Automatic Essay Grading

Statistical Methods in AI Course Project
Pranav Dhakras, Sahil Chelaramani and Josyula Gopala Krishna
Team Number 32

Abstract

In the task of essay grading we assign a grade to an essay in the most human-like manner possible. We tackle this problem by building statistical models which try and predict how a human would evaluate an essay. In this paper, the models we use are Support Vector Machines, Decision Trees, Random forests and Recurrent Neural Networks (LSTMs). We also propose to use the technique of boosting with Decision Trees.

1. Introduction

Essays form a substantial portion of formally testing students' command and fluency in a language. They are useful to judge an individual's grip over language by testing for multiple aspects such as vocabulary, fluidity, grammatical command, originality as well as creativity. The subjective nature of an essay makes it difficult to grade it uniformly across many human graders. In addition, human graders tend to unknowingly grade an essay with their own biases towards the subject matter presented. Manual grading has another major drawback, the time required to grade essays can be significantly high. These were the main motivations to develop an automatic essay grading system. Thus, the system should try to emulate a grading scheme similar to how a tutor would while eliminating the inconsistencies and biases and reducing the cost of grading. There has been research on automatic essay grading since the 1960s. The earliest systems, such as PEG (Page and Petersen, 1995), based their grading on the surface information from the essay. For example, the number of words and commas were counted in order to determine the quality of the essays (Page, 1966). These systems were successful to some extent however, they fail to capture aspects like grammatical correctness and fluidity of language. Much research has been conducted in the field most notably by Educational Testing Service (ETS) which conducts several online as well as offline educational tests and assessments. The problem of automatic essay grading is thus twofold. First, involving identifying and extracting of the key features of an essay and secondly, applying machine learning techniques to grade future essays. Both these task contribute significantly to the effectiveness of a successful automatic essay grading system and hence, in this project we have focused on a balance between the two. We propose to test and tweak multiple popular machine learning techniques such as Support Vector Regressor, Artificial Neural Networks, Random Forests and Boosting on two different sets of features, as described below.

2. Features

2.1. Feature Selection

There are a multitude of ways to eliminate redundant features. The simplest being a brute-force search across the feature space and evaluating every single combination of features for the model. As one might imagine, this would be an expensive prospect, and hence we need a methodical way to select features. There are two popular methods to attack this problem.

1. ***Dimensionality reduction*:** This method effectively combines features in some linear or nonlinear manner and retains a reduced set of features. The problem with this method is that it is expensive to compute for a small number of features and in the case of our own feature set, with the growth in computation power it becomes beneficial to just simply use feature elimination or domain expertise to eliminate features.

2. ***Feature Selection*:** This method iteratively evaluates the validity of features and eliminates them based on their relative importance to the task. As such, paired with cross validation, we can simply evaluate how effective a feature is, based on some metric such as weight assigned to the feature or its weak correlation to the answer. There are two well-known methods for feature selection -

    a. ***Forward Feature selection*:** This method evaluates the features in a bottom up manner and combines the two most effective features iteratively, and reports the best feature selection across all validation runs.

    b. ***Recursive Feature elimination*:** This method first evaluates all the features and trains a model on them. The features that least contribute to the outcome are eliminated. We believe this method is superior to the forward feature selection because forward feature selection may eliminate a pair of features which is strongly correlated to the answer early on based on a biased or faulty metric.

We have naturally used recursive feature elimination in this experiment.

2.2. Our Feature Set

We used Kaggle's Automatic Essay Scoring competition dataset to extract features. The dataset was cleaned and and appropriately pre-processed before extracting the features. We used the Python NLTK library for the tokenizer, extracting Parts-Of-Speech (POS) tags, and also to check for foreign words. In addition, we have used other third party libraries, PyEnchant and Vader for spellchecking and sentiment analysis of essays.

At the syntactic and lexical level we extracted the following features:

- Bing Snippets: This feature was used as a metric to judge how well the essay actually answered the question asked in terms of content. The idea is that we perform a Bing

search for keywords obtained from the question. Now, each of the results of the Bing search have their own descriptions, we extract keywords from these description and count how many occur in the essay.

- Sentiment: The reason for including sentiment was that we think that a well written essay evokes some sense of emotion in the reader. We try to quantify this emotion as positive, negative or neutral. We judge this line by line of the essay using the Vader library by summing the total positive, negative and neutral sentiment values across all lines in the essay.

- Unique N-grams count: Unique unigram, bigram, trigram count extracted from the essay.

- Long Word Count: The number of words which are present in the english language and have a length above seven.

- Part of Speech Count: Count of the number of nouns, verbs, adjectives, adverbs and foreign words in the Essay.

- Spelling Error Count: The number of words that either are not recognized or have possibly been misspelled. We used the PyEnchant library to help us figure this out.

- Essay length: The number of words in the essay.

- Sentence Count: The number of sentences in the essay.

We applied recursive feature elimination for the above features on our dataset to extract the most informative (or discriminative) features. These best features turned out to be:

- Positive sentiment
- Negative sentiment
- Adjective count
- Adverb count
- Long word count
- Unique N-grams count
- Sentence count

Some of these feature selections seem intuitive, for example, a long essay need not always mean that it will be rated well and tutors may actually prefer shorter, less verbose and precise essays. However, others such as elimination of noun and verb counts may be counterintuitive

because these words define the content of the essay and especially the capture the key elements in the text. One possible explanation of this phenomenon would be that many students will perhaps use nouns and verbs in similar numbers. The use of adjectives and adverbs, on the other hand, capture the style of writing, creativity and grip over the language more effectively.

2.3. GloVe

Another alternative feature representation we came across to represent the essay is known as GloVe. GloVe was introduced by Jeffrey Pennington, Richard Socher and Christopher D. Manning in 2014. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. The method works by building a co-occurrence matrix of all words in the corpus (Which words occur with which other words) and then perform a matrix factorization to get our word vectors. The problem with this approach is that high frequency words tend to heavily affect the similarity measure. The way GloVe gets around this problem is, rather than using the counts directly, it uses the ratio of counts. The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well. For this reason, the resulting word vectors perform very well on word analogy tasks, such as those examined in the word2vec package. To deal with co-occurrences that happen rarely or never – which are noisy and carry less information than the more frequent ones – the authors use a weighted least squares regression model. So then they factorize this matrix to yield a lower-dimensional (word x features) matrix, where each row now yields a vector representation for each word. In general, this is done by minimizing a "reconstruction loss" which tries to find the lower-dimensional representations which can explain most of the variance in the high-dimensional data.

3. Architecture of the System

3.1. Overview

There are two commonly used approaches for the essay:

1. The essay is compared to the ones that were assessed by human-graders and the grade given to a new essay is based on similar essays present.

2. The essay is compared to the essay topic and the grade is given based on the similarity to these materials.

In this project we have tried to use the first approach entirely. Along with this we have also tried to incorporate principles behind topic related models. We look for phrases which can aptly describe important concepts present in an essay and then used these concepts as a feature for further analysis. After the features were obtained which shall be further explained in the coming section, we applied four different learning models.

These four models are:

1. Support Vector Regression

2. Decision Tree and Random Forests

3. Boosting with Decision Trees

4. RNNs: Long Short Term Memories

3.2. Support Vector Regression

Support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyse data used for classification and regression. Given a set of labelled points SVMs try and find the best possible separating boundary for them. A SVM only considers the points while lie on the boundary or near the boundary (including outliers) while finding the decision boundary. One of the most important ideas in SVMs is that presenting the solution by means of small subset of training points gives enormous computational advantage. SVMs contain many important characteristics of margin maximization algorithms and a key idea is that a nonlinear function is used to map the given data into higher dimensional kernel induced feature space which may make the classification or regression task much simpler in that space. The effectiveness of the system is not depended on the dimensionality of the higher order feature space but more closely tied to choosing the “right” kernel to perform the mapping to higher dimensional feature space. Another interesting characteristic of SVMs is that the mapping to higher dimensions has little or no effect on the computational complexity of finding the best solution. SVMs are characterized by usage of kernels, absence of local minima, sparseness of the solution and capacity control obtained through the use of a soft margin, or on number of support vectors, etc. SVMs can be applied to both classification as well as regression problems. Since SVMs are not based on iterative or gradient based search techniques they are not likely to be stuck in local minima and usually return the global optimum solution for the problem at hand.

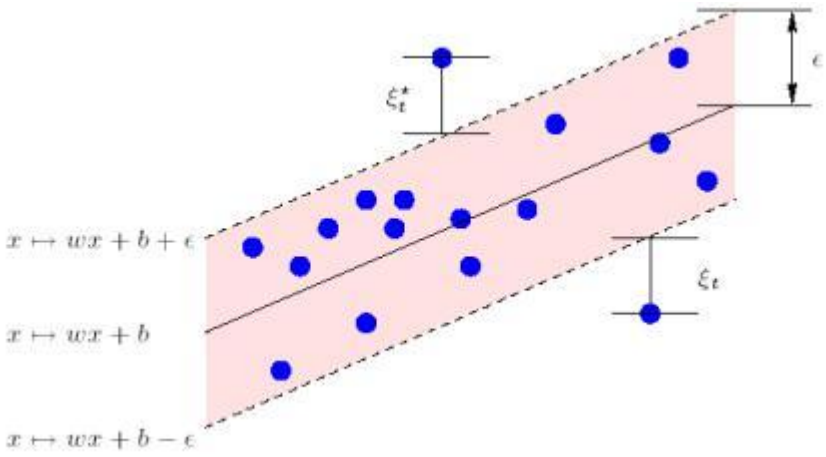

Figure 1: Idea of support vectors for linear classification

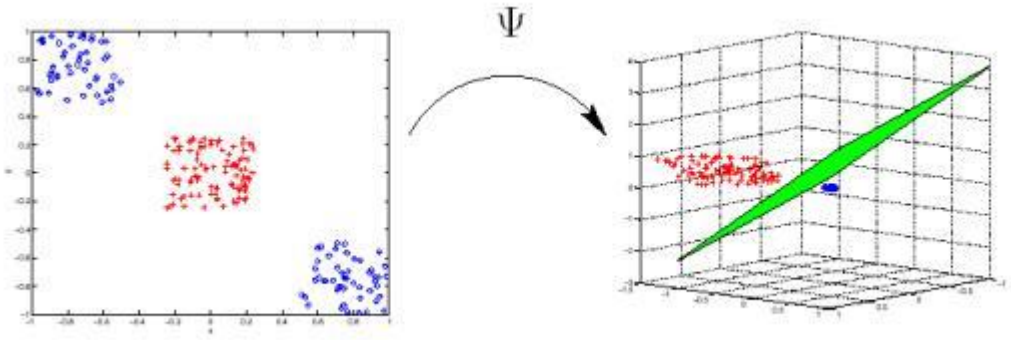

Figure 2: Mapping points to higher dimensional space to make them linearly separable

In SVM regression, the input is first mapped onto a higher dimensional feature space using some fixed (nonlinear) mapping, and then a linear model is constructed in this feature space. It then tries to minimize the margin by performing linear regression in the higher order space. It also introduces slack variables to adjust for the complexity of the model.

For this purpose of SVR, we took help of the Python library SciKit Learn. SVR was tested with 3 different kernels: linear, radial basis function and polynomial kernels.

3.3. Decision Trees and Random Forests

Decision trees are a popular method for various machine learning tasks. A decision tree is a non-cyclic graph like structure, closely resembling a flowchart. In a decision tree each branch takes some decision depending upon specific features while each leaf node represent a class label (in case of classification) or a value (in case of regression). Tree learning is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, decision trees by themselves tend to over fit to the training data, especially if they are allowed to grow very deep. The reason for this is a deep tree will (in the worst case) have a separate leaf not for each data sample and will then fail to provide a generalized solution. Such decision trees have very

high variance. One way to reduce the problem of overfitting in decision trees is to limit the depth of the tree or limit the minimum number of data samples that each leaf node represents. This helps building a more generalized model. Another approach is to use a random forest, i.e., a collection of randomly initialized trees.

A random forest is a meta-estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. By taking the average (or in some cases vote) of the output of multiple trees Random decision forests avoid overfitting to their training set a problem that is often faced by individual decision trees. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance of the final model.

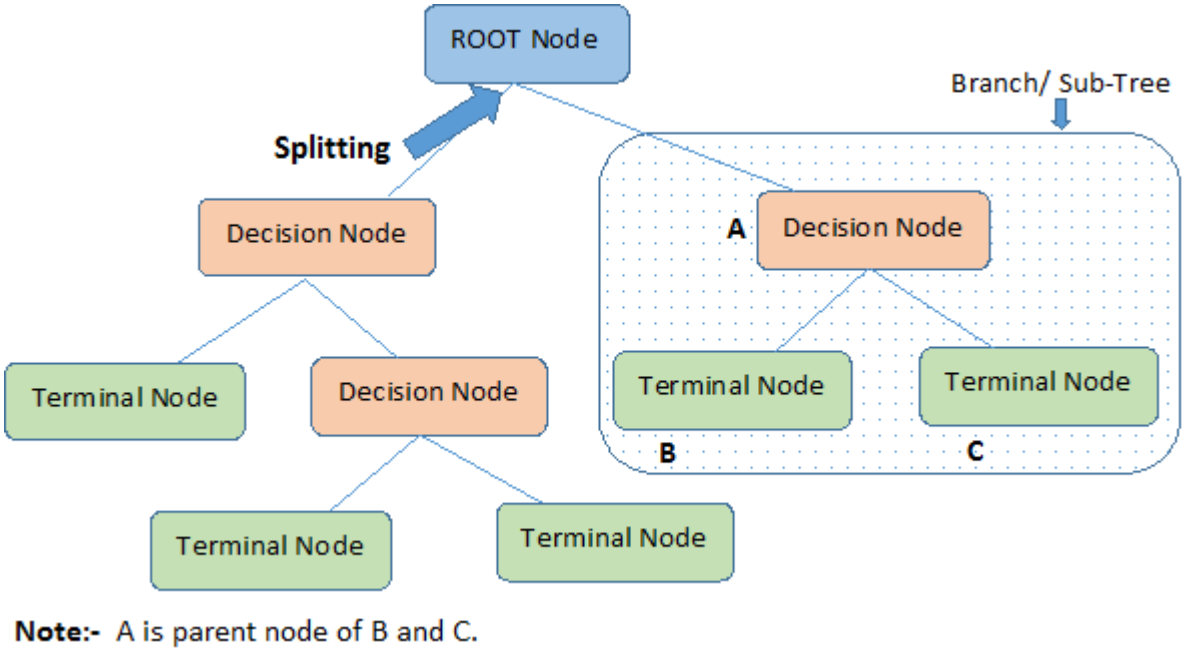

Figure 3: Concept of decision trees

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = x1, ..., xn with responses Y = y1, ..., yn , bagging repeated (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B :

1. Sample, with replacement, n training examples from X , Y ; call these X b , Y b .

2. Train a decision or regression tree f b on X b , Y b .

After training, predictions for unseen samples can be obtained by predicting the sample against multiple decision trees in the forest and then obtaining an average or median or mode of all such predictions.

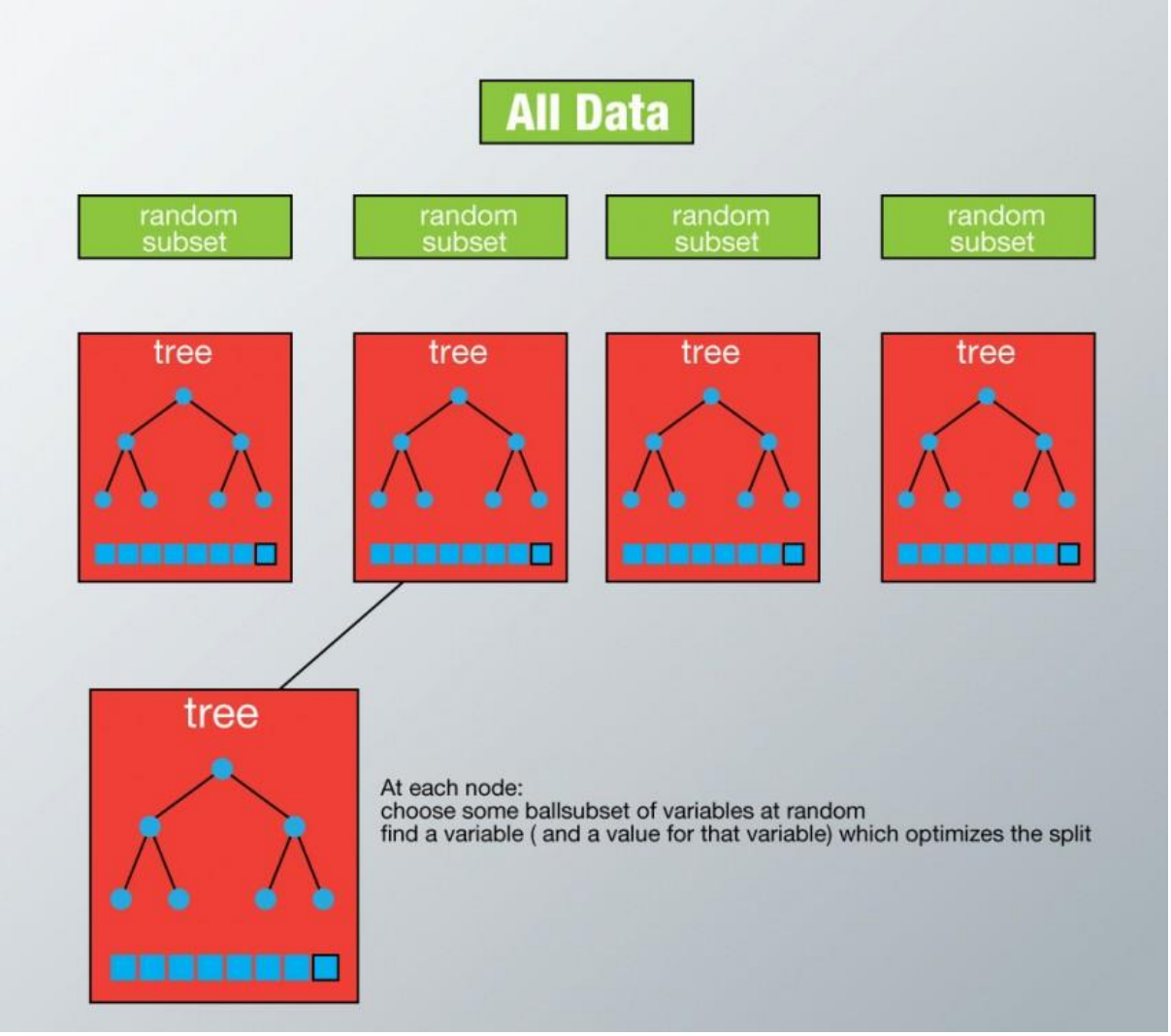

Figure 4: Random Forest – A collection of multiple randomly generated Decision Trees

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. The way to achieve non correlated trees in a forest is to train them all on different subsets of data.

The application of random forest learning for classification as well as regression is very similar with one major difference, that is, in the classification scenario the leaf nodes represent class labels whereas in regression the leaf nodes hold the predicted value of the output variable. Thus for classification the use of a “vote” or mode may be more intuitive while combining multiple decision trees whereas for regression mean or median or mode may be used depending upon the specific use case.

One disadvantage of using random forests for regression is that it cannot really predict a value beyond the range of values it encountered during training. However, in the task of essay grading the output always remains in a predefined range (say a score from 0-10). Thus, our implementation of random forest for essay grading was devoid of this issue.

We have built a random forest model from scratch to train and test over the acquired dataset.

3.4. Boosting with Decision Trees

The idea behind boosting is that given a set of weak learners (Algorithms that can weakly predict the result), we combine them to come up with a strong learner. To find weak rule, we apply learning algorithms with different distributions. If there is any error caused by first learning algorithm, then we pay higher attention to those observations and we apply the next learning algorithm. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule. An illustrative diagram is shown below.

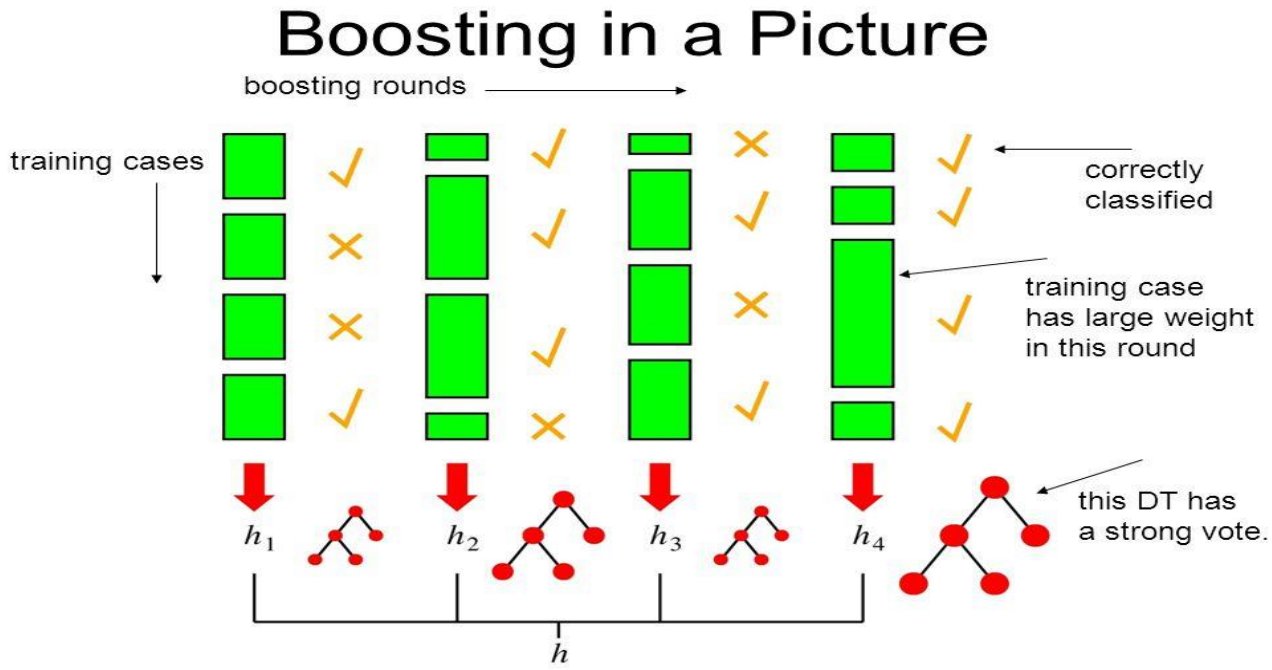

Figure 5: Idea of Boosting algorithm

There are many boosting algorithms which use other types of engine such as:

- AdaBoost (Adaptive Boosting)
- Gradient Tree Boosting
- XGBoost

In this paper we have used Adaboost.

3.5. RNNs - Long Short Term Memory

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. There are two major processes involved in the learning process of a neural network.

1. Feed Forward Operation

2. Back Propagation

The problems with traditional neural networks are that most tasks performed by human beings depend upon pre-learned knowledge about the world. Traditional networks cannot recall ideas they have learned to make more informed decisions. As a method to enable neural networks to "store" their learned knowledge from previous states, we introduce a feedback loop from a neuron to itself, as illustrated in the following diagram:

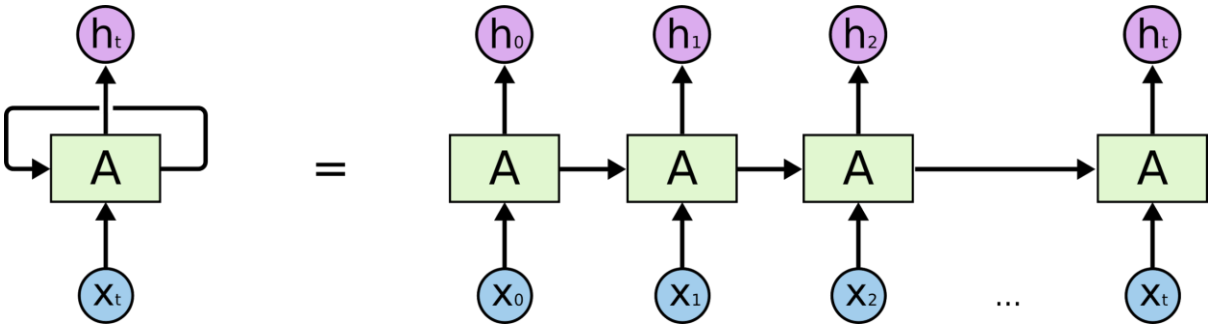

Figure 6: Recurrent Neural Networks - LSTMs

The above structure is known as a recurrent neural network. You can image that the one self-loop is unfolded into N different distinct time steps. The figure on the right closely resembles a list structure used in programming and one can simply infer that this structure is why RNNs can deal with remembering sequences it has already seen before. In theory RNNs are perfect to capture long term dependencies, but don't perform as well as the size of N increases. Bengio et al (2014) go into the details about why they don't work as well. An extension to this idea is that an LSTM, which is a particular architecture of RNNs, allows the network to remember long term dependencies. These kinds of networks also happen to be much easier to stabilize as compared to vanilla RNNs and hence are now the default choice for RNN architectures. In this paper, we use an LSTM model as a basis, and vary different parameters of the network to obtain the results.

4. Experiments

We used two different feature sets. We conducted tests on the test data using multiple models with varied configurations. Later we calculated the quadratic kappa for each of the models on each of the datasets to make an informal comparison between the models.

SciKit Learn library's implementation of SVR was used. It was tested with three kernels, linear, RBF and polynomial. For Random Forest regression we have compared our implementation with that of SciKit Learn library. SciKit Learn Random Forest Regressor was test with 10 and 50 trees for both feature sets. Our own implementation of Random Forest was testing by using various values for both the parameters, number of trees in the forest as well as minimum data at a leaf node.

An LSTM implementation from the TensorFlow library was also tested against both feature sets. The number of hidden layers for LSTM were tweaked to learn about the most suited LSTM architecture for the problem at hand.

Finally, we have also tested the AdaBoost boosting algorithm from SciKit learn for boosting the results of multiple decision trees (again from SciKit Learn).

The results of all the above experiments have been documented below.

Feature set	Classifier	Params	Average Quadratic Kappa
Our features	SciKit - SVR	Linear Kernel	0.7735
Our features	SciKit - SVR	Poly Kernel	0.685
Our features	SciKit - SVR	RBF Kernel	0.7605
Our features	SciKit - RandomForestRegressor	trees = 10	0.7854
Our features	SciKit - RandomForestRegressor	trees = 50	0.8089
Our features	Our RandomForest	trees = 10 min_data = 10	0.8176
Our features	Our RandomForest	trees = 50 min_data = 10	0.8052
Our features	Our RandomForest	trees = 10 min_data = 30	0.8151
Our features	Our RandomForest	trees = 50 min_data = 30	0.696

Our features	LSTM	Hidden layers = 1	0.8673
Our features	LSTM	Hidden layers = 3	0.8979
Our features	LSTM	Hidden layers = 5	0.9081
Our features	AdaBoost + DecisionTrees	max_depth = 1	0.8387
Our features	AdaBoost + DecisionTrees	max_depth = 3	0.8287
Our features	AdaBoost + DecisionTrees	max_depth = 9	0.8251
Glove	SciKit - SVR	Linear Kernel	0.7965
Our features	SciKit - SVR	Poly Kernel	0.696
Glove	SciKit - SVR	RBF Kernel	0.7605
Glove	SciKit - RandomForestRegressor	trees = 10	0.8013
Glove	SciKit - RandomForestRegressor	trees = 50	0.8114
Glove	Our RandomForest	trees = 10 min_data = 10	0.758
Glove	Our RandomForest	trees = 50 min_data = 10	0.7307
Glove	Our RandomForest	trees = 10 min_data = 30	0.7593
Glove	Our RandomForest	trees = 50 min_data = 30	0.7134
Glove	LSTM	Hidden layers = 1	0.8856
Glove	LSTM	Hidden layers = 3	0.9081
Glove	LSTM	Hidden layers = 5	0.9287
Glove	AdaBoost + DecisionTrees	max_depth = 1	0.8176
Glove	AdaBoost + DecisionTrees	max_depth = 3	0.7928
Glove	AdaBoost + DecisionTrees	max_depth = 9	0.7714

We can see that SVR does not perform as well as the other classifiers. In addition, using AdaBoost with decision trees has produced better results than both the Random Forest implementations. Finally, the accuracy of LSTM for essay grading was truly appreciable with GloVe vectors, and even outperform all of the other methods.

5. Future Work

As our experiments show, the best results have been obtained using LSTMs. One problem with LSTMs is that they are painfully slow to train, hence trying out different kinds of deep networks like convolutional networks or different architectures of RNNs may further improve the performance. Additionally, in this paper we use two sets of features GloVe and our own feature set, but these are by no means the only feature spaces that could be considered. Another way to approach this problem is that we could try combining our features with GloVe vectors to capture semantic and syntactic information better.

6. Conclusion

Thus, the project of automatically grading essays looks positive and can be improved with better feature sets which capture details about sentence structure and arrangement of ideas rather than just focusing on the surface features and keywords in the essay. Additionally, by leveraging nonlinear features we address the problem of people gaming the system. Going forward, we believe there will be systems which will effectively grade essays in the most humanlike manner as possible.

7. References

Murray, and Orii *"Automatic Essay Scoring"* http://www.cs.cmu.edu/~norii/pub/aes.pdf

Mahana, Johns and Apte *"Automatic Essay Grading with Machine Learning"* http://cs229.stanford.edu/proj2012/MahanaJohnsApte-AutomatedEssayGradingUsingMachineLearning.pdf

AnalyticsVidhya *"Tutorial on Tree based Modelling"* https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/#one

Attali, Yigal, and Jill Burstein. *"Automated essay scoring with e-rater V. 2."* The Journal of Technology, Learning and Assessment 4.3 (2006). https://www.ets.org/Media/Research/pdf/RR-04-45.pdf

Kaggle. *"Develop an automated scoring algorithm for student-written essays."* (2012). https://www.kaggle.com/c/asap-aes

Robertson, Stephen. *"Understanding inverse document frequency: on theoretical arguments for IDF."* Journal of documentation 60.5 (2004): 503-520.

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. *"Glove: Global Vectors for Word Representation."* EMNLP. Vol. 14. 2014. 10

Huyenn and Lucio Dery *"Neural Networks for Automated Essay Grading"* https://cs224d.stanford.edu/reports/huyenn.pdf

Hochreiter, Sepp, and Jrgen Schmidhuber. *"Long short-term memory."* Neural computation 9.8 (1997): 1735-1780

Colah *"Understanding LSTMs"* http://colah.github.io/posts/2015-08-Understanding-LSTMs/