AUTOMATIC ESSAY GRADING

Vaibhav Kumar, Mrinal Dhar, Chanakya Malireddy
Team Number 30
Statistical Methods in AI Course Project

Abstract

Random forests is a notion of the general technique of random decision forests that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. In this project we have tried to use Random Forest as a regression technique for essay grading. We also used Support Vector Regression(SVR) and Artificial Neural Network(ANN) for the same. Along with this, Hidden Markov Model was implemented for creating part of speech tags (POS tags) in the preprocessing stage. Also, we implemented text rank technique in order to grab semantic information out of the essays.

1. Introduction

Essays are a useful tool tools to assess learning outcomes such as ability to recall, organise, and express one's ideas in writing. Many researchers claim that the subjective nature of essay assessment leads to variation in grades awarded by different human assessors, which is perceived by students as a great source of unfairness. Also one of the main motivations behind developing automated essay assessment systems are to decrease the time in which students get feedback for their writings, and to reduce the costs of grading. The assumption in most of the systems is that the grades given by the human assessors describe the true quality of an essay.

Thus, the aim of the systems is to "simulate" the grading process of a human grader and a system is usable only if it is able to perform the grading as accurately as human raters. An automated assessment system is not affected by errors caused by lack of consistency, fatigue or bias, thus it can help achieving better accuracy and objectivity of assessment (Page and Petersen, 1995).

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 es- says (Page, 1966). Although these

kinds of systems performed considerably well, they also received heavy criticism (Page and Petersen, 1995). Some researchers consider the use of natural language as a feature for human intelligence (Hearst et al., 2000) and writing as a method to express the intelligence. Based on that assumption, taking the surface information into account and ignoring the meanings of the content is insufficient. Then we can witness the advent of essay grading system purely based on semantic techniques like Latent Semantic Analysis (LSA). LSA has produced promising results in content analysis of essays.

In this project we have tried to use different methods for assessment of essays. At first we employed the method of Support Vector Regression. In the next stage we used Artificial Neural Network. Then we employed the Random Forest based regression tree for grading of essays.

2 Essay Grading System

We have developed a system for automated assessment of essays. In this section, we explain the basic architecture of the system and describe the methods used to analyze essays.

2.1 Architecture of the System

There are two commonly used approaches for the essay:

1. The essay graded is compared to the one's they were assigned by the human-graders and the grade given to a new essay is based on the closest similar essays present.

2. The essay is compared to the essay topic related model 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 have tried to look for phrases which can aptly describe important concepts present in an essay. We have then used this concepts as a feature for further analysis.

After the features were obtained which shall be further explained in the coming section, we applied three different learning models.

These three models were based on :

1. Support Vector Regression (SVR)
2. Artificial Neural Network (ANN)
3. Random Forest (Decision Tree)

Some of the concepts which considered and used while extracting out features were:

1. Hidden Markov Model (for POS tags)
2. Latent Semantic Analysis (idea was considered but later discarded)
3. Text Rank (for finding phrase similarity used instead of LSA)

The featured are first fed to these models. When these models get ready after training, the test essays are fed to them. These models then predict outputs for these essays.

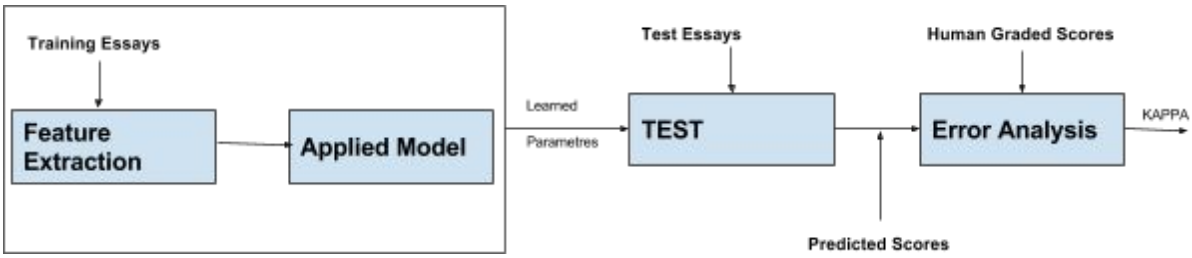
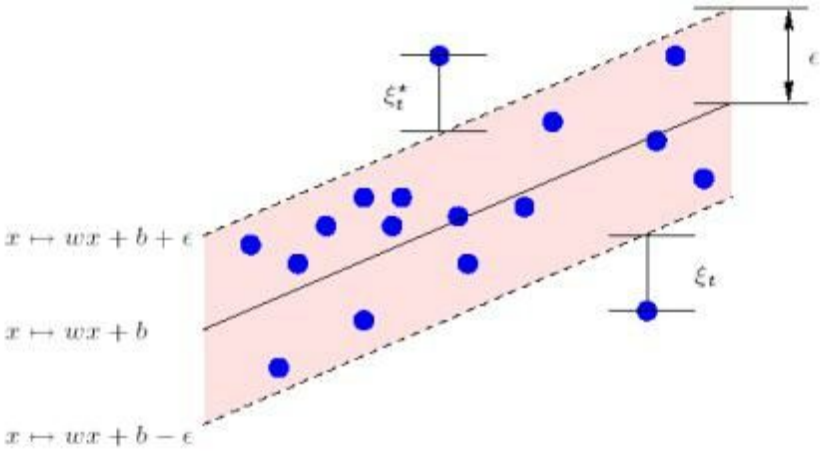


FIG . Displays the overall setting of the system

2.2 Support Vector Regression

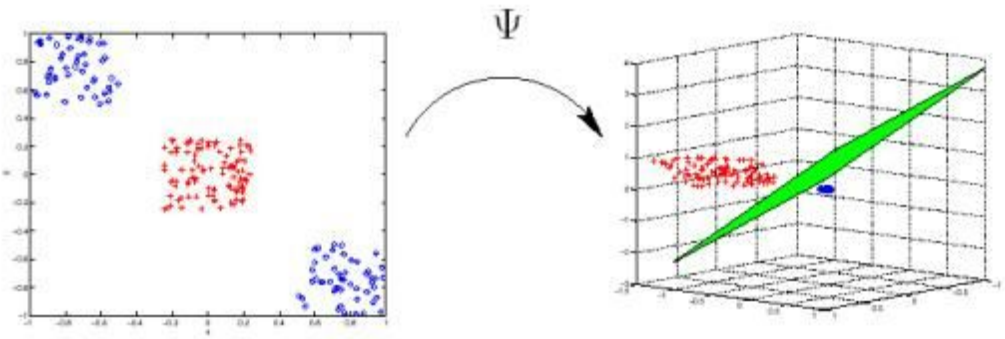
Support Vector Machines are very specific class of algorithms, characterized by usage of kernels, absence of local minima, sparseness of the solution and capacity control obtained by acting on the margin, or on number of support vectors, etc. Support Vector Machine can be applied not only to classification problems but also to the case of regression. Still it contains all the main features that characterize maximum margin algorithm: a nonlinear function is learned by linear learning machine mapping into high dimensional kernel induced feature space. The capacity of the system is controlled by parameters that do not depend on the dimensionality of feature space. One of the most important ideas in Support Vector Classification and Regression cases is that presenting the solution by means of small subset of training points gives enormous computational advantages. Using the epsilon intensive loss function we ensure existence of the global minimum and at the same time optimization of reliable generalization bound.



In SVM regression, the input $\mathbf{x}$ is first mapped onto a m-dimensional feature space using some fixed (nonlinear) mapping, and then a linear model is constructed in this feature space. Using mathematical notation, the linear model (in the feature space).

$$f(\mathbf{x}, \omega) = \sum_{j=1}^{m} \omega_j g_j(\mathbf{x}) + b$$

where $g_j(\mathbf{x}), j = 1, \ldots, m$ denotes a set of nonlinear transformations, and b is the "bias" term. Often the data are assumed to be zero mean (this can be achieved by preprocessing), so the bias term is dropped.



The quality of estimation is measured by the loss function $L(y, f(\mathbf{x}, \omega))$. SVM regression uses a new type of loss function called ε -insensitive loss function proposed by Vapnik:

$$L_\varepsilon(y, f(\mathbf{x}, \omega)) = \begin{cases} 0 & \text{if } |y - f(\mathbf{x}, \omega)| \le \varepsilon \\ |y - f(\mathbf{x}, \omega)| - \varepsilon & \text{otherwise} \end{cases}$$

The empirical risk is:

$$R_{emp}(\omega) = \frac{1}{n} \sum_{i=1}^{n} L_\varepsilon(y_i, f(\mathbf{x}_i, \omega))$$

SVM regression performs linear regression in the high-dimension feature space using ε -insensitive loss and, at the same time, tries to reduce model complexity by

minimizing $\| w \|^2$. This can be described by introducing (non-negative) slack variables $\xi_i , \xi_i^* \quad i = 1,\dots n$, to measure the deviation of training samples outside ε -insensitive zone. Thus SVM regression is formulated as minimization of the following functional:

min $$\frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*)$$

s.t. $$\begin{cases} y_i - f(\mathbf{x}_i, w) \le \varepsilon + \xi_i^* \\ f(\mathbf{x}_i, w) - y_i \le \varepsilon + \xi_i \\ \xi_i , \xi_i^* \ge 0, i = 1,\dots, n \end{cases}$$

This optimization problem can transformed into the dual problem and its solution is given by

$$f(\mathbf{x}) = \sum_{i=1}^{n_{SV}} (\alpha_i - \alpha_i^*) K(\mathbf{x}_i, \mathbf{x})$$ s.t. $0 \le \alpha_i^* \le C$, $0 \le \alpha_i \le C$,

where n_{SV} is the number of Support Vectors (SVs) and the kernel function

$$K(\mathbf{x}, \mathbf{x}_i) = \sum_{j=1}^{m} g_j(\mathbf{x}) g_j(\mathbf{x}_i)$$

For this purpose of SVM, we took help of a python library.

2.3 Artificial neural network

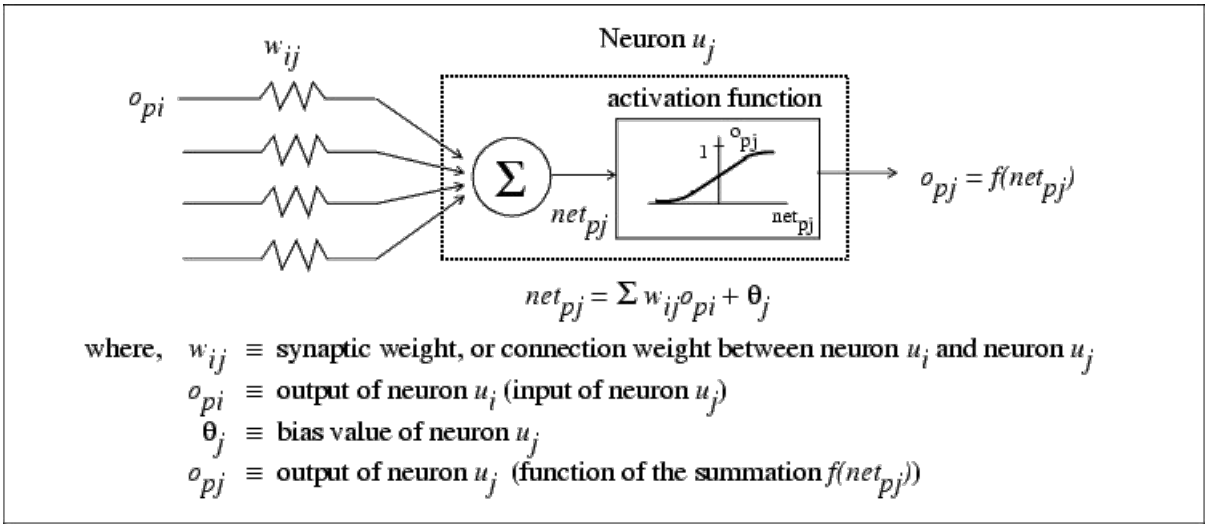
An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system.

It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

There are two major processes involved in the learning process of a neural network.

1. Feed Forward Operation

2. Back Propagation



The backpropagation equations go as follows:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net_j}}\frac{\partial \mathrm{net_j}}{\partial w_{ij}}$$

Calculating the partial derivative of the error with respect to a weight w_{ij} is done using the chain rule twice we obtain the above equation.

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net_j}} = \begin{cases} (o_j - t_j)\varphi(\mathrm{net}_j)(1-\varphi(\mathrm{net}_j)) & \text{if } j \text{ is an output neuron,} \\ (\sum_{l\in L}\delta_l w_{jl})\varphi(\mathrm{net}_j)(1-\varphi(\mathrm{net}_j)) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

To update the weight w_{ij} using gradient descent, one must choose a learning rate, α. The change in weight, which is added to the old weight, is equal to the product of the learning rate and the gradient, multiplied by -1:

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

In a similar fashion to this, we also calculate the change in the weights if the of hidden to the output layer.

We used the sigmoid function as the transfer function at the different layers i.e the output and the hidden layers.

2.4 Random Forest Decision trees

Random forests is a notion of the general technique of random decision forests that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Decision trees are a popular method for various machine learning tasks. Tree learning "come[s] closest to meeting the requirements for serving as an off-the-shelf procedure for data mining", say Hastie *et al.*, because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate.

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, because they have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. 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.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners.

Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, 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.
3.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x')$$

or by taking the majority vote in the case of 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.

Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

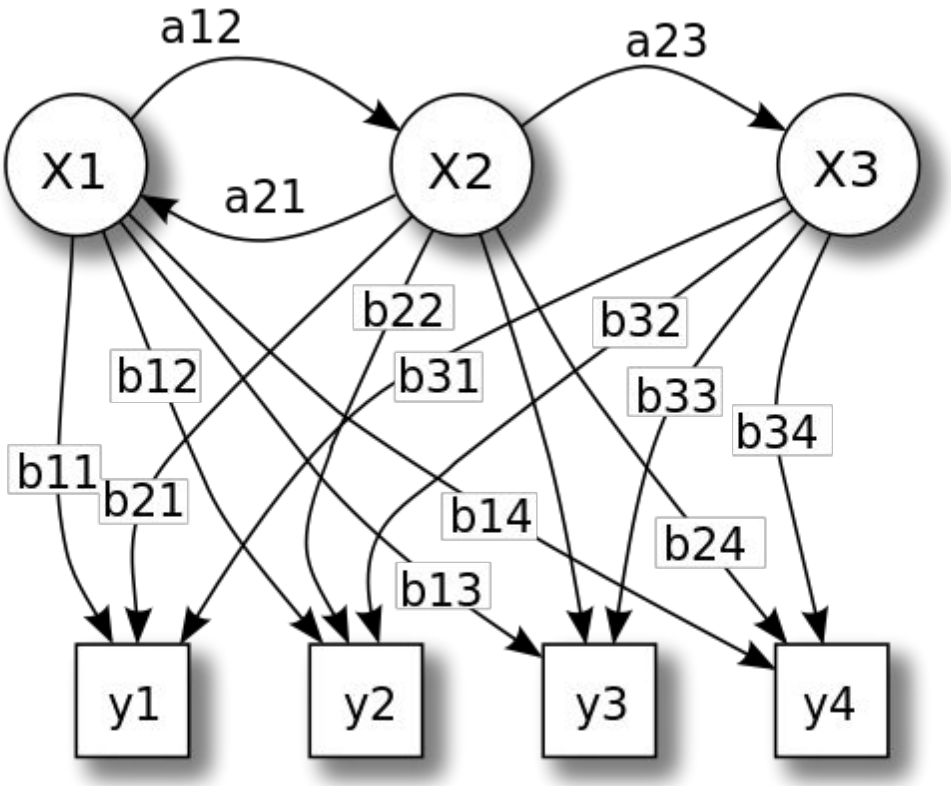
We used the random forests in order to give an output of the score that would be assigned to a test essay that would be provided at that point of time.

2,5 Hidden Markov Model

A **hidden Markov model** (**HMM**) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (*hidden*) states. A HMM can be presented as the simplest dynamic Bayesian network.

A hidden Markov model can be considered a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other.

A pictorial description of the HMM has been provided below:



Our main purpose of using the HMM was to assign POS tags to the words in the sentences presented in the different essays. We then used the Viterbi algorithm for the purpose of decoding the sequence:

Suppose we are given a hidden Markov model (HMM) with state space S, initial probabilities π_i of being in state i and transition probabilities $a_{i,j}$ of transitioning from state i to state j. Say we observe outputs $y_1, \ldots, y_T$. The most likely state sequence $x_1, \ldots, x_T$ that produces the observations is given by the recurrence relations:

$$\begin{aligned} V_{1,k} &= \mathrm{P}(y_1 \mid k) \cdot \pi_k \\ V_{t,k} &= \max_{x \in S} \left(\mathrm{P}(y_t \mid k) \cdot a_{x,k} \cdot V_{t-1,x} \right) \end{aligned}$$

Here $V_{t,k}$ is the probability of the most probable state sequence $\mathrm{P}(x_1, \ldots, x_T, y_1, \ldots, y_T)$ responsible for the first t observations that have k as its final state. The Viterbi path can be retrieved by saving back pointers that remember which state x was used in the second equation.

Let $\mathrm{Ptr}(k, t)$ be the function that returns the value of x used to compute $V_{t,k}$ if $t > 1$, or k if $t = 1$. Then:

$$\begin{aligned} x_T &= \arg\max_{x \in S}(V_{T,x}) \\ x_{t-1} &= \mathrm{Ptr}(x_t, t) \end{aligned}$$

Here we're using the standard definition of arg max.

The complexity of this algorithm is $O(T \times |S|^2)$.

2.6 TextRank

TextRank is a keyphrase extraction algorithm. While supervised methods have some nice properties, like being able to produce interpretable rules for what features characterize a keyphrase, they also require a large amount of training data. Many documents with known keyphrases are needed. Furthermore, training on a specific domain tends to customize the extraction process to that domain, so the resulting classifier is not necessarily portable, as some of Turney's results demonstrate. Unsupervised keyphrase extraction removes the need for training data.

It approaches the problem from a different angle. Instead of trying to learn explicit features that characterize keyphrases, the TextRank algorithm exploits the structure of the text itself to determine keyphrases that appear "central" to the text in the same way that PageRank selects

important Web pages. Recall this is based on the notion of "prestige" or "recommendation" from social networks. In this way, TextRank does not rely on any previous training data at all, but rather can be run on any arbitrary piece of text, and it can produce output simply based on the text's intrinsic properties. Thus the algorithm is easily portable to new domains and languages.

TextRank is a general purpose graph-based ranking algorithm for NLP. Essentially, it runs PageRank on a graph specially designed for a particular NLP task. For keyphrase extraction, it builds a graph using some set of text units as vertices. Edges are based on some measure of semantic or lexical similarity between the text unit vertices. Unlike PageRank, the edges are typically undirected and can be weighted to reflect a degree of similarity.

Example of our keyphrase output for the above paragraph given by our algorithm would be as follows:

similarity
Furthermore
characterize
PageRank
recommendation
different
extraction algorithm
TextRank algorithm
interpretable
TextRank
Essentially
important
particular
intrinsic
structure
training
graph-based
previous training
explicit
specific
keyphrase extraction
arbitrary

3 Experiments

3.1 Procedure and Materials

To analyse how our models fared, we tested it on dataset that was acquired from Kaggle. There was data related to essays based on different topics. First, we had to preprocess the data in order to extract out important and valuable features from it.

We mainly used NLTK for our feature extraction, however the POS Tagger was implemented by us using HMM and then was further used to gather different features. At the syntactic and lexical level we extracted the following features:

1. Numerical Features:
    a. Total word count per essay
    b. Average word length per essay
    c. Noun Count
    d. Verb Count
    e. Adverb Count
    f. Adjective Count
    g. Punctuation Count
    h. Spelling Error
    i. Short Word Count
    j. Long Word Count
    k. Comma Count
2. Lexical Diversity
    a. Yule's I Index
    b. Different types of POS tags available
3. Semantic Concepts
    a. Correlation with the top 10 phrases (keyphrases) that are very essential to the very topic itself.

We used the tokenizer at different levels in order to split the essays into the minimal parts that were required by us to carry this feature analysis. The tokenizer was also implemented by us.

Explanation for the third part above where we talk about finding the semantic concepts is discussed as follows (we used the following procedure):

1. Find the top 10 keyphrases that an ideal essay pertaining to a particular topic would possess. For this purpose we split the data into different parts, one part of which was specifically meant for this purpose. After we extracted the top 10 keyphrases from the all the essays combined at this step we move further. We call these the global keyphrases.
2. We now represent these 10 different keyphrases as 10 different features.
3. For a particular essay in the training set / test set, the top 10 keyphrases were found for it.
4. These top 10 keyphrases were then compared to the global keyphrases.
5. Then the measure of relatedness of these keyphrases were found with that of the global features, An average of all these values were taken corresponding to a particular global value.
6. The way in which we find out the keyphrases is already talked about in the textrank algorithm in the previous section.

We then use the forward feature selection procedure to find out which all featured we could consider finally. To do this, we took a feature and regressed over it individually and if it increased the efficiency then we considered it to added to our final list of features

and if it did not then we left it. In this way we accounted for all the features that we selected.

In the end, we had a combination of three different kinds of features:

1. Lexical
2. Syntactic
3. Semantic

We then implemented our above mentioned techniques and analysed them individually, Our entire implementation was done in python with the help of some other modules that were used for some specific kind of computations.

3.2 Results and Discussions

```
(venv)sankaul@localhost $ python run.py -n ../data/small_TRAIN ../data/models/small ../data_dumps/small -t ../data/test

================================================================================================================================

Training the model...
Processing |################################| 100% | 0 seconds remaining

Model trained and data dumped successfully.

Testing phase begins...
Loading test set 1
Loading test set 2
Loading test set 3
Loading test set 4
... Done.

================================================================================================================================

Processing SVR classifier...

Accuracy per test-set:
1: 0.76849
2: 0.75839
3: 0.79854
4: 0.76600


Processing Neural Net...

Accuracy per test-set:
1: 0.70020
2: 0.69828
3: 0.72918
4: 0.70991


Processing Random Forest...

Accuracy per test-set:
1: 0.80282
2: 0.80011
3: 0.79929
4: 0.81393


================================================================================================================================
```

	SVR	Neural Net	Random Forest
Test set 1	0.76849	0.70020	0.80282
Test set 2	0.75839	0.69828	0.80011
Test set 3	0.79854	0.72918	0.79929
Test set 4	0.76600	0.70991	0.81393
Average	0.772855	0.7093925	0.8040375

The results of our experiments can be seen as tabulated above. We tested our models on 4 different test sets each corresponding to a different topic. We calculated the quadratic weighted kappa for each of the models on each if the datasets.
It can be clearly seen that SVR outweighs that Neural Net and even further, the Random Forest based technique dominates over all of them.

Hence, it can be seen that sometimes even decision trees seem to perform better than other methods like the SVR or ANN.

Throughout the project, we searched for various methods in which we could understand the deep structures hidden inside the essay. It has often been regarded that the essay assessment systems do not consider the imaginative power of the writer. However, we have tried to overcome this by using textrank to find out good phrases that an essay could possess. Earlier, techniques like LSA were available, which are based on singular value decomposition (SVD). These systems, after truncating the matrix, assess the essays based on the closest neighbour of theirs present in the test set. The output that they give is an all over average.

Had we done this, then we would have completely wiped off the syntactic and lexical diversity of an essay. We wanted to account for this factor as well, and along with this, we wanted to come up with a new method to find related concepts of an essay. We had initially tried LSA, but then we came up with a way in which we could utilize the textrank algorithm to identify important concepts in essay and weigh them accordingly.

We then used three different models, namely SVR, ANN and Random Forests, to assess the performance of our system.