International Journal of Computer and Information System (IJCIS)
Peer Reviewed – International Journal
Vol : Vol. 05, Issue 02, June 2024
e-ISSN : 2745-9659
https://ijcis.net/index.php/ijcis/index

Plagiarism Detection Using Artificial Intelligence

1st Ismail Hossain Sadhin, 2nd Tanvir Hassan, 3rd MD. Al Muzahid Nayim
[1,2] Computer Science, American International University Bangladesh (AIUB), Dhaka, Bangladesh
[3]Computer Science, Iowa State University, Ames, Iowa, USA
[1]ismailhossain.sadhin17@gmail.com, [2]tanvirhasan0146@gmail.com, [3]nayim@iastate.edu

Abstract—Presently available plagiarism detection technologies are primarily restricted to string-level comparisons between potentially original texts and suspiciously plagiarized materials. The objective of this research is to enhance the precision of plagiarism identification by integrating Natural Language Processing (NLP) methods into current methodologies. Our proposal is an external plagiarism detection framework that uses various natural language processing (NLP) approaches to examine a set of original and suspicious papers. The techniques not only analyze text strings but also the text's structure, taking text relations into consideration. Preliminary findings using a corpus of short paragraphs that have been plagiarized demonstrate that NLP approach increase the correctness of current methods.

Keywords: NLP, CopyCatch, MOSS, xgboost, Bert, RoBerta, SVM, Glove

I. INTRODUCTION

The Internet's ease of sharing information has made it more common for people to look for literature available online. This has led to more people—especially in academic domains—replicating ideas or works of others without giving due credit. The identification of instances of plagiarism is just as vital as the prevention of such a problem, which should be prioritized for educational reasons.

Numerous approaches have been developed over time to carry out the automatic detection of plagiarism. These include tools for the detection of natural language text and computer programming source code.

Here our suggested system is a model that is based on machine learning. It makes use of Sci-Kit-Learn functionalities that are integrated into modules. In order to detect plagiarism, our proposed method computes many attributes from the programming assignments. From the source code, we calculate many features like the similarity score, the quantity of variables and functions that are underused, etc. Next, we use these features to train an xgboost learning algorithm, and we compare the outcome with Support Vector Machine (SVM). We think that even with heavily obfuscated assignments, our features will be able to identify plagiarism. On the test set, our model achieved 93 percent accuracy using the attributes we suggested.

II. EXISTING APPROACHES

There are various classifications for plagiarism detection techniques. Lancaster (2003) states that techniques can be categorized according to the type of detection methods employed, the accessibility of the system, the number of documents the metrics can process, and the intricacy of the metrics. It has been noted that surface complexity paired measurements have been applied more often than highly accurate multidimensional metrics with substantial structural complexity. This results from the trade-off between precision and processing power. Detection jobs can be quite time-consuming and labor-intensive, even with the aid of powerful computers. The more complex the measurements, the more processing power that is required. For users who have access to personal computers, this is not optimal.

Although Razvan Rosu[IUJI] used BERT and RoBERTa for plagiarism detection, this technique was innovative. They used this technique in hopes that the publications that are comparable to the query document would be rated more pertinently and accurately.

Their goal was to create a reliable plagiarism detection system that can deliver results quickly and accurately. The majority of the time, comparing words to words is necessary for detecting plagiarism, which is laborious to go through a huge document and is susceptible to manipulation by substituting synonyms for words. The use of deep learning, which goes beyond word parsing to comprehend the text, can provide fresh insights into plagiarism detection. The other method they looked at was GLoVE Pennington (2014) in conjunction with TFIDF Ramos (2003) and cosine similarity Thada (2013). This method is chosen because GLoVE can easily achieve two objectives: it can build word embeddings in space vectoring and it takes into account global statistics instead of local ones.

III. METHODOLOGY

The text processing methods and plagiarism detection approaches we implemented in our studies are described in detail in this section.

Pre-processing and NLP techniques:

a) ***Tokenization:*** The practice of dividing text into smaller pieces known as tokens is known as NLP tokenization. Words, sentences, or even characters can serve as these tokens, based on the level of detail needed. A key component of NLP jobs is tokenization, which gives computers the ability to efficiently comprehend and analyse human language. It is useful for machine translation, sentiment analysis, and text classification. To address different language peculiarities, techniques range from basic whitespace-based tokenization to more complex approaches like word embeddings and sub word tokenization.

b) ***Lowercase***: To make the matching more generic, replace all uppercase characters with lowercase letters.

c) ***Stop-word removal***: In text preprocessing, stop-word removal involves filtering out common, non-informative words like "the," "of," and "and," alongside pronouns and articles. These words, known as stop words, carry minimal semantic value and can distort analysis outcomes if retained. By eliminating them, the focus shifts to the substantive content of the text, enhancing the effectiveness of tasks such as sentiment analysis or information retrieval. Techniques utilizing predefined lists or statistical methods are employed to systematically remove these functional words, streamlining subsequent text processing stages.

d) ***Punctuation removal***: Punctuation marks in titles often serve decorative rather than semantic purposes. Their removal streamlines text for NLP models, focusing on content rather than formatting. This pre-processing step ensures cleaner input data, enhancing the model's ability to extract meaningful insights. By disregarding punctuation, the model can better discern linguistic patterns and nuances in the text.

e) ***Part-of-Speech tagging***: Grammar labels such as "noun" or "verb" are assigned to each word using POS tagging, which facilitates linguistic analysis. It helps identify consistent grammatical structures even with word substitutions, enriching NLP tasks. This tagging facilitates parsing, sentiment analysis, and named entity recognition by revealing syntactic functions. By labelling words according to their roles, POS tagging enhances the understanding of text semantics.

f) ***Stemming***: Stemming is a linguistic normalization technique where words are reduced to their base or root forms, facilitating broader analysis. For instance, "product," "produce," and "produced" are all stemmed to "produc," while "computer" and "computers" are both normalized as "computer." This process aids in grouping variations of words under the same root, simplifying tasks like search and classification. Stemming algorithms like Porter and Snowball employ rules to truncate words to their simplest forms, disregarding suffixes and prefixes. Such normalization enhances the efficiency and accuracy of natural language processing applications.

g) ***Lemmatization***: Lemmatisation involves converting terms into their base forms as found in dictionaries to enable generalized comparative analysis. For instance, "produced" is normalized to "produce." This process ensures consistency in representing words, facilitating accurate interpretation and analysis in natural language processing tasks.

h) ***Number replacement***: In number replacement, numeric values are replaced with placeholder symbols to standardize comparison analysis. This technique ensures that numerical data doesn't overshadow textual semantics in NLP tasks. By employing dummy symbols for numbers and figures, the focus remains on linguistic patterns and meaning extraction. This pre-processing step enhances the uniformity and effectiveness of text analysis algorithms.

i) ***Dependency Parsing***: Which words or phrases depend on which other words or phrases is indicated by the dependency structure. Dependency-based parsing is a technique that we apply to examine and deduce relationships between characters in a phrase, as well as structure and semantic dependencies. We utilized the Stanford parser to perform syntactic dependency analysis on each sentence.

j) ***Chunking***: To determine the noun phrase, verb phrase, and other elements in a sentence, we employ shallow parsing. As an example, we extract the following information from the parse tree for the phrase fragment "a basic concept of Object-Oriented Programming," which is generalized to include only the identities and structures of the constituents (shown by the parenthesis): (NP (NP (PP (NP))) where the fragment is entirely covered by the first noun phrase (NP) constituent, consisting of "A basic concept" is covered by the second NP, and A sentence beginning with a preposition (PP) that covers "of Object-Oriented Programming." The final NP, "Object-Oriented Programming," makes up this PP.

IV. IMPLEMENTATION

A built-in library for machine learning tools is called Sci-kit-learn. It has statistical modeling and

machine learning techniques. The suggested technique for text feature extraction makes use of this library. For word embedding, or converting textual data into an array of integers, the Count Vectorizer is utilized. The process of transforming textual data into a vector format is now used to compare two text files for similarities. The cosine of the angle between the two vector representations of text files is calculated using cosine similarity. The process yields a score between 0 and 1, which gives us an idea of how similar the two input files are to one another.

The implementation strategy consists of four essential steps:

- ***Input text file:*** The file is intended to be the system's input for detecting plagiarism. Text format (with a.txt extension) is required.
- ***Pre-processing:*** Here we pre-processed the data for better result and apply our algorithms.
- ***Vectorization of text:*** Sci-kit's built-in capabilities ensure that the words are translated into a vector format from the textual input.
- ***Compute similarity:*** The fundamental idea of Cosine Similarity is used to calculate how similar two text files are to one another. When two text files are represented as vectors, their similarity is calculated using the dot product of both vectors or cosΘ (where Θ is the angle between the two vectors).
- ***Algorithms:*** In this state we will try the ML classification algorithms for our model.

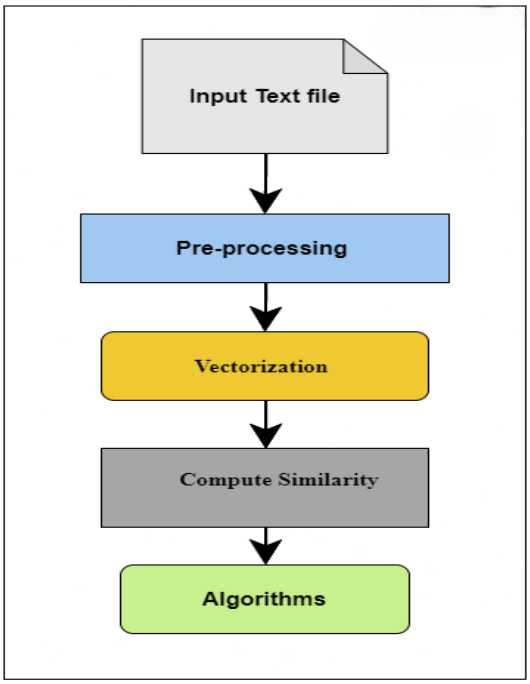

Figure 1. The model Diagram

V. ALGORITHMS

5.1Naïve Bayes Algorithm

Based on the assumption of feature independence, the Naïve Bayes algorithm is a probabilistic classification method based on Bayes' theorem. In many real-world circumstances, Naïve Bayes is remarkably effective despite its simplicity, and it is commonly employed in text categorization, sentiment analysis, spam filtering, and recommendation systems.

a. ***Bayes theorem***: The Bayes Theorem determines the likelihood of a hypothesis based on the available data. In terms of math, it is stated as: (P(E\H) × P(H))/(P(E)) = P (H\E) where P(E\H) is the probability of the evidence provided that the hypothesis is true, P(H) is the probability of the hypothesis being true (prior), and P(E) is the probability of the hypothesis given the evidence.

b. ***Naïve Assumption*****:** Naïve Bayes makes the assumption that a feature's existence in a class is unrelated to the existence of any other feature. This makes calculating probability easier.

c. Method:

- ***Training:*** Using a training dataset, determine the prior probabilities and conditional probabilities of each feature given the class labels.

- ***Prediction:*** Apply the Bayes theorem to determine the posterior probability of each class given the features for a new instance. For each case, the class swith the highest probability is allocated

The reason why we use Naïve Bayes:

A. *Simplicity*: Naïve Bayes is simple to implement and easily understandable, making it suitable for quick prototyping and deployment.

B. *Efficiency*: To estimate the parameters required for classification, a small amount of training data is needed.

C. *Fast Prediction*: Because of the algorithm's computational efficiency, big datasets and real-time applications can benefit from it.

D. *Robustness to Irrelevant Features*: Because Naïve Bayes assumes feature independence, it can effectively handle irrelevant features.

E. ***Good Performance*:** Naïve Bayes frequently exhibits surprisingly good performance in practice, particularly in text classification tasks, despite its simplicity and naïve assumption.

In summary, an easy-to-use yet powerful algorithm for classification problems is Naïve Bayes, particularly in scenarios with large datasets and high-dimensional feature spaces. Numerous machine learning applications choose it for their efficiency, durability, and simplicity.

5.2 Logistic Regression

A classification model called logistic regression is used to estimate the likelihood of binary outcomes. It functions by fitting the data to a logistic curve, which uses the logistic function to convert the linear combination of input features into probabilities. The procedure involves:

a. ***Data Preparation***: Collect and preprocess data, including feature scaling and handling missing values.
b. ***Model Training***: Estimate the parameters of the logistic function using techniques like maximum likelihood estimation or gradient descent.
c. ***Model Evaluation***: Use metrics such as accuracy, precision, recall, and F1-score to evaluate the model's performance.
d. ***Prediction***: Apply the trained model to estimate the probability of class membership for new data points.

We use logistic regression because it's simple, interpretable, and computationally efficient. It provides probabilistic predictions, allowing for uncertainty quantification. Applications for logistic regression can be found in a number of industries, including marketing (predictive customer turnover), banking (credit scoring), and healthcare (disease detection).

5.3LSTM

Long Short-Term Memory (LSTM) is a specialized recurrent neural network (RNN) architecture designed to overcome the vanishing gradient problem and capture long-term dependencies in sequential data. Here's a breakdown of how it operates:

a. ***Memory Cells***: LSTMs contain memory cells that store information over time. These cells have an internal state that can be updated, read, or reset based on the input data.
b. ***Gates***: LSTMs feature three types of gates - input gate, forget gate, and output gate. These gates manage the flow of information into and out of the memory cells, allowing the network to selectively remember or forget data.
c. ***Input Processing***: The input gate controls the flow of new input into the memory cell, deciding which information is relevant to store.
d. ***Forget Mechanism***: The forget gate determines which information from the previous state should be discarded, based on the current input and past memory.
e. ***Output Control***: The output gate manages the output from the memory cell, deciding which information should be passed to the next time step or the output layer.
f. ***Training Procedure***: LSTMs are trained using backpropagation through time (BPTT), which involves computing gradients and updating network parameters. This enables the network to learn and capture long-term dependencies in sequential data.

The reason why we use LSTM:

A. *Long-Term Dependencies*: LSTMs can learn and remember long-term dependencies in sequential data, making them ideal for tasks like natural language processing, time series prediction, and speech recognition.
B. ***Vanishing Gradient*:** LSTMs address the vanishing gradient problem that affects traditional RNNs, allowing for more effective training and better performance on tasks with long sequences.
C. *Flexibility*: LSTMs can be applied to both classification and regression tasks and can handle input sequences of variable lengths.
D. ***State-of-the-Art Performance*:** LSTMs have shown state-of-the-art performance in various sequential data tasks, making them a popular choice in machine learning and deep learning research.

In summary, LSTM networks are a powerful and flexible architecture for modelling sequential data. They excel at capturing long-term dependencies and are widely used in applications where understanding temporal patterns is crucial.

5.4Support Vector Machine (SVM)

Support Vector Machine (SVM) with the Support Vector Classifier (SVC) algorithm is a powerful supervised machine learning model used for classification tasks. Here's how it works:

a. ***Data Representation***: Support Vector Machines (SVM) operate by using vectors to represent data points in a high-dimensional space, where each feature is associated with a dimension.
b. ***Maximizing Margin***: SVC looks for the hyperplane in the feature space that maximizes the

margin between classes while also effectively separating them. The hyperplane is selected so as to maximize the support vectors—the distance between the closest data points from each class.

c. ***Kernel Trick***: By utilizing a kernel trick, SVM can effectively tackle non-linear classification issues. In order to do this, the input features must be transformed into a higher-dimensional space where the data can be separated linearly. Sigmoid, polynomial, linear, and radial basis function (RBF) are examples of common kernel functions.
d. ***Training Procedure***: SVC solves a convex optimization problem to determine the ideal hyperplane. It determines the hyperplane parameters that maximize margin and reduce classification errors.

The reason why we use SVM:

A. *Effective in High-Dimensional Spaces*: Support Vector Machines (SVM) function effectively even when there are more features than samples.
B. *Robustness to Overfitting*: SVM can handle noisy data because of its regularization parameters, which aid in preventing overfitting.
C. *Versatility*: SVM supports various kernel functions, allowing it to handle linear and non-linear classification tasks.
D. *Global Optimality*: SVM finds the globally optimal solution, making it less sensitive to local optima compared to other algorithms like neural networks.

In general, SVM with SVC is frequently utilized because of its higher performance and applicability in a variety of fields, including finance, bioinformatics, image classification, and text classification.

5.5 XGBoost

XGBoost (Extreme Gradient Boosting) is a highly regarded machine learning algorithm frequently used for both classification and regression tasks. Here's an overview of its key features and functionalities:

a. ***Boosting*:** XGBoost leverages ensemble learning by sequentially combining multiple weak learners, typically decision trees, to form a stronger model. Each new model aims to correct the errors of the preceding ones.
b. ***Gradient Boosting:*** The algorithm uses gradient boosting, which optimizes a loss function by adding models that minimize this loss through gradient descent, thereby reducing the overall error.
c. ***Regularization*:** To prevent overfitting, XGBoost incorporates regularization techniques. It includes parameters that control the complexity of individual trees and the model as a whole.
d. ***Handling Missing Values*:** XGBoost is equipped with built-in mechanisms to handle missing values in the dataset, automatically learning how to address them during the training process.
e. ***Parallel Processing*:** Designed for efficiency and scalability, XGBoost supports parallel processing, allowing it to utilize multiple CPU cores during training.

The reason why we use XGBoost:

A. ***High Performance*:** XGBoost is renowned for its superior performance and accuracy, frequently excelling in machine learning competitions and practical applications.
B. *Flexibility*: It supports a variety of objective functions and evaluation metrics, making it versatile for different problem types.
C. *Feature Importance*: XGBoost provides feature importance scores, aiding users in understanding which features most significantly impact predictions.
D. *Regularization*: The built-in regularization mechanisms help guard against overfitting, enhancing its robustness in noisy data environments.
E. *Wide Adoption*: Its effectiveness, scalability, and user-friendly nature have led to widespread adoption in both industry and academia.

In summary, XGBoost is a powerful and adaptable algorithm suitable for a broad range of classification and regression tasks. Its capability to handle large datasets, optimize complex loss functions, and deliver interpretable results makes it a favored tool among data scientists and machine learning practitioners**.**

VI. RESULT AND ANALYSIS

The accuracy performances of various classification models were evaluated in the context of predicting plagiarism. Among the models examined, Support Vector Machine (SVM) and XGBoost exhibited notably higher accuracies compared to Long Short-Term Memory (LSTM), Logistic Regression (LR), and Naïve Bayes.

SVM achieved an impressive accuracy of 91%, closely followed by XGBoost with 93%. These results highlight the efficacy of SVM and XGBoost in capturing complex patterns within the dataset and making accurate predictions.

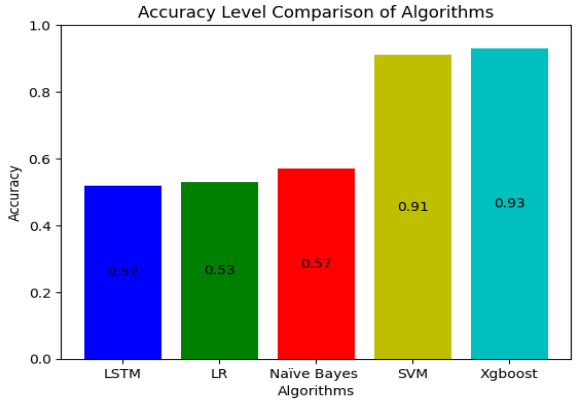

Figure 2. Algorithm's comparison

In contrast, LSTM, LR, and Naïve Bayes demonstrated lower accuracies of 52%, 53%, and 57% respectively. While LSTM is tailored for sequential data, its performance suggests limitations in capturing underlying relationships or temporal dependencies effectively. LR and Naïve Bayes, though simpler models, may have struggled to handle the inherent complexity of the data, resulting in suboptimal predictive performance.

Table 1. Test accuracy of applied algorithms.

No	Algorithm Name	Test Accuracy
1	LSTM	52%
2	LR (Logistic Regression)	53%
3	Naïve Bayes	57%
4	SVM	91%
5	Xgboost	93%

The significant variation in model accuracy highlights the significance of choosing a suitable categorization strategy according to the particular demands of the prediction task. SVM and XGBoost, with their higher accuracies, are particularly suitable for applications where precision is crucial, while LSTM, LR, and Naïve Bayes may find utility in scenarios prioritizing simplicity or interpretability over predictive performance while predicting plagiarism.

VII. CONCLUSION

Our aim in this study was to employ NLP methods to automatically identify instances of plagiarism within texts. The paper outlines how NLP techniques can enhance existing plagiarism detection methods and suggests avenues for further exploration by applying advanced NLP techniques. Our experimental findings demonstrate that these NLP methods significantly enhance the performance of basic detection models. Specifically, the Xgboost feature exhibits promising results in boosting overall detection accuracy.

NLP techniques have been shown to be effective in improving detection accuracy; nevertheless, a number of issues still need to be addressed. These include handling synonymy (the disambiguation of words), generalizing sentence structures, and multilingual detection. We intend to tackle these challenges in our future research efforts.

In conclusion, while more accurate detection methodologies are being developed, human judgment remains essential for assessing cases of plagiarism.

REFERENCES

[1] A. Aiken, "MOSS: A System for Detecting Software Plagiarism," Stanford University, 1994. [Accessed: Mar. 21, 2010]. Available: https://stanford.edu

[2] T. Lancaster and F. Culwin, "Classifications of plagiarism detection engines," unpublished internal 2nd draft, South Bank University, London, UK, 2003, pp. 1-16.

[3] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in Proc. 2014 Conf. Empirical Methods Natural Lang. Process. (EMNLP), Oct. 2014, pp. 1532-1543.

[4] J. Ramos, "Using tf-idf to determine word relevance in document queries," in Proc. 1st Instructional Conf. Machine Learning, vol. 242, Dec. 2003, pp. 133-142.

[5] V. Thada and V. Jaglan, "Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm," Int. J.Innovations Eng. Technol., vol. 2, no. 4, pp. 202-205, Aug. 2013.

[6] D. Klein and C. D. Manning, "Fast Exact Inference with a Factored Model for Natural Language Parsing," in Advances Neural Inf. Process. Syst. 15 (NIPS 2002), Cambridge, MA: MIT Press, 2003, pp. 3-10.

[7] N. Awale, M. Pandey, A. Dulal, and B. Timsina, "Plagiarism Detection in Programming Assignments using Machine Learning," Journal of Artificial Intelligence and Capsule Networks, vol. 2, no. 3, pp. 177-184, Jul. 2020. [Online]. Available: http://irojournals.com/aicn/. DOI: https://doi.org/10.36548/jaicn.2020.3.005

[8] H. Chavan, M. Taufik, R. Kadave, and N. Chandra, "Plagiarism Detector Using Machine Learning," International Journal of Research in Engineering, Science and Management, vol. 4, no. 4, pp. [specific pages], Apr. 2021. [Online]. Available: https://www.ijresm.com. ISSN (Online): 2581-5792.

[9] R. Rosu, A. S. Stoica, P. S. Popescu, and M. C. Mihăescu, "NLP based Deep Learning Approach for Plagiarism Detection," International Journal of User-System Interaction, vol. 13, no. 1, pp. 48-60, 2020.

[10] M. Chong, L. Specia, and R. Mitkov, "Using Natural Language Processing for Automatic Detection of Plagiarism," presented at the [Conference/Workshop], Research Group in Computational Linguistics, University of Wolverhampton, UK.

[11] J. Bao and J. Malcolm, "Text similarity in academic conference papers," in 2nd International Plagiarism Conference, Northumbria University Press, 2006, pp. 19-21.

[12] J. Bao, C. Lyon, P. Lane, W. Ji, and J. Malcolm, "Comparing Different Methods to Detect Text Similarity," Technical Report, University of Hertfordshire, 2007.

[13] A. Barrón-Cedeño and P. Rosso, "Towards the Exploitation of Statistical Language Models for Plagiarism Detection with Reference," in European Conference on Artificial Life, ECAL 2008 PAN Workshop, 2008, pp. 15-19.

[14] N. Kang, A. Gelbukh, and S. Han, "Ppchecker: Plagiarism pattern checker in document copy detection," in Text, Speech and Dialogue, Springer, 2006, pp. 661-667.

[15] T. Lancaster and F. Culwin, "A Visual Argument for Plagiarism Detection using Word Pairs," in Plagiarism: Prevention, Practice and Policies 2004 Conference, Apr. 2004, pp. 1-14..

[16] R. Lukashenko, V. Graudina, and J. Grundspenkis, "Computer-Based Plagiarism Detection Methods and Tools: An Overview," in ACM International Conference on Computer Systems and Technologies, vol. 54, no. 3, pp. 203-215, 2007.

[17] J. Williams, "The plagiarism problem: Are students entirely to blame?" in Proc. 19th Annual Conf. Australasian Society for Computers in Learning in Tertiary Education, ASCILITE, 2009, pp. 934-937