Using Machine Learning for Calculus

Paul Norrigan

8/28/24

Abstract

The goal of this research project is to develop a machine learning program which can take an integral equality (for example, a u-substitution that turns one integral into another) and verify that it is correct. Many other mathematics-related software can only take in a one-sided equation and return a calculated output, but most mathematical proofs are built on relating several successive pieces of logic. For this problem, a language processing approach was taken to convert the integral equations into something that could be understood by a language processing model. From there, several different classifiers were tested, each one returning about a 40-60% accuracy depending on the dataset that was input.

1. Introduction

Just about anyone who has used a software for help on their math homework knows that, generally, you give the software a problem and it returns a numerical answer. Most math software applications function like this, and while it is very useful for helping you with your math homework, it is much less useful for theoreticians in the field trying to make new discoveries. New discoveries aren't built on solving singular problems, but rather on finding general statements which can usually be expressed in the form of a two-sided equation. There are existing software applications that can help do this, such as Wolfram Mathematica and SymPy, but they either require paying for a subscription or knowledge of coding in order to be used.

As someone who has worked on developing various mathematical proofs, I often find myself frustrated by the lack of math software that can help me verify and strengthen my work without requiring expensive payment. Therefore, the ultimate goal of this research project is to create a machine learning software that can work with theoretical mathematicians in the field by providing a manner in which they can verify the validity of their mathematical reasoning. A useful case study is the task of equating two integrals, as this often comes up when developing mathematical proofs.

To tackle this problem, a machine learning program will be built which can intake a two-sided equation involving integrals and then output a probability of whether the equation is true or false, with 1 being true and 0 being false. In order to intake the two-sided equation, the equation will be formatted with LaTeX, a common method of formatting mathematical equations, and then turned into a word vector that can then be analyzed by a variety of different machine learning programs, including linear regression, K-nearest neighbors, MLP, and a multi-layer neural network.

2. Background

Any two integrals are equal to each other so long as there exists a closed-form function u(x) such that:

$$\int_{a}^{b} f(x)dx = \int_{u(a)}^{u(b)} g(u)du$$

This is essentially the basic concept behind using u-substitution to transform one integral into another. The machine learning program is evaluated based on its ability to correctly predict whether an integral transformation is performed correctly. Here is an analysis of what existing

softwares like ChatGPT, Mathway, and Wolfram Alpha can do when it comes to evaluating the correctness of an integral transformation.

Mathway

Mathway, like a vast majority of other math softwares, can only take in a one-sided equation as an input, so it would not be able to evaluate the correctness of an integral transformation. It is possible to input an integral transformation, but Mathway will not be able to understand it, and will instead just solve the integral on the left side of the equation. Another downside of Mathway is that, it gives you a limited response unless you pay for an upgraded version.

Wolfram Alpha

Wolfram Alpha is similar to Mathway, except that it is made to input, understand, and output mathematical reasoning rather than just calculations, making it similar to a language model. Out of the three options discussed here, it is the only one that can take an integral transformation and accurately evaluate whether it is done correctly. However, it has a similar limitation to Mathway in that, beyond basic calculations, more complex evaluations require upgrading to a very expensive paid version.

ChatGPT

ChatGPT isn't technically a math software, but it is a language model capable of reasoning, and if you give it a math-related question, it will give you an answer. These answers, however, tend to be highly erroneous, especially when it comes to integrals. ChatGPT has faulty and inconsistent mathematical reasoning, mainly because of the fact that it was not made to understand mathematical reasoning, and so it bases its answers mainly off of educated guesses. Thus, if you give it an integral transformation, it will more than likely output false negatives and false positives, where it classifies a correct transformation as incorrect and vice versa.

3. Dataset

For each of the four datasets that would be used, they were input into an excel sheet with two columns, one for the LaTeX form of the equation and one for true (1) or false (0). The following is an example of the LaTeX conversion of an equation in the fourth dataset:

$$\int_0^1 x^2 dx = \int_0^1 \frac{\sqrt{u}}{2} du: \backslash int\_\{0\}\hat{}\{1\}\, x\hat{}2\, \backslash,\ dx\ =\ \backslash int\_\{0\}\hat{}\{1\}\, \backslash frac\{\backslash sqrt\{u\}\}\{2\}\, \backslash,\ du$$

Then, this excel sheet would be read by the program and turned into a list before a new list wordVector is created which stores the word vectorization of each LaTeX equation in the first column of the list in the same indices as well as its boolean value in the second column. Then, the data is split to be 80% training and 20% testing data, and then four different classifiers are tested: Logistic Regression, K-Nearest Neighbors, Multi-Layer Processing, and Convolutional Neural Network.

It was very hard to find any existing datasets that could be input into the machine learning model, so a dataset had to be generated. For the dataset, a 50/50 split between correct and incorrect integral equalities would be needed to train the machine learning model. The first method of doing this was asking ChatGPT to generate the integral equalities, hoping that its inaccuracy would lead to about a 50/50 split between correct and incorrect equalities. However,

nearly all of the resulting equalities were incorrect, and half of them had to be manually sorted through and corrected to obtain a 50/50 split between correct and incorrect equalities. This dataset contained 50 entries in total.

The next method was asking ChatGPT to generate a long list of functions that could be turned into integrals, manually performing u-substitutions on them, and then inputting that equation into the machine learning model. This process went by much more quickly and generated much more samples. This dataset contained 200 entries in total.

Even with this new process, there was potential for optimization, so a better prompt for ChatGPT was needed where it would be able to output integrals whose values were all the same (thus implying that the integrals are equal to each other). However, this approach turned out to be much slower because, even with the new prompt, ChatGPT had a hard time coming up with large amounts of integrals of different functions that were equal to the same value. These samples were then combined with the previous two datasets to contain 400 entries in total.

The final method was similar to the second one where a long list of ChatGPT-generated functions would be turned into integrals, only this time, instead of performing random u-substitutions on each integral, the same substitution could be performed on each one in the list, and then another list could be made with a different substitution performed, and so on. This was by far the fastest and most effective method since the same format could be used with multiple equations consecutively when performing the substitutions. This dataset contained 400 entries in total.

4. Methodology / Models

In order to allow the machine learning model to understand the integral equations, LaTeX, a common method of writing out mathematical expressions using only letters and symbols available on the keyboard, was used to express the equations. These LaTeX equations were then passed through a word vectorizer that turned each equation into a word vector nine units long that could be then passed into the machine learning model (nine unit long word vectors seemed to produce the best results with everything else being at default settings). For this research project, four different machine learning models were used to see which one would provide the best results: Logistic Regression, K Nearest Neighbors, Multi-Layer Perceptron, and Convolutional Neural Network.

Logistic Regression

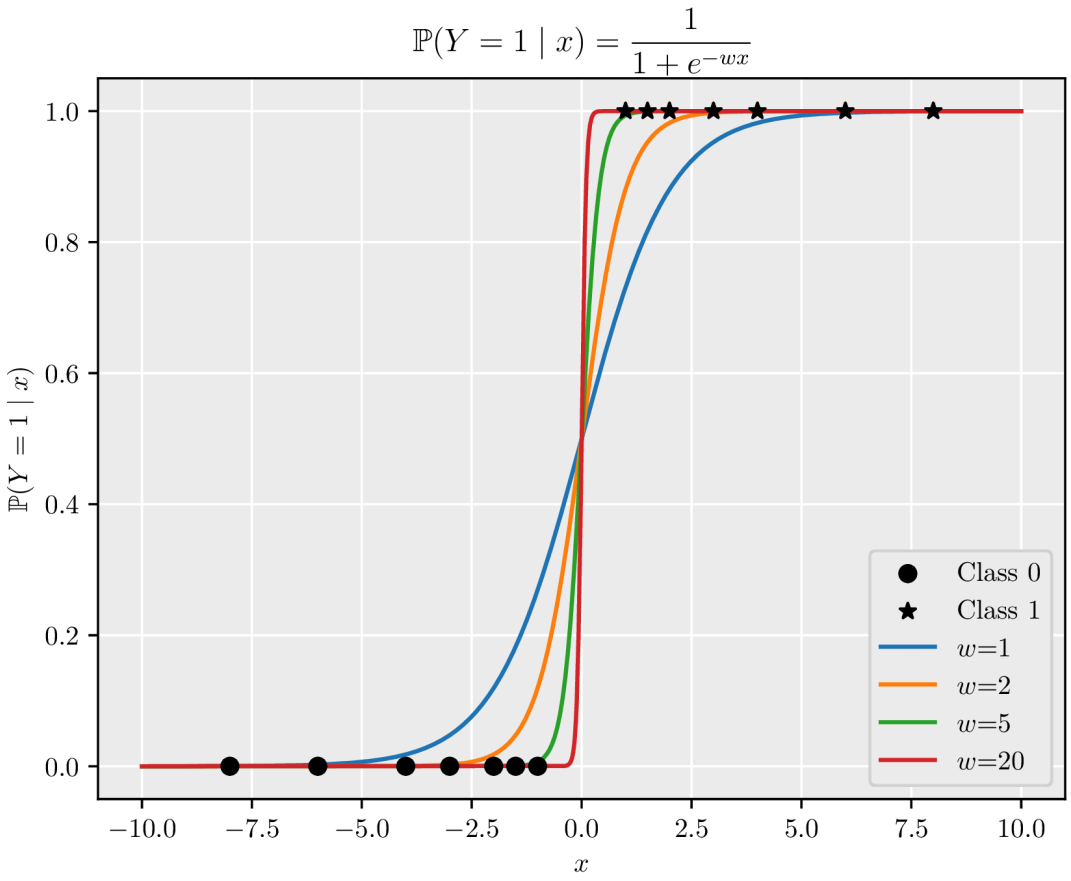

Figure 1: An example of a logistic curve being fitted to data.

A logistic regression classifier attempts to fit a logistic curve to its input data in order to determine the likelihood that it is of one type or another. It plots data on an X-Y plane with X being the independent variable and Y being a value between 0 and 1 (which represent the different possible types), and then it fits a logistic curve to the data and calculates the boundary value (the X-value which differentiates the data of one type from the data of the other). This type of model is expected not to be very useful for this problem because this problem does not really involve an independent variable, but rather it involves word vectors. Still, this model was tested in order to see how effective it would be, if at all, for this problem.

K Nearest Neighbors

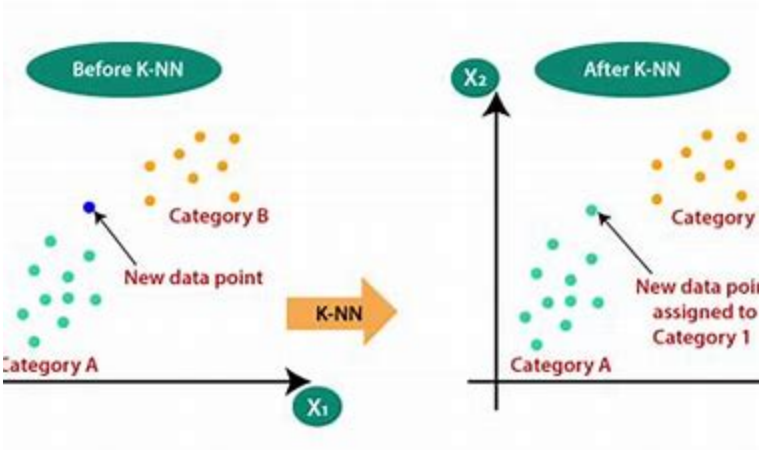

Figure 2: An visualization of how a K Nearest Neighbors model classifies data.

A K Nearest Neighbors machine learning model uses the training data to define separate categories. Then, whenever it takes an input, it will search K neighbors around that input to determine which category the input belongs in. The value of K can be altered in the initiation of the model. This will likely be very useful for this problem, especially with the fourth (and, to an extent, the second) data set, since there will be so many integral equations taking the same format due to having the same substitution performed on them. For this problem, K will be equal to 5.

Multi-Layer Perceptron

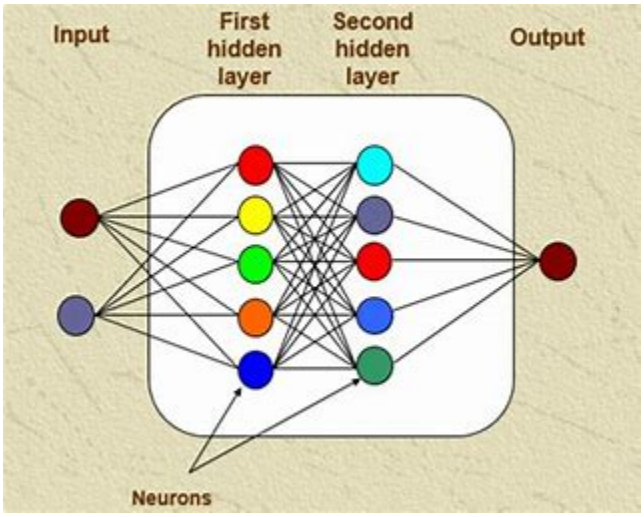

Figure 3: A visualization of a neural network with an input size of 2 and 2 hidden layers of size 5.

A Multi-Layer Perceptron is a neural network, meaning it relies on ‘neurons’ which give signals to neurons in the next layer. Those neurons then send another signal to the next layer depending on what it receives, and so on, until the last layer gives an output. The intermediary layers are known as hidden layers. Each neuron has an associated weight with it that determines the strength of the signal that it will output. The machine learning model determines the weight that each neuron will have using the training data. An MLP classifier, however, is a specific

version of a neural network designed specifically to work with image classifying, and therefore will likely not be useful for this problem, but it will still be tried to test how relatively effective it would be. For this problem, the hidden layer sizes will be set to 3.

Convolutional Neural Network

A Convolutional Neural Network is a general neural network that is designed for making decisions on its input. It is highly configurable in its input size, amount of hidden layers, the density of each layer, and the epochs of training it receives. Furthermore, a Keras tuner can be used to automate the process of finding out what combination of hidden layers, hidden layer density, and epoch count will optimize the neural network's accuracy. For this problem, the Convolutional Neural Network is expected to be most effective for datasets such as the first and third which involve much more complicated integral transformations, as it is able to understand much more complex operations. For this problem, the Keras tuner will be used to optimize the accuracy of the CNN model.

5. Results and Discussion

The first dataset achieved an accuracy of 40% from the Logistic Regression model, 60% from the MLP model, 40% from the KNN model, and 60% from the CNN model. The second dataset achieved an accuracy of 60% from the Logistic Regression model, 50% from the MLP model, 42.5% from the KNN model, and 50% from the CNN model. The third dataset achieved an accuracy of 48.75% from the Logistic Regression model, 51.25% from the MLP model, 38.75% from the KNN model, and 51.25% from the CNN model. The fourth dataset achieved an accuracy of 48.75% from the Logistic Regression model, 48.75% from the MLP model, 51.25% from the KNN model, and 52.5% from the CNN model.

In terms of the datasets used, the second and fourth datasets, as expected, returned the highest average accuracy rates of 50.625% and 50.3125%, respectively, and the third dataset returned the lowest average accuracy rate of 47.5%. This is likely because the second and fourth datasets were the most consistent in the samples that they used, while the third dataset had a mix of many different methods of integral transformation, as well as many arbitrary transformations that even the average mathematician could not perform.

In terms of the models used, the best performing model was the Convolutional Neural Network, with an average accuracy of 53.4375%. Surprisingly, the Multi-Layer Perceptron was the second best performing (despite being a model designed for image processing), followed by the Logistic Regression model and then the K Nearest Neighbors model in last with an average accuracy of 43.125%. This is highly surprising, considering that the KNN model was expected to be one of the best-performing due to its ability to predict an equation's validity based on a similar equation. However, one possible reason for this surprisingly poor performance is that the KNN model heavily struggled with the inconsistency of the third dataset, but performed much better on the fourth dataset, which was the most consistent.

Dataset	Logistic Regression	K Nearest Neighbors	Multi-Layer Perceptron	Convolution-al Neural Network	Average
1	40%	40%	60%	60%	50%
2	60%	42.5%	50%	50%	50.625%
3	48.75%	38.75%	51.25%	51.25%	47.5%
4	48.75%	51.25%	48.75%	52.5%	50.3125%
Average	49.375%	43.125%	52.5%	53.4375%	

Table 1: A table representing the accuracy for each model and dataset used, as well as the average accuracies of the models on the bottom row and the average accuracies of each dataset on the right column.

Perhaps the most surprising result is the low performance across the board of all of the models. Even the CNN model with an average accuracy of 53.4375% is only slightly better than a coin toss at predicting the validity of an integral transformation, and most of the models (Logistic Regression and KNN) are actually performing worse than a coin toss. The original purpose of this machine learning program was to be able to predict the validity of an integral transformation, and while this program could likely do it better than ChatGPT, Mathway, and most similar softwares, Wolfram Alpha still offers a much better alternative since it is much more accurate and can do most integrals without paying for a subscription.

One possible explanation for the low performance of the program is that the word vectorization of the integral equations was not describing the integrals in an accurate manner. The only necessary parts of the integrals were the bounds and the function inside the integrand. However, the word vectorizer was capturing unnecessary parts of the integrals, such as the differential and the various extraneous symbols present in the LaTeX conversion of each integral like \int and \frac. It was also possibly inaccurately capturing the important parts, especially since the entire function in the integrand was technically only one word, and thus stored as a single value in the entire word vector, and the same thing occurred with the bounds of each integral. Adding to the possibility of this being the reason for the model's inaccuracy is that, for each dataset, changing the way that the models were used did not have a significant impact on their accuracy. For example, the CNN model returned the same accuracy during all stages of the implementation of the Keras tuner (the Keras tuner was implemented in various stages, one for each part of the model that was being tuned). This means that the error is arising from a stage before the models are being used, which is likely in the word vectorization stage. An alternate approach to vectorizing the integral equations would be extracting the lower and upper bounds and the function in the integrand from both integrals in each equation and storing those into a vector that represents the equation. However, there would be notable difficulty in finding a way to store the functions in the integrand into the vector in a way that would allow the machine learning program to find a pattern and thus improve its accuracy.

6. Conclusions

Overall, the usage of machine learning models to verify the validity of various integral transformations has proved to be ineffective, with all the models used being only slightly better than a coin toss or even worse. Thus, machine learning would likely not be very helpful in the development of mathematical proofs involving integral transformations unless a better approach can be developed. If a better approach could be developed, then such a software could prove useful for anyone looking to verify whether they have done a certain integral transformation correctly.

Acknowledgments

Erick Ruiz has mentored me throughout the entire process of building the machine learning program, analyzing and improving the results as much as possible, and writing the research paper.

References

"Pandas - Python Data Analysis Library," from pandas - Python Data Analysis Library (pydata.org)

"NumPy," from NumPy -

"MatPlotLib - Visualization with Python," from Matplotlib — Visualization with Python

"Gensim - Word2Vec Embeddings," from models.word2vec – Word2vec embeddings — gensim (radimrehurek.com)

"Scikit Learn - Machine Learning in Python," from scikit-learn: machine learning in Python — scikit-learn 1.5.2 documentation

"Keras Documentation - KerasTuner," from KerasTuner