

Utilizing Machine Learning Algorithms to Automate the Exam Grading Process

Chris Karvelas

University of Manitoba

DATA 4010: Data Science Capstone Project

Instructors: Margherita Maria Ferrari, Vitalii Akimenko, and Carson Kai-Sang Leung

April 7, 2024

Table of Contents

1. Introduction	2
1.1. The Problem	2
1.2. Background and Project Goals	2
2. Data	3
3. Methodology	7
3.1. Data Preprocessing	7
3.1.1. Old Exams	7
3.1.2. Image Classification Code Preprocessing	9
3.2. Models	10
3.2.1. Model 1: K-Nearest Neighbours (KNN)	10
3.2.2. Model 2: Support Vector Machines (SVM)	12
3.2.3. Model 3: Convolutional Neural Network (CNN)	16
3.2.4. Model 4: Large Language Models – ChatGPT	18
4. Results and Analysis	22
4.1. Image Classification	22
4.1.1. KNN	22
4.1.2. SVM	23
4.1.3. CNN	23
4.2. LLMs	24
4.2.1. W2023 Final	25
4.2.2. New Model	26
4.2.3. Q1 Mock Exam	26
4.2.4. Q2 Mock Exam	28
4.2.5. Answer Key Only	29
5. Conclusion, Issues, and Future Work	30
5.1. Conclusion	30
5.2. Limitations, Challenges, and Remedies	31
5.3. Future work	35

1. Introduction

1.1. The Problem

When it comes to writing exams, the first thought that comes to mind are the hours of preparation a student must put in to succeed. A major aspect of the examination process that does not normally come to mind is the job of the graders. A grader may have to read hundreds or even thousands of examinations and questions and assign an accurate grade to each student. This process is both time-consuming and stressful, as there are a wide range of “correct” and “incorrect” answers that the grader must accurately determine the score of. Additionally, graders typically need to be trained, or come from a certain background of knowledge to properly grade an exam. It is a bad idea to get a history professor to grade a calculus exam and vice versa. If the grader makes a mistake, they run the risk of giving a student an unfairly poor, or an undeservingly high grade. Additionally, the grader must ensure all exams are accurately graded and returned within a few days, or students will get impatient, wanting their marks back.

As a teaching assistant in the Statistics department at the University of Manitoba, I have ample experience grading quizzes and assignments for many painful hours on end. During this process, it is exceedingly difficult to remain consistent and mistake-free. If you change your mind on how to grade something, you need to go back to ensure that all previous instances are marked accordingly.

Fulfilling grading duties is an arduous task and typically requires teams of multiple people working together to grade an exam. This approach is more time-efficient, but also very costly, as graders need to be paid for their work. Overall, there are many downsides to the current way in which exams are graded, as large courses will require many working hours to mark their exams.

Despite these drawbacks, human graders are very well-equipped to mark exams, as they can use their best judgment to score answers and are particularly good at interpreting a solution as being worthy of a certain score. This trade-off begs the question, can we use computers and machine learning to grade exams in a way that is both quick and cost-effective, but also accurate?

1.2. Background and Project Goals

At universities, some first-year courses are required for a wide variety of programs and every semester, hundreds or even thousands of students register in these courses. One such example at the University of Manitoba is MATH 1500: Introduction to Calculus I. This course is required for many programs and is taken by nearly a thousand students each semester. Grading these exams, unsurprisingly, is a massive undertaking that requires a team of graders to be organized and the tasks divided amongst themselves to efficiently grade thousands of exam pages.

Additionally, in reviewing the results from the sample exams, students typically fared poorly. Taking off marks takes time as the proper number of marks to deduct must be determined and comments made to explain where and why a student has lost marks. Doing so severely hinders the grading process as it requires focus, attention to detail, and consistency. Maintaining such a prominent level of consistency is difficult when completing a repetitive task, as factors

such as fatigue can hinder performance. Additionally, grading these exams is a costly procedure, as graders must be paid for the hours worked to grade the exams.

This project aims to serve as a starting point for solving this problem of grading, by finding a quick, accurate, and cost-effective way to grade exams. It attempts to use machine learning to grade MATH 1500 exams in a way that gives accurate results with a truncated grading scheme and offers a path forward for further research and development towards this goal.

This project utilizes several machine learning algorithms for image classification, as well as pre-trained Large Language Models (LLMs) to classify the student answers into grades. The project also attempts to provide a clear set of instructions on how it got the exams graded and to identify areas of interest for future research and development. Finally, I analyze the grading accuracy with the truncated schemes to determine the potential of the methods used. The hope is that over time, methods and technology will be improved to a point where machine learning can be used to grade exams as accurately, if not more accurately, than a human.

2. Data

Over the course of this project three datasets were used. Initially, final exam scans from the Summer 2023 (S2023) semester for MATH 1500 were used. To guarantee anonymity, all information identifying students was removed. The exams were organized into individual questions and tests were performed on select questions from each exam (see section 3.1.1). The criteria for a question to be selected was that it had to be straightforward and have a more objective standard of being right or wrong. For example, the only question that was used from the S2023 Final was a question asking students to find four different limits from a provided graph (see Figure 1). This question required little-to-no explanation to be done and had mostly objective answers. For these reasons it was the most ideal starting point for analysis in this project.

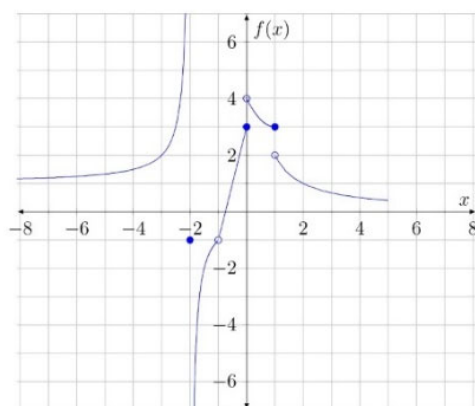
The dataset for the final exams were taken from a website called Crowdmark which is what many departments at the University of Manitoba use to grade assessments. Crowdmark allows exams to be graded digitally, simplifying the grading process. Students write their exams as normal, then all the exams are scanned and uploaded to Crowdmark. Crowdmark automatically assigns the student's exam to their profile via a QR code on each page (see Figure 2). The booklet is assigned to a student based on their name written on the first page (example omitted for anonymity reasons). Crowdmark also automatically separates each exam into questions so that a grader can easily grade the same question in multiple exam booklets. The grader gives a score for each question and can leave comments explaining their decisions. When every booklet is graded, the score for each student is automatically tabulated and the results are published. Students can then view their graded booklet. Crowdmark works best with Portable Document Format (PDF) files but also accepts a wide variety of file types. Students can complete online exams within Crowdmark as well.

Unfortunately, none of the models trained provided any meaningful results (see section 4.1.). The reason I first thought of why this was the case was that I did not have enough data to sufficiently train the models, so I changed datasets to scans of the Winter 2023 (W2023) semester Final. 511 students wrote this exam, up from 253 exams from the S2023 semester.

Analysis was performed on a definite integral question, which had a numeric answer so there should have been minimal variance with correct student answers (see Figure 2). Unfortunately, the increase in sample size did not improve results at all, so a change in the approach was again required (see section 4.2.).

UNIVERSITY OF MANITOBA
COURSE: MATH 1500
DATE: TBA
Final Exam
DURATION: 2 hours

- [4] 2. The graph of the function $f(x)$ is given in the figure. Find the following values if they exist - no justification/work is required. If not, explain why not.



- (a) $\lim_{x \rightarrow -2^+} f(x)$
- (b) $\lim_{x \rightarrow 0} f(x)$
- (c) $\lim_{x \rightarrow 1^+} f(x)$
- (d) $\lim_{x \rightarrow 1^-} f(x)$

Figure 1: The question from the S2023 Final used for analysis.



209D6FDE-F335-4679-92D3-9D1F5F66458A

23wmath1500final

#232 Page 14 of 16

00

Final Exam

Winter 2023

9. (4 points) Find $\int_{-2}^2 \sqrt{4-x^2} dx$ using the signed area interpretation of definite integrals.

Figure 2: The question from the W2023 Final used for analysis. Notice the QR code on the top left.

The W2023 dataset was too large for the new approach (see section 5.2.), so the third and final dataset was a mock exam that I created with 150 answers for each of three different

questions. Of these three, only the first two questions were used in any tests (see Figure 3). The dataset had three classes: 0% answers (class 0), 100% answers (class 2) and neither 0% nor 100% answers (class 1). The classes were grouped together when writing, where class 0 were answers 1-50, class 2 were 51-100, and class 1 were 101-150. Each set of solutions on the mock exam had this answer design. The mock exam answers were where most of the important analysis was performed on as it created an ideal exam scenario to test if our methodology had any chances of success.

Finally, each of the three datasets of images were converted to LaTeX code using a website called Mathpix, creating an additional dataset for each question. Images of the code for the final exam questions were initially used in the image classification, but the code from the W2023 Final and the mock exam were later used in text processing. For the W2023 Final question, full corrections were not made to the LaTeX code due to time and resource constraints, but some issues were removed (see section 3.1.1.). For the mock exam, since those solutions were solely used for text processing, the errors in the LaTeX code conversion were corrected so they exactly matched the written solutions.

1. Determine $\lim_{x \rightarrow \infty} \frac{x + \pi \sin(2x)}{3x + \sqrt{x} + 4}$.

$$\begin{aligned} \lim_{x \rightarrow \infty} \frac{x + \pi \sin(2x)}{3x + \sqrt{x} + 4} &= \lim_{x \rightarrow \infty} \frac{1 + \pi \frac{\sin(2x)}{x}}{3 + x^{-1/2} + \frac{4}{x}} \\ &= \frac{1}{3}. \end{aligned}$$

2. Fix any $x \neq 0$. For any $q \in \mathbb{R}$ and differentiable function f , define $g(x) = \frac{f(xq) - f(x)}{qx - x}$.
Determine $\lim_{q \rightarrow 1} g(x)$.

Let $xq = x + h$ so that $q = 1 + h/x$. Thus $q \rightarrow 1$ iff $h \rightarrow 0$. Then

$$\begin{aligned} \lim_{q \rightarrow 1} g(x) &= \lim_{q \rightarrow 1} \frac{f(xq) - f(x)}{qx - x} \\ &= \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h} \\ &= f'(x). \end{aligned}$$

Figure 1: Questions 1 and 2 on the mock exam and their answer key solutions.

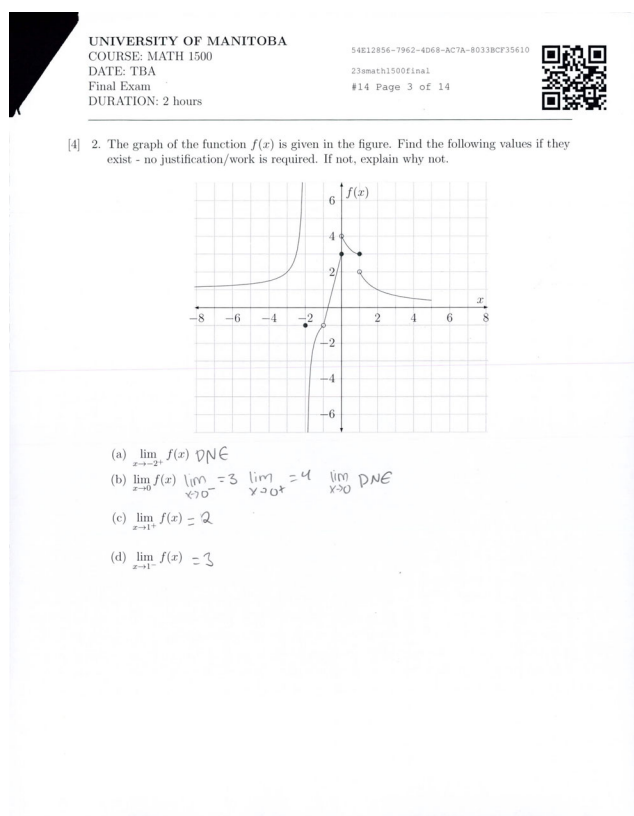


Figure 2: Sample student answer from the S2023 final used in analysis.

69. Let $x_n = x + h$, so $n = 1 + \frac{h}{x}$
 so as $n \rightarrow 1$, $h \rightarrow 0$,
 then

$$\lim_{n \rightarrow 1} \frac{f(x_n) - f(x)}{x_n - x}$$

$$= \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$= f'(x)$$

Figure 3: Sample mock exam answer for Q2.

3. Methodology

3.1. Data Preprocessing

3.1.1. Old Exams

The datasets came in single PDF files which had several thousand pages in each. Additionally, several pages were scanned out of order or missing, which threw off the ordering of questions in some spots.

When it came to preparing the data for analysis, several steps were taken to allow the data to be usable. First, each page of the PDF was converted into a Joint Photographic Experts Group (JPG) image, this process was performed using Adobe Acrobat 2020. Once the pages were converted to JPG images, the images were renamed using the Advanced Renamer software. Figure 6 shows the file naming format for each exam.

For the S 2023 Final	"S23 Final-Page1", "S23 Final-Page2"....
For the W 2023 Final	"W2023_1"," W2023_2"....

Figure 4: How the files were named so they could be easily accessed and sorted.

The images were then loaded into Jupyter Notebook and the answers to the desired questions were separated into folders. Answers written on blank pages were omitted from any analysis. The operation was performed using regular expressions based on the modulo of its page number with the number of pages in an individual exam booklet (see GitHub). It was for this reason the page numbers were included in the file names. In other words, Q2 would be all pages in the S2023 Final with a modulo of three, since Q2 occurred on the third page of the test (the title page was page 1). Unfortunately, not every page was scanned in the correct order, and some were missing, so the modulo result for a question would change. Several page ranges had to be made to ensure no questions were mismatched (see GitHub). I performed this task with the *move* function from the *shutil* Python library and the file paths of both the original image and the desired destination folder. This task was only performed for the questions that I analyzed.

Once the exam answers were converted to images, each image varied slightly in size, but were all approximately 3200×2500 pixels. That meant that an image had around eight million pixels, so in the W2023 final dataset, there were over four billion values per question. Images needed to be shrunk, and made into a standard size, as otherwise the models would not work. Each image was shrunk to be 640×500, maintaining the aspect ratio so that minimal information would be lost. This task was performed using the *Image.resize* function in the Python Imaging Library. This action reduced the overall dataset size by a factor of twenty-five, making it much more manageable to use and it also standardized the image sizes. The names of the shrunk images for the S2023 exam were again changed with the Advanced Renamer program to match the order they appeared in the folder, for example: "001", "002",.... The W2023 final did not have its names changed. This approach allowed for two separate ways of reading in the data to be demonstrated (see GitHub).

This shrunken size still proved to be too much for the computer memory when performing some of the tests (see section 3.2.2.2.), so the images were further shrunk to 320×250 in those instances.

The other bit of data creation I did was converting the images to LaTeX code. I took the full-sized JPG images for a particular question and used Adobe Acrobat to create a PDF of the solutions for each question. Once I had the PDF, I uploaded the file to Mathpix and “snipped” it, which converted the answers into LaTeX code. This process only took a couple of minutes to perform. I then exported the file with the LaTeX code as a PDF and converted and resized each image to 640×500 . I did minimal analysis with these images as it was not the best use of the LaTeX code.

The other issue I had to fix with was that the dataset containing the target scores was in a completely different order than the PDF data. I manually added an order variable to the target dataset that ensured ordering was consistent between both datasets. During this process, I noticed a couple of booklets were in one dataset and not the other. Each of those values were deleted from their respective dataset and the file numberings were adjusted accordingly.

After this, no further pre-processing was done on the S2023 Final, as shortly after the initial testing, the dataset was dropped in favour of the W2023 Final and was never used again. After switching the approach to LLMs, I had to create a new dataset, an Excel spreadsheet that stored the LaTeX code and the corresponding grade for each answer. Before creating the spreadsheet, I had to clean the data up by removing all questions that were incorrectly snipped (see Figures 7 and 8). All unnecessary noise was removed from the data as well (see Figure 30). After cleaning, the W2023 Final consisted of 455 answers, down from 511. I also had to preserve order and enumerate each answer to distinguish easily between separate answers. After that task was done, I manually copy-and-pasted each value into an Excel spreadsheet and took the corresponding target scores from the original score dataset.

264

9. (4 points) Find $\int_{-2}^2 \sqrt{4-x^2} dx$ using the signed area interpretation of definite integrals.

$$\int_{-2}^2 \sqrt{4-x^2} dx$$

$$= \int_{-2}^2 (4-x^2)^{1/2} dx$$

$$= \int_{-2}^2 \frac{1}{2} (4-x^2)^{3/2} \cdot (-2x) dx$$

$$-\int_0^{-2} \sqrt{4-x^2} + \int_0^2 \sqrt{4-x^2}$$

265

9. (4 points) Find $\int_{-2}^2 \sqrt{4-x^2} dx$ using the signed area interpretation of definite

Figure 5: Example of output of a poorly converted answer with an image.

```
\end{aligned}
$$



1. (4 points) Evaluate the following limit
```

Figure 6: How an image appears in the LaTeX code.

With the mock exam, each answer was numbered when it was written to distinguish between answers and to keep track of class sizes. The solution PDF was snipped on Mathpix, and the LaTeX code was adjusted to accurately match the solutions and no answers were removed. The data was then copied into an Excel spreadsheet with the target data as before. The answer key was also snipped so the question and solution could be included in prompts for the LLMs (see section 4.2.5.).

3.1.2. Image Classification Code Preprocessing

Before training the image classification models, more pre-processing had to be done. The exam data was imported using the scikit-image (skimage) Python library and its `io.imread` function and the file path of the folder of images. The target data was imported with the pandas (pd) Python library and its `read_excel` function and the file path of the dataset. The target data

had to be transformed so it was binary, as I wanted to start with a binary classification problem. The questions analyzed for the final exams had many students either scoring 0% or 100%. Therefore, classes were made so that the highest frequency score was one class (0 if it was 0% or 1 if it was 100%, on the question) and the other class contained all answers that scored differently. Unfortunately, results were so skewed on Q9 of the W2023 Final that even with this most generous classification system for the minority class, almost 90% of solutions belonged to class 0. Around 40% of students scored 100% on Q2 of the S2023 Final, so that dataset had more balanced classes than the other.

The images were then reshaped into a 2D array using the NumPy (np) Python library and its *reshape* function. The result was a dataset of 511 rows and 320 000 columns (equal to the number of pixels in each image). For the Convolutional Neural Network (CNN), the images were left as is and not reshaped. The dataset was then split into training and testing data, using an 80-20 training and testing split. The split was made using the sci-kit learn (sklearn) Python library and its *train_test_split* function.

3.2. Models

3.2.1. Model 1: K-Nearest Neighbours (KNN)

3.2.1.1. Mathematical Background

The KNN algorithm is a **supervised learning** algorithm, meaning it is trained with labelled data (“What is Supervised Learning?,” n.d.). At a high level, the KNN model classifies a point based on the classifications of the nearest points around it in N-dimensional space, where N is the number of columns in the dataset. The model uses straight-line, or Euclidean distance (see Figure 9). Euclidean distance is calculated as follows:

$$d(x, y) = \sqrt{\sum_{i=1}^N (x_i - y_i)^2} \quad (1)$$

where $d(x, y)$ is the distance between points x and y in the dataset and $\sqrt{\sum_{i=1}^N (x_i - y_i)^2}$ is the Euclidean distance between two points, $x_i, y_i \in \mathbb{R}^N$, where i is an integer (Gotke, 2020).

One of the **hyperparameters**, a value that controls the learning process of a machine learning algorithm, is the number of neighbours, K , where K is an integer (Nyuytiymbiy, 2022). The model takes a new point, x , and compares it to the K nearest values and the class with most of the K nearest points to x will be the predicted class for x .

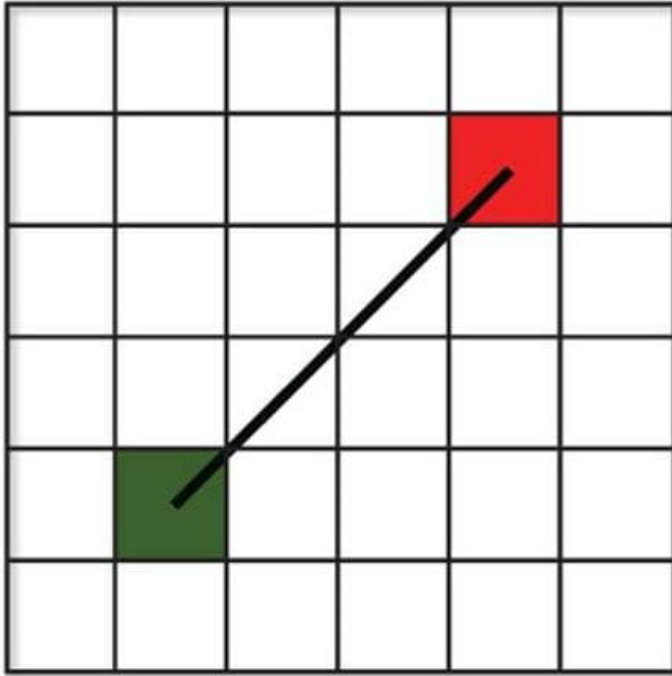


Figure 7: Example of Euclidean distance in 2-dimensions (From: Gotke, 2020).

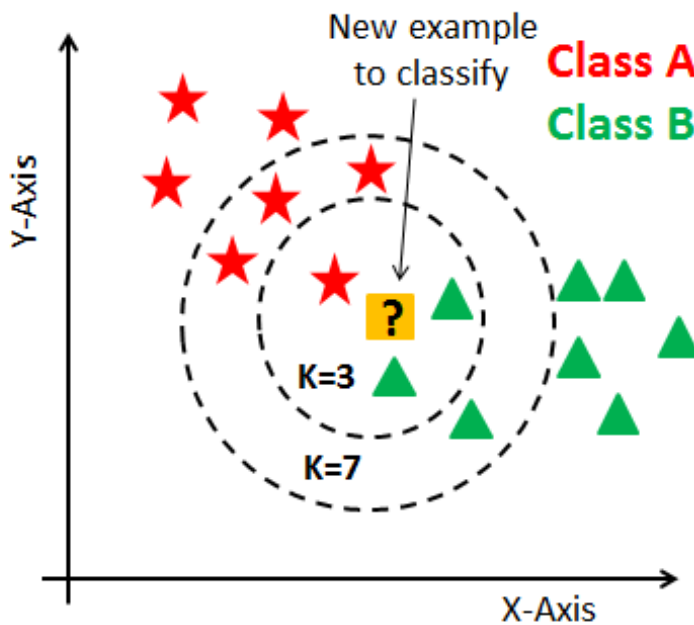


Figure 8: An example of KNN in 2-dimensional space for two different values for K , $K=3$ and $K=7$. At $K=3$, the new example would be classified as being in Class B and at $K=7$, it would be classified as being in Class A (From: Gotke, 2020).

At times where $K/2$ nearest neighbours are in each class, the ties are broken randomly (Zapp, 2024).

3.2.1.2. Training Overview

The KNN model was trained using the function *KNeighborsClassifier* from the sklearn library. A **grid search** (a test of what combination of hyperparameters train the best model) was performed on the following hyperparameters:

1. The number of neighbours: {1, 2, 3, 4, 5, 6, 7, 8, 9}.
2. The weights used to calculate: *uniform* (all K nearest points count the same), or *distance* (the closer of the K nearest points are more influential) (“sklearn.impute.KNNImputer,” n.d.).

This task was performed using sklearn’s *GridSearchCV* function. The search used ten **cross-validation folds**, in other words it split the training data into ten random parts and used nine of them for training the model and one for testing it. It repeated this process for each “fold” and each hyperparameter combination, measuring how accurate each model was. This means that with the nine nearest neighbour values and two distance measures, there were eighteen unique hyperparameter combinations, which combined with the ten folds meant the model was fit 180 times. The grid search and model fitting combined, took approximately two minutes to run and the optimal hyperparameter combination (the one that created the most accurate model on its testing folds) was found to be four nearest neighbours and *uniform* weights. All tests and training in this project were timed using the *time* function in the time module in Python.

3.2.2. Model 2: Support Vector Machines (SVM):

3.2.2.1. Mathematical Background

The SVM algorithm is another supervised learning algorithm that aims to classify a point based on its location. In the binary classification case, consider the set, $S = \{y_1, x_1, \dots, y_L, x_L\}$ in N -dimensional space. Then each training point, $x_i \in R^N$ belongs to one of two classes, $y_i \in \{-1, 1\}$ (transformed to $\{-0, 1\}$ in this project), for $i = 1, \dots, L$ (Lin & Wang, 2002). The goal is to find a **hyperplane**, or an N -dimensional boundary that separates some N -dimensional space into two parts (Weisstein, n.d.). This task is difficult to perform in the **input space**, the set of all potential values for the input of the function. A typical solution for this problem is to map this input space into higher-dimension **feature space**, a set consisting of the features which define a provided dataset (“What Is Input Space,” 2022; Lin & Wang, 2002). In this feature space, an optimal hyperplane is found where:

$$w^T z + b \geq 0, \quad (2)$$

which is defined by a pair $\{w, b\}$, where:

$$y_i \geq f(x_i) \geq \text{sign}(w^T z_i + b) = \begin{cases} 1, & \text{if } y_i = 1 \\ -1, & \text{if } y_i = -1 \end{cases} \quad (3)$$

where z is the corresponding feature space mapping from R^N to the feature space, Z , $w \in Z$, $b \in R$, and y is the classification. S is said to be **linearly separable** (i.e. a hyperplane can fully divide two classes), if for all elements of S :

$$= \begin{cases} w^T z_i + b \geq 1, & \text{if } y_i = 1 \\ w^T z_i + b \leq -1, & \text{if } y_i = -1 \end{cases} \text{ where } i = 1, \dots, L. \quad (4)$$

When S is linearly separable, an optimal hyperplane can be found that maximizes the space (margin) between the two linearly separable classes (see Figure 11).

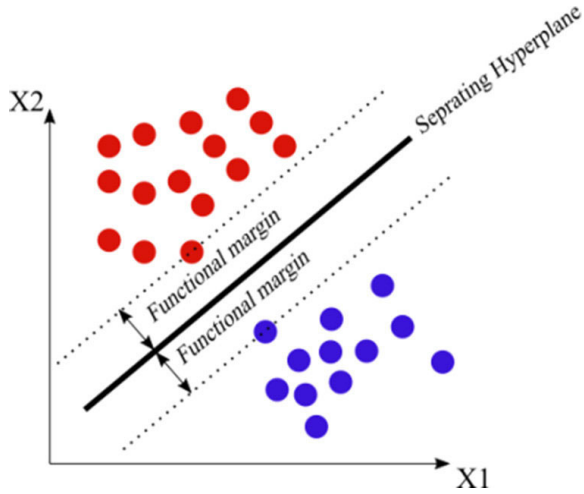


Figure 9: A linearly separable dataset in 2D (From: Saeidpour, 2016).

When S is not linearly separable, classification errors must be allowed and we introduce non-negative variables, ξ_i , where:

$$y_i (w^T z_i + b) \geq 1 - \xi_i, \text{ where } i = 1, \dots, L \text{ and } \xi_i \geq 0. \quad (5)$$

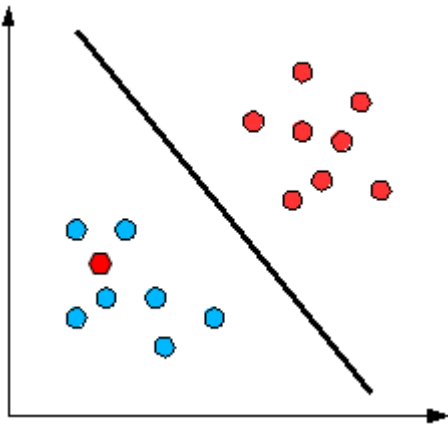


Figure 10: Non-linearly separable data (From: "When Data is NOT Linearly Separable", n.d.).

The next problem is to construct the optimal hyperplane, which can be found with the following quadratic programming problem:

$$\text{minimize } \frac{1}{2} \|w\|^2 + C \sum_{i=1}^L \xi_i, \text{ subject to (5),} \quad (6)$$

where C is a constant, called a regularization parameter, that controls the trade-off between margin maximization and allowed classification violations (see Figure 13). The term $\sum_{i=1}^L \xi_i$, is a measure of the amount of misclassification.

SVM Parameter C

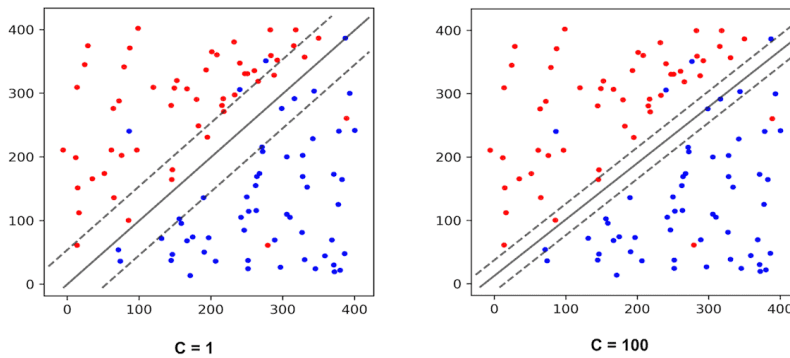


Figure 11: The difference between high and low C -values on the margins and misclassification allowed (From: Sharma, 2021).

(6) is a difficult problem to solve, but we can simplify it by using Lagrangian multipliers and transforming it into the dual problem:

$$\begin{aligned} \text{Maximize } & \sum_{i=1}^L \alpha_i - \frac{1}{2} \sum_{i=1}^L \sum_{j=1}^L \alpha_i \alpha_j y_i y_j z_i z_j \\ \text{Subject to } & \sum_{i=1}^L y_i \alpha_i = 0, 0 \leq \alpha_i \leq C, i = 1, \dots, L, \end{aligned} \quad (7)$$

where $\alpha = [\alpha_1, \dots, \alpha_L]$ is the vector of non-negative Lagrange multipliers satisfying the constraints in (5).

A particular solution, α_i , of (7) must satisfy the following, according to the **Kuhn-Tucker theorem**:

$$\alpha_i (y_i \cdot w \cdot z_i - b) = 0, \text{ where } i = 1, \dots, L \quad (8)$$

$$C - \alpha_i = 0, \text{ where } i = 1, \dots, L. \quad (9)$$

From this equality it follows that the only nonzero values α_i in (8) are those for which the constraints in (5) are satisfied with the equal sign. The point, x_i corresponding with $\alpha_i > 0$ is called a **support vector**, the point in a class that is the closest to the margin (see Figure 14).

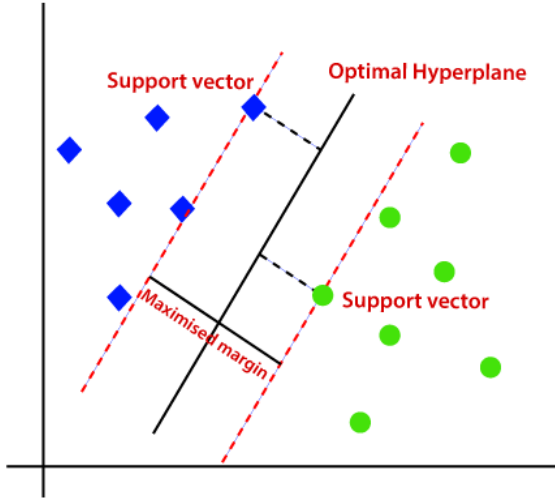


Figure 12: Labelled diagram of a support vector machine (From: Saini, 2024).

There are two types of support vectors in the non-linearly separable case. The first is when $0 < \alpha_i < C$, the support vector x_i satisfies the equalities $y_i \cdot w^T z_i = 1/b$ and $\xi_i > 0$. In the case where $\alpha_i > C$, ξ_i exists and the support vector x_i does not satisfy (4), these support vectors are called **errors**. The point x_i corresponding with $\alpha_i = 0$ is classified correctly and is the maximized margin (see Figure 14).

Then, to create the optimal hyperplane, $w^T z_i = 1/b$, we have that:

$$w = \sum_{i=1}^L \alpha_i y_i z_i. \quad (10)$$

The scalar, b can be found from the Kuhn-Tucker conditions (8 and 9). The final issue that must be addressed is that we do not know anything about the feature space, Z . To address this issue, we can use a **kernel** that can compute a dot product of data points (x_i, x_j) , in Z . The kernel function is:

$$K(x_i, x_j) = \sum_{l=1}^L \quad (11)$$

where d is the degree of the kernel.

Finally, we get the following problem for finding the optimal hyperplane:

$$\begin{aligned} W &= \sum_{i=1}^L \alpha_i^2 / 2 - \sum_{i=1}^L \sum_{j=1}^L \alpha_i \alpha_j y_i y_j z_i z_j K(x_i, x_j) \\ \text{Subject to } &\sum_{i=1}^L y_i \alpha_i \geq 0, 0 \leq \alpha_i \leq C, i = 1, \dots, L. \end{aligned} \quad (12)$$

We finally get a **decision function**, which calculates which class the new point belongs to:

$$f(x) = \text{sign} \left(w^T z - 1/b \right) = \text{sign} \left(\sum_{i=1}^L \alpha_i y_i K(x_i, x) / b \right), \quad (13)$$

which then outputs the predicted class of x (Lin & Wang, 2002).

3.2.2.2. Training Overview

The SVM was trained using sklearn's `svm.SVC` function. The function determined classes using an ovo ("one-vs-one") decision function shape, which split multi-class problems into binary problems with each pair of classes, determining which class each point should belong to. This approach only tests one pairing in the binary case (Brownlee, 2021). Another grid search was performed for hyperparameter tuning with ten cross validation folds. The hyperparameters searched for were:

1. The C value: {0.1, 0.5, 1}.
2. The type of kernel function: *linear*, or *poly* (polynomial).
3. the degree of the polynomial kernel {1, 2, 3}.

A C-value of 1, with a *poly* kernel of degree one was found to be the optimal set of hyperparameters. The search and model fitting took 4 minutes for Q2 of the S2023 Final .

Additionally, the SVM algorithm is a very computationally demanding algorithm, so it required a large amount of computer memory to compute, especially for large datasets. For the model to train on my system, the images had to be further shrunk by a factor of four to 320×250.

3.2.3. Model 3: Convolutional Neural Network (CNN):

3.2.3.1. Mathematical Overview

The CNN is a model that consists of several "layers" that each work together to train the model. The first layer is called a **convolutional layer**, which extracts features from the data. In the 2-dimensional case, it uses the following formula:

$$x_{s,t} - w_{\sigma,\tau} \odot x_{s'',t''} \quad \sigma, \tau \leq w; s, t \leq x \quad (14)$$

where x is the dataset, or image in this case, w is a set of weights called **filters**, or **kernels**, s, t are indices based on the size of x and σ, τ are indices used to help with feature extraction and are less than or equal to s and t in magnitude. Convolutional layers require an **activation function** which transforms the data. One of the activation functions used was the **rectified linear activation function** (ReLU), which outputs the input, x , if it is positive, and otherwise outputs 0 (Brownlee, 2020). The formula is as follows:

$$f(x) = \max(0, x) \quad (15)$$

The other activation function used was the **Softmax** activation function, which calculates a set of probabilities that a specific output of the network belongs to a certain class (Priya, 2023). The formula for the Softmax function for an N-class classification problem, where $N \in \mathbb{R}$, z is the output of the neural network, and e is Euler's number, is as follows:

$$\text{Softmax}(z) \quad (16)$$

After the convolutional layer, a **max pooling layer** is typically introduced, which condenses the information, computing the maximum over the different regions of the image, I used a region size of 2×2 (see Figure 15). The purpose is to shrink the amount of information, only focusing on the important regions.

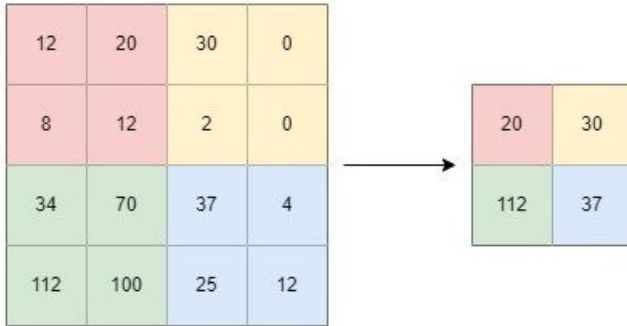


Figure 13: Example of a 2×2 max pooling layer (From: Nanos, 2024).

After, there is a **dropout layer**, which stops the impact of some of the **neurons** (inputs) in the network, reducing the total amount of information and preventing **overfitting** (learning too much about the training data so that the model performs poorly with new data).

Before the results can be outputted, the data must be transformed into a 1-dimensional vector, which is performed using a **flatten layer**, which “stacks” the data into a vector (see Figure 16) (Shah, n.d.). Finally, a **dense** (or **fully connected**) layer is used which takes in the features that have been used by the network and classifies each image based on their output (Dumane, 2020).

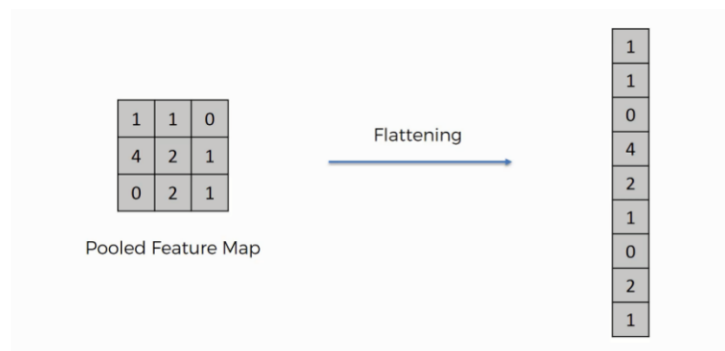


Figure 14: Example of a flatten layer (From: “Convolutional Neural Networks (CNN): Step 3 – Flattening”, 2018).

3.2.3.2. Training Overview

Before training the CNN, the images had to be loaded into a dataset. This task was performed using the Miscellaneous operating system interfaces (os) Python library and its `listdir` function to get the list of all file paths of each data image. The individual files paths were joined using the `os.path.join` function and each of those were added to a vector which created a direct link to each image. Then using the `pd.Series` and `concat` functions, all the file paths were added to a `pd` data frame.

Once I had this list of file paths, I took each image and ensured they were greyscale (this removed a dimension in which colour values were stored) and normalized the values by dividing them by 255 (colour values range from 0 to 255 inclusive). I did this by first checking if the file path existed, using the *path_exists* function in the *os* library, and then used the *load_img* and *img_to_array* functions from the TensorFlow (TF) library. TF was accessed via the Keras library, which serves as the Application Program Interface (API) of TF. The functions loaded the images into an array so they could be used with the CNN. The data was then split into training and testing as done before and the training set was further split into an 80-20 training-validation split, again using sklearn's *train_test_split* function. The **validation data** acts as test data within the training set and aids in training the model.

The model was then created using the TF library through the Keras library and its *Sequential* function and consisted of the following:

- An input layer with:
 - A *Conv2D* layer, with 32 filters, each with size 5×5 and the ReLU activation function, with an input shape of 640×500 .
 - A *MaxPooling2D* layer, with a pool size of 2×2 .
 - A *Dropout* layer of 25% of information.
- A hidden layer with:
 - A *Conv2D* layer with 64 filters of size 5×5 .
 - A *MaxPooling2D* layer identical to the one previous.
 - A 25% *Dropout* layer.
- An output layer with:
 - A *Flatten* layer.
 - A *Dense* layer with 32 neurons and a ReLU activation function.
 - A 50% *Dropout* layer.
 - A *Dense* layer with an Softmax activation function and two neurons, one for each class.

The ultimate result of this CNN was a binary classification for the exams. The model was trained using 50 **epochs** (50 passes through the dataset). It used the Adam optimizer, which dynamically adjusts the **learning rate** (how much the model changes when the weights are updated), and measured loss via **sparse categorical cross-entropy** (the difference between the discovered probability distribution of the model and the values the model predicted) (Brownlee, 2021; Brownlee, 2020; Pykes, 2024). The model took slightly over 1.5 hours to train.

3.2.4. Model 4: Large Language Models - ChatGPT

3.2.4.1. Overview of ChatGPT

ChatGPT is a popular LLM, created by OpenAI. GPT stands for Generative Pre-trained Transformer and the models are used for in many different applications such as the Bing search engine and the Jasper writing tool. ChatGPT models were trained on enormous amounts of unlabelled text data (almost the entire open internet at the time of training). It used neural networks plus a few ground rules to learn the rules and relationships that form comprehensible text. This is how the model can understand a **prompt** (a message written to the model asking it to perform a specific task) and respond with text that is coherent.

ChatGPT uses what is called a **transformer model**, a kind of neural network that reads all the words in a sentence at once, instead of going one-word-at-a-time like the old recurrent neural networks that were popular in earlier LLMs. The model reads words as **tokens**, which are bits of text that are stored as a **vector** (a number with position and direction), so the transformer can remember the information. This allows vital information from earlier in a paragraph to be remembered which results in a more accurate comprehension of the language by the model. *GPT-3* was trained on five hundred billion tokens, it is unknown how many tokens *GPT-4* was trained on, but it is likely more, as it performs much better. In general, little information has been made public on the *GPT-4* model. The massive amount of training data means that the models have been trained on more-or-less the sum of human knowledge, so they can give reasonably accurate responses on almost any topic. Additionally, the complexity of the model is exceedingly high, *GPT-3* has about 175 billion **parameters** in its neural network. When you provide a prompt, it uses the different values, weights, and some randomness that were given to these different parameters, to determine the “best” response. It is likely that *GPT-4* has more parameters, which are also trained better, as that model is a significant improvement from its predecessors. This combination of factors is the probable cause of the improvement, as overly complex parameters on their own can lead to overfitting, which would be detrimental to the model.

The results from the internet training were not ideal, as the model was given minimal guidance, so developers implemented **Reinforcement Learning from Human Feedback** (Guinness, 2023). A prompt was created, and the model gave several responses, then a human ranked the responses from best to worst so the model could learn how to best respond to prompts. The result was an efficient training method that improved the model’s ability to answer prompts, making it suitable for widespread use (Ramponi, 2022).

When a prompt is provided, the transformer model breaks down the sequence of words into tokens and tries to determine the most important parts of the prompt. Once that step is completed it runs the neural network again, this time generating what it believes to be the best response. The randomness that is included with the parameters allow for the same prompt to not receive the exact same answer twice (Guinness, 2023).

Overall, the model takes in the prompt as a series of tokens and due to its extensive training, knows how tokens relate to each other. Using a complex neural network framework, it then outputs a response. ChatGPT does not understand English, nor any other human language, but rather knows how the token values work together and uses that mapping to produce a response that is (usually) correct and understandable by a human reader.

3.2.4.2. Code Preprocessing, Prompt, and Test Overview

I accessed ChatGPT via its API, and the OpenAI Python library, through its *chat.completions.create* function. Using an API key (a set of unique characters linking the program to an API account), I created a for-loop that cycled through the dataset and went answer-by-answer using a prompt. The first model I used was the *GPT-4-Turbo-Preview* which was OpenAI’s newest model at the time.

The data was imported using the *read_excel* function and every blank cell (times when students did not answer the question) was replaced by a period. This replacement was necessary since the model could not grade fully blank responses.

I provided the model with the following prompt when grading Q9 of the W2023 Final: “only respond with a score out 0 for a completely wrong answer and 1 for any other answer: *LaTeX code of Question*. Solution: *LaTeX code of solution*.” The idea behind this prompt was to give the model some instruction on how to grade and to provide it with the official solution. This prompt provided zero training data and followed an extremely basic marking scheme. It was also important to instruct the machine to only respond with the grades, as otherwise it explained what was wrong with each solution. Having it provide a 0 or 1 allowed for easy conversion of the output from a string to an integer, so I could compare the accuracy with the target data. The model took a little over 5 minutes to mark all 455 solutions in the dataset.

I then provided training data by adding solutions and their corresponding scores to the prompt. The training data replaced the “Solution: *LaTeX code of solution*” part of the previous prompt and in between each training example, I added “scores a *target_value*,.”

After including five training examples per class, I could no longer increase the amount of training data in the W2023 Final question due to usage caps (see section 5.2.), so I switched to the mock exam for the remaining tests. This smaller dataset was also preferred to the S2023 Final, since it was shorter and thus more cost-effective and efficient to run.

Initially, I performed tests on the mock exam with the *GPT-4-Turbo-Preview* model, however, I realized that this dataset was small enough that I could try using the more expensive and lower usage-capped *GPT-4* model. The *GPT-4* model did a much better job of grading (see Figure 22).

I had initially considered trying **prompt engineering**, seeing which prompt structures performed the best, but with this more expensive model, I decided against pursuing that task. I instead decided to re-use the same prompts and include different amounts of training data. I graded the two questions from the mock exam using two different schemes: 0% vs 100% (excluding all results from class 1) and using all three classes.

I also tried to preserve the efficiency of the program by combining multiple solutions into a single entry in the data vector, separated by a pipe symbol (`|`), and instructed the model to grade several solutions at once. This mostly worked with ten answers per iteration, however, at times the LLM would give an inconsistent number of grades (see Figure 31). When that inconsistency arose, I iterated through the dataset one solution at a time. I also changed the target vector, so the 100% class was class 1, for the binary test only.

The full prompt I used for the binary Q1 tests was as follows, I avoided using some punctuation at the end of the prompt to save tokens:

respond with 0 for a wrong answer and 1 for a right answer, only answer with 0 or 1, the question is: *LaTeX of question*. Here are some examples: *Example solutions separated by a “scores a” and the classification of that answer*. Each content has ten answers, each separated by the pipe symbol. Please provide 10 answers for each, separated by a comma

This prompt was designed to tell the machine what to do and how to distinguish between separate solutions. The last clause was necessary as the default output was to separate each answer by the pipe symbol. Using the comma provided a nicer looking output. The values were then split into individual outputs using Python’s *string.split* method and cast to integers and

stored in a vector so it could be compared with the target data. The no-training case did not include the parts of the prompt relating to the training data.

Since the data was initially organized by class, I used sklearn's *train_test_split* function to split the data randomly into training and test and stratified on the target data to ensure that each group had an even amount of training representatives. I used the *random_state* parameter each time I split to ensure reproducibility, as the splits would be the same each time. With no training data, I could not use this function, so I randomly permuted the indices using the *np.random.permutation* function to have a random order. I used a This action was done since the solutions were grouped by class in the original dataset.

For the 3-class classification, I followed the same steps as before, except I did not usually combine solutions. I also adjusted how the prompt was structured, but kept the main points the same and added some additional information:

respond with 0 for a wrong answer, 2 for a right answer, and 1 for an answer that is partially correct. Only answer with 0 or 1 or 2, the question is: *LaTeX of question*. a 2 must be given when the correct answer, $1/3$ is written and a sufficient amount of work is shown, a 1 should be given when either some correct work is shown, or the correct answer is given with no work. The training values are as follows: *training examples formatted the same way as the binary case*.

I added extra details to distinguish between classes since the difference between classes was not as stark as with the binary classification. Only in the no-training case was I able to do ten grades per content chunk, so I added in the extra instruction about the ten answers at the end of the prompt only in that test.

For Q2, the prompt was adjusted to match this new question. For the binary classification, I used the following:

respond with 0 for a wrong answer and 1 for a right answer, only answer with 0 or 1, the question is: *LaTeX code of question*. Here are some examples: *training data formatted the same way as in Q1*. Each content has 10 answers, each separated by the pipe symbol. Please provide 10 answers for each, separated by a comma.

For the 3-class case, I added extra instructions to help distinguish between the classes:

respond with 0 for a wrong answer, 2 for a right answer, and 1 for an answer that is partially correct. Only answer with 0 or 1 or 2, the question is: *LaTeX code of question*. Part marks should be given when some correct steps are shown and the answer is incorrect or incomplete, or where the answer is given but little to no work is shown. The correct final answer is: *the final answer to the question*. *Training examples when necessary*.

Again, only in the no-training example, was I able to combine the solutions, so I included that last part about the 10 answers to that prompt.

Finally, I ran four tests using just the answer key as training data, one on each type of classification for each mock exam question. I kept the prompt simple otherwise:

respond with 0 for a wrong answer and 1 for a right answer, only answer with 0 or 1, the question is: *LaTeX of question*. The answer is: *LaTeX of answer key answer*. Each content has 10 answers, please provide 10 answers for each, separated by a comma.

In the multi-class case, I adjusted the first line to match the different class system: “respond with zero for a wrong answer, 2 for a right answer, and 1 for an answer that is partially correct. Only answer with 0 or 1 or 2, the question is: ...”.

4. Results and Analysis

4.1. Image Classification

This project took two separate approaches, image classification (models 1-3) and Natural Language Processing using pre-trained LLMs (model 4). See the GitHub repository for examples of each kind of model and test. Most numeric results below are displayed using a **confusion matrix**, where values on the main diagonal are the correctly classified student answers and values off that diagonal are incorrectly classified. **Accuracy** was calculated by dividing the number of observations on the main diagonal (i.e. correctly graded ones) by the total number of observations in the matrix. When multiple tests are displayed at once, their accuracies are presented using line graphs. The confusion matrices were created using the *confusion_matrix* and *ConfusionMatrixDisplay* functions from the sklearn library, and all plots were shown using the *plot* function from the matplotlib Python library through its API, pyplot.

4.1.1. KNN

Figure 17 shows the results of the KNN model on the Q9 of the W2023 exam.

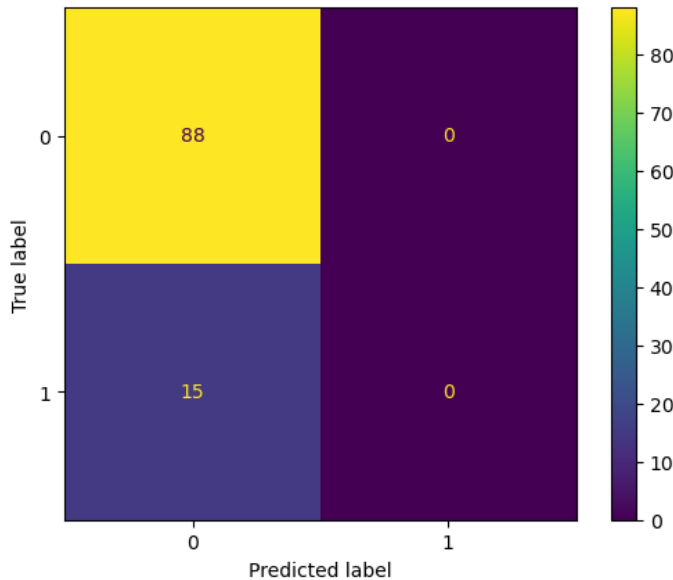


Figure 15: The confusion matrix from the KNN on the image data from Q9 of the W2023 Final.

Unfortunately, the algorithm assigned every value to class 0, which was the class that most of the answers in the dataset scored on that question. In fact, 443/511 were included in class

0 meaning almost 87% of students scored 0/4 on that question. This result occurred with all KNNs trained on each question tested.

For models 1 and 2, I got the same kind of results for each question graded, so only one test is shown for each, as adding more would be redundant.

4.1.2. SVM

The results of the SVM did not differ from the KNN, so no further hyperparameter searches were performed as they would have been unlikely to have achieved any meaningful improvement. Figure 18 shows the results of the SVM on the S2023 Final.

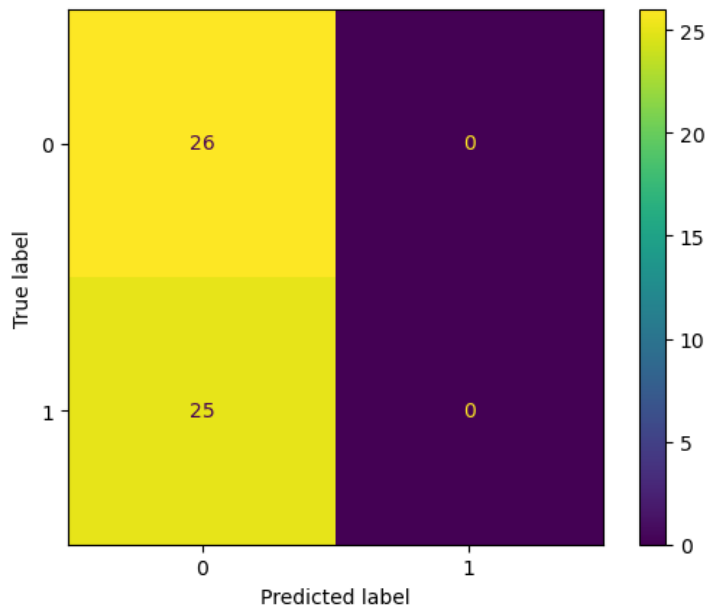


Figure 16: The results for Q2 on the S2023 exam.

For Q2 on the S2023 Final, all values were again classified as belonging to the majority class. In Q2, the classes were more balanced than Q9 on the W2023 Final, as class 1 had 107 members and class 0 had 145 members. Class 1 were all the answers that scored 100% and class 0 were all the other answers. Unfortunately, this closer balance did not alter the result. The SVM did not provide any more meaningful results.

4.1.3. CNN

Figure 19 shows the results of the CNN on Q9 of the W2023 Final.

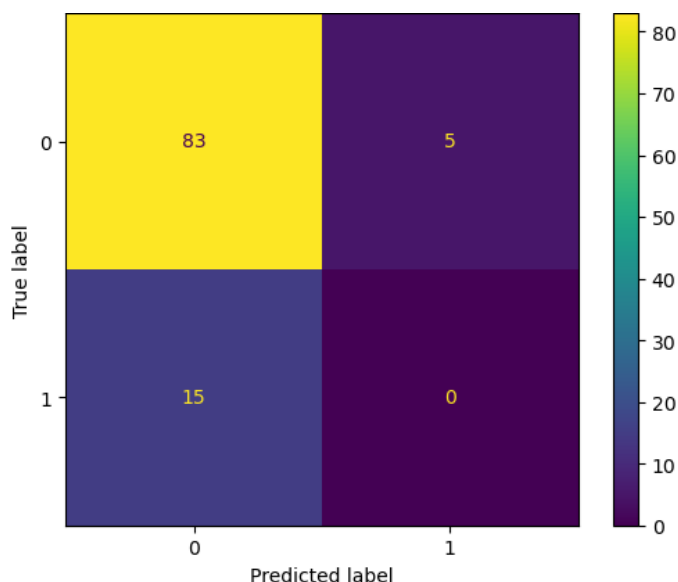


Figure 17: The result of the CNN on Q9 of the W2023 Final.

The CNN did not assign every value to the same class, however, it only assigned five values to class 1 and none of those predictions were correct. Consequently, this model did not provide any more meaningful results than the other models. This result, combined with the long training time made me decide not to pursue further model improvement. The S2023 final also produced meaningless results and are therefore not displayed.

The LaTeX code was also tested as images with these algorithms, however, this approach was ill-suited for that data and the results were just as meaningless as before, so they will not be displayed.

4.2. LLMs

The image classification algorithms did not work at all. All images were assigned to one class in each test. This seemed to show that there were not enough differences between a “correct” and “incorrect” answer to be easily distinguished. This problem was confirmed by Dr Christopher Henry of the Computer Science department at the University of Manitoba, who explained that for image classification to be effective, especially with CNNs, it requires a problem where someone with no prior knowledge could distinguish between classes. In other words, for these approaches to work, someone who does not have any background in mathematics, English, or even numbers, should be able to tell the difference between a right and wrong answer, without having access to an answer key every time. This is clearly not possible, as right and wrong answers depend on seeing the correct answer and/or correct steps in the student’s work. If the machine only sees images of writing or text, it is unable to distinguish between a right or wrong answer since it cannot parse through the individual written characters when looking at the entire exam page. This is why a CNN performs well at differentiating between a picture of a dog versus a cat, as such a difference is much more distinct. Someone could tell that they are different animals even if they had never seen either creature before.

This meant that a different approach had to be determined and after speaking with Dr. Tristan Miller of the Computer Science department at the University of Manitoba, the decision

was made to use pre-trained LLMs to grade the LaTeX code. I settled on using the ChatGPT models as the LLMs of choice.

4.2.1. W2023 Final

Figure 20 shows the result of having no training data on the W2023 Final.

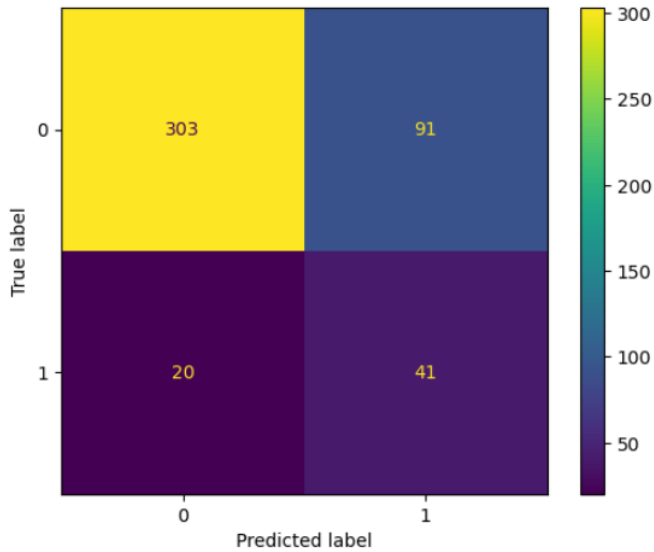


Figure 18: The result of GPT-4-Turbo-Preview on Q9 of the W2023 Final with no training data.

The accuracy of this test was 67%, which was not great, however, it was a much more promising result than before. The model did not assign everything to one class and had correct classifications in each class. This occurrence was a sign that the model was distinguishing between the classes, albeit inaccurately.

Figure 21 shows the results of having five training examples per class (10 total).

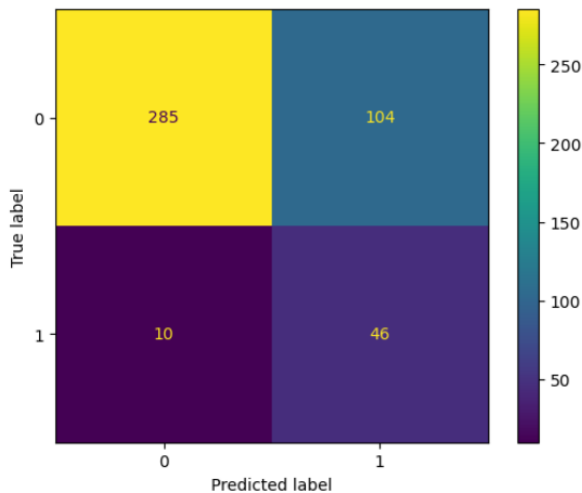


Figure 19: Results for Q9 of the W2023 final, with five training examples per class.

The accuracy increased to around 75% with just a few training examples provided. This result seems to suggest that if some exams are graded, or if sample solutions are provided, it can improve the results of the grading. Ten total training examples was just over 3% of the dataset. Token limits prevented me from adding any additional training data.

4.2.2. New Model

With the mock exam, I decided to compare the performance of the *GPT-4-Turbo-Preview* model with *GPT-4*. Figure 22 shows the result of using the exact same data and prompt with the *GPT-4-Turbo-Preview* and the *GPT-4* models.

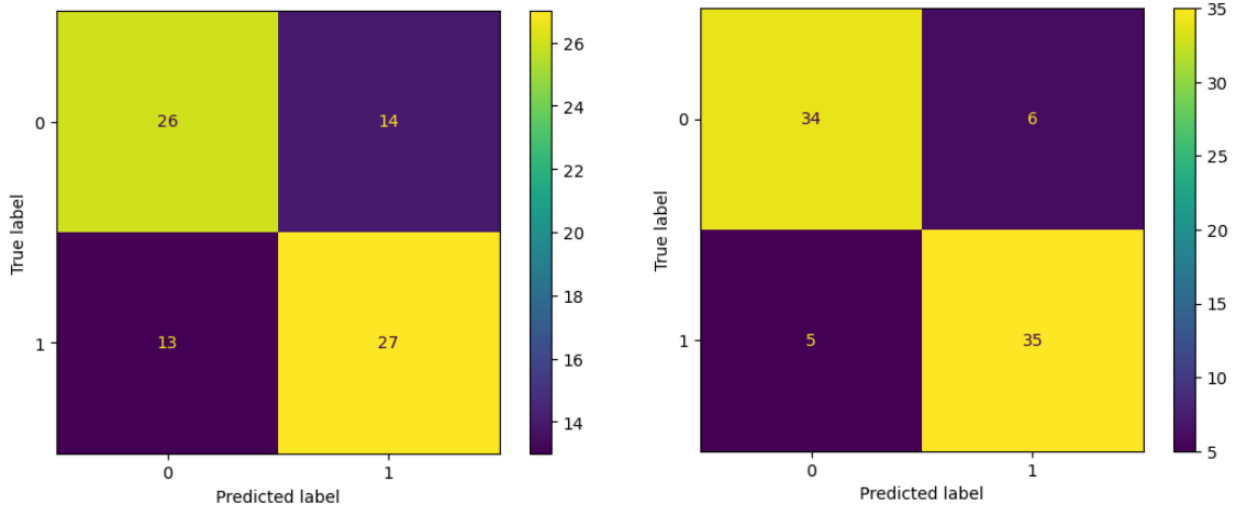


Figure 20: The same prompt and data run on two different models *GPT-4-Turbo-Preview* (left) and *GPT-4* (right).

With all-else-equal, the *GPT-4* scored 14% higher in accuracy, 86% versus 72%. From this point forward, I abandoned the Turbo model in favour of the *GPT-4* model for all subsequent tests.

4.2.3. Q1 Mock Exam

Figure 23 shows the accuracy of the 2 and 3-class classification tests at the various levels of training data.

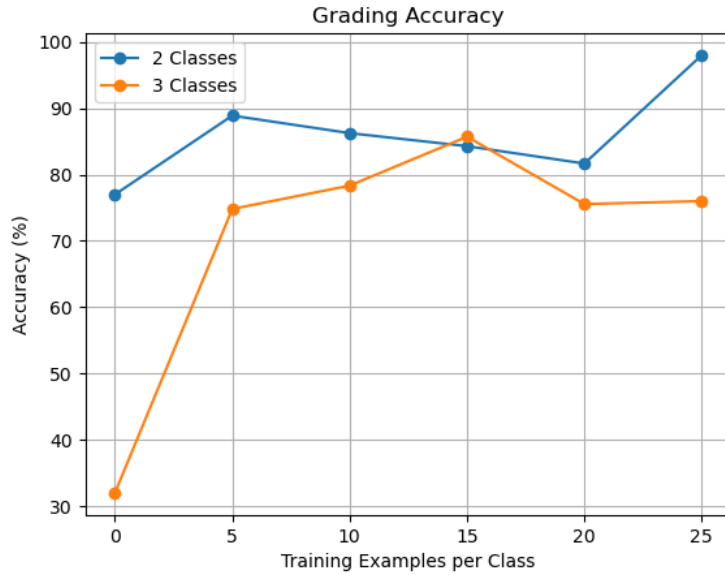


Figure 21: The accuracy of the grading of Q1 on the mock exam at various training levels.

We see that when I added a few training examples, the accuracy increased, especially in the multi-class case. There was a significant increase in accuracy within the multi-class case from using no training to five training examples, and a lesser, but still sizable increase in the binary case. Oddly enough, as I added more training examples to the binary case, the accuracy decreased until I used twenty-five examples in each class (a 50-50 train-test split), where the accuracy was almost perfect. Also, fifteen training examples per class (a training size of 30%) had the best accuracy score for the 3-class case, scoring in the mid eighties.

Unsurprisingly, the binary classification scored better, since the classes had a larger difference between them, as no part-mark solutions were provided. However, getting an accuracy in the mid eighties for the 3-class classification is still a very promising result.

Figure 24 shows the best result from each classification.

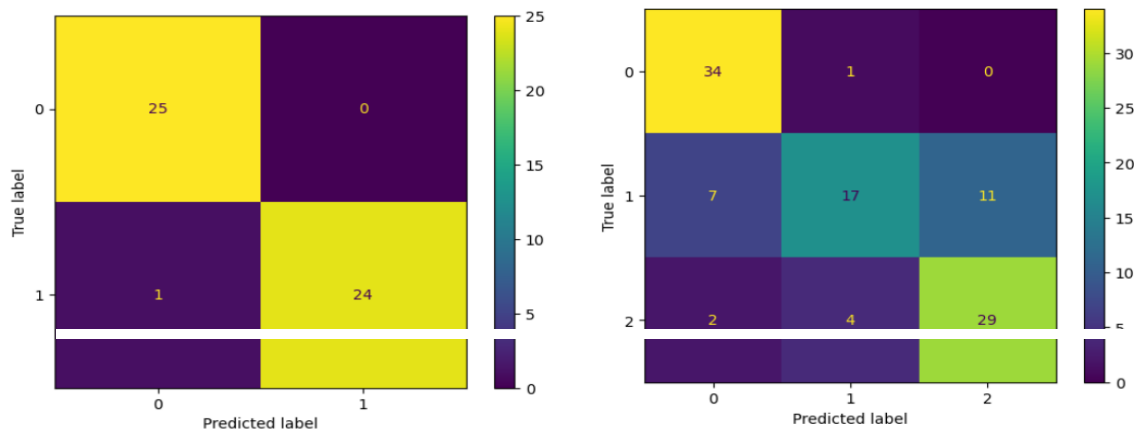


Figure 22: The best results for each classification type from Q1.

As shown above, the binary classification performed very well. The 3-class classification also did well, but there was some trouble with class 1, which had less than 50% accuracy within it. This was not a surprising result, as there were many ways to lose marks, such as skipping steps or using incorrect variables. These mistakes may be hard to define as partially correct/incorrect as opposed to fully correct/incorrect.

In terms of time, the grading was quick. For the binary case where I combined the answers into groups of ten, the time varied from 10 to 15 seconds to grade each question and converting the output to strings took negligible time. For the multi-class problem, the first solution took around 20 seconds, which was the only test that I was able to combine solutions on. The other examples where the prompt ran through each test answer individually ranged from 60-80 seconds to grade.

4.2.4. Q2 Mock Exam

Figure 25 shows the accuracy results for each test at different training levels. The class types are separated into different graphs since I used different amounts of training data in each test.

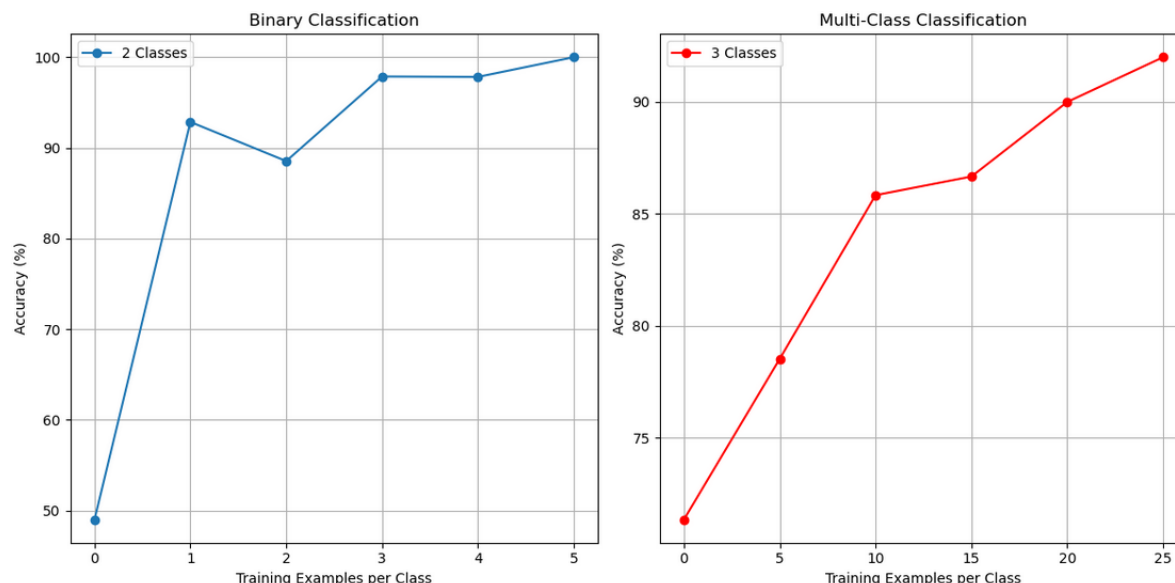


Figure 23: The accuracy results for the Q2.

From the above graphs, adding any training data made an enormous difference in this problem. In the binary case, simply adding one example per class increased the accuracy by over 40%. Having five examples per class (10% of data as training) gave a perfect score. For the multi-class case, using half the dataset as training resulted in an accuracy over 90%. These results show that this LLM is well-suited for grading derivation-type problems and is perhaps even better at it than grading numeric computation questions. Both class breakdowns performed better in the derivational Q2 than in the computational Q1.

Figure 26 shows the best results.

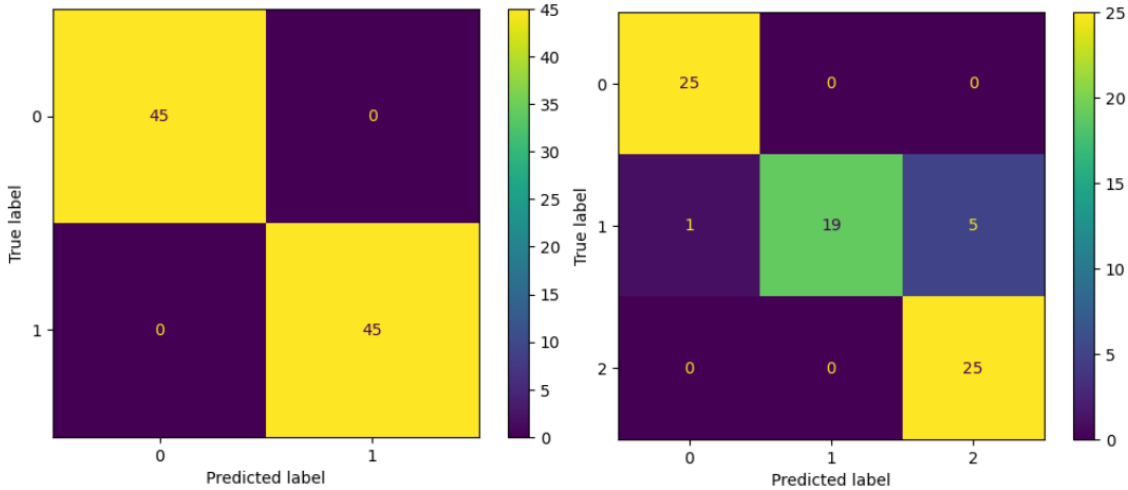


Figure 24: Best results for Q2 on the mock exam.

We see the perfect classification for the 5-training example binary test, but in the multi-class case, the accuracy for class 1 was much better than in Q1, scoring around 75% accuracy. This is reflected in the higher overall accuracy score. We also see that all examples from classes 0 and 2 were classified correctly.

The tests took at most 26 seconds to run for the binary classification with the combined answers. The multi-class problem took a lot longer, but the longest grading only took a little over 4 minutes to run; many orders of magnitude faster than any human could grade.

4.2.5. Answer Key Only

Figure 27 shows the results for Q1.

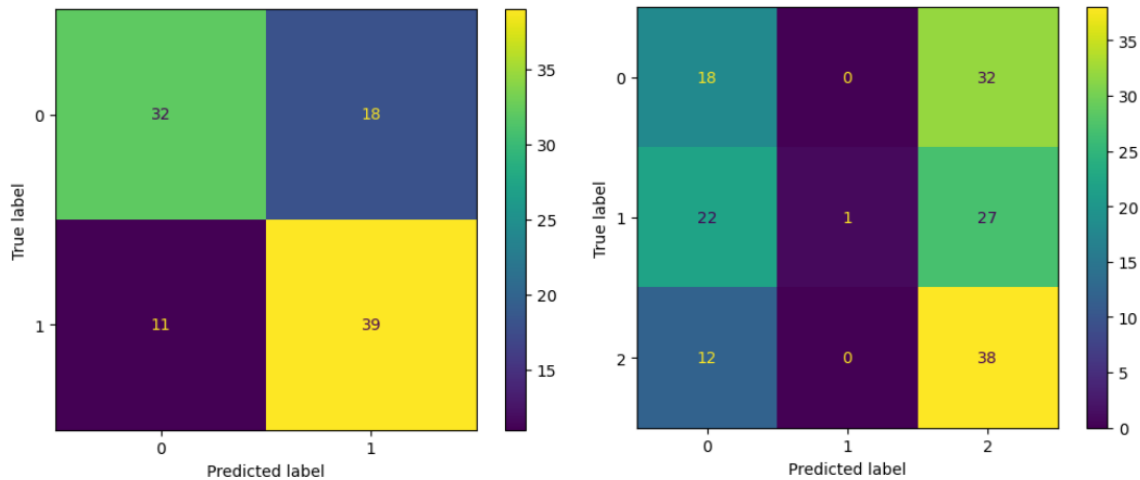


Figure 25: The results from the two types of tests for Q1 of the mock exam where the answer key solution was provided instead of providing training examples.

In the binary case that the accuracy increased from 69% to 71% when compared to using no training data and in the multi-class case, the accuracy decreased from 63% to 38%. It does not

appear that including the simple answer key instead of adding training data is a good idea. Class 1 was again classified poorly.

Figure 28 shows the results for Q2.

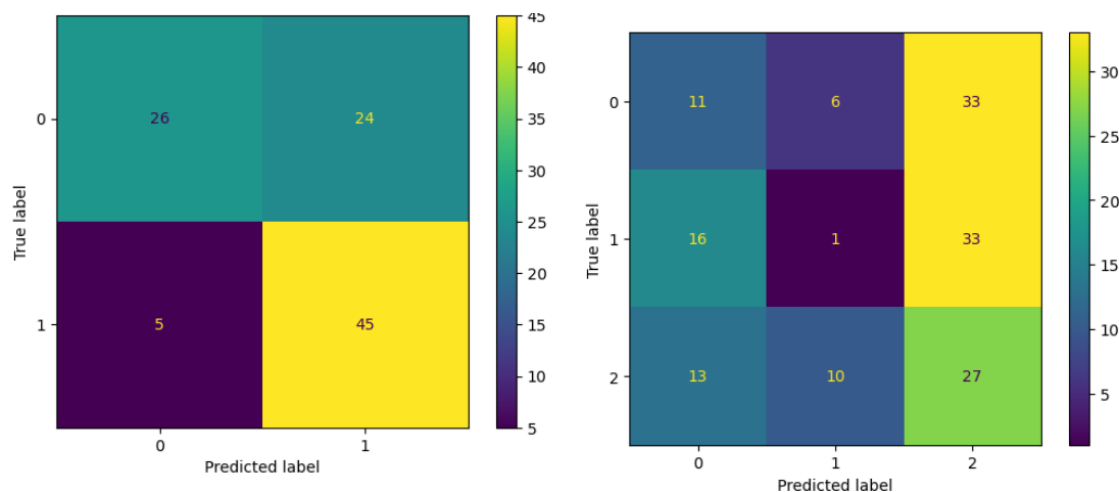


Figure 26: The results from the two types of tests for Q2 of the mock exam where the answer key solution was provided instead of providing training examples.

Compared to no training data, the binary case accuracy increased from 49% to 71%, which is a substantial increase, but is still less accurate than having just a single training example per class. The multi-class test accuracy decreased from 71% to 26% when compared to no training data. This again indicates that providing the solution key instead of training is not a good approach.

5. Conclusion, Issues, and Future Work

5.1. Conclusion

Overall, this project intended to serve as a baseline for solving the problem of finding a quick, accurate, and cost-effective way to grade exams. It also aimed to grade MATH 1500 exams in a way that produced accurate results and offered a path forward for further research and development into the issue (see section 5.3.).

In terms of accuracy, the project saw good results with the truncated grading scheme of fully correct versus fully incorrect. To a lesser extent, the project also had promising results with a third class that included solutions that were neither fully correct nor incorrect. In the mock exam, which served as a “perfect scenario,” since it was written with even class sizes and the answer key as a basis, the binary case performed very well. It scored in the high nineties in accuracy for Q1 and was perfect in Q2. The multi-class problem scored in the mid eighties in accuracy for Q1 and over 90% in Q2. These results occurred with at most half of the dataset being used as training data, a small amount as far as machine learning is concerned; the failed image classification models used 80% of the dataset as training. This high accuracy grading took no longer than a few minutes to perform in the worst case and mostly took just a few seconds, so it clearly achieved the goal of finding a fast way to grade. Furthermore, this increase in accuracy for the derivation question indicates that this model is adept not only in numerical calculation

questions but is potentially better with non-numeric questions. This appears to indicate a high level of robustness with ChatGPT for this problem. Finally, human graders do make mistakes, so even when humans are grading, the accuracy will not be 100%. Overall, this project achieved its goals in terms of serving as a baseline for grading accuracy as well as finding a fast way to grade exams.

The project left something to be desired in terms of its goal of being cost-effective. The most expensive test, according to the OpenAI pricing guide and their tokenizer machine, cost a little under \$20 USD (or \$27 CAD at the time of writing) (“Pricing, n.d.; “Tokenizer”, n.d.). This was the test where I used a 50% train-test data split for the multi-class case on Q2. When I used smaller amounts of training data, or combined solutions, however, the cost went down. In this worst-case scenario, a 10-question exam with the same amount of training data and five times the number of solutions per question would cost around \$1000 USD (\$100 USD per question), which is around \$1350 CAD. This price is equivalent around seventy grader-hours, assuming a \$20 CAD/hour rate of pay. This is hardly an improvement from how much it currently costs to grade an exam by hand.

Although this price is not ideal, it is not a bad place to start, as that price calculation was the worst-case scenario. For example, the 5-training example test for the same question cost around \$7.50 USD and the no-training example, with combined strings, cost just \$0.56 USD. If reliable ways to consistently grade multiple questions at once can be determined, the prices will decrease substantially. With some work, this approach can become very cost-effective.

Overall, this combination of price, accuracy, and speed, especially when compared to the results of the image classification algorithms, indicate that using LLMs like ChatGPT are the way forward for future work on this problem. This project achieved its goals for accuracy and speed and demonstrated a promising baseline for grading exams in a cost-effective manner.

5.2. Limitations, Challenges, and Remedies

There were several issues that arose with this project that will need to be addressed in future work.

Mathpix does a very good job of converting writing to LaTeX code, however, it reads values left to right and sometimes misses values, so the results can be inaccurate (see Figure 29). Mathpix does allow you to edit the output code and provides what the output looks like for easy reference, but when you have large datasets, the website is slow and causes problems while editing. A very good computer would be required to clean the data on there. Additionally, if Mathpix cannot read something, it leaves an image of it and puts a link to the image in the code (see Figures 7 and 8). A user who is particularly adept in LaTeX can add the answers in, but either way it would require someone to input a student’s answer, which can lead to mistakes, and it is a very time-consuming process. A remedy to this problem would be that such poorly converted questions be flagged and then marked by hand and the LaTeX for that question discarded. Additionally, Mathpix does not have an option to export the data to a spreadsheet, so the transfer must be inefficiently performed by hand. Finally, Mathpix does cost money to use, around \$10 USD per month (“Mathpix Pricing”, n.d.).

$$f(x) = \frac{2}{3}(4-x^2)^{\frac{3}{2}} \cdot \frac{\ln x}{-2} + c$$

cancel factor of $\frac{3}{2}$ cancel $-2x$ from chain rule hard to cancel result of product rule

$$f(x) = \frac{2}{3}(4-x^2)^{\frac{3}{2}} \cdot \frac{\ln x}{-2} + c$$

cancel factor of $\frac{3}{2}$ cancel $-2x$ from chain rule result of product rule

Figure 27: How Mathpix reads left to right and misses values.

To address some of these problems with converting, the exams would have to be formatted differently. First, each page would need an explicit ID number so that the pages can be uploaded in order and to allow for easy differentiation between answers in the LaTeX code. This is necessary so the data can be easily transferred to a spreadsheet. Additionally, the exams must be formatted in such a way that students are forced to do their work down the page, to prevent various parts of their answers being converted together and thus limiting conversion mistakes. Furthermore, the exams likely should not be made for Crowdmark and must have limited instructions on the question pages. The Crowdmark ID and text on the top of the pages added a lot of noise to the data that had to be removed (see Figure 30). This removal was among the main cleaning actions performed on the answers from Q9 of the W2023 final. Finally, although no such instances were analyzed in this project, multiple questions should not be included on the same page, as more work will have to be done to separate each question to grade, as the current methods do not allow for questions on the same page to be separated automatically.

매집

2131A57A-1EB2-41B2-900E-B38C8164F34D



23wmath1500final

Hat #104 Page 2 of 16

回新落 10

Final Exam

Winter 2023

1. (4 points) Evaluate the following limit

$$\lim_{x \rightarrow -3} \frac{\sqrt{x+7} - 2}{x+3}$$

Figure 28: Noise added by Crowdmark details, taken from the Mathpix output.

For the LLMs, the key issues were the cost and usage limits. This project cost over \$500 CAD to perform all the tests and tinkering. This prevented me from doing a lot more analysis, such as prompt engineering and using the S2023 Final with the LLMs. Additionally, the usage limits prevented me from doing more with the larger exams and forced me to create and use the mock exam. I did not use the *GPT-4* model on the W2023 exam for this same reason.

Tokens are how the API counts usage and calculates the cost. The API for *GPT-4* costs \$30/million tokens input and \$60/million tokens output. This high output cost is one reason I was

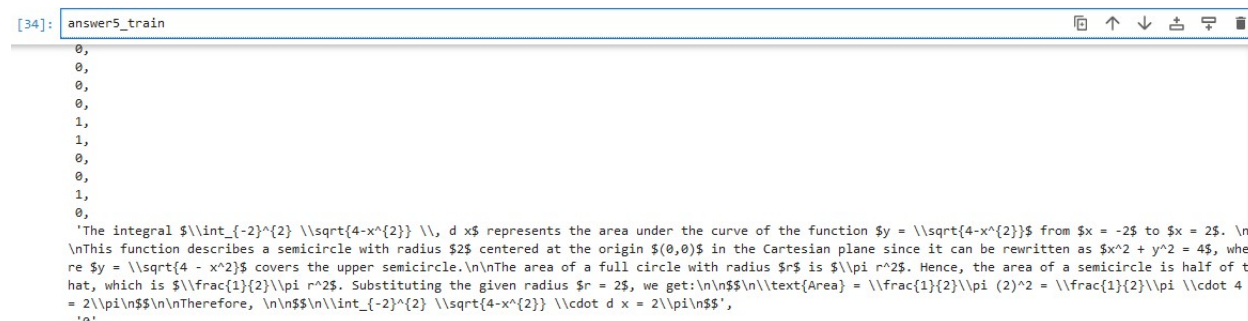
so explicit on limiting the output to just one number per solution. The *GPT-4-Turbo-Preview* model is cheaper, being \$10 and \$30/million tokens input and output respectively, but much less accurate.

Each time you iterate through a new entry in the dataset, the prompt is re-inputted. For example, if your prompt is 1000 tokens and you must run the loop 100 times, then the prompt contributes 100 000 tokens on its own. If the dataset is another 100 000 tokens total, then the entire question costs 200 000 tokens to grade (output tokens were negligible in my tests). For the maximum example from before, the prompt was 8731 tokens, which was run for each of the 75 test examples and the number of tokens for the test data was 8239. Therefore, that test ended up using $(8731 * 75) + 8239 = \mathbf{663\ 064\ tokens}$. From that math, prompt length and iteration numbers easily contribute the most to the token count whilst performing these tests. This reality is why I tried to combine solutions so I could grade multiple solutions at a time. Having the model output two solutions per run would half the token usage and having ten solutions per would reduce it by ten times. Unfortunately, in the multi-class case the machine would not give the same number of output grades every time, making it impossible to compare the answers to the target data (see Figure 31). This problem meant I had to go with the inefficient and expensive 1-answer-per-run approach. This same output problem happened when I tried grading all solutions at once, ten solutions worked the best for most tests. Additionally, due to the size of the W2023 Final, I did not perform any more tests on it, since the dataset itself was 85 000 tokens, if combining answers did not work, I would have had to run the prompt hundreds of times, making the tests too big and expensive to run.

```
'0, 2, 2, 0, 2',
'0,0,0,0',
'0, 0, 2, 0, 0',
'2, 2, 2, 0, 1',
'1, 2, 1, 0, 2',
'2, 2, 0, 0, 2',
```

Figure 29: The machine outputting an inconsistent number of grades per iteration. Here it was instructed to output five grades, but the second line above only outputs four grades.

Even with the 1-answer-per-run approach, the model would not always print out a class name. Sometimes it would say: “1/3 scores a 1”, which I would manually adjust to equal one, but other times doing so would not be possible (see Figure 32), so I would have to re-run the grading until it gave a proper set of grades.



```
[34]: answer5_train
0,
0,
0,
0,
1,
1,
0,
0,
1,
0,
'
The integral  $\int_{-2}^2 \sqrt{4-x^2} \, dx$  represents the area under the curve of the function  $y = \sqrt{4-x^2}$  from  $x = -2$  to  $x = 2$ .
This function describes a semicircle with radius 2 centered at the origin  $(0,0)$  in the Cartesian plane since it can be rewritten as  $x^2 + y^2 = 4$ , where  $y = \sqrt{4-x^2}$  covers the upper semicircle.
The area of a full circle with radius  $r$  is  $\pi r^2$ . Hence, the area of a semicircle is half of that, which is  $\frac{1}{2}\pi r^2$ . Substituting the given radius  $r = 2$ , we get:

$$\text{Area} = \frac{1}{2}\pi (2)^2 = \frac{1}{2}\pi \cdot 4 = 2\pi$$

Therefore,  $\int_{-2}^2 \sqrt{4-x^2} \, dx = 2\pi$ .'
0'
```

Figure 30: A bad output from the model.

The usage limits are in place, so you do not use the models too much. For *GPT-4* the usage limit was 300 000 tokens per minute at usage level 4, so if make your test was too big, it would not run (“Rate limits”, n.d.). Our longest test worked since it took a couple of minutes to run, keeping it below the per minute request limit. Additionally, the prompt length in the multi-class tests became too long when I tried doing more than twenty-five training examples per class, so I could not run anything more than that. Token limits were the reason I could not use more than five training examples in the W2023 tests. Limits do increase with time and usage, and I reached usage level 4. This level allowed me to run larger tests using *GPT-4*, which had a much lower usage limit than *GPT-4-Turbo-Preview* (whose level 4 limit was 800 000 tokens per minute), however, I only reached level 4 because I had an account for 14 days and spent over \$250 USD on the API.

Finally, this combination of cost and token constraints is why I did not work on prompt engineering, the S2023 Final, nor experimenting with adding the full solution and some training data. Those tests would have been too costly and should be saved for future work. I also had a third mock exam question but did not pursue that question due to these constraints.

Another issue was that there was some degree of randomness to solutions and running the prompt twice with the same data and model would give different results. For example, the binary five training values per class score on Q1 of the mock exam differed wildly on some runs (see Figure 33). This tendency, combined with the output error in Figure 32 causing me to have to re-run grading multiple times, means this is a significant issue with these models.

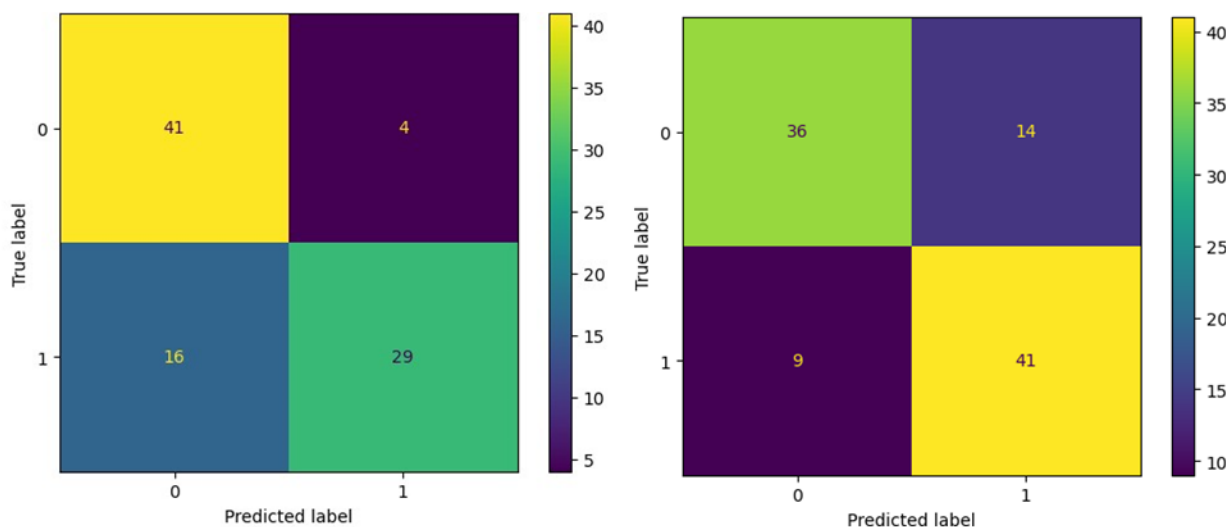


Figure 31: The results of the same prompt and data being run two separate times.

For this method to work better in the future, a partnership with OpenAI and the University of Manitoba will likely need to occur where there are neither usage limits, nor a per-token cost for API usage for the purpose of grading exams. Doing so will be the most cost-effective way to grade exams on a large scale.

5.3. Future work

This project is just the beginning of a process to automate exam grading. As AI improves and text-to-LaTeX conversions become more accurate, the pre-processing end of things will be streamlined. The pre-processing was by far the most time-consuming task, taking many hours for each question. Additionally, as LLMs become more advanced, sophisticated, and (maybe) cheaper to use, the grading accuracy should increase as well. Hopefully, this grading method will eventually spread to other departments in the Faculty of Science at the University of Manitoba, and perhaps beyond.

Further focus on prompt engineering and methods to reduce the overall token usage would be a good start. Part of prompt engineering could be to provide more details on how to score answers, as class 1 in the 3-class problem still has a lot of room for improvement. Such details would have to give clear cutoffs between what is considered partially correct versus not partially correct. Additionally, including a detailed answer key alongside the training data in the prompt could lead to improvements. For the answer key, a more detailed one than I used that breaks down the part marks could be helpful to use on its own and with training data to improve results. Overall, if effective prompt engineering is utilized, it could reduce the need for substantial amounts of training data and allow for shorter prompts and cheaper grading.

Another potential fix for this issue with part marks would be to have each mark tested, where every mark on an exam is determined via a binary classification problem of correct vs incorrect. The grader could tell the machine what to look for with each mark and the outputs would be added together at the end to give the final score for a student. For example, if a question is worth four marks, then the model grades it four times, each time focusing on a particular part of the question that is worth one mark. This may be more costly, but that increase

in prompt usage could be offset by each problem being smaller and maybe requiring less training data.

Another area to investigate would be how the models handle noise. Unnecessary noise such as Crowdmark formatting and un-converted text images, were removed from the LaTeX, however, how noise impacts the performance of the models was not investigated. If the models still perform well with that noise, then less pre-processing will be required, and the process will be accelerated.

It would also be important to see if the proportion of training to testing examples is significant, or if the number of training examples is more influential. If a 500-person exam requires a 50-50 train-test split, it would not be very efficient or cost-effective, whereas if it only requires a certain number of training examples per grade, then that could be beneficial.

The other area that should be investigated is increasing the number of classes to better reflect and hopefully match the number of grades. Only three grade classes were looked at in this project, so there is a lot of room for improvement in this area. To do so, more prompt engineering must be tested to help the model distinguish grades. A more detailed answer key may also help with this issue.

This project provided a good start to solving this problem, as it clearly identified LLMs as the premier method. As more development and advancement in the underlying models and programs used occur, combined with some tweaks to how exams are formatted, results should improve. This approach of using LLMs has the potential to speed up and perhaps even cheapen the grading process, preventing the problems caused by human graders, and doing so without sacrificing any accuracy in grading.

Acknowledgements

A special thanks to Dr Christopher Henry, and Dr Tristan Miller for providing me with much needed guidance and answering all my questions when this project hit a rut. Without their help I would not have been able to find a better approach to this problem after the image classification techniques failed.

Finally, a huge thanks to my domain expert, Dr. Shaun Lui. This whole project was his idea and he provided me with much needed support, guidance, and suggestions. He also funded the project, which allowed me to perform extensive tests with the ChatGPT API, as well as Mathpix and Adobe Acrobat. His help and funding are the only reason this project returned any promising results.

References

- Brownlee, J. (2020a, August 20). *A gentle introduction to the rectified linear unit (ReLU)*. Machine Learning Mastery. <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>
- Brownlee, J. (2020b, September 12). *Understand the impact of learning rate on neural network performance*. Machine Learning Mastery. <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>
- Brownlee, J. (2021a, January 13). *Gentle introduction to the adam optimization algorithm for deep learning*. Machine Learning Mastery. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>
- Brownlee, J. (2021b, April 27). *One-vs-rest and one-vs-one for multi-class classification*. Machine Learning Mastery. <https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/>
- Convolutional Neural Networks (CNN): Step 3 - Flattening*. SuperDataScience. (2018, April 17). <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening>
- Dumane, G. (2020, March 2). *Introduction to convolutional neural network (CNN) using tensorflow*. Towards Data Science. <https://towardsdatascience.com/introduction-to-convolutional-neural-network-cnn-de73f69c5b83>
- Gotke, S. A. (2020, November). *Most popular distance metrics used in KNN and when to use them*. KDnuggets. <https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html>
- Grosse, R. (n.d.). *Lecture 11: Convolutional Networks*. Lecture.
- Guinness, H. (2023, September 6). *How does chatGPT work?*. Zapier. <https://zapier.com/blog/how-does-chatGPT-work/>
- Lin, C., & Wang, S.-D. (2002). Fuzzy support vector machines. *IEEE Transactions on Neural Networks*, 13(2), 464–471. <https://doi.org/10.1109/72.991432>
- Mathpix Pricing*. Mathpix. (n.d.). <https://mathpix.com/pricing>
- Nanos, G. (2024, March 18). *Neural networks: Pooling layers*. Baeldung. <https://www.baeldung.com/cs/neural-networks-pooling-layers>
- Nyuytiymbiy, K. (2022, March 28). *Parameters and hyperparameters in machine learning and Deep Learning*. Medium. <https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac>

Pricing. OpenAI. (n.d.-a). <https://openai.com/pricing>

Priya, B. C. (2023, June 30). *Softmax activation function: Everything you need to know*. Pinecone. <https://www.pinecone.io/learn/softmax-activation/>

Pykes, K. (2024, January 11). *Cross-entropy loss function in machine learning: Enhancing model accuracy*. DataCamp. <https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning>

Ramponi, M. (2022, December 23). *How CHATGPT actually works*. AssemblyAI. <https://www.assemblyai.com/blog/how-chatGPT-actually-works/>

Rate limits. OpenAI. (n.d.-b). <https://platform.openai.com/docs/guides/rate-limits/usage-tiers?context=tier-free>

Saeidpour, A. (2016, March). *Figure 3: SVM classification for non-linearly separable data points*. ResearchGate. https://www.researchgate.net/figure/SVM-classification-for-non-linearly-separable-data-points_fig4_303469788

Saini, A. (2024, January 23). *Guide on Support Vector Machine (SVM) algorithm*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/>

Shah, S. M. A. (n.d.). *What is a neural network flatten layer?*. Educative. <https://www.educative.io/answers/what-is-a-neural-network-flatten-layer>

Sharma, S. (2021, October 26). *SVM: What makes it superior to the maximal-margin and support vector classifiers?*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2021/05/support-vector-machines/>

Sklearn.impute.KNNImputer. scikit. (n.d.). <https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html>

Tokenizer. OpenAI. (n.d.-c). <https://platform.openai.com/tokenizer>

Weisstein, E. W. (n.d.). *Hyperplane*. Wolfram MathWorld. <https://mathworld.wolfram.com/Hyperplane.html>

What is input space?. Pachyderm. (2022, April 22). <https://www.pachyderm.com/glossary/what-is-input-space/>

What is Supervised Learning?. Google. (n.d.). <https://cloud.google.com/discover/what-is-supervised-learning>

When Data is NOT Linearly Separable. 10.3 - When Data is NOT Linearly Separable | STAT 897D. (n.d.). <https://online.stat.psu.edu/stat857/node/242/>

Zapp, T. (2024, January). *January 11. COMP 4630: Machine Learning*. Winnipeg; Manitoba.