RESEARCH INSTITUTE FOR SYMBOLIC COMPUTATION

JOHANNES KEPLER UNIVERSITY LINZ

Is ChatGPT Smarter Than Master's Applicants?

Bruno Buchberger

January 2023

RISC Report Series No. 23-04

ISSN: 2791-4267 (online)

Available at https://doi.org/10.35011/risc.23-04

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Is ChatGPT Smarter Than Master's Applicants?

Bruno Buchberger

RISC (Research Institute for Symbolic Computation)
Johannes Kepler University
Linz – Hagenberg, Austria

January 2013

Summary:

My Experiment:

In the course of the selection procedure for a special informatics fellowship program sponsored by Upper Austrian companies, I ask the applicants a couple of simple technical questions about programming, etc., in a Zoom meeting.

I put the same questions to the dialogue system ChatGPT, [ChatGPT].

The Result

The result surprised me:

Nearly all answers of ChatGPT were completely correct and nicely explained. Also, in the dialogues to clarify some critical points in the answers, the explanations by ChatGPT were amazingly clear and goal-oriented.

In comparison: I tried out the same questions in the personal Zoom interviews with approximately 30 applicants from five countries. Only the top three candidates (who had a GPA of 1.0, i.e., highest possible GPA, in their bachelor's study), performed approximately equally well in the interview. All the others performed (far) worse than ChatGPT.

And, of course, all answers from ChatGPT came within 1 to 10 seconds, whereas most of the answers of the human applicnats needed quite some lengthy and tough dialogues.

Program Synthesis by ChatGPT from Natural Language Problem Specifications

I am particularly impressed by the ability of ChatGPT to extract meaningful and well-structured programs from problem specifications in natural language.

Additional Questions Requesting to Invent Proofs for Natural Language Statements

In this experiment, I also added some questions that ask for proofs for simple statements in natural language. This is something I do not attempt to ask in the master's student interviews. The performance of ChatGPT was quite impressive as far as formalization and propositional logic are concerned. In examples where predicate logic reasoning is necessary, the ChatGPT answers are not yet perfect.

I am pleased to see that ChatGPT tries to present the proofs in "natural style" This is something that I had as one of my main goals when I initiated the Theorema project, [Theorema 1997, 2016] in 1995. I think we achieved this already in the early stage of Theorema, and I think we achieved this slightly better and more systematically than ChatGPT does.

I also tried to develop a natural language input facility for Theorema in 2017, i.e. a tool to formalize natural language statements in predicate logic. However, for a couple of reasons, unfortunately, I was not able to continue this research. Now I see that ChatGPT more or less achieved this goal.

A Research Proposal

Thus, I think that

- the "natural style" proving methods that we developed within Theorema (for the automated generation of programs from specifications, the automated verification of programs in the frame of knowledge, and the automated proof of theorems in theories, see in particular the "Lazy Thinking Method" for algorithm synthesis from specifications, [Buchberger 2002])
- combined with the natural language formalization techniques of ChatGPT could result in a major leap forward.

I propose this as the topic of a research project and invite colleagues and students to get in contact with me and join me in this effort: Buchberger.bruno@gmail.com.

This project falls into my general view:

"AI = Symbolic Computation + Machine Learning", see my paper [Buchberger 2022].

Namely:

- algorithms on formal systems (like knowledge theories, problem specifications, theorems, proofs) is branch of Symbolic Computation

- Natural Language Processing: A branch of Machine Learning.

The Protocol of My Interview With ChatGPT

BB (Bruno Buchberger):

Write a program for the following task: Given a set of appointments in a calendar and a new appointment, check whether the new appointment collides with one of the appointments in the calendar. An appointment consists of a start time, an end time, and text describing the content of the appointment. Code in your favorite programming language.

ChatGPT:

Sure, here is a solution in Python:

```
class Appointment:
    def __init__(self, start, end, text):
        self.start = start
        self.end = end
        self.text = text

def check_collision(appointments, new_appointment):
    for appointment in appointments:
        if appointment.start < new_appointment.end and appointment.end >
            new_appointment.start:
            return True
```

```
    return False

# Example usage
appointments = [Appointment(10, 11, "Doctor's appointment"), Appointment(12, 13,
"Lunch with John")]
new_appointment = Appointment(11, 12, "Meet with Jane")

if check_collision(appointments, new_appointment):
    print("Collision detected")
else:
    print("No collision detected")
```

This solution creates a `Appointment` class to represent an appointment, with `start`, `end`, and `text` as its attributes. It then defines the `check_collision` function which takes in a list of `appointments` and a `new_appointment` and checks whether there is a collision by iterating through the list of appointments and checking if the start or end time of any of the appointments fall within the start and end time of the new appointment. If a collision is detected, the function returns `True`, otherwise it returns `False`.

I hope this helps! Let me know if you have any questions.

BB:

Have you got any idea how the computation time of the program check_collision in your previous answer could be improved significantly?

ChatGPT:

Yes, one way to improve the computation time of the `check_collision` function would be to use a more efficient data structure to store the appointments.

Currently, the function is using a simple list to store the appointments and it has to iterate through the entire list to check for collisions. If the list is long, this can be time-consuming.

One solution would be to use a data structure that allows for faster lookup, such as a hash table or a binary search tree. This would allow the function to quickly find and compare appointments, rather than having to iterate through the entire list.

Another solution would be to use a calendar library or framework that is optimized for storing and querying appointments. These libraries often use advanced data structures and algorithms to manage appointments efficiently.

Finally, if the number of appointments is very large, it may be necessary to use a distributed system or database to store and query the appointments, rather than using a data structure in memory. This would allow the appointments to be distributed across multiple machines, improving the speed and scalability of the system.

I hope these suggestions help! Let me know if you have any other questions.

BB:

Let's assume the list of appointments in your above program check_collision is sorted how would you then formulate the program that checks for collision with a new appointment?

ChatGPT:

Sure, here is a revised version of the `check_collision` function that takes advantage of a sorted list of appointments to improve the computation time:

```
def check_collision(appointments, new_appointment):
   # Binary search for the first appointment that ends after the new appointment starts
   left = 0
   right = len(appointments) - 1
   while left <= right:
      mid = (left + right) // 2
      if appointments[mid].end <= new_appointment.start:
         left = mid + 1
      else:
         right = mid - 1

   # Check for collisions with the appointment to the right (if it exists)
   if left < len(appointments) and appointments[left].start < new_appointment.end:
      return True

   # Check for collisions with the appointment to the left (if it exists)
```

```
if right >= 0 and appointments[right].end > new_appointment.start:
    return True

# No collision detected
return False
```

This revised version of the `check_collision` function uses a binary search to quickly find the first appointment that ends after the start time of the new appointment. It then checks for collisions with the appointment to the right (if it exists) and the appointment to the left (if it exists). If no collision is detected, the function returns `False`.

I hope this helps! Let me know if you have any questions.

BB:

Can you give a formula for the runtime of the algorithm in your last answer?

ChatGPT:

Sure, the runtime of the revised version of the `check_collision` function that I provided in my previous answer is O(log n), where n is the number of appointments. This is because the function uses a binary search to find the first appointment that ends after the start time of the new appointment, and the runtime of a binary search is O(log n).

To be more precise, the function performs at most two comparisons for each level of the binary search tree, so the maximum number of comparisons is 2 * log(n). This means that the function will be able to find the correct appointment in the list of appointments in O(log n) time.

I hope this helps! Let me know if you have any other questions.

BB:

Let's assume you have a database of one million customers and you want to sort them by age. Which sorting algorithm do you choose?

ChatGPT:

There are several sorting algorithms that could be used to sort a large database of customers by age, but some algorithms may be more suitable than others depending on the specific characteristics of the data and the desired performance.

One algorithm that could be suitable for sorting a large database of customers by age is merge sort. Merge sort is a divide and conquer algorithm that has a runtime of O(n * log(n)). This means that it can sort a large number of items relatively quickly, making it a good choice for sorting large databases.

Another algorithm that could be suitable for sorting a large database of customers by age is quick sort. Quick sort is another divide and conquer algorithm that has a runtime of O(n * log(n)) on average, but it can be slower than merge sort in the worst case.

There are other algorithms that could also be suitable for sorting a large database of customers by age, such as heap sort and radix sort. It is generally a good idea to test and compare the performance of different algorithms in order to choose the best one for a specific use case.

I hope this helps! Let me know if you have any other questions.

BB:

Let me be more specific: Let's assume that the age of a person is given by years. Knowing this, which sorting algorithm would you use in the previous task?

ChatGPT:

If the age of each person is given as an integer number of years, and the database is large (e.g. one million customers), then a comparison-based sorting algorithm such as merge sort or quick sort would be a good choice. These algorithms have a runtime of O(n * log(n)) on average and are efficient at sorting large datasets.

One advantage of merge sort is that it is stable, meaning that it preserves the relative order of items with equal keys (in this case, items with the same age). This can be useful if you want to maintain the original order of the items as much as possible.

Another advantage of merge sort is that it has a consistent runtime, meaning that it performs at the same speed regardless of the input data. This can be useful if you want to ensure that the sorting process always completes within a certain time frame.

On the other hand, quick sort has the advantage of being faster than merge sort in the average case and requiring less auxiliary space, but it can be slower than merge sort in the worst case. This may or may not be a concern depending on the specific requirements of your application.

I hope this helps! Let me know if you have any other questions.

BB:

How would you apply radix sort in this task and what would be the runtime?

ChatGPT:

To apply radix sort to the task of sorting a large database of customers by age, you would need to first convert the ages from years to some other representation that can be sorted using radix sort. One option would be to represent the ages as integers in a fixed number of digits (e.g. two digits for ages 0-99, three digits for ages 100-999, etc.).

Once the ages have been converted to a fixed-width integer representation, you can use radix sort to sort the database. Radix sort is a non-comparison based sorting algorithm that works by sorting the items based on each digit of the integer representation, starting with the least significant digit and working towards the most significant digit.

The runtime of radix sort depends on the number of digits in the integer representation and the number of items being sorted. In the worst case, the runtime is O(w * n), where w is the number of digits in the integer representation and n is the number of items being sorted. In the best case, the runtime is O(w * n), where w is the number of digits in the integer representation and n is the number of items being sorted.

One advantage of radix sort is that it can be very fast for sorting large datasets of integers with a small number of digits. It is also a stable sorting algorithm, meaning that it preserves the relative order of items with equal keys (in this case, items with the same age).

On the other hand, radix sort may not be the most efficient algorithm for sorting a large database of customers by age if the ages are represented as a large number of digits (e.g. four or more digits). In this case, a comparison-based sorting algorithm such as merge sort or quick sort may be a better choice.

I hope this helps! Let me know if you have any other questions.

BB:

Assume we have 10 red cards with numbers 1, ..., 10 ; 5 blue cards with numbers 1, ..., 5; 7 green cards with numbers 1, ..., 7. What is the probability of picking a blue card with the number 2 or 3.

ChatGPT:

To calculate the probability of picking a blue card with the number 2 or 3, we need to consider the total number of possible outcomes (i.e. the total number of cards) and the number of successful outcomes (i.e. the number of blue cards with the number 2 or 3).

There are a total of 10 + 5 + 7 = 22 cards. Of these, 5 are blue cards, and 2 of them (the cards with the numbers 2 and 3) are successful outcomes.

Therefore, the probability of picking a blue card with the number 2 or 3 is 2/22 = 1/11.

I hope this helps! Let me know if you have any other questions.

BB:

Do you know Bayes' Theorem? How could you apply Bayes's Theorem in the previous task?

ChatGPT:

Yes, Bayes' Theorem is a mathematical formula for calculating the probability of an event based on prior knowledge of conditions that might be related to the event. It is often used in machine learning and statistics to make predictions or infer the likelihood of an event based on data.

Bayes' Theorem can be written as follows:

P(A|B) = P(B|A) * P(A) / P(B)

where:

- P(A|B) is the probability of event A occurring given that event B has occurred
- P(B|A) is the probability of event B occurring given that event A has occurred
- P(A) is the probability of event A occurring
- P(B) is the probability of event B occurring

To apply Bayes' Theorem to the task of calculating the probability of picking a blue card with the number 2 or 3, we need to define the events A and B and calculate the probabilities of each event.

For example, we could define the event A as "picking a blue card with the number 2 or 3" and the event B as "picking a blue card". We can then use the information provided in the question to calculate the probabilities of each event as follows:

- P(A) = 2/22 = 1/11 (since there are 2 successful outcomes out of a total of 22 cards)
- P(B) = 5/22 = 5/22 (since there are 5 blue cards out of a total of 22 cards)
- P(B|A) = 1 (since all successful outcomes are blue cards)

Substituting these values into Bayes' Theorem, we can calculate the probability of picking a blue card with the number 2 or 3 given that we have picked a blue card:

P(A|B) = P(B|A) * P(A) / P(B) = (1) * (1/11) / (5/22) = 2/5

Therefore, the probability of picking a blue card with the number 2 or 3 given that we have picked a blue card is 2/5.

I hope this helps! Let me know if you have any other questions.

BB:

Assume we have an O(n log n) algorithm and an O(n2) algorithm for the same problem, and we run it on the same computer. For a fixed n, say n = 1000, can the O(n2) algorithm run faster than the O (n log n) algorithm?

ChatGPT:

In general, the O(n^2) algorithm will not run faster than the O(n * log(n)) algorithm for a fixed value of n (e.g. n = 1000). This is because the O(n * log(n)) algorithm has a lower order of growth than the O(n^2) algorithm, meaning that it will generally be more efficient for large values of n.

To be more specific, the O(n * log(n)) algorithm will have a runtime of approximately 2000 * log(1000) = 6000 for n = 1000, while the O(n^2) algorithm will have a runtime of approximately 1000^2 = 1000000. This means that the O(n * log(n)) algorithm will be significantly faster than the O(n^2) algorithm for n = 1000.

Of course, there are many factors that can affect the actual runtime of an algorithm, such as the specific implementation, the hardware, and the input data. It is always a good idea to test and measure the performance of different algorithms in order to determine the best one for a specific use case.

I hope this helps! Let me know if you have any other questions.

BB:

I think your answer is not completely correct. Note that my question was whether, for the specific value n=1000, the O (n^2) algorithm *could* run faster than the O(n log n) algorithm. In other words, the question is whether the two algorithms may have certain properties that entail that, for n=1000, the O(n^2) algorithm may run faster than the O (n log n) algorithm. Can you imagine such properties?

ChatGPT:

I apologize for the misunderstanding. You are correct that it is possible for the O(n^2) algorithm to run faster than the O(n * log(n)) algorithm for a specific value of n (e.g. n = 1000), even though the O(n * log(n)) algorithm has a lower order of growth and is generally more efficient for large values of n.

There are several factors that could contribute to the O(n^2) algorithm running faster than the O(n * log(n)) algorithm for a specific value of n:

- Constant factors: The O(n^2) algorithm may have a smaller constant factor than the O(n * log(n)) algorithm, meaning that it takes fewer steps to complete each operation. This can make a significant difference for small values of n.
- Hardware: The hardware on which the algorithms are running may be better optimized for one algorithm over the other. For example, if the O(n^2) algorithm makes heavy use of the CPU cache, it may be faster on a machine with a large cache.
- Input data: The input data may be structured in such a way that one algorithm is faster than the other. For example, if the input data is already sorted, an O(n) sorting algorithm may be faster than an O(n * log(n)) sorting algorithm.

It is always a good idea to test and measure the performance of different algorithms in order to determine the best one for a specific use case.

I hope this helps! Let me know if you have any other questions.

BB:

You need not apologize. It is a common misunderstanding! Anyway, could you figure out an example in which an O (n^2) algorithm would run faster than the O(n log n) algorithm for the specific value n = 1000?

ChatGPT:

Sure, here is an example of a situation in which an O(n^2) algorithm could run faster than an O(n * log(n)) algorithm for the specific value n = 1000:

Suppose we have two algorithms for sorting a list of 1000 integers:

- An O(n^2) sorting algorithm that uses a simple bubble sort implementation.
- An O(n * log(n)) sorting algorithm that uses a merge sort implementation.

In this example, it is possible that the O(n^2) bubble sort algorithm could run faster than the O(n * log(n)) merge sort algorithm for n = 1000. This could be due to a combination of factors, such as the constant factors of the two algorithms and the specific hardware on which they are running.

For example, suppose that the bubble sort algorithm has a constant factor of 10 (i.e. it takes 10 steps to complete each operation) and the merge sort algorithm has a constant factor of 100 (i.e. it takes 100 steps to complete each operation). In this case, the bubble sort algorithm could be faster than the merge sort algorithm for n = 1000, even though the merge sort algorithm has a lower order of growth.

This is just one possible example, and the actual performance of the algorithms would depend on a variety of factors. It is always a good idea to test and measure the performance of different algorithms in order to determine the best one for a specific use case.

I hope this helps! Let me know if you have any other questions.

BB:

Can you please give me a definition of what it means that a function f on the natural number has order O(g(n))?

ChatGPT:

In computer science and mathematics, the notation O(g(n)) is used to describe the order of growth of a function f(n) as n (the input size) increases. Informally, this means that the function f(n) grows at most as fast as g(n) as n becomes very large.

More formally, we say that a function f(n) has order O(g(n)) if there exist positive constants c and n0 such that f(n) is at most c * g(n) for all n > n0. In other words, the function f(n) is "bounded" by the function g(n) as n becomes very large.

For example, suppose we have a function f(n) that represents the number of steps required to perform some operation on a list of n items. If f(n) has order O(n), this means that the number of steps required to perform the operation grows linearly with the size of the input (n). If f(n) has order O(n^2), this means that the number of steps required to perform the operation grows as the square of the size of the input (n^2).

The notation O(g(n)) is often used to describe the worst-case runtime of an algorithm, meaning that it represents an upper bound on the number of steps required to perform the algorithm for the worst possible input. However, it can also be used to describe the average-case runtime of an algorithm, meaning that it represents the expected number of steps required to perform the algorithm for a "typical" input.

I hope this helps! Let me know if you have any other questions.

BB:

Could you please formulate the negation of the following sentence: "for all indices i < the length of the list, the element at position i in the list is <= the element at position i+1 in the list".

ChatGPT:

Sure! Here is the negation of the following sentence:

"For all indices i < the length of the list, the element at position i in the list is <= the element at position i+1 in the list."

"There exists an index i < the length of the list such that the element at position i in the list is > the element at position i+1 in the list."

I hope this helps! Let me know if you have any other questions.

BB:

Bravo!

ChatGPT:

Thank you! I'm glad that I could help. Let me know if you have any other questions.

BB:

Consider the following problem: The salespersons of a company that produces and sells bikes give discounts to potential customers to increase sales. The discounts are decided according to “feeling”. How would you set up a machine learning approach to maximize the company's profit by setting discounts appropriately?

ChatGPT:

Here is a general outline of how you could set up a machine learning approach to maximize the company's profit by setting discounts appropriately:

1. Collect data: The first step would be to collect data on past sales, including the size of the discount offered, the resulting sale price, and the profit made from each sale. You may also want to collect data on other relevant factors, such as the type of bike being sold, the customer's location, and the time of year.
2. Preprocess the data: Next, you will need to preprocess the data by cleaning it and preparing it for analysis. This may involve removing any missing or invalid data, handling outliers, and normalizing or standardizing the data.
3. Split the data: Split the preprocessed data into a training set and a test set. You will use the training set to train your machine learning model, and the test set to evaluate its performance.
4. Train a model: Using the training set, train a machine learning model that can predict the profit made from a sale based on the size of the discount offered and any other relevant factors. You may want to try out several different models and choose the one that performs best.
5. Fine-tune the model: Once you have chosen a model, fine-tune it by adjusting its hyperparameters and/or adding regularization to prevent overfitting.
6. Evaluate the model: Use the test set to evaluate the performance of the fine-tuned model. If the model's predictions are not accurate enough, you may need to go back to step 4 and try a different model or adjust the preprocessing of the data.
7. Implement the model: Once you are satisfied with the performance of the model, you can implement it in the company's sales process and use it to guide the decision-making of the salespersons.

This is just a general outline, and the specific steps you take will depend on the details of the problem and the available data. I hope this helps! Let me know if you have any other questions.

BB:

Could you please give a couple of other relevant factors that may play a role in this project!

ChatGPT:

Here are a few other relevant factors that may play a role in a machine learning project to maximize the company's profit by setting discounts appropriately:

- The customer's income level: Customers with higher income levels may be more willing to pay higher prices for bikes, so offering them smaller discounts may still result in a profitable sale.
- The customer's perceived value of the bike: Customers who value the bike more highly may be willing to pay a higher price, even with a larger discount.
- The customer's perceived quality of the bike: Customers who perceive the bike as being of high quality may be willing to pay a higher price, even with a larger discount.
- The customer's perceived competition: Customers who perceive that they have few other options for purchasing a similar bike may be willing to pay a higher price, even with a larger discount.
- The customer's location: Customers in certain geographic regions may be more or less willing to pay higher prices, depending on factors such as local economic conditions and the availability of other options.
- The time of year: The demand for bikes may vary seasonally, with higher demand resulting in the ability to offer smaller discounts and still make a profit.

These are just a few examples of the many factors that may play a role in setting discounts to maximize profit. The specific factors that are relevant will depend on the details of the company's business and the available data.

BB:

These factors sound very reasonable. Thank you! Any ideas how these factors could be measured when we try to build up a training set?

ChatGPT:

Here are a few ideas for how the relevant factors I mentioned earlier could be measured when building a training set for a machine learning model to maximize the company's profit by setting discounts appropriately:

- Customer income level: This factor could be measured by asking the customer directly for their income level, or by inferring it from other available data such as their job title or the area in which they live.

- Customer's perceived value of the bike: This factor could be measured by asking the customer directly for their perceived value of the bike, or by inferring it from other available data such as their willingness to pay a higher price or the features of the bike that they express interest in.
- Customer's perceived quality of the bike: This factor could be measured by asking the customer directly for their perceived quality of the bike, or by inferring it from other available data such as the brand or model of the bike and the customer's willingness to pay a higher price.
- Customer's perceived competition: This factor could be measured by asking the customer directly about their perceived competition, or by inferring it from other available data such as the availability of similar bikes in the area and the customer's willingness to pay a higher price.
- Customer's location: This factor could be measured by collecting the customer's postal code or other geographic data.
- Time of year: This factor could be measured by collecting the date of the sale.

These are just a few examples of how the relevant factors could be measured when building a training set. The specific approach will depend on the details of the problem and the available data.

BB:

How big should the training set be in this particular project so that there is a chance that the sales persons could rely with good reason on the support of the trained model?

ChatGPT:

It is difficult to give a general answer to the question of how big the training set should be in this particular project, as it will depend on the complexity of the problem and the amount of variation in the data. In general, a larger training set can lead to a more accurate model, but there is a point of diminishing returns where adding more data to the training set does not significantly improve the model's performance.

One way to determine the appropriate size of the training set is to split the available data into a training set and a test set, and then evaluate the performance of the model on the test set as the size of the training set is increased. If the model's performance on the test set improves significantly as the training set size is increased, it may be worth adding more data to the training set. If the model's performance on the test set does not improve significantly, it may indicate that the training set is large enough.

Alternatively, you could use a cross-validation approach, in which the data is split into multiple folds and the model is trained and evaluated multiple times using different combinations of folds as the training and test sets. This can give a more robust estimate of the model's performance and help ensure that the model is not overfitting to the training data.

Ultimately, the appropriate size of the training set will depend on the specific details of the problem and the available data, and may require some experimentation to determine.

BB:

What do you mean by overfitting?

ChatGPT:

Overfitting occurs when a machine learning model is trained too closely to the training data, and is not able to generalize well to new, unseen data. This can lead to poor performance on the test set or in real-world applications.

There are a few ways in which a model can overfit to the training data. For example, if the model is too complex, it may learn patterns in the training data that do not generalize to new data. This is known as "high variance" overfitting. On the other hand, if the model is too simple, it may not be able to capture the underlying patterns in the training data and will perform poorly on both the training data and new data. This is known as "high bias" overfitting.

To prevent overfitting, it is important to use a sufficient amount of training data, use appropriate model complexity, and use techniques such as regularization to prevent the model from learning excessively complex patterns in the data. It is also a good idea to evaluate the model's performance on a separate test set to ensure that it is able to generalize to new data.

BB:

What do you mean by regularization?

ChatGPT:

Regularization is a technique used to prevent overfitting in machine learning models. It does this by adding a penalty term to the objective function that the model is trying to optimize. The penalty term encourages the model to simplify its decision boundary, making it less complex and more likely to generalize to new, unseen data.

There are a few different types of regularization that are commonly used, including:

- L1 regularization: This adds a penalty term to the objective function that is proportional to the absolute value of the model's weights. It encourages the model to use a sparse set of weights, with many weights set to 0.
- L2 regularization: This adds a penalty term to the objective function that is proportional to the square of the model's weights. It encourages the model to use a small set of weights, but does not encourage sparsity as much as L1 regularization.
- Elastic net regularization: This is a combination of L1 and L2 regularization, and allows the user to control the balance between the two types of regularization.

Regularization is often used in conjunction with other techniques, such as cross-validation, to prevent overfitting and improve the generalization performance of the model.

BB:

Assume we have the following tables in our database system:

performers:

ID	Performer	TypeOfMusic	Description
4	Abba	Pop	See Wiki: "Abba"
5	Rollins	Jazz	See Wiki: "Rollins"
6	Deep Purple	HeavyMetal	See Wiki: "Deep Purple"

events:

ID	PerformanceDate	Performer	Venue
1	2022-04-30	6	3
2	2022-05-05	4	2
3	2022-06-15	5	2

venues:

ID	Venue	Address
2	Posthof	Linz, Hafenstraße 78
3	AEC	Linz, Hauptstraße 1

Write a query to find what dates the Posthof is booked.

Write a query to find which performers are playing at the Posthof and when.

ChatGPT:

To find what dates the Posthof is booked:

```
SELECT PerformanceDate

FROM events

JOIN venues ON venues.ID = events.Venue

WHERE venues.Venue = 'Posthof'
```

To find which performers are playing at the Posthof and when:

```
SELECT Performer, PerformanceDate

FROM events

JOIN venues ON venues.ID = events.Venue

JOIN performers ON performers.ID = events.Performer

WHERE venues.Venue = 'Posthof'
```

I hope these queries are helpful! Let me know if you have any other questions.

BB:

Express the following English sentence as a rule (clause) in predicate logic: If there exists a z such that z is a child of x and y is a child of z then y is a grandchild of x.

ChatGPT:

Here is a clause in predicate logic that expresses the given English sentence:

(∃z)(child(z, x) ∧ child(y, z)) → grandchild(y, x)

BB:

Prove that the following two statements are equivalent:

- for all x, y if there exists a z such that P(x, z) and Q(z, y) then R(x, y)

- for all x, y, z if P(x, z) and Q(z, y) then R(x, y)

ChatGPT:

To prove that these two statements are equivalent, we need to show that each statement implies the other.

First, let's consider the first statement: "for all x, y, if there exists a z such that P(x, z) and Q(z, y), then R(x, y)."

We can prove that this statement implies the second statement as follows:

- Assume that the first statement is true.
- Consider the second statement: "for all x, y, z, if P(x, z) and Q(z, y), then R(x, y)."
- We want to show that this statement is true.
- Let x, y, and z be arbitrary.
- Assume that P(x, z) and Q(z, y) are true.
- By the first statement, we know that if there exists a w such that P(x, w) and Q(w, y), then R(x, y).
- Setting w = z, we see that there exists a w such that P(x, w) and Q(w, y).
- Therefore, by the first statement, R(x, y) is true.
- Since x, y, and z were arbitrary, the second statement is true for all x, y, and z.

Now, let's consider the second statement: "for all x, y, z, if P(x, z) and Q(z, y), then R(x, y)."

We can prove that this statement implies the first statement as follows:

- Assume that the second statement is true.
- Consider the first statement: "for all x, y, if there exists a z such that P(x, z) and Q(z, y), then R(x, y)."
- We want to show that this statement is true.
- Let x and y be arbitrary.
- Assume that there exists a z such that P(x, z) and Q(z, y).
- By the second statement, we know that if P(x, z) and Q(z, y), then R(x, y).
- Therefore, by the second statement, R(x, y) is true.
- Since x and y were arbitrary, the first statement is true for all x and y.

Since we have shown that each statement implies the other, we can conclude that the two statements are equivalent.

Additional Questions: Proof Problems

BB:

What is the definition of the notion: function f has O(h(n))?

ChatGPT:

In computer science, the "big O" notation is used to describe the asymptotic behavior of a function. Informally, if a function f has O(h(n)), it means that the growth rate of f is no worse than the growth rate of h(n). In other words, there exists some positive constants c and n0 such that for all n > n0, the following inequality holds:

f(n) ≤ c * h(n)

Here, h(n) is a function that describes the upper bound on the growth rate of f. The function h is called an "upper bounding function" for f.

...

BB:

Please prove the following lemma: If the functions g and f have order O(h(n)) then the function f+g also has order O(h(n)).

ChatGPT:

To prove that the function f + g has order O(h(n)) if the functions f and g have order O(h(n)), we need to show that there exist positive constants c1, c2, and n0 such that f(n) is at most c1 * h(n) for all n > n0 and g(n) is at most c2 * h(n) for all n > n0, and that f(n) + g(n) is at most (c1 + c2) * h(n) for all n > n0.

To show this, we can start by assuming that there exist positive constants c1, c2, and n0 such that f(n) is at most c1 * h(n) for all n > n0 and g(n) is at most c2 * h(n) for all n > n0. We can then write the following inequalities:

f(n) <= c1 * h(n) for all n > n0 g(n) <= c2 * h(n) for all n > n0

Adding these inequalities, we get:

f(n) + g(n) <= c1 * h(n) + c2 * h(n) for all n > n0

Since h(n) is a positive function, we can rearrange the terms and divide both sides by h(n) to get:

(f(n) + g(n)) / h(n) <= (c1 + c2) for all n > n0

This means that (f(n) + g(n)) / h(n) is at most c1 + c2 for all n

BB:

Please look at your proof again. The overall flow of proof is not bad. However, in the second line you say "we need to show that there exist positive constants c1, c2, and n0 such that f(n) is at most c1 * h(n) for all n > n0 and g(n) is at most c2 * h(n) for all n > n0". Two lines later, you say "we can start by assuming that there exist positive constants c1, c2, and n0 such that f(n) is at most c1 * h(n) for all n > n0 and g(n) is at most c2 * h(n) for all n > n0". Thus, you assume what you first say you need to show. Something is wrong here. Can you improve your proof and eliminate this logical weakness?

ChatGPT:

You are correct that there is a logical weakness in the proof as written.

Here is the revised proof:

Suppose that f and g have order O(h(n)). This means that there exist positive constants c1 and c2, and a value n0 such that for all n > n0, the following inequalities hold:

f(n) ≤ c1 * h(n) g(n) ≤ c2 * h(n)

Now consider the function f + g. For all n > n0, we have:

(f + g)(n) = f(n) + g(n) ≤ c1 * h(n) + c2 * h(n) = (c1 + c2) * h(n)

Therefore, f + g has order O(h(n)).

This proves the lemma.

I apologize for the mistake in the original proof. I hope this revised proof is clearer. Let me know if you have any other questions.

BB:

Yes, this is better. There is only a slight weakness: By the assumptions and the definition of O(n) you cannot assume that you can take the same n0 in the expansion of the definition of f is O(h(n)) and g is O(h(n))! However, let's stop this discussion here and move to another topic

Please formalize in predicate logic: a function f is dominated by the function g if there exist numbers n0 and c such that, for all n > n0, f applied to n is smaller or equal to c times g applied to n.

ChatGPT:

Here is the formalization of the statement in predicate logic:

∀n (n > n0 ⟶ f(n) ≤ c * g(n))

Explanation:

- The symbol ∀ is the universal quantifier and it means "for all".
- The predicate (n > n0) is true for all values of n that are greater than n0.
- The predicate (f(n) ≤ c * g(n)) is true if the function f applied to n is smaller than or equal to c times the function g applied to n.
- The symbol ⟶ is the logical implication symbol and it means "if ... then".

So, the statement can be read as: "For all values of n, if n is greater than n0, then the function f applied to n is smaller than or equal to c times the function g applied to n."

BB:

You only formalized the last part of the statement. Let's try it again: f is dominated by g iff there exist c and n0 such that, for all n>n0, f(n) <= c g(n).

ChatGPT:

Here is the formalization of the statement in predicate logic:

∃c ∃n0 (∀n (n > n0 ⟶ f(n) ≤ c * g(n)))

Explanation:

- The symbol ∃ is the existential quantifier and it means "there exists".
- The predicates (c) and (n0) are existentially quantified, which means that there exists a value for c and a value for n0 that make the entire statement true.
- The predicate (∀n (n > n0 ⟶ f(n) ≤ c * g(n))) is true for all values of n that are greater than n0 and for which the function f applied to n is smaller than or equal to c times the function g applied to n.
- The symbol ⟶ is the logical implication symbol and it means "if ... then".

So, the statement can be read as: "There exists a value for c and a value for n0 such that, for all values of n that are greater than n0, the function f applied to n is smaller than or equal to c times the function g applied to n."

BB:

Formalize in predicate logic: isDominated(f,g) iff there exist c and n0 such that, for all n>n0, f(n) <= c g(n).

ChatGPT:

Here is the formalization of the statement in predicate logic:

isDominated(f, g) ⇔ ∃c ∃n0 (∀n (n > n0 ⟶ f(n) ≤ c * g(n)))

Explanation:

- The predicate isDominated(f, g) represents the statement "f is dominated by g".
- The symbol ⇔ is the logical biimplication symbol and it means "if and only if".
- The symbol ∃ is the existential quantifier and it means "there exists".
- The predicates (c) and (n0) are existentially quantified, which means that there exists a value for c and a value for n0 that make the entire statement true.
- The predicate (∀n (n > n0 ⟶ f(n) ≤ c * g(n))) is true for all values of n that are greater than n0 and for which the function f applied to n is smaller than or equal to c times the function g applied to n.
- The symbol ⟶ is the logical implication symbol and it means "if ... then".

So, the statement can be read as: "f is dominated by g if and only if there exists a value for c and a value for n0 such that, for all values of n that are greater than n0, the function f applied to n is smaller than or equal to c times the function g applied to n."

BB:

Prove the following lemma: If isDominated(f,h) and isDominated(g,h) then isDominated (f+g,h).

ChatGPT:

Here is a proof of the lemma:

Assume that isDominated(f, h) and isDominated(g, h) are true.

By the definition of isDominated, this means that there exist values c1 and n1 such that, for all n > n1, f(n) ≤ c1 * h(n), and there exist values c2 and n2 such that, for all n > n2, g(n) ≤ c2 * h(n).

Let n0 = max(n1, n2). Then, for all n > n0, we have that n > n1 and n > n2.

Therefore, for all n > n0, we have:

f(n) ≤ c1 * h(n) g(n) ≤ c2 * h(n)

Adding these two inequalities, we get:

f(n) + g(n) ≤ (c1 + c2) * h(n)

Therefore, isDominated(f + g, h) is true, where c = c1 + c2 and n0 = max(n1, n2).

This proves that if isDominated(f, h) and isDominated(g, h) are true, then isDominated(f + g, h) must also be true.

BB:

Try to prove the following lemma: If isDominated(f,h) and isDominated(g,h) then isDominated (f*g,h).

ChatGPT:

Here is a proof of the lemma:

Assume that isDominated(f, h) and isDominated(g, h) are true.

By the definition of isDominated, this means that there exist values c1 and n1 such that, for all n > n1, f(n) ≤ c1 * h(n), and there exist values c2 and n2 such that, for all n > n2, g(n) ≤ c2 * h(n).

Let n0 = max(n1, n2). Then, for all n > n0, we have that n > n1 and n > n2.

Therefore, for all n > n0, we have:

f(n) ≤ c1 * h(n) g(n) ≤ c2 * h(n)

Multiplying these two inequalities, we get:

f(n) * g(n) ≤ (c1 * c2) * h(n) * h(n)

Since h(n) * h(n) = h(n)^2, we can simplify the inequality to:

f(n) * g(n) ≤ (c1 * c2) * h(n)^2

Therefore, isDominated(f * g, h) is true, where c = c1 * c2 and n0 = max(n1, n2).

This proves that if isDominated(f, h) and isDominated(g, h) are true, then isDominated(f * g, h) must also be true.

COMMENT: Here, one sees that ChatGPT is losing correctness of reasoning! Rather, it seems that it always wants to come up with a proof, even for false statements!

BB:

Try to prove the following lemma: If isDominated(f,h) and isDominated(g,h) then isDominated (f*g,h * h).

ChatGPT:

Here is a proof of the lemma:

Assume that isDominated(f, h) and isDominated(g, h) are true.

By the definition of isDominated, this means that there exist values c1 and n1 such that, for all n > n1, f(n) ≤ c1 * h(n), and there exist values c2 and n2 such that, for all n > n2, g(n) ≤ c2 * h(n).

Let n0 = max(n1, n2). Then, for all n > n0, we have that n > n1 and n > n2.

Therefore, for all n > n0, we have:

f(n) ≤ c1 * h(n) g(n) ≤ c2 * h(n)

Multiplying these two inequalities, we get:

f(n) * g(n) ≤ (c1 * c2) * h(n) * h(n)

Since h(n) * h(n) = h(n)^2, we can simplify the inequality to:

f(n) * g(n) ≤ (c1 * c2) * h(n)^2

Therefore, isDominated(f * g, h * h) is true, where c = c1 * c2 and n0 = max(n1, n2).

This proves that if isDominated(f, h) and isDominated(g, h) are true, then isDominated(f * g, h * h) must also be true.

BB:

Try to prove the following lemma: If isDominated(f,h) then isDominated (1/f,h).

ChatGPT:

Here is a proof of the lemma:

Assume that isDominated(f, h) is true.

By the definition of isDominated, this means that there exist a value c and n0 such that, for all n > n0, f(n) ≤ c * h(n).

Let 1/c = d. Then, we have that d * c = 1.

Therefore, for all n > n0, we have:

1/f(n) = d * (1/f(n)) = d * (f(n)^(-1)) = d * (f(n)^(-1) * c) = d * c * f(n)^(-1) = 1 * f(n)^(-1) = f(n)^(-1)

Multiplying both sides of the inequality f(n) ≤ c * h(n) by f(n)^(-1), we get:

1 = f(n)^(-1) * f(n) ≤ f(n)^(-1) * c * h(n) = d * h(n)

Since h(n) is positive for all n, we can divide both sides of the inequality by h(n) to get:

1/h(n) ≤ d

Therefore, isDominated(1/f, h) is true, where c = 1/d = c and n0 = n0.

This proves that if isDominated(f, h) is true, then isDominated(1/f, h) must also be true.

BB: Find a counterexample to the following conjecture: If isDominated(f,h) then isDominated (1/f,h).

ChatGPT:

Here is a counterexample to the conjecture:

Let f(n) = n and h(n) = n^2 for all positive integers n.

Then we have:

f(n) = n h(n) = n^2

It is easy to see that f(n) is not dominated by h(n). In fact, f(n) grows faster than h(n) as n increases.

However, 1/f(n) = 1/n is dominated by h(n) = n^2.

To see this, let c = 1 and n0 = 1. Then, for all n > n0, we have:

1/f(n) = 1/n ≤ c * h(n) = 1 * n^2 = n^2

Therefore, isDominated(1/f, h) is true, where c = 1 and n0 = 1.

But isDominated(f, h) is not true.

This is a counterexample to the conjecture, because it shows that the implication "if isDominated(f, h) then isDominated(1/f, h)" is not always true.

COMMENT: Again, ChatGPT loses logical correctness in such proofs. It just tries "to satisfy the questioner!

BB:

Let's define that a person is the grandchild of another person if there exists a third person who is a parent of the first person and a child of the other person. Let's also define that a person is a grandparent of some other person if there exists a third person who is a child of the first person and a parent of the other person. Prove: a person is a grandchild of another person iff the other person is a grandparent of the first person.

ChatGPT:

This statement can be proven by showing that if a person is a grandchild of another person, then the other person must be a grandparent of the first person, and that if a person is a grandparent of another person, then the other person must be a grandchild of the first person.

To prove the first part, suppose that a person A is a grandchild of a person B. This means that there exists a person C who is a parent of A and a child of B. By definition, this means that B is a grandparent of A.

To prove the second part, suppose that a person A is a grandparent of a person B. This means that there exists a person C who is a child of A and a parent of B. By definition, this means that B is a grandchild of A.

Therefore, we have shown that if a person is a grandchild of another person, then the other person must be a grandparent of the first person, and that if a person is a grandparent of another person, then the other person must be a grandchild of the first person. This proves that a person is a grandchild of another person if and only if the other person is a grandparent of the first person.

References

[Buchberger 2002]
B. Buchberger.
Invention by Lazy Thinking. In: A Min Tjoa @ Work – In Honor of the 50th Birthday of A Min Tjoa, ed: R. Wagner, pp. 11-31. 2002, Austrian Computer Society, ISBN 3-85403-164-5.

[Buchberger, 2022]
B. Buchberger.
Symbolic Computation and Artificial Intelligence. Research Institute for Symbolic Computation (RISC) Johannes Kepler University, Linz / Schloss Hagenberg , Austria.January 25, 2022 JKU, AI Lecture Series. Available from buchberger.bruno@gmail.com.

[ChatpGPT] ChatGPT: Optimizing Language Models for Dialogue. https://openai.com/blog/chatgpt/

[Theorema 1997] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, D. Vasaru.
A Survey oft he Theorema Project. In: Proc. Of ISSAC '97, Maui, Hawaii, July 21-23, 1997. Ed.: W. Küchlin, pp. 384-391, 1997. ACM Press, ISBN 0-89791-875-4.

[Theorema 2016] B. Buchberger, T. Jebelean, T. Kutsia, A. Maletzky, W. Windsteiger. Theorema 2.0: Computer-Assisted Natural Style Mathematics. JFR 9 (1), pp. 149-185, 2016, ISSN 1972/5787.