Optimizing and Diversifying Machine Learning Ensembles

Benjamin Rapkin, Rohit Pokhrel, and Toby Otto

December 2023

1 Introduction

Machine learning algorithms are ubiquitous today. With the increase in demand for these algorithms, a common technique that is employed to achieve increased accuracy is ensemble learning (Opitz). An ensemble consists of multiple models whose predictions are combined, with the hope of "averaging" the errors of individual models and gaining a higher accuracy. Two big questions that arise when employing ensemble learning are: what models to use and how much training time to allocate to each model. We try to address the latter question. When training models, data scientists have a limited amount of total time they can allocate for training, and ideally, they want to make best use of this time. We have devised an optimization problem that closely resembles Harry Markowitz's portfolio optimization strategy (Markowitz). In essence, we want a portfolio of ordered pairs, where the first entry in the ordered pair represents a model, and the second entry in the ordered pair represents the optimal training time for that model. The data scientist can then interpret this portfolio as an educated guess for allocating time to different models.

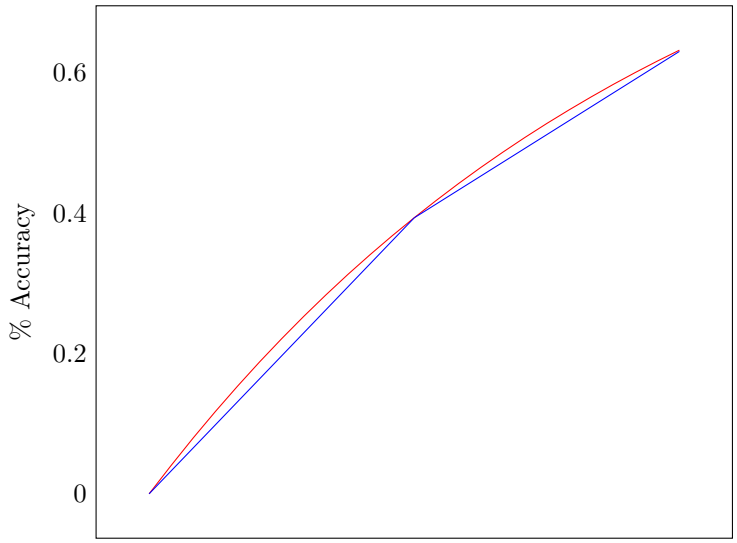
2 Setup

In our approach to portfolio optimization, we have m models which each are allowed to train for t_m time steps. However, unlike with stock returns, accuracy at any given time step is not a linear function of time but instead exhibits diminishing returns. This proves challenging to model as an accurate equation would require a regression to be performed on each ML model's performance and add a series of exponential constraints similar to

$$Accuracy = \sum_m \sum_i a_m - b_m \cdot e^{-c_m x_i}$$

This proved to be infeasible however (see 5.2). But we did not want to lose all the details of diminishing return, so instead we modeled each epoch as its own stock, with its own linear return function defined only over that time step. This allows us to linearize the graph and create a much more truthful representation.

Linearization of Accuracy for Two Time Steps



With each epoch of each model now its own 'stock', the original discount function is no longer applicable. Unfortunately, there is not a simple solution to

the as the original function discounted based on time and our analogue would be to discount based on how similar the data sets are to whatever data set we plan to run our ensemble learner on. This presents two particulate issues, this first being one may not have the data yet. An example of this is if a project were in the budgeting stage and the team needed to predict what kind of cloud computer resources would be needed such as CPUs, GPUs, or TPU as the efficiency of each various depending on which ML algorithms are being run. So knowing the makeup of an ensemble learning will greatly affect what you rent and for how long, which in turn informs the budget going forward.

The second, and likely largest issue is that tests which characterize the similarity of distributions, such as the Kolmogorov–Smirnov test or Wasserstein metric, require samples to have the same dimensionality, which is often not the case between different data sets. There is the option to create an embedding function which maps one of the data sets into a either a higher or lower dimension, however, embedding functions often introduce significant bias and if we care about which distribution is most similar it may be hard to guarantee that any ordering we produce of most to least similar data sets are reliable. Therefore, more research is needed for the creation of a reliable discount function.

With all this said, the last step is to calculate the variance and average accuracy for each 'stock' over the unique data sets. With this we can then create a covariance matrix akin to that in the original problem, allowing us to maintain the objective function

$$\text{minimize} \quad -rx + u \cdot x^T Cx$$

3 Contraints

For any given model, it is only reasonable that we choose only one epoch to invest in as it makes no sense to invest some amount into two different time stamps of a model. Mathematically this holds as our epochs are there to allow us to represent our diminishing returns with a detailed piece wise linear function.

A natural way to express this then is for each model epoch e_{jo} (model $j's$ o^{th} epoch) we invest $0 \leq x_{jo} \leq 1$ which is equivalent to the proportion of our total computational time that we spend training model j. These investments must follow the constraint

$$x_{jo} \cdot x_{jo'} = 0 \ \ \forall o \neq o'$$

Which states that there can only be one non-zero epoch investment per model

Now that we have constrained the model to only invest in one epoch, it must also only invest the amount of time that epoch spans. For each model epoch e_{jo}, we have some starting time s_{jo} and ending time t_{jo}. As we explored before, for each epoch we have the option to invest (given that no other epochs from this model have been selected) or not. If we invest in model epoch e_{jo}, then the allocation x_{jo} must follow

$$(x_{jo} - s_{jo}) \cdot x_{jo} \geq 0 \text{ and } (t_{jo} - x_{jo}) \cdot x_{jo} \geq 0$$

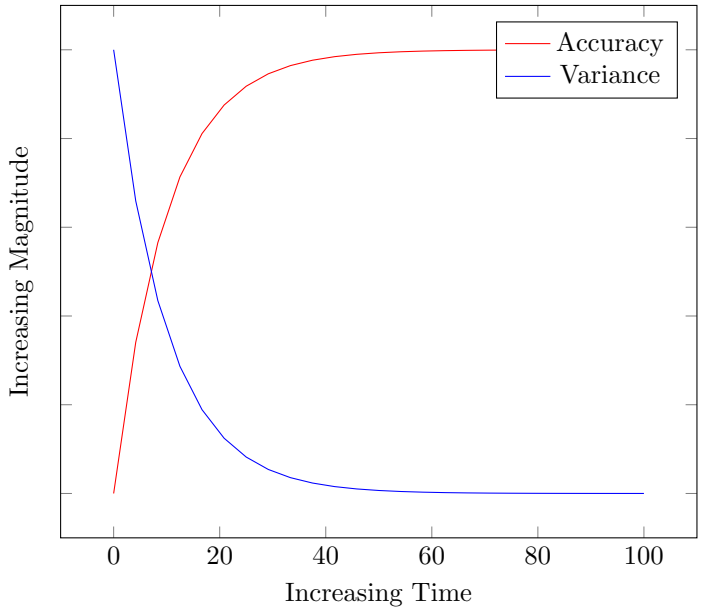
Notably however, this presents some issues as these produce quadratic constraints which means this problem is not necessarily convex. To get around this, we can reformat the constraints in a manner that insures they are convex by introducing p, a matrix of binary variables:

$$x_{jo} \leq p_{jo} \text{ where } p \in \{0, 1\} \text{ and } \sum_{o} p_{jo} = 1$$

$$x_{jo} \leq t_{oj} p_{jo}$$

$$x_{jo} \geq s_{oj} p_{jo}$$

In addition to the feasibility constraints on x, we want the ensemble model to be accurate, so we must introduce something analogous to the original return constraint. However, if we take a look at the graph below, it is evident why this cannot just be the time investment multiplied by the mean accuracy at that time like in the original problem



As we can see, because a ML model makes smaller and smaller adjustments at each time step, the model's variability will inevitably decrease in an exponential fashion. So if we just asked our program to solve the problem where

$$\text{Accuracy} \ \geq \ \text{Average Return} \cdot x_{jo}$$

Then it would pick an ML model which crosses the accuracy threshold then invest entirely into it as that will give it a lower variation then investing between two models which meet the threshold. So average accuracy is a poor heuristic for our ensemble accuracy. Therefore, we instead proposed a different algorithm

Let a_{jo} denote the accuracy of model j at epoch o

$$Score = \sum_{j} \sum_{o} a_{j(o-1)} + (a_{jo} - a_{j(o-1)}) \cdot \frac{x_{jo} - t_{j(o-1)}}{t_{jo} - t_{j(o-1)})}$$

This says that if we invest in a given model and epoch, its return is the accuracy at the last epoch, plus the increase in accuracy over the current epoch

multiplied by the proportion of the epoch that we train for. Using this metric, we can set score/accuracy bounds ≥ 1, which forces the program to diversify its time investment over several ML models as any given model cannot reach an accuracy > 1

4 Problem Statement

Finally, now that our objective function and all the constrains have been explained, we can formally state our optimization problem as:

$$
\begin{aligned}
\text{minimize} \quad & -rx + u \cdot x^T Cx \\
\text{subject to} \quad & 0 \leq x_{jo} \\
& x_{jo} \leq p_{jo} \text{ where } p \in \{0, 1\} \\
& x_{jo} \leq t_{oj} p_{jo} \\
& x_{jo} \geq s_{oj} p_{jo} \\
& \sum_o p_{jo} = 1 \\
& \sum_j \sum_o x_i = 1 \\
& Score(x) \geq A
\end{aligned}
$$

5 Implementation

5.1 Data Gathering

To create the underlying performance data for each ML model analogous to Markowitz's stocks, we trained several models for various times. These include a feed forward neural network (NN), a random forest (RF), and a gradient boosted tree (GBT) [1]. Additionally, we trained deterministic classifiers such as,

[1] As the tree algorithms are deterministic, training time was instead based on the number of sub-trees they were allowed to generate before predicting

a decision tree, a naive Bayes classifier, a svm, and a least squares regression. The reasoning for adding these is that they are incredibly cheap to run and do not improve, meaning we could reliably expect to see small investments in them which would provide a litmus test for whether our model was working as intended. Additionally, they are often used in industry alongside more complex ensemble models.

The algorithms were run on a wide range of classification data sets designed to illustrate the far-reaching applications of machine learning and the wide impact that improvements to ensemble learning could have. They include classifying clothing item by images, the health of fetuses from a variety of vital signs, types of glass from chemical test results, whether loan would be approved for a given applicant, oil patch classification from satellite imagery, and determining if a celestial body was a pulsar.

In addition to the conceptual variety, significant attention was paid to ensuring that these data sets provided a good variety of mathematical features which would affect convergence rates. Number of classes, bias in proportion of classes, data types[2] and data distributions varies greatly between the data sets.

5.2 Optimization and Results

Once we had the performance of each model over each epoch, we used Gurobi's python api to find an optimal solution for our model. Unfortunately, this presented many issues. The largest of these was the score parameter. As we mentioned at the start, the original approach had been to make an exponential function for an accuracy measurement. However, in order to search for an efficient frontier we must pass in an accuracy method which is required to be a function of the Gurobi wrapper classes for x.[3]. This is difficult as we cannot directly access any x_i from the wrapper and as such cannot do functions which

[2]This is referring to continuous vs discrete data which was dealt with using standard OneHot encoding techniques

[3]The function can technically be anything that produces numerical output, however, if you want to use the data which corresponds to a given solution, i.e. the x vector, all manipulations must be done with x as a Gurobi object, not as standard array, meaning how we can access the data is extremely limited.

are not expressible as linear algebra [4] Similar issues appeared when attempting to implement our piece wise score function, however, we have been able to implement a correlated version which is off by a constant amount per x_i.

If we use the relaxed score function, then Gurobi will successfully optimize the model to meet a minimum accuracy threshold with the least variability possible. In a test example with two algorithms for 10 epochs each, it split $\sim .247\%$ of it's compute time on a neural network, setting it at epoch six, and $\sim .753\%$ of it's resources on a random forest with eight trees. Attempts to scale this to the other trained models have been difficult. When scaled to three algorithms with 10 epochs of GBT, and 20 for both RF and NN, the pre-solved model has approximate 1000 constraint terms. These must be hard coded and Gurobipy offers no way to print a models current constraints making debugging extremely difficult and time consuming. Therefore at the moment the model is only partially functional for specific conditions with 3 sub-models.[5]

6 Further Research

6.1 Discount Functions

While some of the issues regarding an analogous discount functions have been touched upon earlier in the paper it is an area that would be well served by more research. While brute force calculations of distribution's similarities are difficult, there could be hope in instead using metrics such as similarity in comparing class biases between two data sets. Some of the data sets had few-shot or even one-shot learning features where a given class have very little representation in the training sets which requires algorithms which are very good at high level

[4]The documentation lists a get attribute function that should allow access to the x_i entry, however, every time we attempted to use there errors were throw due to the index being lazily evaluated. Additionally, Gurobi does not allow a variety of functions to be applied to many of its variables. For example, any form of division where x_i is the divisor is forbidden, forcing the introduction of dummy variables in the constraints. So it is unclear if we were able to get x_i whether the API would allow e^x and I do not know how this could be circumvented with dummy variables.

[5]It will currently only solve the model when $\sum_i x_i < .5$ which I believe is due to an error with the epoch exclusivity or bound constraints but I have been unable to track these down.

generalization. So we may want to discount algorithms performance on data sets without these features or other meta-parameters.

6.2 Augment Historical Data

The Markowitz Model's data is based on historical trends in the stock market, which makes perfect sense for the original problem. However, in trying to stay analogous to that, our model only uses the specific ML model's performance on a few data sets to guide its choices. However, each ML model is just a numerical method, so a large next step would be to augment that observed performance with what we can say about a model's best case, worst case, and expected convergence rate based on what we know about the data the model will be tested on. Depeneding on how tight we can get these bounds, this would prove to be a large boon to our fairly small set of sample

7 Code

Please find our code hosted at kaggle

References

1. Opitz, D. W., and Maclin, R. (1999). Popular Ensemble Methods: an Empirical study. Journal of Artificial Intelligence Research, 11, 169–198. https://doi.org/10.1613/jair.614

2. Markowitz, Harry M. 1952. "Portfolio Selection." Journal of Finance