Filesystem Machine Learning Analysis

Nathan Tibbetts

April 16, 2020

Abstract

Given a proposed method of visually representing a user's filesystem as a tree in order to improve human-computer interactions, we perform an analysis of file metadata to show the effectiveness of using machine learning to estimate relative directory importance, in place of expensive recursive metrics. We conclude that machine learning models, even when trained on a user's specific system, cannot easily be used to accurately perform these estimates in a realistic situation, but may be able to model them, and explain why. We also do some cluster analysis of the data.

1 Problem Statement and Motivation

1.1 Background and Motivation

We are given a proposed system of visual representation for a user's filesystem, designed to improve the efficiency with which a user can find and organize files and folders, which involves drawing a visual, fractal-like representation of their computer's directory tree. Each subsequent depth layer of nodes in the tree is drawn smaller and closer-packed than the last, with traces connecting them to their parents, so as to visually divide and segment information so that the user can mentally parse it very quickly. Since drawing such a tree is a recursive process, and since a given computer contains hundreds of thousands to millions of files, it is impossible to draw and label everything. Therefore, we must only draw what is most "important."

However, since the visual tree is recursive, we must not only think about drawing what is most important, but about how important in total each item is together with its descendants, amortized by their depth in the tree, so that we can allocate to each item a space in which to draw which seems fair, relative to the others. We call this "recursive importance." This problem does not have one particular solution, since file "importance" is ambiguous and would depend largely on user preferences, but designing a specific solution is an easy recursive problem, once given file metadata such as whether files are hidden, how large they are, how deep they are in the tree, and how frequently the user accesses them. However, we further tease apart the problem by trying to estimate only what we define as "recursive weight," which is a representation of the visual size of a branch of the tree, a sum of the weight of an object with the depth-amortized weights of all its children, *without* respect to the relative importance of specific files.

1.2 Mathematical Description

Let F be the filesystem, or the set of i indices for nodes in the filesystem.

Let F_i be the sub-tree of F, starting with node i as root.

Let W_i be the Recursive Importance of node i.

Let w_i be the weight, or importance, of node i. To remove importance as a factor and find Recursive Weight we set $w_i = 1, \forall i$.

Let d_i be the depth of node i in the tree.

Let $\alpha = \frac{1}{4}$ be the depth-amortization factor.

Then, the general formula for Recursive Importance, or Recursive Weight when $w_j = 0, \forall j$, is:

$$W_i = \sum_{j \in F_i} \alpha^{d_i - d_j} w_j$$

This will be the recursive value we try to approximate with machine learning for each node $i \in F$, using $\alpha = \frac{1}{4}$, and $w_j = 0, \forall j$.

1.3 Problem Statements

All this does not require machine learning, but since these methods are recursive algorithms performed on massive trees, potentially making them hugely inefficient, we pose it as a machine learning problem to non-recursively estimate the value of the Recursive Weight W_i, treating as true labels or ground truth the outputs of the simple equation given above performed on every node in F. A machine learning replacement for this algorithm, if accurate, would be useful in reducing the processing time required to draw or update the filesystem tree F for this user interface. This is the primary problem we are trying to solve.

Secondarily, we are interested in classifying or grouping files, based on their path strings alone. For example, how much information is contained in them - can we observe meaningful clusters in the data? If accurate, this could be useful in improving our definitions of file "importance". Or even better, can we perhaps recover enough recursive information from the paths to estimate Recursive Weight in this way?

As most of our other metadata is in the form of boolean variables, we did not expect this data to be continuous enough to be properly clusterable - but we show later that we were wrong.

2 Related Research

Regarding the first problem, we have heard of no research regarding the estimation of recursively based metrics via machine learning or neural networks. Surely some related proprietary studies have been done by companies such as Microsoft and Apple in order to improve search suggestions on the operating systems of personal computers, but these are likely representing simple "importance" of files, rather than "recursive importance". There are some examples, though, of recursive metrics without machine learning, of course. [1]

As for the second question, clustering files based on path names, this is almost a forray into user and programmer psychology and naming conventions, and fits with very little other research known to us. The Perception, Control, and Cognition Lab at BYU has done some work in clustering and classification on sentence encodings[2], but with no exploration of file path strings.

Both of these problems, however, aim to explore and improve human-computer interaction and user interface efficiency - an area in which much research has been done, but little of it to do with directory tree visualization, except for what you see already implemented in existing interfaces.

3 Ethical Implications

We see this potentially being implemented into a user's operating system as part of a user interface. Since such an algorithm would only be gathering data from itself in order to be more accurate for the user, and has no need to share its data with other programs, we see no more adverse ethical implications from the use of this algorithm than from the everyday use of one's own computer filesystem; as long as it is done within a relatively secure program, it should pose no risk to personal data.

On the other hand, such research could potentially be used in other data visualization software, such as in knowledge graph exploration, family pedigree and descendancy tree visualization, social network exploration, etc. While such things could obviously be used nefariously, we generally consider the increase of knowledge in the field of data visualization an important step forward for the many good uses that could also clearly come of it.

Thus, the main ethical question for us would be whether to patent and protect work in this area of research, or to release it (with other protections) into the public domain. However, as this applies more to future work than to the limited scope of our current project, this is a question to be answered another day.

4 Data

Description: All file and folder paths and metadata from the filesystem of one computer, which describes file/folder properties, immediate sizes, ownership, and protections, including some feature-engineered information such as one-layer (immediate) child count and depth found in the tree. We also include an estimate of how many days ago each file was modified, in place of access or modification dates.

Source: The data we use was scraped from my personal laptop's filesystem, then cleaned and feature-engineered.

Suitability: This choice of source will clearly contain some biases in the Userspace, particularly in file names, since each user uses their system differently. However, it is sufficient to show what we want to show here - which is to demonstrate how well machine learning can generally be expected to work on this type of data and problem posed.

Reliability: From previous analyses of our data, we know that there are likely many "ghost" or temporary OS files included with the data - however, this only makes the problem more realistic, and more difficult for an algorithm to properly sort through and distinguish effectively. Since the data was scraped and engineered programmatically, we expect it to entirely accurate. We consider the data thus reliable.

5 Methods

The first problem can be formulated as a regression-type machine learning problem. We compute our approximations $\tilde{W_i}$ of the ground truth Recursive Weight W_i by building a simple feed forward neural network with ReLU activations on each of 4 hidden layers with 100 nodes each, and a single non-activated scalar output. We begin with such a small network under the assumption that the data is relatively simple, and we keep this structure for consistent comparability across various results. We use the L1 distance $L(i) = |\tilde{W_i} - W_i|$ as our loss function. Initially we used a train / validation / test split of about 700000 / 100000 / 200000, but this changed when we needed to use a specific subset of our data. Note that overfitting will not occur with only one training epoch, which we can do initially because we have such a large dataset.

For the second problem, we first treat it as an encoding problem in order to convert our strings into vectors, and then treat it as a clustering problem. We begin by converting file paths to "sentences" by replacing "/" characters with spaces. Because we assume that there is a lot of information contained in the meanings of the words and names contained in file paths, and not just in the relative letters and words composing them, we elect to use Google's Universal Sentence Encoder 4 (which encodes a lot of semantic information as well as structural) to generate high-dimensional vector representations of our paths. We then UMAP on these vectors to cluster our data and visualize it, as well as using UMAP on our other metadata, treated as vectors containing mostly boolean entries. We color the clusters by Recursive Importance as a test. If cluster divisions are apparent, we will will use a neural network structure equivalent to the one previously used to estimate Recursive Importance. If not, we will consider this step unproductive, as the patterns in the data would thus appear to have little to do with recursive metrics.

6 Results

6.1 Problem 1: Approximating W_i

Initially the neural network learned the most common value in the dataset for W_i, which is near 1, and achieved very good loss on it because our dataset was terribly unbalanced in the distribution of W_i. See the training loss results in Table 1.

Normally any loss value below about 3 would perhaps be acceptable for this application - except we see in Figure 1 that it failed to learn the outliers - which are important in our application, as these are the interesting folders with many descendants. We afterwards attempted to learn $log(W_i)$, which also did not smooth out, and produced even worse results than before when we computed the true distance, $|W_i - e^{L_log(i)}|$. Next, we created a class-

	μ	$\lvert W_i - \mu\rvert$	Train $L(i)$	Test $L(i)$	Accuracy
Full Dataset	1.333	0.614	0.333	0.358	0.417
Balanced Dataset	15.407	14.864	11.752	11.497	0.227
UMAPed Balanced	15.473	14.934	4.328	4.545	0.687
UMAPed Path Vecs	15.473	14.934	7.290	12.613	0.132

Table 1: Results table. Note that those containing a subscript i in the column header are averages over all i. Training loss here is the average loss on the training data, but after training has already been completed.

balanced subset of our dataset with much more even stratifications of W_i, and trained again. This subset contained around 5000 entries having $W_i > 15$, 6000 entries with $4 <= W_i <= 15$, and 6000 with $1 < W_i < 4$, dropping all entries with $W_i = 1$ as these have no children and are easy to identify non-recursively anyway. We then use a train / test split of around 14000 / 3000. However, even training for 40 epochs on this smaller, more even dataset, it appeared to have learned nothing but the average value, and had to run tests to prove that it could overfit, until we compared it with the expected value for a random choice in this situation, whereupon we saw that it was significantly better than random. We also tried varying batch sizes, network size and depth, and even removing batches. All of our results were somewhat comparable to these. See Table 1 for results, and Analysis for details. All further examples are done with a batch size of 20 and trained for 40 epochs, with the network structure as described previously.

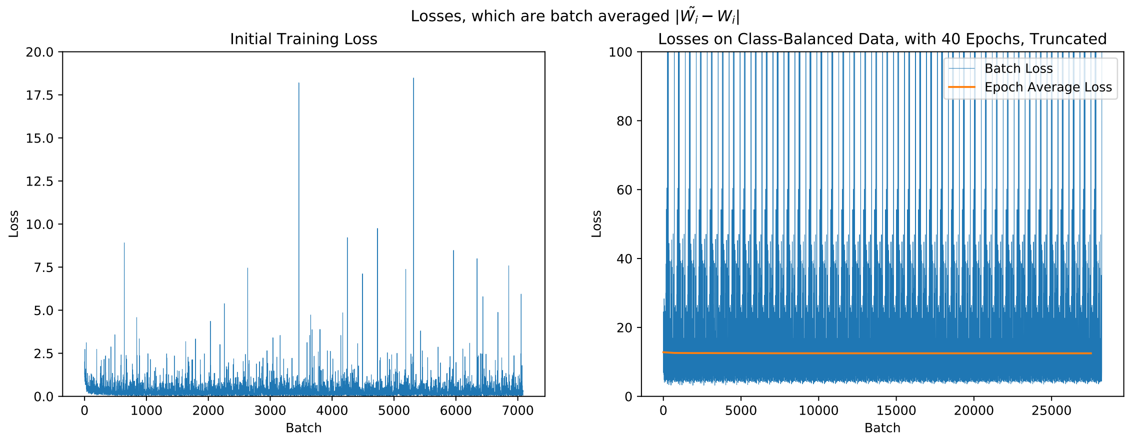

Figure 1: Left: Loss value throughout training with basic setup. Here we used 1 epoch and batch size of 100. Right: Loss value of network trained on W_i class-balanced dataset with our standard 40 epochs and batch size 20.

6.2 Problem 2: Visualizing Clusters

Encoding path strings as vectors and then performing UMAP on them turned out to be effective for visually identifying clusters. With some experimentation, we decided to use the parameters n_neighbors=8 and min_dist=.5. We color using a log scale of Recursive

Importance, and see that this appears to work surprisingly well. We then tried UMAP on our original data, to see that it also produces even clearer clusters. We show how each transformation function UMAP learned generalizes to a set of test data.

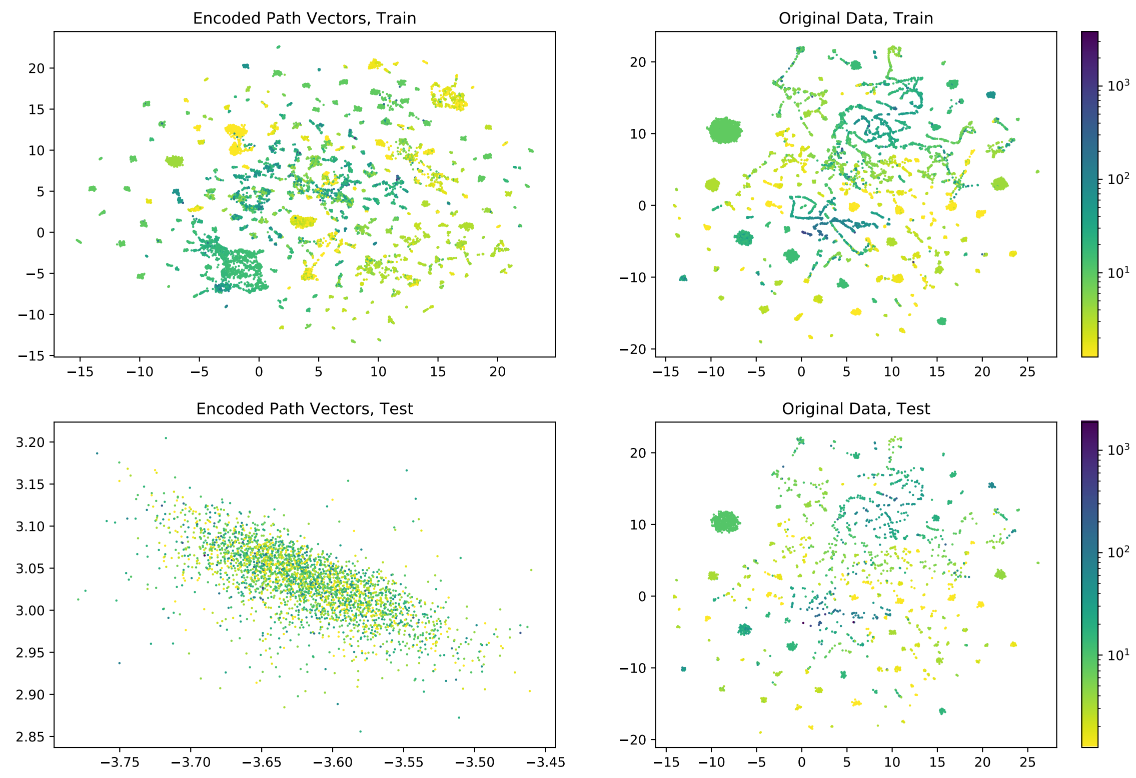

Figure 2: UMAP Transformation on our class-balanced dataset's encoded path vectors as well as original information. First fit on training sets, and then showing how the train fit transfers over to the test sets. These graphs are colored by Recursive Importance, showing clear patterns in the data.

We trained our neural network on 3D UMAP outputs, and show that UMAP does aid the training, and with our original vectors even generalizes well, but that path vector encodings simply confuse the issue.

7 Analysis

Let $\mu_i = mean(W_i)$. Then a classifier which always guesses μ, or is randomly centered around it, should have an average loss $mean(L(i) = |\tilde{W_i} - W - i|)$ equal to $mean(|W_i - \mu|)$, the average distance from the mean. Note that this latter value is akin to a non-squared version of variance. We defined the model's accuracy to be

$$Accuracy = \frac{mean(|W_i - \mu|) - mean(L(i))}{mean(|W_i - \mu|)}$$

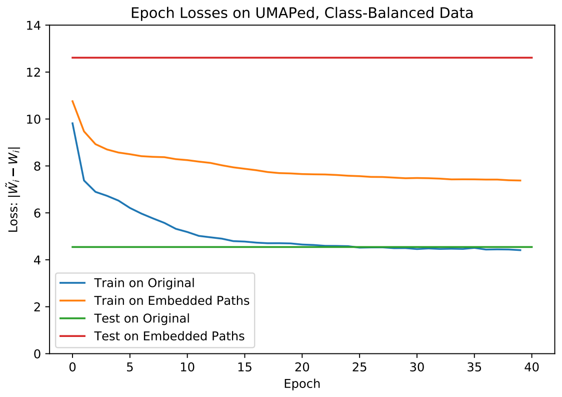

Figure 3: Epoch-averaged train and test losses on UMAP transformed data. As the epoch-per-epoch loss looks just as spiky as before, we have omitted it.

Meaning that 0% accuracy is the equivalent of mean-centered random choices, and 100% accuracy is a perfectly accurate model.

Normally when you train a neural network, you want to see the loss function make a nice exponential decay function, but in the first graph we see it quickly drops to the average value of W_i, and stays there because it finds a lower loss there, despite ignoring the few it gets very wrong. This explains the failure of the first training to learn anything. However, after evening out the stratfications in the data so that there were just as many high values of W_i as intermediate and low values, the fact that it still learned no apparent pattern by which to better its approximations caused us to believe that the data was very uncorrelated - essentially, that it was all noise. However, upon computing $mean(|W_i - \mu|)$ and seeing that it was better than random, and upon seeing the *very clear* patterns in the colors of UMAP clusters, we learned that there is actually quite a strong pattern. This was a rather surprising result for us - we expected to find such a pattern in the path strings, but not in the file metadata.

We assume that the folder path vectors show clear enough color separation in the clusters because the words in the paths relative to each other include information about the tree structure, and that this causes them to be encoded nearby in sentence space. However, we see that just as much recursive information is contained in the original metadata as well. Also, notice how well the UMAP transformation algorithm generalized on the test set for our original data, and how awfully it did on the test set for encoded path vectors (compare Figure 3 lower right and lower left, respectively). It becomes clear why both of the neural networks trained on UMAPed vectors performed well, but one miserably failed its test accuracy, doing almost as bad as completely random. If information is lost at one step in the pipeline, it no longer exists for the next step in the pipeline to see. We assume sentence vectors contained so much more variance in UMAP because there are potentially many new words in an unfamiliar path name, but relatively few possible new combinations of metadata properties.

In the future, it would be wise to perform the preparatory work of visualizing the patterns in any new dataset before doing deep learning on it, in order to get some idea of the complexity of the problem, and potentially adding a shortcut to faster learning by putting UMAP into the pipeline - though we suspect that this would be better for a small network, and that for a robust, large DNN UMAP would eventually get in the way of learning finer-tuned representations because of how it forces clusters together.

Finally, we submit that our new algorithm of using UMAP on the Google Universal Sentence Encoder, and then trying a neural network on top of that, will not work effectively for any system which has to manage new input, but may work for analysis of a static system that receives no new data - unless you want to retrain UMAP each time you add data, which is a valid option.

8 Conclusion

Our two problems ended up being tied together. As we began to see the results of the second problem, we found that our data was not as noisy and random as we thought, and that it could be learned, and that that learning may be aided by transforming the data appropriately. Furthermore, we conclude that recursive patterns appear to be just as strong or stronger in standard directory metadata as in the path strings representing them.

While the results show that with a large enough neural network we could potentially learn recursive filesystem metrics reasonably accurately, we do not think that a machine learned algorithm that is lightweight enough to fit in a user's operating system interface would be able to accurately enough estimate the Recursive Weight metric necessary for the implementation of a visual, fractal-style tree interface. Such a network, even once trained, would likely be bulky and slow, and would not give accurate results large directories because they appear as outliers to the system. However, this work validates that such models could work, and they may be useful in studying user behavior and filesystem properties. It also warrants further research into performing clustering algorithms on OS and user metadata for the same reasons.

9 Future Work

- Experiment with larger and more varied models, perhaps such as a random forest, and attempt to achieve high accuracy, including on outliers. Experiment with the algorithms outputs in specific situations instead of in general, in order to see what kinds of values it outputs.
- Further study user behavior and filesystem properties through clustering of the file metadata and encoded paths. Combine this with file usage frequency to develop a research-based measure of file/directory importance to incorporate into recursive importance metrics.
- Develop an efficient recursive algorithm for computing Recursive Weight in real time, as opposed to an estimate. Alternatively, develop a filesystem format which allows the OS to store extra recursive metadata so that it does not need to be recomputed frequently.

References

[1] Kdirstat, 2016.

[2] Nathan Tibbetts. The analyst, a toolset for studying high-dimensional embedding spaces, 2018.