Deniz Soral
18 February 2025
Link to Self Score
Link to Colab
Link to Notes

Data Finding

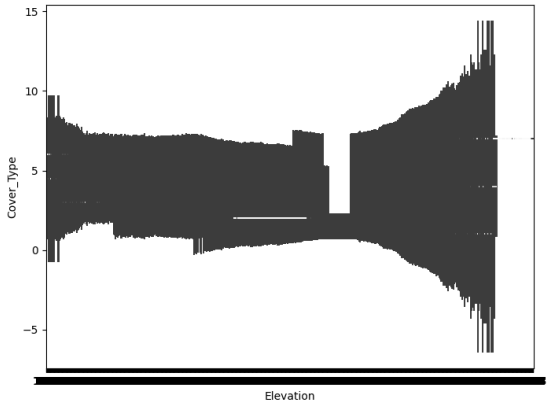
I found the forest coverage dataset pretty quickly. My criteria remained the same: something I can understand, varied, and interesting enough. I also knew I wanted to do a classification model because regression sounded less fun. I just scrolled through some tabs on UCIrvine, and this one stood out as it had a lot of features, instances, and was very basic.

The data has 54 features, each a different aspect of a patch of forest. Examples include elevation, shade at certain times of day, different soil types, and distance to roadways and water. There are 7 targets, each a different kind of tree coverage: Spruce/Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir, and Krummholz (windblown trees that cannot grow).

Feature Engineering

I didn't spend too long on this, as the model was already one-hot-encoded for me (which was kind of annoying, as I would have liked to do it myself and write about it here).

First, I shrunk my data to 500 random samples per class, evenly split so the model would be trained properly without bias towards one target. This was also the only reasonable way for me to efficiently train and visualize my data; otherwise, it would have looked like this (pretty silly).



Next, I used StandardScaler to standardize my data. I opted not to normalize because I thought standardization—which sets the mean to 0 and the standard deviation to 1 (per feature)—would allow for more variance within my data than just constraining it to [-1, 1]. In reality, it probably does not matter *that* much, as long as I preserve the data's, albeit few, negative values.

I also visualized my data (without standardization, of course) to decide on whether or not I wanted to bucket; here's what I ended up with.

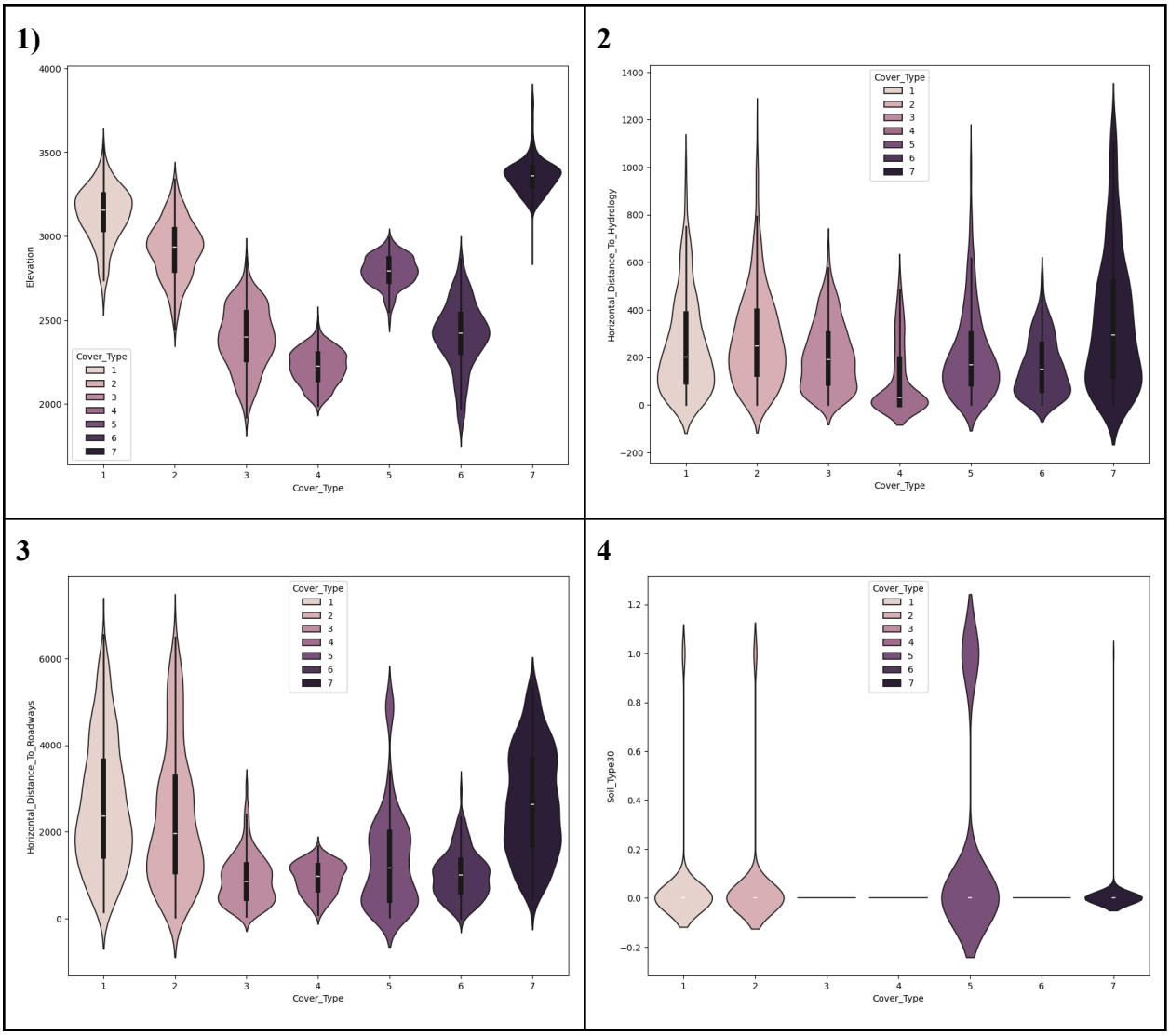


Figure: *Graphs of 4 features over target Cover_Type . 1) Elevation. 2) Horizontal_Distance_To_Hydrology. 3) Horizontal_Distance_To_Roadways. 4) Soil_Type30*

There are only a few features as distinct as Elevation (1) while most look like Horizontal_Distance_To_Hydrology (2) and Horizontal_Distance_To_Roadways (3), meaning bucketing wouldn't have done much good, and probably lose a lot of minute distinctions between classes. 40 out of the 50 features looked like Soil_Type30, which, surprise! ranges from 1–40. I also just don't think bucketing makes sense for a neural network. Maybe for normal classification models, where you can bucket to make the model less overfit, or make the model more interpretable and generalized. From my understanding though, for a DNN, the whole point is to create small distinctions between weights, which bucketing would remove. Also

interpretability doesn't really feel like a concern here, because we have no idea what happens in hidden layers and how the model makes decisions, unlike an easily visualizable Decision Tree.

Furthermore, some of the features are very similar between targets, like the ones that measure shade, which is literally just the position of the sun, so it would make sense that shade would be relatively similar. But the data was distinct enough that I decided to keep it.

Finally, I split my data into training and testing groups with a 80–20 train–test ratio.

Model Setup & Loop

As I mentioned earlier, the dataset I used has 40 Binary Soil_Type features, meaning there's really only 15 or so *unique* features that it provides me. If I were to just throw in every single feature into the input layer, I knew the model would be extremely biased towards whatever soil type was most commonly associated with a given tree coverage, which might make a accurate model if the soils are hyper specific; but, it would definitely not be a very applicable model, as it would require those specific measurements and designation of soil—which the UCI repo page does not name. Furthermore, each Soil_Type is pretty useless by itself because they're all binary—many of the Soil_Types are only turned on for 1 or 2 targets. On the other hand, I didn't want to completely remove the Soil_Data, as it probably could help the model become more accurate and applicable (if, say, those measurements were common or easy to take). To reconcile the issue, I decided to make an 'ensemble' model, where I would train the Soil_Type data into one input/node, then feed that into the rest of the model. This would help ensure that the data was used, but not very important in the grand scheme of the model's weights. Here's a quick sketch of what should look like:

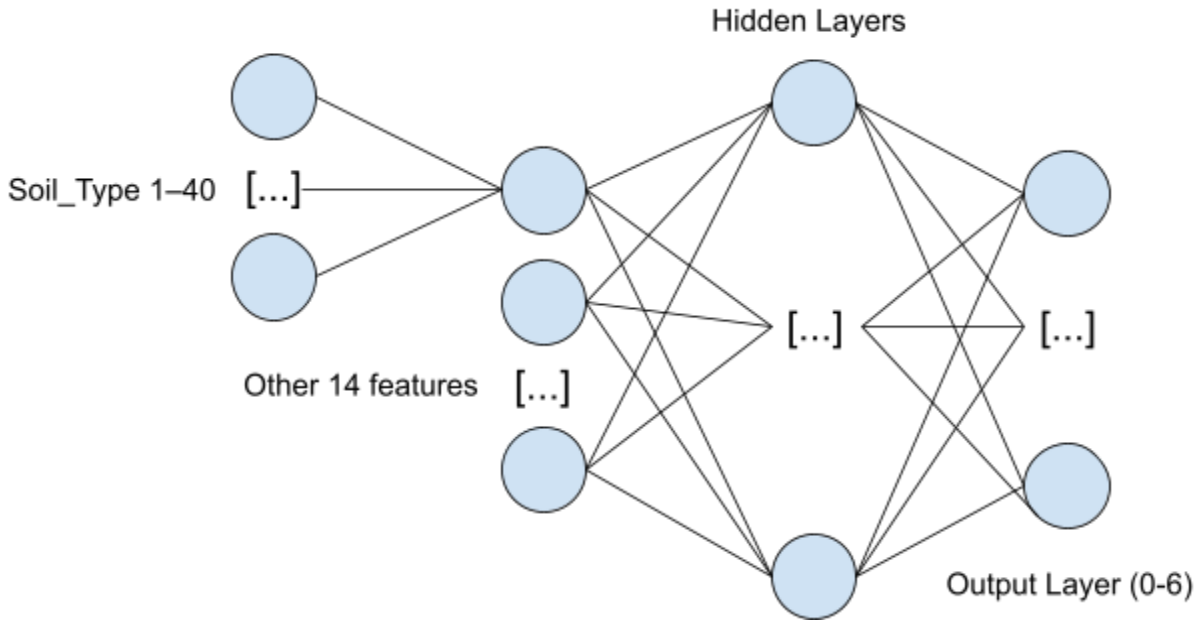


Figure: *Super cool sketch of my 'ensemble' model that should definitely give me a 3 in training loops or something.*

The training loop for this was pretty straightforward. I split my features into a soil or no_soil group, then fed both of them into the model. Then, I trained the soil data to 1 node—from 40→64→32→1—using powers of two to make convergence faster. Then, I concatenated the new single node as a new column to the no_soil tensor, then ran that through the model—15 (14 other + 1 soil)→32→64→128→64→32→16→7.

After each layer I used ReLU as my activation function (definitely started with this). I tried a couple others, mainly Sigmoid, and they did not work nearly as well. The loss would stay super high (~1.8) and my accuracy/precision/recall was horrible. The moment I swapped to ReLU the model started to work miraculously well. Honestly, I have no clue why it's so much better with ReLU, especially because StandardScalar creates negative values that ReLU isn't preserving. I hypothesize that it may make the gradient more navigable by creating sparsity from turning a lot of unnecessary neurons off that would otherwise be negative. Really, though, it work good so me no fix.

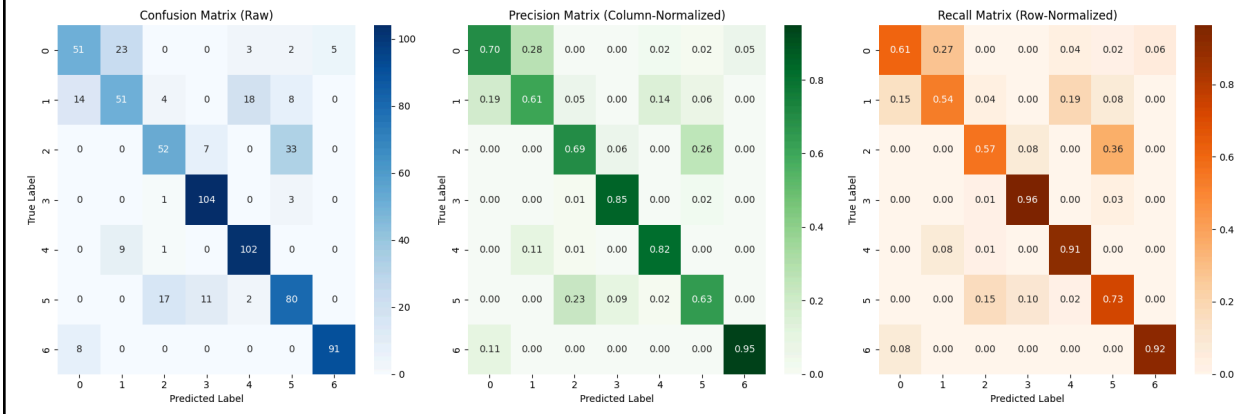
I played around with the learning rate a bit, testing values ranging from 0.1 to 0.01. I found that a learning rate of 0.05 worked best for the model. Anything higher would make the loss skip oscillate a ton and struggle to settle in a local minimum, other learning rates were inefficient. I also used an epoch of 7500, which is where the model tended to plateau in terms of precision and recall (and accuracy, if that really matters).

I used CrossEntropyLoss as my loss function, which is pretty common for non-binary classification models. It does some wacky math to penalize a model's ouput probabilities for each class (worse guesses means higher loss). The most important thing is that CrossEntropyLoss automatically applies softmax, which makes all final outputs add up to one—actually making them probabilities—allowing me to validate my model later.

Realistically, I could have condensed my model further as I did with the Soil_Type features. There are a few other groups I could have made, namely the hillshade features and Wilderness_Area features. These changes may have made the model better and reduced bias towards the specific characteristics they were measuring, but it was honestly a really big pain to set up the separate layer for Soil_Type, it made the model much slower, and the groups would have been pretty small thus making the bias negligible, so I opted not to create more 'ensemble' layers to save my soul.

Validation

To validate my model, I used confusion matrices to visualize accuracy, precision, and recall. Here are those figures now.




```
Class-wise Metrics:
Class 0: Accuracy = 60.71%, Precision = 69.86%, Recall = 60.71%
Class 1: Accuracy = 53.68%, Precision = 61.45%, Recall = 53.68%
Class 2: Accuracy = 56.52%, Precision = 69.33%, Recall = 56.52%
Class 3: Accuracy = 96.30%, Precision = 85.25%, Recall = 96.30%
Class 4: Accuracy = 91.07%, Precision = 81.60%, Recall = 91.07%
Class 5: Accuracy = 72.73%, Precision = 63.49%, Recall = 72.73%
Class 6: Accuracy = 91.92%, Precision = 94.79%, Recall = 91.92%
```

0 = 0.6496439610936663
1 = 0.5730280552419005
2 = 0.6227304886769964
3 = 0.904387221151198
4 = 0.8607531128742687
5 = 0.6779661870503596
6 = 0.9333294199560817

Figure: *Runthrough of the model, with accuracy, precision, and recall matrices on top, class-wise (without comparison to other classes) values in the middle, and f-1 scores on the bottom.*

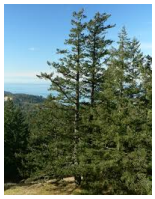
This is pretty decent for discerning between 7 classes. If this was a binary classification (and I had the same average values) I would be much more concerned. In the context of trees, I don't think there's a bit difference between prioritizing precision or recall. Recall in this scenario would just be like asking "out of all the Krummholz tree covers, how many of them did I miss?" Precision would be "out of all the things I labeled as Spruce/Fir, how many weren't actually Spruce/Fir?" Under the context I worked on, there isn't a huge difference between each of these, so taking the harmonic mean of the two (essentially trying to maximize for both) with the f1-score is probably a more holistic approach. Say, though, if a certain cover type is more prone to fire and recognizing all instance is very important, then maximizing recall for that type would

probably be more important than precision or f-1 score, and model probably wouldn't be great at this. Generally, I don't really see a real world application for this model, considering that anything important like ecological conservation or fire prevention would need *much* higher precision and recall.

My guess for the low numbers for classes 0, 1, 2, and 5—corresponding to Spruce/Fir, Lodgewood Pine, Ponderosa Pine, and Douglas-Fir—are because the trees are pretty similar in type. These trees are all conifers, meaning they grow and germinate in similar ways, all have needle like structures, and possess similar barks. Furthermore, these trees all grow in similar areas—mainly in the taigas of the Northern Hemisphere—meaning its likely that the relative position of the sun, soil composition, and slope are similar due to this fact. Each other tree, Cottonwood/Willow, Aspen, and Krummholz, are wildly different from the rest of the data set, which is likely why they have much better precision, recall, and f-1 scores (like really really different, you can tell just from their pictures).

My mean f1-score is ~74%, which again, pretty solid. Before I made all the adjustments I described earlier it was in the mid-50s, so I'm glad to say the model improved substantially. So yeah

Let's play a game and see if you can verify this for yourself! I'm going to show you an example of each one of these trees with labels, then a mixed up set of different specimens of each tree unnlabeled. See if you can guess what each one is, the answers will be on the page after.

Spruce/Fir	Lodgepole Pine	Ponderosa Pine	Cottonwood/Willow	Aspen	Douglas-Fir	Krummholz
						

QUIZ (don't scroll past this)

ANSWERS

Aspen	Ponderosa Pine	Krummholz	Spruce/Fir	Lodgepole Pine	Douglas-Fir	Cottonwood/Willow

I'm not entirely sure how this is related to validation, but my train loss graph also looks about as normal as I would expect it to, which means the model is probably finding a local minimum and not skipping around the gradient.

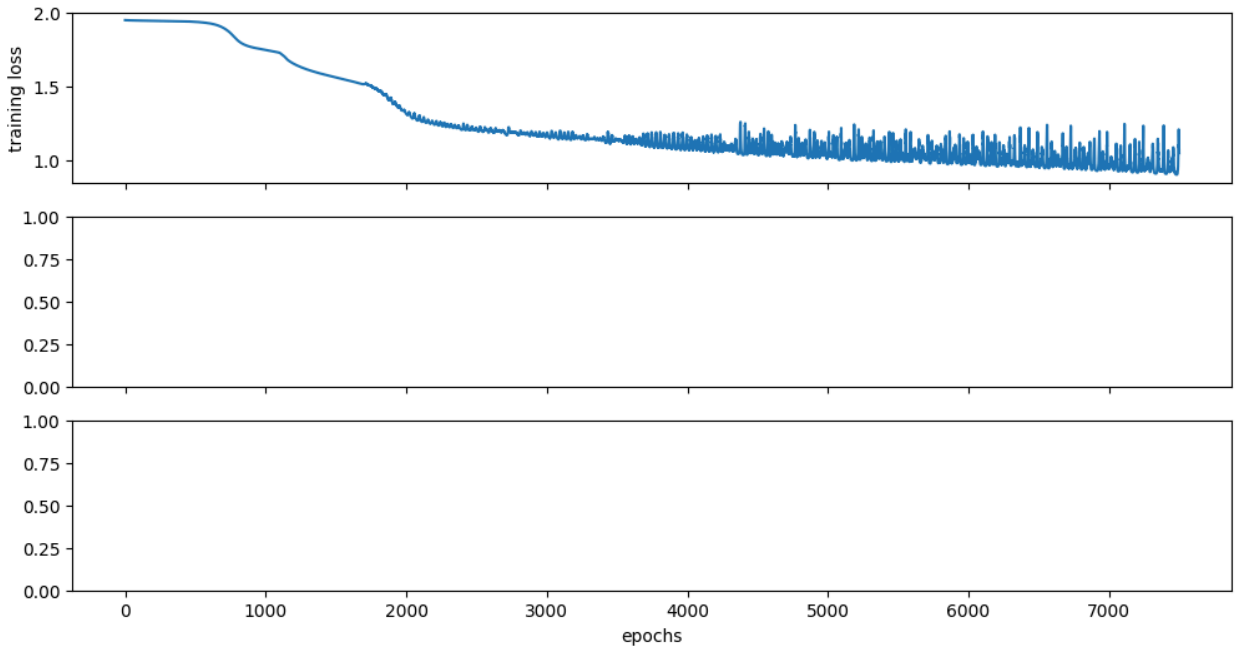


Figure: *Training loss over epochs.*

Those oscillations at the end are a bit concerning, though. I probably should have used a scheduler (or whatever it's called) that gradually lowers the learning rate the lower the loss gets. I tried doing this, but my code kept on breaking so I didn't bother. My guess is that it would have improved the model at least a little bit and make the model more consistent.