Machine Learning Engineer Nanodegree

Capstone Project

Brendan Connolly
January 1st, 2019

Definition

Project Overview

This project is centred on soil analysis in Africa. This is a vital area of research that allows forward planning for the best use of land and improved resource management. Definition: “Soil functional properties” are those properties needed to support human activities such as nutrient and water retention, and resistance to soil erosion. Obtaining “soil functional properties” of un-sampled land has previously been too expensive an activity for the poorer countries of interest in this project. New (cheaper) infrared spectroscopy and geo-referencing techniques mean that these key “soil functional properties” could now be inferred via machine learning without having to sample the soil using conventional techniques.

I have a personal interest in this area of research as I am a keen vegetable grower myself. I know the value of good soil and know the difference that soil quality can make to productivity. This project also has humanitarian advantages to it. Allowing poorer countries to make the best use of their land could prevent future famines and give them the chance to achieve a little more prosperity.

Note: This problem and all supporting material have come from kaggle:
https://www.kaggle.com/c/afsis-soil-properties

In terms of data, there are two similar sets of data provided by kaggle for this project. The first is the training set (1158 rows) and the second is the test set (728 rows). Each row in the data set is a sample of soil taken from somewhere on the continent of Africa. The data contains a value for absorption of light at particular discrete wavelengths for each soil sample. There are also some other spatial features given relating to the soil samples that I will go into more detail on in the data analysis section. The training data contains values for the target variables. Unfortunately, the test data provided by the competition organisers does not. This means that the test data provided is useless for the purposes of this exercise. As a result our training/test set will be made up from the 1158 row data set.
In terms of the original problem, the feature set is the inexpensively sourced satellite data. The features we are trying to predict are the “soil functional properties” obtained via the more expensive conventional sampling techniques.

Problem Statement

In this project I will be predicting soil properties based on infrared spectroscopy and spatial measurements. The properties I will be predicting are the SOC (soil organic carbon content), pH, Ca (calcium content), P (phosphorous content) and Sand (sand content). These properties are good indicators of how "useful" the soil is for human activities.

A suitable solution to this problem should take the form of a csv of results. For each soil sample in the test data set, there should be a prediction for each of the five target variables. However, this only really applies to a real world application of these algorithms. For the purposes of this project, I won't be producing a csv and will instead focus on the performance of my algorithms using the evaluation metrics (discussed below). The actual results would be fairly meaningless to anyone correcting this report.

The following tasks will be undertaken to produce a solution:

- First of all I will load in the training data using numpy's csv reading functionality.
- Then I will do some high level analysis of the data to gain a firmer understanding
- Then I will do some pre-processing on the data. As yet, this step is not well defined. Some of the data provided has been scaled and mean centred, however it would seem that the light absorption features have not. The actions taken here will largely depend on the observations in the data exploration in the previous step.
- After that I will perform feature selection on the dataset. This is important since, in our data set, we have roughly triple the number of features as data points.
- I will split the data into a training and a test set
- Now for the actual model building. I will use a combination of two models (taking an average of the results for both as my final predictions).
  The first method I will use is the sklearn "RandomForestRegressor" class.
  The second model I will use will be a Neural Network built using a combination of keras and sklearn.
- I will then evaluate my predictions using the test set based on the evaluation metric discussed below and compare my score to the score of a benchmark model.

Please note that all of the above steps will be more thoroughly discussed and justified in the relevant sections of the main report.

Metrics

An evaluation metric is provided by the organisation that was running the competition on kaggle. Any solution should be evaluated by using the mean column wise root mean squared error (MCRMSE). The formula is as follows:

$$\text{MCRMSE} = \frac{1}{5}\sum_{j=1}^{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{ij} - \hat{y}_{ij})^2}$$

In the equation, y and ŷ are the actual and predicted values respectively.

This is appropriate for an evaluation metric since the values I will be predicting are continuous. The root mean squared error is a simple but effective way of calculating the error between two vectors (actual and predicted). Bigger errors get punished more than smaller errors. Making the formula apply a column wise mean allows the evaluation method to take into account all five of the target variables in the one calculation.

Analysis

Data Exploration

The dataset for this problem contains 1158 rows with each row being a sample of soil from a particular African location. The data contains columns for five of the target variables described in the problem statement. As well as that, each soil sample has a value for absorption of light at particular discrete wavelengths. 3578 wavelength features are present in the data. The wavelengths are for infrared bands of light and vary between 7497.96 cm -1 and 599.76cm -1. Theoretically, absorption at different wavelengths hints at different soil constituents that may or may not be present in that soil sample. There is also an indication in the data as to whether a sample is top or sub soil. Other information is provided such as BSA (Black Sky Albedo) – a measure of how much a soil sample reflects direct sunlight, EVI (Enhanced Vegetation Index) – a measure of "greenness", LST (Land Surface Temperatures) and others. More information can be obtained in the links provided. A single row from the data is provided below. Please note that the vast majority of the features have been left out as they are wavelength absorption measurements and will not be much different in structure to those provided.

PIDN | "09gt9UK5"
m7497.96| "0.210899"
m7496.04| "0.209221"
m7494.11| "0.207645"
.....
m603.617| "1.68849"
m601.688| "1.67928"
m599.76 | "1.67214"
BSAN | "-0.492753624916077"
BSAS | "-0.536363661289215"
BSAV | "-0.582306802272797"
CTI | "0.00638961791992188"
ELEV | "1.17884182929993"
EVI | "0.396501451730728"
LSTD | "-0.00939411297440529"
LSTN | "0.197884991765022"
REF1 | "-0.622545480728149"
REF2 | "-0.548237502574921"
REF3 | "-0.637681186199188"
REF7 | "-0.413306444883347"
RELI | "-0.13013930618763"
TMAP | "0.15401329100132"
TMFI | "-0.0339054465293884"
Depth | "Subsoil"

Gaining meaningful insights from a statistical analysis of this data set may be quite difficult. My relative lack of knowledge in the area of spectroscopy is working against me here. However, there are some things that I can do.

- I can break down our data using the only categorical tag "Depth" to see if there is any heavy emphasis on either top or subsoil in the data. Doing this analysis gives a roughly 50-50 split which is roughly what you would expect and warrants no further comment.
- Another thing that I can do is calculate some very simple statistics for a subset of the light absorption columns to see if there are any that stick out as being abnormal. Any abnormalities could be taken into account when analysing or possibly even pre-processing the data. See results in figure 1.

	m5139.42	m3583.13	m2373.97	m3872.41	m6572.29	m3689.2	m6169.24	m1789.64	m1830.14	m1631.5	m6003.39
count	1157.000000	1157.000000	1157.000000	1157.000000	1157.000000	1157.000000	1157.000000	1157.000000	1157.000000	1157.000000	1157.000000
mean	0.317898	1.282517	0.613348	0.446102	0.261122	1.298581	0.256853	1.184753	1.084870	1.650546	0.256900
std	0.120155	0.288000	0.140811	0.135757	0.113597	0.334517	0.113194	0.120465	0.109553	0.149876	0.113247
min	0.031786	0.464656	0.256785	0.114143	-0.015421	0.364791	-0.013609	0.842285	0.796547	1.132820	-0.012506
25%	0.236548	1.078290	0.509095	0.346271	0.183226	1.064380	0.179799	1.106900	1.017370	1.544340	0.180016
50%	0.320848	1.321930	0.620299	0.463566	0.266741	1.329460	0.262675	1.188940	1.087640	1.664380	0.262053
75%	0.389376	1.510560	0.718264	0.544074	0.328830	1.550480	0.322819	1.261620	1.155060	1.760780	0.322063
max	0.819433	1.886600	0.942427	0.866830	0.761321	2.037170	0.757292	1.589090	1.482850	2.044230	0.756739

Figure 1

By looking at these simple statistics there doesn't seem to be any need to remove outliers. One thing to note is, looking at the means, that there seems to be more absorption at the lower wavelengths. This means that they cannot be treated in the same manner without further processing. The answer here is to normalise the data to some common range and scale.

As part of the competition guidelines, it is suggested that the spectra associated with CO 2 (m2379.76 -> m2352.76) be taken out of the data set in order to obtain a more accurate model. I may do this if I'm struggling to obtain decent results.
Another step that will have to be taken is the changing of the Depth characteristic into a numerical value so it can be run through the regression model builders

All data was obtained from Kaggle: https://www.kaggle.com/c/afsis-soil-properties/data

More in depth and complete description of the data fields:
http://africasoils.net/services/data/remote-sensing/land/

Exploratory Visualization

In this section I plotted the absorption across all of the given wavelengths for a random sub-sample of data points. This is a good visualisation as it will give us an idea which wavelengths we are most likely to get interesting results from. It can give also us an idea of any general trends in the data.

Figure 2 is the result of my efforts.

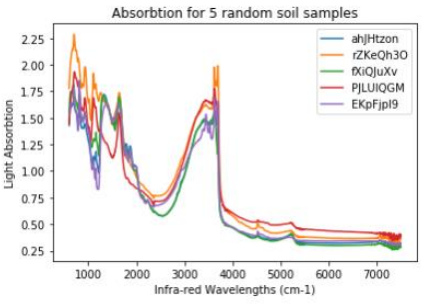

Figure 2

Analysis:

One thing to note is that there seems to be a general trend that each of the samples follow. There are major peaks in absorption at ~1000cm-1 and ~3600cm-1. There is a local trough at around 2300cm-1. The absorption patterns can be quite unpredictable from 700-1200cm-1 while after 2500cm-1, they all follow a fairly rigid pattern. The zones described here could well be good indicators for our target variables so I will keep them in mind as I go through the rest of this exercise.

Algorithms and Techniques

Note: All parameters described below are subject to change upon actual implementation.

Feature Selection

This is a key step since we have roughly triple the number of features as data points. Feature selection is used to reduce the number of features that are used for the training phase. This means that the training process will be more efficient i.e. if we only use the most important features, we should still get good results without having to train on the entire set of features. We also reduce the effect of the curse of dimensionality. The curse of dimensionality arises when you have many features to consider and very few data points. Each new feature adds an extra dimension to the problem and squares the number of points needed to get the same coverage than the lower dimension problem. Lowering the number of dimensions here means that we will be decreasing the sparsity of data points in the problem space considerably.

I will perform feature selection for each target variable separately using a model based approach. By this I mean I will train regression models to predict all of the variables independently. The algorithm I will use computes a "p-value" for each feature with respect to the target variable. That is, how much of the target variables variance can be described by the feature being tested. Only the features with the highest “p-values” will be selected. This approach has the potential to be quite slow but, since our dataset is relatively small, it should be done very quickly.

In terms of variables, the main one we have to worry about here is the number of features we want to take for each target. For now, I am going to take 50 features from the data set for each target variable. I have no other reason than intuition to pick that number, however the same goes for any other number. If results are poor at the learning stage, I may revisit this number as the solution evolves.

Another approach I'm taking into account is principal component analysis (sklearn) to further reduce the number of features to be trained on. Again, this would be performed for each of the target variables independently. Once again I will use this if the results at the learning stage are poor.

Model building

I will use a combination of two regression methods to obtain an output in this exercise as described below.

The first is the sklearn “RandomForestRegressor” class. Since this task involves prediction of continuous valued variables, a method that supports regression is required. Using a random forest regressor rather than a standard regressor has the added benefit of reducing overfitting. Using a single decision tree would have a high chance of finding a relationship that is present in a subset of the data but not in testing. Averaging over a number of decision trees reduces this risk considerably as a majority of trees would have to overfit for the model to overfit as a whole. In order to obtain the best meta parameters for the model I will use 10 fold cross-validation on the training set.

For this model, the main parameters that I'm concerned with are "n_estimators" (the number of decision trees to average over), max_depth (number of splits done before the algorithm stops building the trees) and min_samples_split (Number of samples at which the algorithm will stop trying to split the data). The initial values I will be using for this are:

n_estimators = 50 - This seems like it's enough to get a good average. Since there are less than 1158 rows in our training set it would probably not give any advantage to put this number any higher as the chance of any new tree telling us anything different is quite low
max_depth = None - This means that the algorithm will keep building the trees until it has only min_samples_split points in a leaf node. Normally this would lead to overfitting but since we are using a random forest, that drawback should be balanced out
min_samples_split = 2 - No point in stopping as we have such little data, we don't need the performance boosts

The second model I will use will be a Neural Network built using a combination of keras and sklearn. There is a specific sklearn class for regression neural networks and this is what I will be using. My initial network topology will be wide and relatively shallow. A wide base will hopefully pick out as many of the important patterns in the data as possible. The network topology will quite possibly change as I tune this part of the overall model due to the experimental nature of neural network layout definition. For this reason, I won't go into major detail about the parameters I'm planning on using to start with. I'm going to use the Neural Network for two reasons: (1) Intuition tells me that the patterns in the given data set may be reasonably complicated and well hidden. Since Neural Networks seem to be able to recognise patterns on a number of levels (from very simple to more complicated) it seems like a good choice. (2) We have very little training data here so training time looks like it may not be too much of an issue as long as I keep my network topology reasonably simple. Since training time shouldn't be much of an issue, it makes sense to bring one of the more advanced but intensive algorithms to the table.

My final result will be some kind of weighted average between the predictions of the two models above. The weights I assign to each model will be decided when I see the performance of the models on the training data. This will be documented in the evaluation section.

Benchmark

A benchmark model is provided by the winner of the kaggle completion. It utilises a complex set of methods to make predictions for each of the target variables. The solution methods will be different but the results of the two models will be directly comparable since each model is solving the same problem.
This model could be compared to my own by using the evaluation metrics described in a previous section. The model in question obtained a MCRMSE of 0.46892. This model was put together by an expert in the field so I won’t be expecting to achieve these kinds of results. Thus, should be a good measure of what is possible.
A pdf will be provided in this submission containing a description of the benchmark model. The code for this model can be found at: https://github.com/Yasssser/AFSIS2014

Methodology

Data pre-processing

The pre-processing techniques that I used were normalisation, feature mapping and feature selection.

To address the wide ranging absorption values for the different frequency bands, normalisation was used on the absorption measurements. The other features were pre normalised as part of the

competition data set. In this case I used a simple min/max normalisation technique. Take the minimum value away from each data point and then divide by the maximum value. This ensures that all data lies in the range of 0 to 1. When this technique is applied, each of the absorption wavelengths will get given equal treatment in any kind of regression algorithm i.e. if a certain wavelength has particularly high absorption level naturally, it won't skew any line of best fit.
Normalisation was performed using scikit-learn's built-in sklearn.preprocessing.MinMaxScaler. The documentation for this can be found at: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Feature mapping was used to change the depth into a numerical feature rather than a categorical feature. This is performed by changing all instances of "Subsoil" into 0's and all instances of "Topsoil" into 1's. This feature can then be used in any regression algorithm that I choose to use. I implemented this change using my own code.

I performed the feature selection process independently for each of the five target variables. The selection step was performed with the sklearn objects sklearn.feature_selection.SelectKBest and sklearn.feature_selection.f_regression.
The code for this section takes the data and returns a dictionary of target variables to lists of the most important features for those targets. This dictionary is then used when the data is being split into training and test to only pick out the features that are considered important for that target. The data needs no manipulation for input to this step as sklearn just requires a pandas data frame for a feature set and a numpy array for a set of targets to train against. This is how the data is naturally throughout the solution.
I wrote the code in such a way that the number of features selected is a parameter for the implementation stage and can change during the learning process. This allows for a more flexible approach.

Before moving onto the algorithm implementation phase, I split the data using scikit learn's train_test_split functionality. I used a training to test ratio of 4:1. In keeping 80% of the available data for training I'm reducing the effect of having such little data as much as possible. This introduces the risk that the test data is not representative but I think this risk is worth taking.

Implementation

I trained two algorithms to compile my final model. These algorithms were a random forrest regressor and a neural net regeressor. The random forrest regressor used was provided by the scikit learn suite, the documentation can be found at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
For my neural net I used tensorflow. The KerasRegressor wrapper provided for integration with scikit learn was also used so that the neural net would be easily comparable with the random forrest regressor. I used a lot of keras functionality to build the neural net and the documentation can be found at: https://keras.io/

Once again, I trained a different model for each of the target variables. This presented a challenge for how the data should be structured. Throughout my solution, I decided to store the items in dictionaries with an entry for each target variable as this allowed for the cleanest and most readable solution.

For my initial try, I decided to use 50 features for each target variable. That's just over 1% of the total number of features which I think represents a good basis. The features were selected using the feature selection process outlined above.

The metric I used to score the trained models was mean squared error. This is a simple scoring metric that is suitable for regression models. A line of best fit is composed using the training set. Points then have their distance measured against this line. Points that are further away from the line raise the error disproportionately to points that are closer to the line. Essentially, outliers are punished heavily with this system. I used 10 fold cross validation to get a well balanced estimate for the error. This means that I took the training data and split it up into 10 sections. The algorithm then loops over each of the 10 sections using it for testing at each iteration. The other 9 sections are used for training at each iteration.
I used scikit learn functionality to achieve these ends. The particular object used was: sklearn.model_selection.cross_val_score, documentation can be found at https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

I have detailed my initial parameters for the random forest regressor above so I will not reiterate them. The implementation wasn't difficult and threw up no major hurdles. I simply had to create an instance of the regressor, get my training data, call the sklearn cross validation function and save a mean of the resulting validation errors for examination at the refinement stage.

The neural net was more challenging, to make that work I had to interface with both keras and scikit learn. Overcoming this challenge meant a lot of trial and error as well as researching the relevant documentation for both utilities.
Another challenge that I faced was that the input layer dimensions of the neural net had to be the same as the number of features that I selected to train on at the previous step. This meant that I had to re-architect the solution slightly to have the number of selected features available at the time of training rather than as a hard coded parameter.
Based on training time, I chose the number of training epochs for the neural net to be 40. There wasn't much to go on initially so going for a number with a reasonably quick training time and a bit of time for the network to evolve seemed like a good way to go.

Refinement

I decided to refine the models independently and then combine the refined models at the end for the final model. I used mean squared error as my evaluation metric via 10 fold cross validation. That gave me an error for each of the five target variables. I decided to take a mean of the five errors to judge each model. The initial results were as follows:
Random Forrest: -0.473998488812
Neural Net: -0.494581291742
Without diving into the figures too much, suffice to say that if we can reduce these two numbers, this will represent an improvement in our methods.

One change I made applied to both models. I changed the number of selected features from 50 to 100. While this increased training time and makes the data more sparse (see section on curse of dimensionality), it reduces the risk that we are missing out on important features due to our feature selection process.

For the Random Forrest model, I changed two parameters to obtain a better result. The max depth of each of the estimators was changed from being theoretically infinite to being capped at 20. The minimum number of samples that can be split on was also increased from 2 (default) to 10. Both of these measures provide a better guard against overfitting. They effectively curtail the training process before each estimator has a chance to make a rule for each individual data point supplied. Following the same line of reasoning, these revisions will decrease training time as the trees created will be smaller.

For the neural net model, I changed the architecture of the network that I was using. As a direct result of increasing the number of features trained on, I had to widen the network to take 100 features as an input rather than 50. I also made the network deeper by adding an extra hidden layer with 50 nodes present. This was done in the hope that the network would pick up more complex patterns in the data. To reduce overfitting I added two dropout layers with a dropout of 30%. These layers "turned off" 30% of the nodes in the next layer on each epoch. This means that the network doesn't get to rely on any one feature too much to make decisions.
When this new architecture was tested using a new round of 10 fold cross validation, the mean squared error actually rose above 0.5. Clearly I needed to revisit some decisions.

In the second round of refinement I changed the activation functions for each of the layers to be a hyperbolic tan rather than relu activation. I also revised the number of nodes in the second to last layer downwards to 12 from 25. As is sometimes the case with neural nets, I cannot claim any deep insights that led to the changes specified. I tried the above settings out and it provided a better result.

After the refinement stage, the final mean squared error results (broken down by model once again) were:
Random Forrest: -0.432118542088
Neural Net: -0.478087080456

Again, without diving into the numbers too closely, we can see that the mean squared error has reduced using our refined methods. Due to the complexity of the problem and the lack of training data, it is hard to imagine a much better result without a drastic re-write.

Results

Model Evaluation and Validation

For final model evaluation, I will compute the MCRMSE as defined above using the models and parameters discovered at the methodology stage.

The refined models were taken and used to predict the values for each of the five target variables in this problem. In order for this to be done, the two models need to have their predictions combined in some way. For this purpose I looked at how well the models predicted each of the variables in the cross validation stage. It turns out that the random forest was the most accurate on 4 out of 5 variables. The neural net only came out on top for phosphorous prediction. I computed the ratio of the inverses of the errors and used these ratios to weight the predictions of the models. The effect being that the random forest model had its prediction weighted higher for all targets except phosphorous where the neural net was weighted higher.

At this stage the predictions could be computed and compared to the actual values for the test set. I implemented my own version of the MCRMSE metric using in-built numpy functions. My final model achieved a MCRMSE of 0.76947. This value will be put into context in the next section.

All that remains is to examine how robust the model I created is. For this purpose I saved the results of the cross validation from the training phase. Cross validation trains the models on different sub-sections of the training data and tests on the remaining. For a robust model, we would expect generally similar results for each cross validation run. As usual, I have done this independently for each model and each target variable. (Look for the robustness evaluation in the accompanying jupyter notebook.)
For the random forest model, the model appears to be fairly robust for each of the target variables except for phosphorous. They all seem to hover around the same value with one or two outliers pushing the standard deviation higher. Phosphorous is a different case and the model looks to be fairly inconsistent when predicting those values. The mean squared error gets very high for some validation runs. For the scope of this project, it doesn't merit a complete re-write but I will discuss this issue in the "improvements" section.
The neural net model gives more or less the same results. Once again, phosphorous proves the least stable target for our model with the results varying wildly and a large standard deviation. The same conclusions apply here as for the random forest.

Justification

My results compare relatively poorly to the benchmark model that I chose in the beginning. The winner of the competition got an MCRMSE of 0.46892. There are two obvious reasons for this difference in results.

First, it can be assumed that the winner of the competition was an experienced data scientist with many years of machine learning behind them. I am just starting out in my machine learning career and as such must be given some sort of benefit of the doubt.

Second, the training set I had to use was 80% the size of what the eventual winner was able to use. The test set was roughly a third of the size. This is of particular relevance because the data set is so small to begin with. Both the testing and training would be affected by this lack of data so the problem would run right through this solution.

A more positive justification can be obtained by seeing how I would have placed in the kaggle competition had I have entered. It turns out I would have come 1089th out of 1233. While this may not sound very positive, I take it as such. It means I beat 144 other competitors. Considering this was my first proper project, it seems like this is a good attempt and the model I built is reasonable at the worst.

Additionally, one of the benchmarks added to the competitors list was an all zeros benchmark. That is, predict all 0's for all values. This comes in 1150th place with an MCRMSE of 0.91393. My score of 0.77425 beats this benchmark comfortably so the model seems to be solid if unspectacular.

Conclusion

Free-Form Visualization

Figure three represents mean light absorption across all samples at each of the frequencies (light blue undulating line). The absorption values shown are from the pre-normalized stage. The vertical lines represent the frequencies that our feature selector picked out as being important for the given target variable.

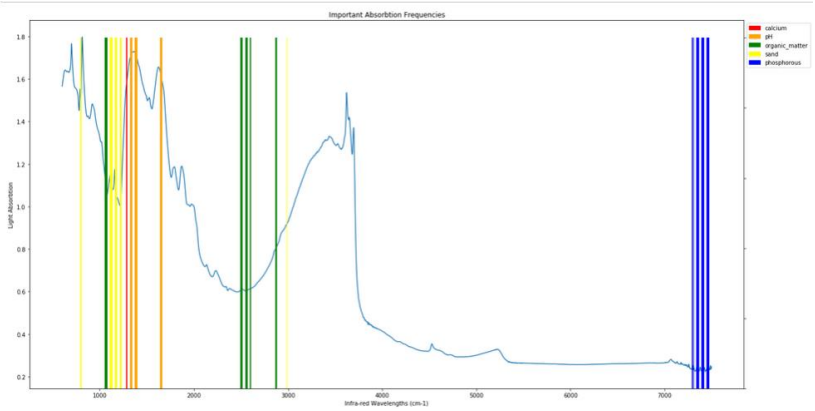

Figure 3

In the plot we see that for some of the targets, the areas of interest are fairly well defined. The best example of this is phosphorous which seems have its presence indicated by the higher frequencies of light. An interesting insight that could be gained here is that our neural net model performed best in testing (out-performed the random forest) for predicting phosphorous levels. Perhaps a case could be made that neural nets perform better when the results are better organised when analysing spectral data. Strangely, both models give inconsistent results at cross validation stage for Phosphorous even though it seems the most well behaved in figure 3. Organic matter, sand and pH all have their own "areas" in the chart however they are also more spread out than phosphorous. This increases my confidence that the feature selection is picking up worthwhile features in the data. These targets are more complicated than phosphorous and calcium (discussed below) and are often made up of more than just a single element. Therefore it makes sense to me that they have more than one important frequency.

It can be seen that there are large areas of frequencies that don't matter at all according to the feature selector. The long and relatively straight area between 4000 and 7000 cm-1 is a good example. This makes sense to me as the data looks like it lacks any defining features at those frequencies when compared to the relatively volatile area between 500 and 2000 cm-1

Another interesting point is that calcium only appears to have one frequency (or a very small band) that matters. When looking at the features selected, you can see that the non-spectral features are selected as being mostly important for calcium level detection. Based on nothing more than intuition, I would have expected pH to be more linked to the non-spectral features as it is more of a complex human abstraction than simple elements like Calcium and Phosphorous.

Reflection

To summarize, I first analysed the data to see what abnormalities I would need to iron out at the data preparation stage. The main realisations were that I would need to normalise the data and perform feature selection to reduce the feature space.
I then performed the said data pre-processing steps. As part of this process I also split the data into training and test sets.
After that, I constructed the two models that would make up my overall solution. These were a random forest regressor and a neural net regressor.

I refined each model independently using results from 10 fold cross validation of the initial models. I also figured out how I would combine the two models for the final output again using the results from the 10 fold cross validation on each of the target variables.
I then took my final model and evaluated it using the MCRMSE metric against the held out test data. I then compared my results to those of my chosen benchmark model and analysed the differences in results.

The main interesting thing about this project was the subject matter. Due to my personal interest in the area of agriculture, identification of new ways to profile soil is a very interesting area to work in. Knowing that I am working with actual data collected by satellites for a real-world problem makes the whole process more approachable.
The other thing I found interesting was working with scikit learn on my own project. Everything I have done so far with scikit has been prescribed by the course so has been in a very sheltered environment. Having to figure everything out from start to finish makes the learning process far more efficient and thus enjoyable. When writing code, I always find it interesting to find out how other people have solved problems and then incorporate the best solution with my requirements.

The most difficult part of the project was refining the neural net. A major problem was the time taken to train the model even after making only the slightest change. The long breaks in concentration and work are hard to deal with as it can take me a long time to get back to full focus levels when a trail of thoughts is broken. I didn't realise the difference that training the neural net on the GPU, as was done in the main deep learning module, makes to the process! The other aspect is that training neural nets is such an experimental process that there isn't a huge amount of supporting material. As a coder making changes and taking a "we'll see how it goes" attitude is a stark contrast with the way I usually solve problems.

The final model that I came to was broadly in line with what I expected at the outset. I knew that the solution would have to be some kind of regression model and I knew that the result would be a single number (MCRMSE). I also expected the results to be poor in comparison to the competition winner due to the complexities in the domain being analysed.
As far as I can tell, with a few code changes and a more generalised loader, my solution would be useable in real-world settings. It could be used to make predictions about soil contents. I don't think I would use it though. As can be seen in the competition leader board, there are many better solutions out there and they would be better used in a real world scenario.

Improvement

One thing that is worth noting is that it would quite possibly be easier to produce a better model with some more training data.

In terms of my implementation, the main improvement I would suggest is to remove the neural net and (possibly) replace it with something else. The main bulk of my time when implementing the project was spent trying to figure out how to improve the performance of the neural net. That's not to mention the training time. It didn't seem to be that well suited to the job either, only excelling for one of the five target variables. For that one variable, the results were unstable so it's hard to see the added value.

Perhaps a more radical approach and one followed by my benchmark model is to train a model specifically for each target variable. This means more time spent implementing and training but

ultimately could yield better results. Trying to apply a single machine learning algorithm to all of the target variables may have been where my solution fell down in terms of its performance. In particular, it seems like the models that I trained above were pretty ill-suited for predicting the phosphorous levels in the soil samples. The model gives big differences in results with minor changes to training data. The prediction of phosphorous levels could be an area that would be vastly improved if a model was trained per target rather than training a single model for all.