PHASE 2

Prof. Nathan Karst

Nishtha Gupta & Jacob Nyamu

Introduction:

In this paper, we build upon our findings from the Phase 1 paper. First, we construct two separate models i.e. KNN and a Classification Tree to predict using our dataset. Thereafter, we implement different controlling strategies that prevent our models from being overfit/too complex. Lastly, we evaluate their performances, review the pros and cons of each model and then choose the most appropriate one for our dataset.

KNN Model Construction

After loading our dataset into R-Studio, we managed the dataset. Since KNN models predict based on distance of variables, the inputs have to be numerical e.g. elevation, aspect, slope. Therefore, we dropped all other categorical variables e.g. ID, Wilderness_Area, Soil_Type, except the Hillshade and Cover_Type, which is our target (what we're trying to predict).

```
#MANAGE
df$Id= NULL
df$Hillshade_9am = as.numeric(df$Hillshade_9am)
df$Hillshade_Noon = as.numeric(df$Hillshade_Noo
df$Hillshade_3pm = as.numeric(df$Hillshade_3pm)
df$Wilderness_Area1 = NULL
df$Wilderness_Area2 = NULL
df$Wilderness_Area3 = NULL
df$Wilderness_Area4 = NULL
df$Soil_Type1 = NULL
df$Soil_Type2 = NULL
df$Soil_Type3 = NULL
df$Soil_Type4 = NULL
df$Soil_Type5 = NULL
df$Soil_Type6 = NULL
```

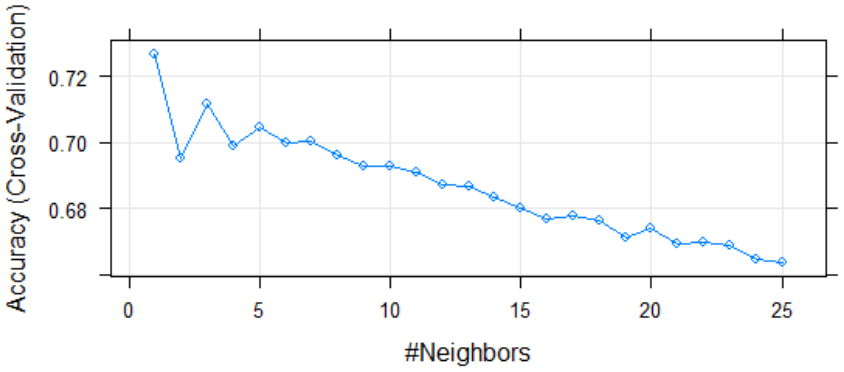
KNN Model Interpretation

After standardizing the remaining numerical variables, setting seed to 1234, and partitioning our model on a 60-40 training-test split, we created our model using K=1 based on knnCrossVal() which predicts best possible value (where K is the distance between the numerical variables). After running the models, we have the following matrix;

	observations						
predictions	1	2	3	4	5	6	7
1	386	124	3	0	7	5	46
2	130	361	19	0	27	14	7
3	2	29	382	31	10	108	0
4	0	1	66	573	0	43	0
5	32	88	12	0	597	11	0
6	2	29	145	33	13	473	0
7	87	16	0	0	0	0	624

From it, we can infer that our model predicted 386 times when Forest Cover type 1 was and was actually the correct Forest Cover Type in the observations split of the testing dataset. On the flip side, we can infer that the number of times our model predicted forest cover type 7 as it was in the test split, was 624.

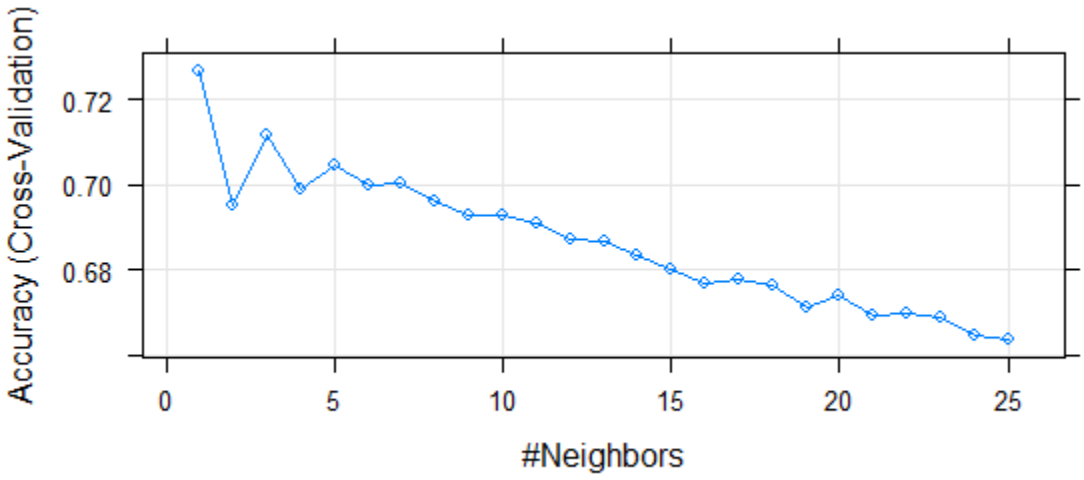
Additionally, the result shows that we could choose K around 1 , which we'll get the highest cross-validation accuracy.



KNN Performance Evaluation

After running our model, we calculated the accuracy rate - how often our model predicted the cover type correctly. The KNN model predicted a 75% accuracy rate and a 25% error rate. To further evaluate this, we used a benchmark error rate - the rate at which one would blindly predict the cover type without the help of a model - which came up to 86%.

values	
accuracy_rate	0.748677248677249
error_bench	0.861772486772487
error_rate	0.251322751322751



- Training accuracy rises as model complexity increases
- Testing accuracy penalizes models that are too complex or not complex enough
- For KNN models, complexity is determined by the value of K (lower value = more complex)

K=1 provides the best accuracy score. One reason for this may be that the data are fairly dense, at least for the non-dummy variable features. This makes it more likely that a close training data point can be found with a small k value.

CART Model Construction

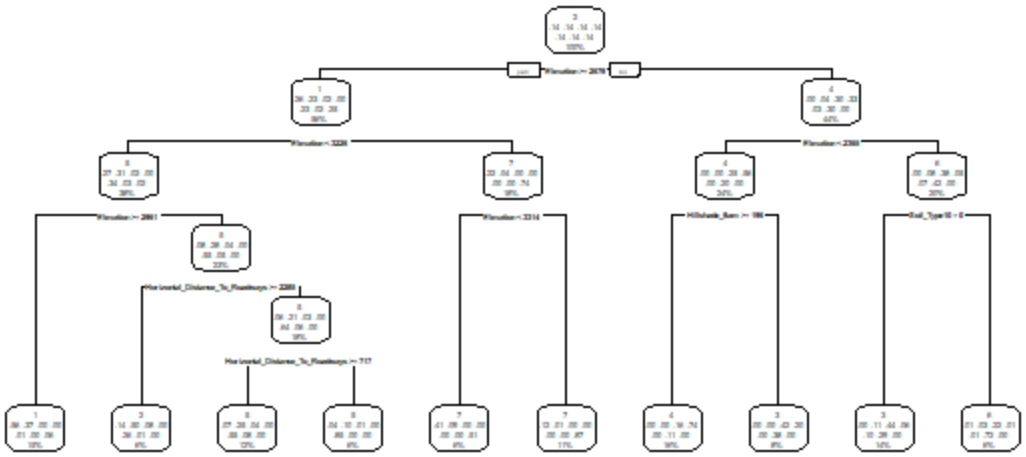
After loading our dataset into R-Studio, we managed the dataset. Since Classification models predict based on categorical and numerical variables, the inputs have to be numerical and categorical e.g. hillshade, elevation, aspect, slope. Here, the target is to predict Forest Cover Type .

```
#MANAGE
df$Id= NULL
df$Hillshade_9am = as.numeric(df$Hillshade_9am)
df$Hillshade_Noon = as.numeric(df$Hillshade_Noon)
df$Hillshade_3pm = as.numeric(df$Hillshade_3pm)
df$Wilderness_Area1 = as.factor(df$Wilderness_Area1)
df$Wilderness_Area2 = as.factor(df$Wilderness_Area2)
df$Wilderness_Area3 = as.factor(df$Wilderness_Area3)
df$Wilderness_Area4 = as.factor(df$Wilderness_Area4)
df$Soil_Type1 = as.factor(df$Soil_Type1)
df$Soil_Type2 = as.factor(df$Soil_Type2)
df$Soil_Type3 = as.factor(df$Soil_Type3)
df$Soil_Type4 = as.factor(df$Soil_Type4)
df$Soil_Type5 = as.factor(df$Soil_Type5)
df$Soil_Type6 = as.factor(df$Soil_Type6)
df$Soil_Type7 = as.factor(df$Soil_Type7)
df$Soil_Type8 = as.factor(df$Soil_Type8)
df$Soil_Type9 = as.factor(df$Soil_Type9)
```

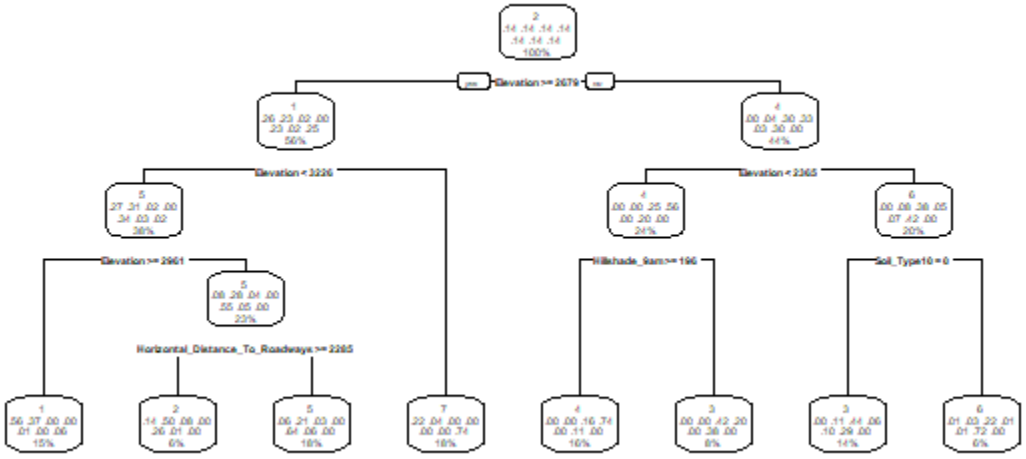
CART Model Interpretation

After standardizing the remaining numerical variables and converting the categorical variables into factor, setting seed to 1234, and partitioning our model on a 60-40 training-test split, we created our model After running the models, we create tree both before and after pruning/cut the tree with optimal value of parameter CP to remove unnecessary nodes.

Tree Plot Before Pruning



Tree Plot After Pruning



predictions \ observations	1	2	3	4	5	6	7
1	549	197	0	0	5	1	58
2	204	507	8	0	55	13	3
3	0	19	587	52	14	157	1
4	0	0	48	798	0	39	0
5	13	86	32	0	771	14	1
6	5	31	187	17	13	652	0
7	81	11	0	0	0	0	819

From it, we can infer that our model predicted 549 times when Forest Cover type 1 was and was actually the correct Forest Cover Type in the observations split of the testing dataset. On the flip side, we can infer that the number of times our model predicted forest cover type 7 as it was in the test split, was 819.

CART Performance Evaluation

After running our model, we calculated the accuracy rate - how often our model predicted the cover type correctly. The CART model predicted a 77% accuracy rate and a 22% error rate. To further evaluate this, we used a benchmark error rate - the rate at which one would blindly predict the cover type without the help of a model - which came up to 86%.

accuracy_rate	0.770337301587302
accuracy_rate_pr…	0.774305555555556
error_bench	0.859292328042328
error_rate	0.229662698412698
error_rate_pruned	0.225694444444444

The accuracy of the model on the test data is better when the tree is pruned, which means that the pruned decision tree model generalizes well and is more suited. However, there are also other factors that can influence decision tree model creation, such as building a tree on an unbalanced class. These factors were not accounted for in this demonstration but it's very important for them to be examined during a live model formulation.

Pros and Cons of the KNN and CART Model

KNN Model

Pros-

1. KNN is easy to implement as it takes two parameters i.e k and distance function.
2. Since the KNN algorithm requires no training before making predictions, new data can be added seamlessly which will not impact the accuracy of the algorithm.

Cons-

1. We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.

CART Model

Pros -

1. Trees can be displayed graphically and can be easily interpreted by non-experts.
2. Decision trees can easily handle qualitative (categorical) features without the need to create dummy variables.

Cons-

1. As the tree grows in size, it becomes prone to overfitting and requires pruning.
2. A small change in the data set can make tree structure unstable and cause variance.

Which model is better?

The KNN model brought down the error rate by 75% from the benchmark error as compared to the CART model accounting for an error rate by 77% from the benchmark error. Thus, it is observed that CART model is more accurate as compared to KNN model to predict the forest cover type on the test data available.