Forest Cover Type Classification Report

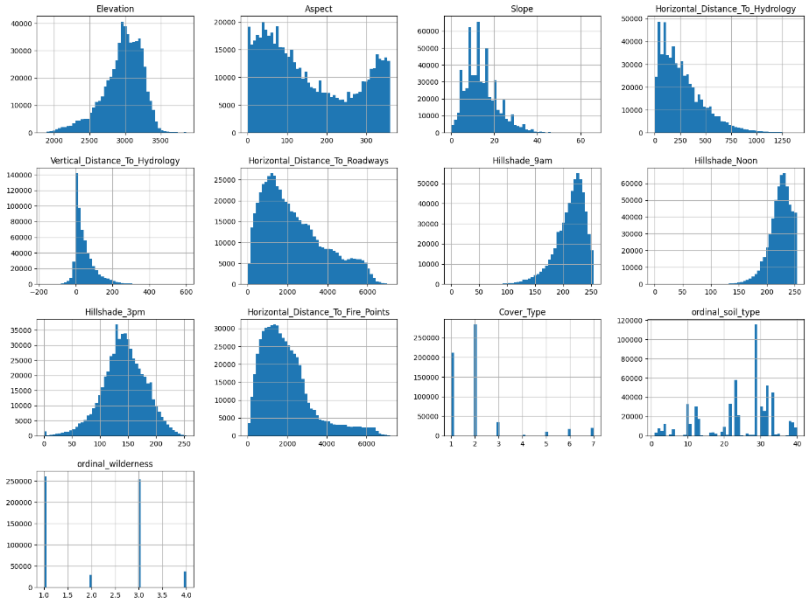
Data preprocessing

- Converting one hot encoded data into ordinal encoded

Due to the large amount of soil types (40 types), visualizing data and understanding it would be hard. So, two columns were added to the data set ('ordinal_soil_type' and 'ordinal_wilderness')

- Visualizing Data to understand data distribution

Comments:



- Most numerical data does have a near normal distribution, except for Aspect feature. (Meaning having a defined structure)
- In addition, most distributions are skewed to one side more than the other
- Data distribution is categorical is unbalanced, having a class represented more than the other classes. -> they can be used in stratified splitting for better evaluation, or under sampling and oversampling can be used, but might ruin data due to having many classes.

- Using the stratified split sets instead of the randomly split sets

Training and test sets were split based on ordinal_soil_type column because the difference in representation in soil types in the data is high, which might cause the model to train more on certain types than others, and even might not train on some types if the data was split randomly.

- Correlations and feature engineering

Comments:

- Elevation
  - Elevation and ordinal_soil_type has a high correlation, but it also might not be indicative of causation since ordinal representation isn't for ranking or level.
  - Elevation and Slope has a good correlation
  - Elevation and Horizontal_Distance_To_Hyrdology, and Elevation and Horizontal_Distance_To_Roadways also has a good correlation
  - Aspect has generally low correlations with most features, but high correlations with HillShade (Might drop Hillshade features)

- Slope
  - Slope has a good correlation also with Vertical_Distance_To_Hyrdology
  - Slope has good correlations with Hillshade feature (which supports more the dropping of the features)

- Horizontal and Vertical distance to hydrology
  - Horizontal and Vertical distance to hydrology also have a high correlations (Might multiply them and add them as a new feature)
  - Otherwise both features don't have high correlations with other feautures

- Horizontal distance to Roadways
  - Horizontal_Distance_To_Roadways doesn't have significant correlations with other features except Horizontal_Distance_To_Fire_Points, which can be inferred from elevation.

- Hillshade Features
  - Hillshade features don't have that great correlation, but logically the amount of sun/shade an area receives does affect cover type (So features can be multiplied or added to see their effects)

- Horizontal distance to Fire Points
  - Horizontal_Distance_To_Fire_Points doesn't have alot of especially with Cover Type, but with wilderness it does. From my knowledge in Environmental Science, I know that an area with alot of wildfires does lead to difference in soil and cover. So, I think it should be added as feature and is more influential than Horizontal_Distance_To_Roadways.

Feature Engineering:

Experimental training set was created in order to drop columns that I think are unnecessary and added the following features:

- Distance to Hydrology: Took the average between Horizontal_Distance_To_Hydrology and Vertical_Distance_To_Hydrology, and dropped the columns.

- Hillshade_Total: Added up Hillshade_9am, Hillshade_Noon and Hillshade_3pm , and dropped the columns.

- Aspect and Horizontal_Distance_To_Roadways were dropped because their correlation with Cover_Type are low. And they have high correlation with other features that were not dropped, so keeping them would be redundant

- Ordinal encoded columns were also dropped since the one hot encoded columns, although they add complexity, but are better suited for the data set since the categorical data types do not represent level, rank or something where order is important. So, keeping using the ordinal data instead of the one hot encoded but cause the model to misinterpret data.

- Scaling Data

Standardization was used was the scaler for all columns because most columns don't have an exactly normal distribution, so standardization is a better choice across the board.

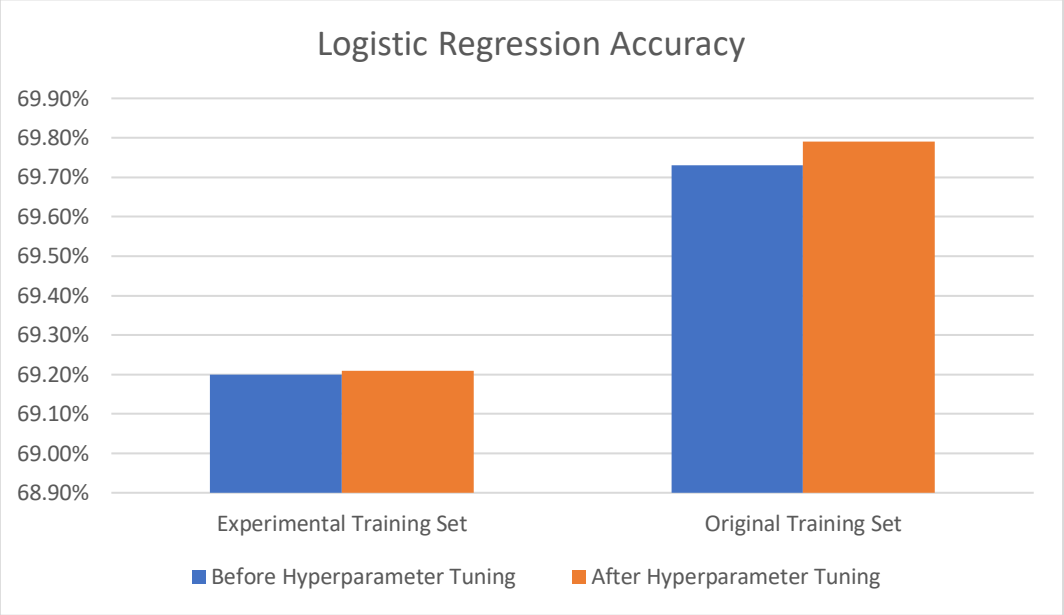
Comparing different models' accuracies:

The system is accuracy based, since false positives and false negatives have equal costs, so we focus more on the overall accuracy of getting the classes right.

So, in the report accuracy is going to be compared, whilst other metrics like confusion matrix, recall, precision and f1 score are included in the .ipynb file.

For each algorithm, two models were trained, one of the experimental training set, and on the original training set.

- Logistic Regression

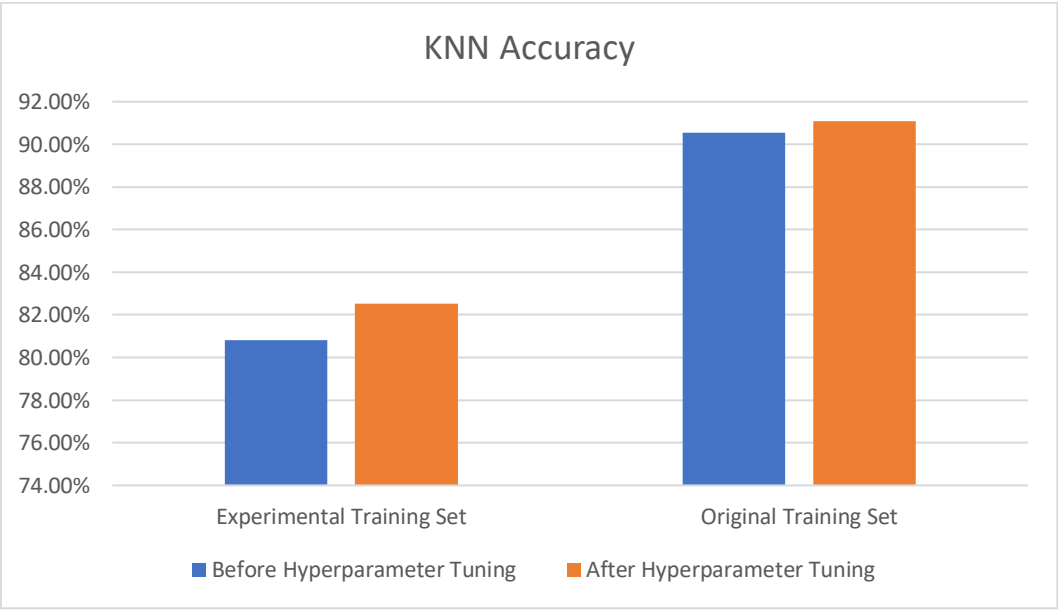


The best model was for logistic regression was after hyperparameter tuning on the original training set, which had an accuracy of 69.65% on the test set.

Generally, accuracy improved after hyperparameter tuning but very slight to even be considered.

The model performed better on the original training set, which shows that the featured columns didn't aid in improving the accuracy of the algorithm.

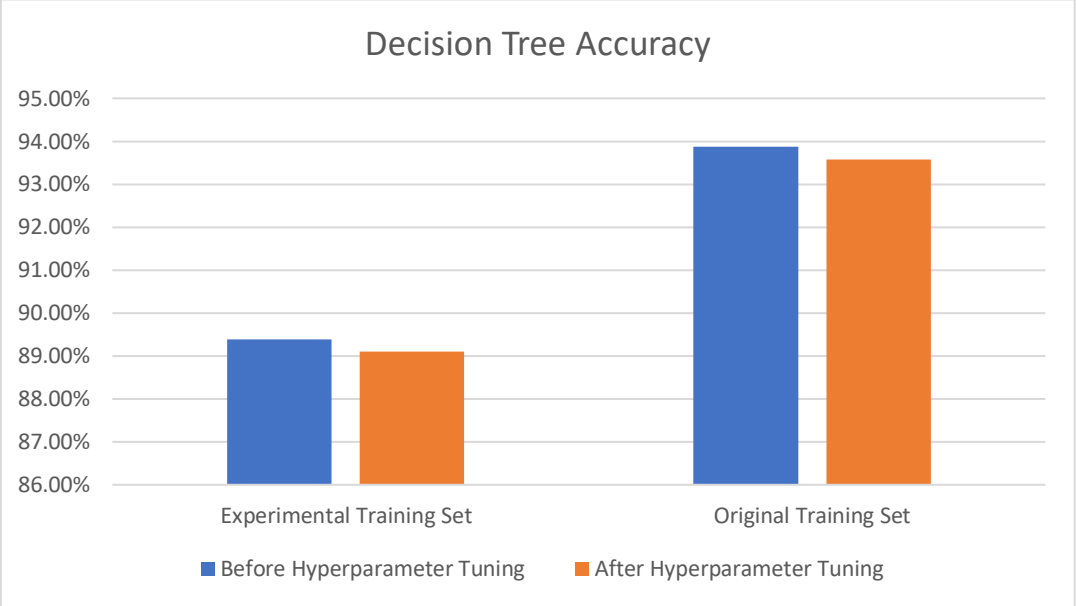
- KNN



Again, the model performed much better on the original training set.

In both, the accuracy improved after hyperparameter tuning, but also slightly.

- Decision Tree



The model also performed better on the original training set.

But in both sets, accuracy actually decreased using hyperparameter tuning, which shows that the ranges given were not the best for the parameters, and that the default parameters are better in our case.

Due to time limitation, and that we have already achieved the objective, I will go on to train to the original data set, and will not reperform hyperparameter tuning on this training set.

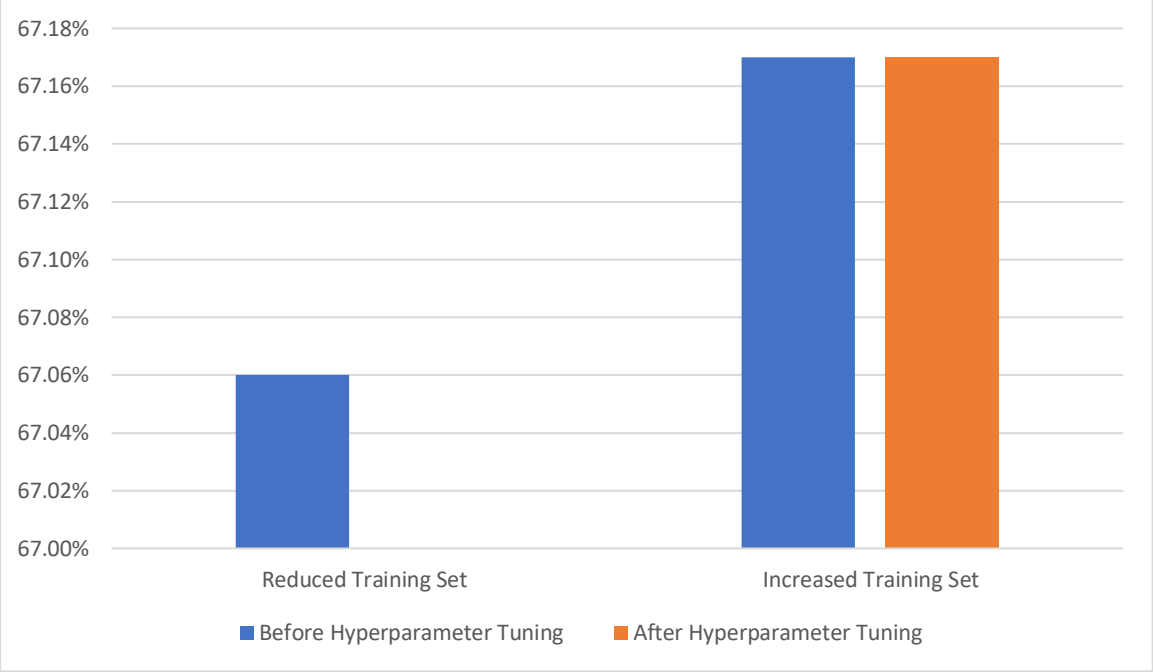
- SVM

The methodology that was followed in the above models didn't work in the case of the SVM model due to the large number of columns and rows (high complexity). So, SVM is not the best model for our data.

So, ordinal columns were used instead of the one hot encoded columns, which were dropped. This decreased complexity of the columns.

In addition, linear kernel (LinearSVC) was used instead of the normal SVC, as it is faster.

At the beginning, the model was training on a very small training set (51% of the original training set), then after seeing the bad accuracy, which was likely due underfitting because of the small training set size, another model was trained on an increased training set (75% of the original training set).



Even after increasing the training set size, the model's accuracy did not change significantly at all.

Generally, the model performs badly mostly due to using a linear kernel only, usually svm performs badly on large datasets, hence it is not the best model for our case.

It also might be performing badly due to the ordinal encoding that might indicate importance of one type over the other.

Final model with best accuracy:

Out of all the models, the decision tree model was the best model to perform on both the experimental training set and the original training set.

It did perform better on the original data set, giving an accuracy of 93.87% on the test set, which is much greater than the objective.

The default parameters of the model performed the best, as after hyperparameter tuning, the model's accuracy decreased.

To ensure that the model does perform well on different training and test sets, cross validation was used, in which the model gave the following scores:

```
array([0.93187539, 0.93060605, 0.93113315, 0.93193993, 0.93146588])
```

Showing that the model does perform well on unseen data in different situation and that it is stable overall.