Dishant D. Bhatt and Christian Haus

Comparing Classification Models on the Forest Cover Type Dataset

***Abstract* — A big question in machine learning is what kind of model should be used with your dataset. We were very interested in this, and decided to test out four different kinds of models (Neural Net, Decision Tree, Logistic Regression, and Bagging Classifier) on a dataset of 581,000 about classifying trees. We found while the BaggingClassifier was overall the most accurate, our favorite was the Decision Tree because of its impressive speed and high accuracy (nearly on par with the BaggingClassifier).**

**The way we found these results were through a series of tests on the data set and comparisons between different model’s performance metrics. The main metrics looked at were time, precision, recall, F1 score, and R2 score.**

1. Introduction

For our project, we decided to use the forest cover type dataset found on Kaggle. Our goal was to compare four different kinds of models (neural net, decision tree, logistic regression, and bagging classifier) and compare them to see which models would work best with our data.

1. Dataset

Our dataset was about forest cover type (tree type) based off features ranging from soil types found in the area, to sunshine, distance to water and more. The dataset, which was taken from Kaggle, is decently sized with about 581,000 different measurements and 54 different features.

1. Major Challenges

Due to the size of our data set, one of the biggest challenges we faced was model run time. For example, KNN took around 30 mins to fit the data and after it got stuck when making the predictions. SVM never ended up fitting the data. Even logistic regression standardized with StandardScalar, took over 900 seconds. This posed a problem when running multiple tests in order to compare results and see how the models performed using different parameters. Due to this we had to research which models are most efficient with large data sets. This led to not only quicker model evaluations, but much more accurate results.

1. Experiments

**Neural Net**

The first model we tested was the neural net. We used the sklearn MLPClassifier (multi-layer perception classifier), and adjusted different hyper-parameters to see what would work best. We expected this to work fairly well as we felt the large data set would provide ample training data. First off, we know that we should standardize data when using neural nets, as otherwise certain features will have much more effect on the function, therefore making it take a longer time to converge.

In order to test this out, we ran 3 different tests using the default MLPClassifier with different standardized data: (1) all of the data standardized, (2) none of it standardized, and (3) all but the bools (0 or 1 values) standardized. Here were the results:

Time (s) f1-score

1) 123.4 0.86

2) 193.8 0.75

3) 223.7 0.87

You can clearly see that standardizing the data does have a marked impact on the processing time. Next, we look at the score. The f1-score is a measure of the tests accuracy, and takes into account both the precision and the recall. Now what’s interesting to see is that the non-boolean standardized model had a slightly better score than the all standardized model. However, since they were so close, and standardizing the data took about half the time, that is what we will use for future tests as we care about both performance and time.

Next, we need to see if we can increase this 0.86 score by varying the hyper parameters of our neural net. We have decided to mess with the solver type, activation function, and hidden layers to see what we can do. First, we attempt different solvers: we start with (1) ‘adam’ (default), then have (2) ‘sgd’, (3) ‘lbfgs’. Here are the results:

Time (s) f1-score

1) 123.4 0.86

2) 326.5 0.85

3) 316.7 0.82

It is pretty clear that the ‘adam’ solver, which is a stochastic gradient-based optimizer, works best in terms of both time and score. This makes sense, as ‘adam' is the recommended solver for problems with large sets of data, and will be our choice moving forward.

After choosing our solver, we then wanted to find the best activation function for the hidden layer. The activation function determines what nodes will output based on their input. We tested out (1) ‘relu’ (default), (2) ‘identity’, (3) ‘logistic’, and (4) ‘tanh’.

Time (s) f1-score

1) 224.9 0.86

2) 36.3 0.71

3) 482.7 0.89

4) 276.1 0.88

The quickest function by far was the ‘identity’ function, which makes sense, as this function merely returns f(x) = x, taking no computation. However, the identity function also has the lowest score of them all, and the best choice for our problem (trying to optimize both speed and score as much as possible) seems to be the ‘tanh’ function. This function, which returns f(x) = tanh(x), has the second highest score, and a more reasonable time compared to the logistic function which only beat its score by 0.01.

Finally, one of if not the most important part of our neural network to figure out is the hidden layers. We need to decide how many layers we want, and how wide these layers should be. We tested out one, two, and three layers each with 100 or 500 nodes each:

1 layer:

Time (s) f1-score

100) 123.5 0.86

500) 593.4 0.91

2 layers:

Time (s) f1-score

100) 361.5 0.92

500) 2787.1 0.94

3 layers:

Time (s) f1-score

100) 417.3 0.95

500) 13730.2 0.96

By increasing both the depth and width of these layers, we were able to improve the score quite a bit. However, by increasing the width of these layers too much, as seen with the 500 node examples (especially the 3 layer one), the time can go up dramatically. This leads us to believe that keeping a smaller width, while going a bit deeper, is definitely the better option here. Therefore, we decided to run a couple more tests based of this where we decreased the number of nodes, but increased the depth:

4 layers:

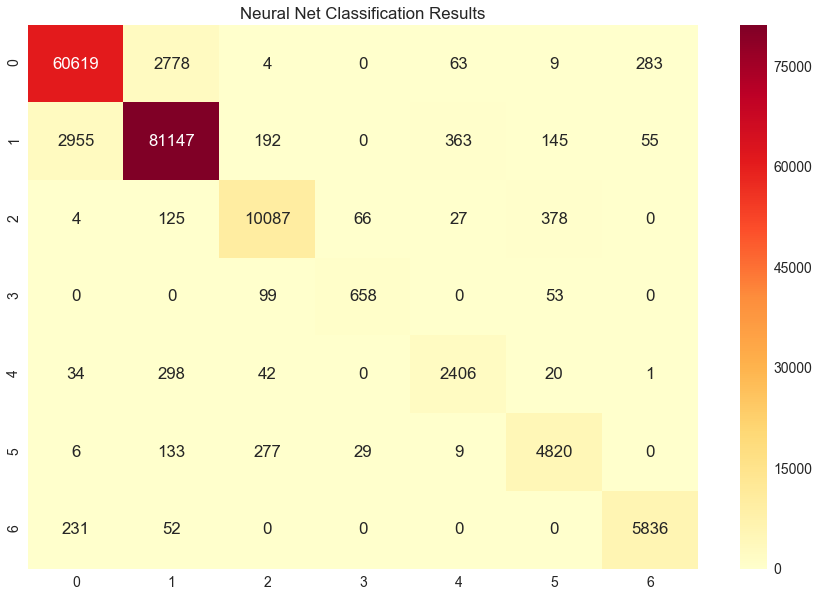
Time (s) f1-score

50) 499.8 0.92

75) 463.4 0.94

These tests did have pretty good scores in much more reasonable amounts of time than the 3x500 net, however, they did not beat out the 3x100 net in either time or score, and while adding another layer to the 3x100 neural net would likely improve accuracy, it would increase the time more than desired. Therefore, we will stick with the 3x100 model.

Finally, we want to put all of these hyper-parameters together and see if we can get something even better. Our final model has 3 hidden layers, each with 100 nodes, uses that ‘adam’ solver, and implements the ‘tanh’ activation function. After all is said and done, our final model ends up with a f1-score of 0.95. It is a pretty accurate model that doesn’t take too long to create, but can we do better?



A heatmap of the Neural Net classifications results is shown above. This was made using the confusion matrix and the heatmap is showing values for actual vs. predicted.

**Decision Tree**

Another model we worked on with surprising results to me was the decision tree. To do this, we implemented the sklearn DecisionTreeClassifier. This model works by building a tree of simple decision rules, where leaf nodes are a classification of the data. First off, we again decided to test out the different standardizations of data, those being: (1) all data standardized, (2) none of it standardized, and (3) all but the bools (0 or 1 values) standardized:

Time (s) f1-score

1) 5.9 0.94

2) 6.1 0.94

3) 5.9 0.94

Unlike many other models, it seems that decision trees do not require data standardization, and this makes sense. The decision trees work based of comparisons, and whether the data is standardized or not won’t affect these. Therefore, moving on, we will actually use the non-standardized data merely because it is slightly faster with the same score.

Next, we will try and improve this already highly impressive model with different hyper parameters, the first one being the splitter type. This parameter decides the strategy used to choose the split at each node in the tree, and the options are (1) ‘best’, which merely chooses the best split, or (2) ‘random’, which chooses a random best split to add some possible variety to the tree:

Time (s) f1-score

1) 5.9 0.94

2) 2.4 0.93

The random splitter type manages to shave off a couple of seconds while only losing 0.01 score. This is because it does not have to compare every possible split in order to find the best one, and instead just picks a random one that is good enough. This could be very useful if the tree were to take a long time to fit, however, with a higher score after only a couple seconds, ‘best’ seems to be the better splitter type for our situation as 3 seconds saved is unimportant.

Another hyper-parameter we altered was the criterion, which is how the quality of any split in the tree is measured. There are two options here: (1) ‘gini’ for the Gini impurity (which is a measure of how often a random element from the set would be labeled improperly if it was labeled randomly), or (2) ’entropy’ for the information gained from this split.

Time (s) f1-score

1) 5.9 0.94

2) 6.1 0.94

Again, neither option seems to really offer much of a clear advantage over the other. In fact, this tree seems to already be about as good as it will get with this data set. You can alter other hyper parameters like the max depth or amount of leaf nodes, but setting a limit on these only serves to decrease the score, as it needs at least a certain amount to be sufficiently accurate. Therefore, it seems that the basic default decision tree really can be much improved upon with this data set, yet it does not really need to be. A score of 0.94 within seconds is very impressive, especially when compared with our other models.

A screenshot of a cell phone

Description generated with high confidence

A heatmap of the Decision Tree classifications results is shown above. This was made using the confusion matrix and the heatmap is showing values for actual vs. predicted.

**Bagging Classifier**

We decided to use a BaggingClassifier model due to its effectiveness with large datasets. Models like KNN and SVM have too high of a time complexity to efficiently use with a dataset as large as ours. Since models like BaggingClassifier work with random subsets of the training set and combine the individual results to a final prediction, we guessed that it would work well with our dataset. We also tested it using the default Decision Tree Model.

Note: Data has been standardize using sklearn.preprocessing’s StandardScalar. The standardization of the data (whether we used StandardScalar or MinMax, etc.) did not affect the performance for this model.

**BaggingClassifier (Decision Tree):**

Time: **--- 70.20564389228821 seconds ---**

Score: **0.958910868368**

Classification Report:

precision recall f1-score support

1 0.95 0.96 0.96 63416

2 0.97 0.97 0.97 84871

3 0.95 0.96 0.95 10746

4 0.91 0.84 0.88 878

5 0.93 0.82 0.87 2928

6 0.94 0.90 0.92 5217

7 0.97 0.94 0.96 6248

avg / total 0.96 0.96 0.96 174304

As you can see with a simple default BaggingClassifier using decision tree and no max samples or features selected, the model does extremely well in classifying the data. However, it is important to note that the model did take 70 secs to fit the data- which is considerably longer than just a normal decision tree. However, it was slightly more accurate. In the next step we will reduce the max features and samples.

**Max\_features and max\_samples = .5:**

Time: **--- 21.32273840904236 seconds ---**

Score: **0.896278914999**

Classification Report:

precision recall f1-score support

1 0.90 0.89 0.90 63416

2 0.89 0.93 0.91 84871

3 0.85 0.91 0.88 10746

4 0.91 0.68 0.78 878

5 0.93 0.51 0.66 2928

6 0.88 0.65 0.75 5217

7 0.97 0.85 0.90 6248

avg / total 0.90 0.90 0.89 174304

Reducing the max features and samples greatly improved the time it took for the model to run. However, the predictions are much worse. This is likely because we are working with a larger number of subsets of the data.

**Removing Highly Correlated Features:**

Below is a list of the top most correlated features in the data set (See BaggingClassifier Notebook):

Wilderness\_Area4 Soil\_Type10 0.485031

Soil\_Type10 Wilderness\_Area4 0.485031

Hillshade\_Noon Slope 0.526911

Slope Hillshade\_Noon 0.526911

Soil\_Type29 Wilderness\_Area1 0.550549

Wilderness\_Area1 Soil\_Type29 0.550549

Aspect Hillshade\_9am 0.579273

Hillshade\_9am Aspect 0.579273

Hillshade\_Noon Hillshade\_3pm 0.594274

Hillshade\_3pm Hillshade\_Noon 0.594274

Vertical\_Distance\_To\_Hydrology Horizontal\_Distance\_To\_Hydrology 0.606236

Horizontal\_Distance\_To\_Hydrology Vertical\_Distance\_To\_Hydrology 0.606236

Elevation Wilderness\_Area4 0.619374

Wilderness\_Area4 Elevation 0.619374

Aspect Hillshade\_3pm 0.646944

Hillshade\_3pm Aspect 0.646944

Hillshade\_9am Hillshade\_3pm 0.780296

Hillshade\_3pm Hillshade\_9am 0.780296

Wilderness\_Area3 Wilderness\_Area1 0.793593

Wilderness\_Area1 Wilderness\_Area3 0.793593

Let’s remove some and make our new training and testing sets and run the model:

df\_nocorr = df.drop(['Hillshade\_3pm', 'Horizontal\_Distance\_To\_Hydrology','Wilderness\_Area1'], axis = 1)

Time: **--- 53.63902997970581 seconds ---**

Score: **0.958715806866**

Classification Report:

precision recall f1-score support

1 0.95 0.96 0.96 63423

2 0.97 0.97 0.97 85218

3 0.94 0.96 0.95 10770

4 0.90 0.84 0.87 815

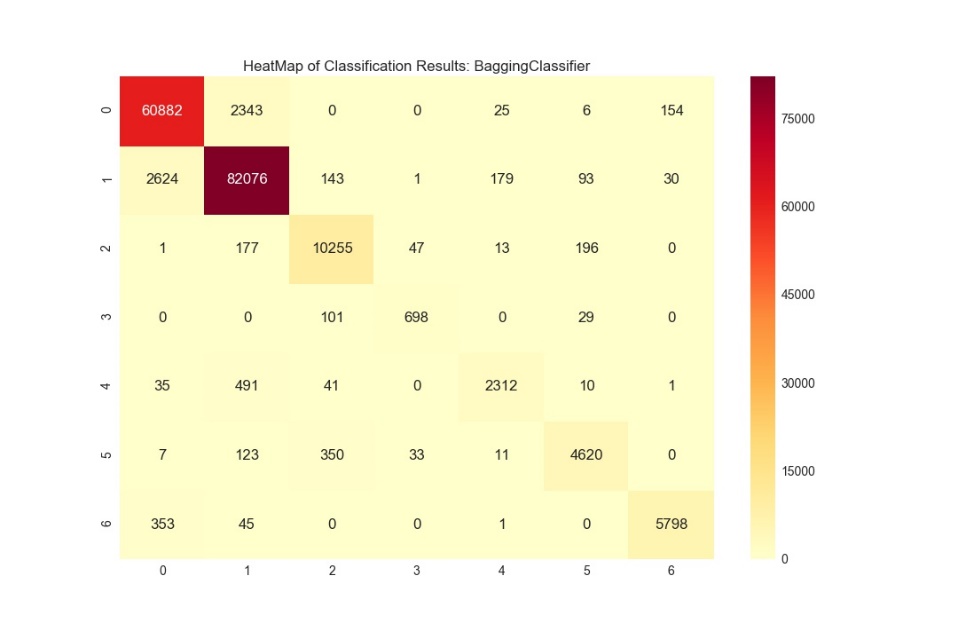
5 0.92 0.81 0.86 2780

6 0.94 0.90 0.92 5175

7 0.97 0.95 0.96 6123

avg / total 0.96 0.96 0.96 174304

As you can see the model took slightly longer to run but did not have a higher score or more accurate results. This means that the correlated values do not have a high enough correlation to skew the model in any way.



This heatmap of the Classification results (made using the confusion matrix) of Bagging classifier shows us that a high number of trees are of class 0 or 1. As you can see the predictions that are wrong are very low in number compared to the right ones (on the diagonal). This is a visualization of how accurate this model is.

**Logistic Regression**

For Logistic Regression we knew it would not be very efficient or accurate. Research has shown that logistic regression is not the best when it comes to large datasets combined with multiple features such as in our case.

We used the ‘sag’ solver because it performs well on large datasets and had to increase max iterations to 400 because the model was not converging otherwise.

Note: All Non-Boolean values were standardized by StandardScalar. Later we will see how MinMax would affect the performance. Logistic regression is often greatly affected by how the data is scaled.

**StandardScalar:**

Time: **--- 910.6872494220734 seconds ---**

Score: **0.035415136772535341**

Classification Report:

precision recall f1-score support

1 0.71 0.69 0.70 63564

2 0.74 0.80 0.77 85161

3 0.62 0.85 0.72 10625

4 0.66 0.31 0.42 772

5 0.26 0.01 0.02 2825

6 0.42 0.09 0.14 5184

7 0.72 0.53 0.61 6173

avg / total 0.70 0.72 0.70 174304

As you can see Logistic Regression (data scaled using StandardScalar) did not perform nearly as well as the other models. However, after seeing the score we were skeptical on how accurate it would be, since the score was extremely low. It turns out it was accurate, just not nearly as other models.

**MinMaxScalar:**

Time: **--- 98.98852276802063 seconds ---**

Score: **0.035415136772535341**

Classification Report:

precision recall f1-score support

1 0.71 0.69 0.70 63564

2 0.74 0.80 0.77 85161

3 0.62 0.87 0.72 10625

4 0.63 0.26 0.36 772

5 0.27 0.01 0.02 2825

6 0.42 0.07 0.12 5184

7 0.72 0.53 0.61 6173

avg / total 0.70 0.72 0.70 174304

Using the MinMaxScalar greatly improved the time but did not have much effect on the results. This is likely because MinMaxScalar transforms data into the range [0,1]. Because this matches our Boolean values, logistic regression ran much faster.

**Taking Out Highly Correlated Variables:**

df\_nocorr = df.drop(['Hillshade\_3pm', 'Horizontal\_Distance\_To\_Hydrology','Wilderness\_Area1', 'Slope'], axis = 1)

Time: **--- 67.75202965736389 seconds ---**

Score: **0.034852900679272991**

Classification Report:

precision recall f1-score support

1 0.71 0.68 0.69 63854

2 0.73 0.79 0.76 85078

3 0.60 0.88 0.71 10565

4 0.68 0.21 0.32 822

5 0.31 0.01 0.02 2758

6 0.41 0.04 0.07 5152

7 0.70 0.53 0.60 6075

avg / total 0.70 0.71 0.69 174304

Taking out some of the most correlated features lowers the model’s performance (As seen with the lower recall an f1 score. There is evidence that these features do not skew the model in any way, but actually helps it in certain cases. 

Like our BaggingClassifier, logistic regression shows that many of the predictions fall into classes 0 and 1. However, a brief look at this heatmap makes it very evident that there are some large numbers not in the diagonal (meaning wrong predictions). This further shows that logistic regression is far from the best model to use for this dataset.

1. Conclusions and Future Works

Out of the four models we assessed, the most accurate model was the BaggingClassifier, using the default Decision Tree parameter. It got an F1 score of .96, at a very reasonable amount of time. However, lowering the max features and max values decreased the accuracy of the BaggingClassifier model considerably. In terms of time, the Decision Tree outperformed all other models by a large margin. It was also almost as accurate as the BaggingClassifier, in a fraction of the time. While the Neural Net had potential to produce accurate results on par with the decision tree, it took much longer than both the Decision Tree and BaggingClassifier. Logistic regression was by far the most inaccurate and time consuming of the models. It only had an F1 score of .70 and using the StandardScalar it took over 900 seconds. This time was greatly reduced by using a MinMaxScalar and was lower than the Neural Net, however, the Decision Tree was still much faster. If a model were to be chosen for this data set, we would recommend the Decision Tree model. This is because it is basically just as accurate as our most accurate model, however, it is considerably faster and more efficient.

While this data set is by no means small, in the future it would be worth considering how datasets many times larger than this could affect the time it would take for our models to fit. Would our models scale properly? And would Decision trees be able to keep both their time and score so promising if we had, say, ten million measurements? In the end, we found that our different models worked very differently with our data, and that it is important to find the right one for your particular machine learning problem.