MAP553 Machine Learning
Project Report

Team XSY : Yunhao CHEN, Sicheng MAO, Xiaozhen WANG

January 5, 2022

Contents

1 Introduction

The goal of the project is to predict seven different cover types in four different wilderness areas with the best accuracy. It is a multi-classification problem.

The data is in raw form (not scaled). It contains both quantitative features (such as elevation, aspect and slope) as well as qualitative features(such as soil type and wilderness area.

Our workflow is:

1. Understanding the task and the data
2. Data analysis
3. Feature engineering and selection
4. Comparison between several machine learning models
5. Hyperparameter Tuning on the best model
6. Evaluation

The first step is to understand the physical meaning of each variable. Then, after checking the anomaly, we conduct the data analysis by drawing scatter plots with respect to different features. The next step is the most important one: feature engineering and selection. Several linear combinations of the existing features turn out to be useful as new features in classification. We also come up with some great ideas in feature design. Moreover, we have tried different ways of coding for Id, soil type and wilderness area. After comparing different machine learning models, we find that Extra Tree has the best performance. We also used **Auto machine learning** tool to automatically select model and tune hyperparameters. By stacking multiple models, it can achieve an accuracy of **86.76%** on the Kaggle public leaderboard.

2 Data analysis

We conduct an exploratory data analysis on the training set.

Name	Measurement	Description
Elevation	meters	Elevation in meters
Aspect	azimuth	Aspect in degrees azimuth
Slope	degrees	Slope in degrees
Horizontal Distance To Hydrology	meters	Horz Dist to the nearest surface water features
Vertical Distance To Hydrology	meters	Vert Dist to the nearest surface water features
Horizontal Distance To Roadways	meters	Horz Dist to the nearest roadway
Hillshade 9am	0 to 255	Hillshade index at 9am, summer solstice
Hillshade Noon	0 to 255	Hillshade index at noon, summer soltice
Hillshade 3pm	0 to 255	Hillshade index at 3pm, summer solstice
Horizontal Distance To Fire Points	meters	Horz Dist to the nearest wildfire ignition points
Wilderness Area (4 binary columns)	0 or 1	Wilderness area designation
Soil Type (40 binary columns)	0 or 1	Soil Type designation
Cover Type	1 to 7	Forest Cover Type designation - Response Variable

First of all, from the Kaggle data description, we know that the training set is identically distributed on all the 7 cover types. Then we draw some scatter plots with respect to certain combinations of features. Here we only put up the figures which demonstrate some obvious features.

Id: Unexpectedly, the Id column does contain some patterns. Apart from type 1 and type 2(later we find these two types occur most in test set), there are gaps of Id number for other cover types, which may be useful in distinguishing them from the first two types.

Elevation, Slope, Aspect: The cover type has a clear change with elevation rising up and there are more diversity on a relatively low elevation level. From other perspectives of this 3d scatter plot, slope and aspect do not have any significant influence.

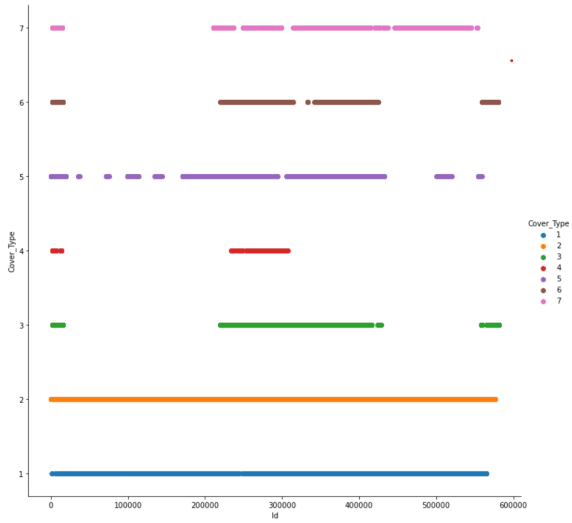

Figure 1: Id-cover type scatter

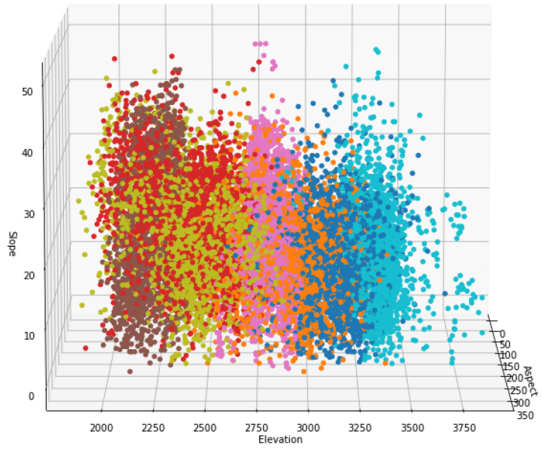

Figure 2: 3D scatter with respect to elevation, slope and aspect

Hillshade: The three hillshade data regularly distributed on a plane with a curve boundary due to their geometric nature. However, there is an abnormal line on the corner which implies some missing values on the hillshade_3pm column.

Distances to hydrology or firepoint: Different cover types form clusters with different size in this feature space.

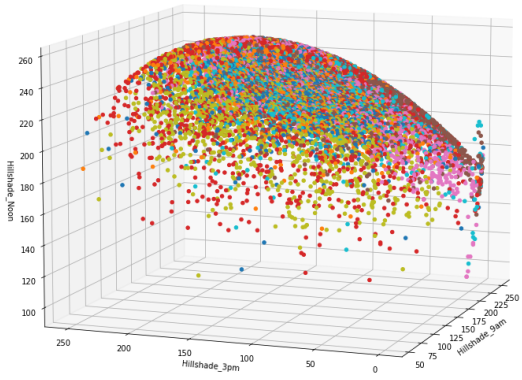

Figure 3: 3D hill shade scatter

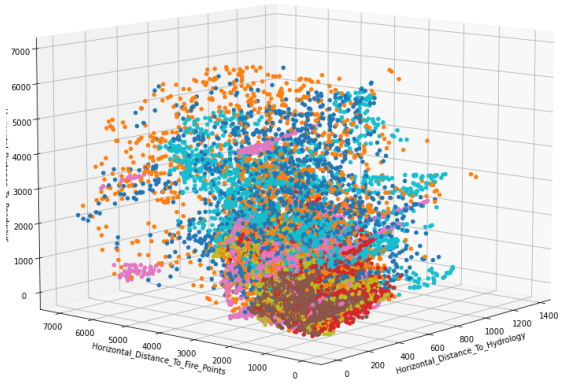

Figure 4: Scatter 3D with respect to distances

Soil type: Scatter by soil type and reduced soil type(by merging those whose first two numerals are identical. There are gaps mainly over the last five cover types.

3 Feature Design and Selection

3.1 One-hot encoding vs integer encoding

In the raw data, there are 40 different soil types and 4 different area types with one-hot encoding representation, while the integer labeled encoding is more adapted to the tree classifiers according to

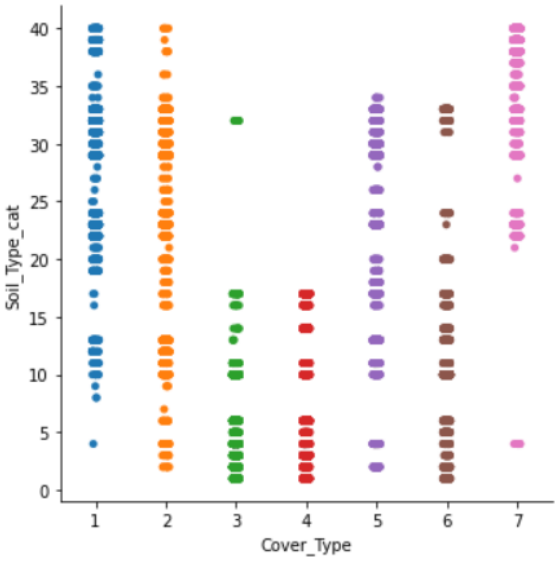

Figure 5: Soil type-cover type scatter

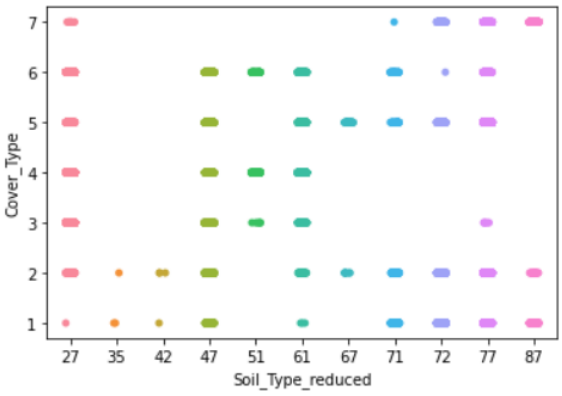

Figure 6: Reduced soil type-cover type scatter

[8]. Therefore we also add the integer labeled representation to these features.

Moreover, according to the description of the soil codes, we know that the last two digits has no special meaning, so we can merge those whose first two digits are identical. By this we create a reduced soil type feature which has 11 different categories in total.

3.2 Create new features

After the exploratory data analysis, we could also create some new features to help this classification task.

- **Aspect:** We can see that the distribution of aspect is not uniform, between 150° and 300° is relatively small. Therefore, we consider shift the values at 180° in degrees azimuth to construct a new set of aspect.

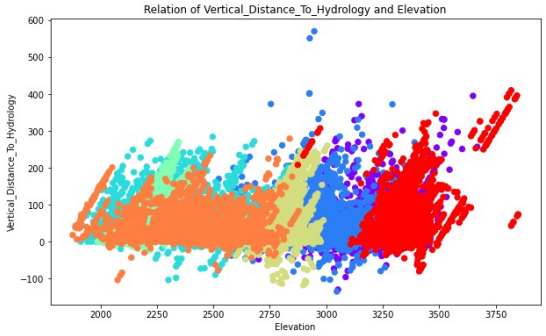

Figure 7: Vertical Hydrology and Elevation

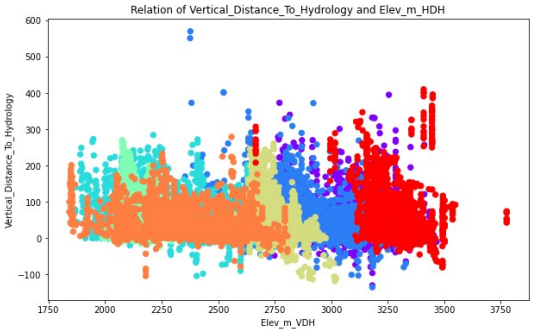

Figure 8: Elev_m_HDH and Vertical Hydrology

- **Vertical Distance To Hydrology:** On the one hand, we observe that Vertical Distance To Hydrology have some negative values. It may be a good idea to create a variable which indicates if it is positive or negative. We found that for each tree, the elevation minus the distance to Hydrology in the vertical direction, which is the elevation of the water source near this tree, is significant for the classification. The following diagram illustrates this significance.
- **Elevation:** In the elevation distribution map of the data analysis, we can see that the different tree species are distributed in bands at different level of elevation. This is also in accordance with the nature of vertical zones in physical geography. We create some features between elevation and water source.
- **Id:** The Id spectra of different cover types has some gaps. In order to better extract this separation feature, we cut the Ids into 10, 100 and 1000 intervals, giving new group IDs to each data point in the same group as new features.
- **Features with similar meanings:** The final construction of features is linear transformation of variables with similar physical significance. For example, the sum, difference, average of the distances in the horizontal direction, the geometric mean of hill shadow in different time(e,g: Square root of product of hillshades at 9AM and Noon), etc.

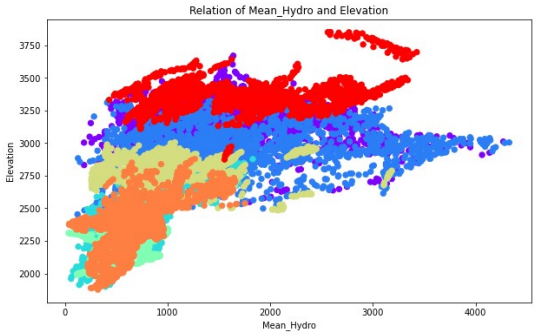

Figure 9: Mean_Hydro and Elevation

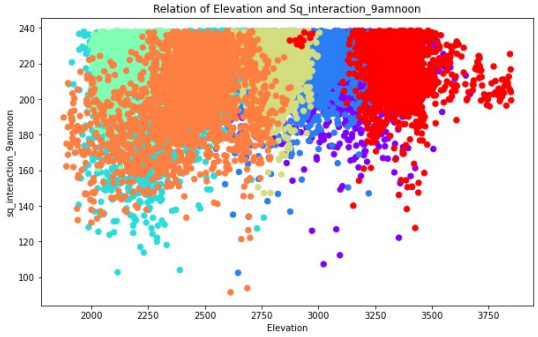

Figure 10: square root of 9am × Noon and Elevation

3.3 Feature engineering table

We organized the newly created features into the following table:

New features	Description
Soil1	Natural code for soil type in different 40 classification
Soil2	Natural code for soil type in different 11 classification
Wilderness_Area	Natural code for Wilderness type
Wilderness_Area2	Square of natural code for Wilderness type
Aspect2	shift the Aspect values at 180° in degrees azimuth
Dist_to_Hydrology	Square root of the sum of the squared horizontal and vertical distances to water
Highwater	The sign of the value Vertical_Distance_To_Hydrology
Elev_m_VDH	The elevation of the water source nearest
Elev_m_HDH	Elevation minus 0.28 times Horizontal_Distance_To_Hydrology
Id2	Group Id for every 10 Ids
Id3	Group Id for every 100 Ids
Id4	Group Id for every 1000 Ids
Cos_slope	The cosine of the slope

New features	Description
Mean_Hillshade	Mean hillshade of the hillshade at 9AM, Noon, and 3PM (0-255)
sq_interaction_9amnoon	Square root of product of hillshades at 9AM and Noon(0-255)
sq_interaction_noon3pm	Square root of product of hillshades at Noon and 3PM(0-255)
sq_interaction_9am3pm	Square root of product of hillshades at 9AM and 3PM(0-255)
Mean_Hydro	Mean Horizontal Distance of Hydrology, Fire Points and Roadways
Hydro_p_Fire	Horizontal_Distance_To_Hydrology plus Horizontal_Distance_To_Fire_Points
Hydro_m_Fire	Horizontal_Distance_To_Hydrology minus Horizontal_Distance_To_Fire_Points
Hydro_p_Road	Horizontal_Distance_To_Hydrology plus Horizontal_Distance_To_Roadways
Hydro_m_Road	Horizontal_Distance_To_Hydrology minus Horizontal_Distance_To_Roadways
Fire_p_Road	Horizontal_Distance_To_Fire_Points plus Horizontal_Distance_To_Roadways
Fire_m_Road	Horizontal_Distance_To_Fire_Points minus Horizontal_Distance_To_Roadways

3.4 Selection of features

The number of features also affects the training time and the training effect of the model, and the features that have a greater impact on the results are selected. This part mainly uses the tools in `sklearn.feature_selection`.

- **Removing features with low variance:** We use `VarianceThreshold` to filter some features with small variance. In particular, when the variance is taken as 0, the 15th soil type does not appear in the training set according to the classification of 40 cover types.
- **Features importance:** We can use `f_classif` to get the impact of each feature on the results and rank them. Next, we use the cross-validation method `RFECV` to delete the features one by one to find the optimal number of features. And in the end, based on the ranking, we use `SelectKBest` to select the K optimal features. Here, after calculation based on the **Extra Tree** model, we get $K = 20$ and the ranking is as following figure. And this set of 20 features is used to train the model.

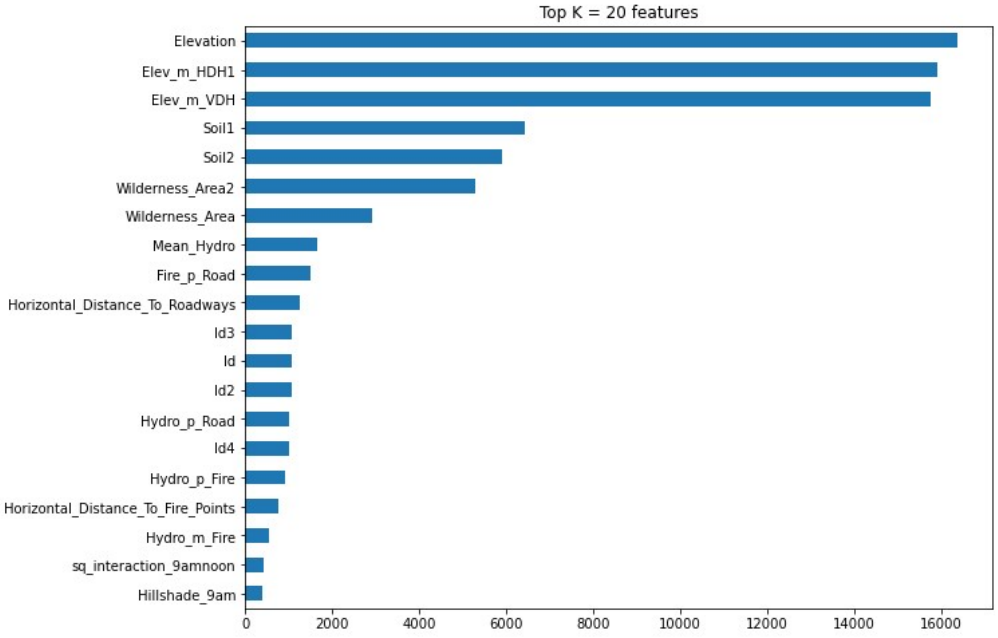

Figure 11: Ranking Top 20 Features

4 Algorithms & Performances

4.1 Gain from past notable studies

The earliest use of the dataset was in a doctoral dissertation. Jock Blackard and Denis Dean at Colorado State University used Linear Discriminant Analysis to obtain a 58% accuracy and Artificial Neural network to achieve 70% accuracy. The paper concludes by saying that ANN does have its drawbacks, it is still more viable than traditional methods, by which we think he refers to logistic regression, LDA and Support Vector Classifiers.

In 2007, Jain & Minz (Journal of Indian Agricultural Statistics) applied Rough Set Theory as an alternative to Machine learning approaches to achieve equally good but using lesser attributes (only 29) to achieve 78% accuracy for the forest cover dataset. This is especially relevant in real world classification applications in remote sensing involving large datasets. In 2004, in Systems, Man and Cybernetics an IEEE International Conference publication, Xiaomei Liu, Kevin W. Bowyer of the University of Notre Dame use the forest cover dataset to conjecture something interesting.

Through these parts, as a classical topic, Tree based algorithm is more suitable for this topic. We first tried the decision tree. Then the ensemble methods (such as Random forest, LGBM and extra tree) have been employed. And we have built a 15 layers neural network in order to check its performance. In the following, we will mainly talk about Extra Tree, LGBM and neural network.

In order to evaluate the performance of different models, we divide the training set into two parts: one true training set and one validation set. Here, the proportion of these two sets is 4:1 (validation_size = 0.20).

4.2 Extra-Tree

4.2.1 Method Algorithm

Extremely Randomized Trees Classifier (ERT) is an ensemble learning technique that aggregates the results of multiple de-correlated decision trees collected in a forest to output classification results. Each decision tree of Extremely Randomized Trees is constructed from the original training samples. At each test node, each tree has a random sample of k features, and each decision tree must select the best features from these feature sets and then split the data based on some mathematical metric (typically the Gini index). This random sample of features leads to the creation of multiple unrelated decision trees.

In the process of building the forest, for each feature, the normalized total reduction of the mathematical metric used to split the feature decisions (e.g., using the Gini index) is calculated, and this value is called the importance of the Gini elements. After the Gini importance is ranked in descending order, the top k features can be selected as needed.

There are two main differences with other tree based ensemble methods. First, it splits nodes by choosing cut-points fully at random, Second, it uses the whole learning sample (rather than a bootstrap replica) to grow the trees. In the code implementation, we call the sklearn.ensemble.ExtraTreesClassifier to train the model.

4.2.2 Parameter Adjustment

- **Max_features and n_estimators :** Here we adjust the parameters in the extra-tree model. First is **max_features**, using the automatic model selection tool sklearn.model_selection.GridSearchCV, we can get a better result when max_features = 0.38. The second is the number of estimators, here generally take a larger case, the results will be better, here we take n_estimators = 2000.

- **Class_weight :** In addition to the parameters of the model itself, we want to improve the results by adjusting the classification weights. Here we observe that the class_weight can change with the results of the last model and converge to a better result by a certain number of iterations(e.g: 5 times). Therefore, we adjust the weights of the model each time except for the first time, and the ratio is chosen to be the inverse of the last classification weight, which will improve the accuracy of the results after several iterations (about 0.5%).

4.2.3 Extra_tree performance

In the initialization phase, we construct the model by taking n_estimators = 2000 and max_features = 0.38. As mentioned earlier, we divide the whole training set again according to 4:1 to verify the accuracy of the model. On the new training set, the accuracy can reach 100%, while on the new test set, the accuracy can also reach **91.73%**. When predicting the final result, we trained the model on the whole training set, predicted it on the test set, and then iterated the result 5 times to adjust the class_weight, and measured this final result by the Kaggle system, and the best result we got is **85.84%**.

4.3 LGBM (Light Gradient Boosting Machine)

LightGBM is an algorithm based on Gradient Boosting Decision Tree(GBDT). The main idea of GBDT is to apply boosting meta-learning method on the weak learners as decision trees. However, the traditional GBDT algorithm is time-consuming. In order to improve the efficiency, LightGBM uses the techniques as Exclusive Feature Bundling to combine the sparse features and Leaf-wise Tree Growth to save the computation resources. LightGBM also inherits the idea of bootstrap aggregation method which focus only a subsample of the training set drawn at random without replacement, increasing the running speed as well as reducing the overfitting.

Even without a complicate feature engineering, the LGBM wins an accuracy around 75% and 83% after fine-tuning.

4.4 Neural network

We have used Tensorflow to build our neural network. The network is full connected, with 11 hidden layers, including 5 layers of 64 neurons, 5 layers of 32 neurons and 1 output layer. Relu is used as the activation function. Stochastic gradient descent has been employed as the optimizer, with a decay learning rate. Batch size is 64 and number of epochs is 1500.

We found that the performance of neural network here was in general worse than tree-based methods. It may be due to the characteristics of data. There is a combination of numbers (quantitative variables) and one-hot coding (qualitative variables). Moreover, the order of magnitude of the quantitative variables are not the same. All these factors would affect negatively the performance of neural network. On the other hand, tree-based models are indeed good at solving this kind of problem. The different orders of magnitude would not affect their performance.

Finally, we got **66%** as the accuracy on test set. The figures below show respectively the evolution of accuracy and loss function during the training process.

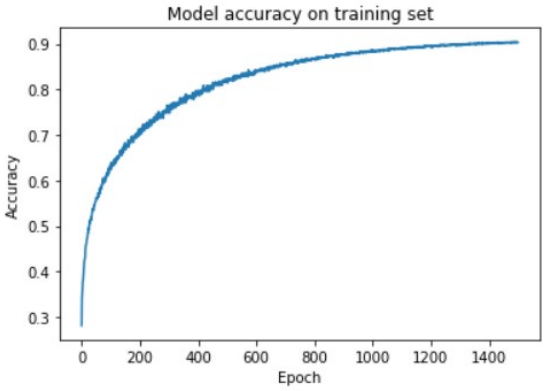

Figure 12: training accuracy with #epochs

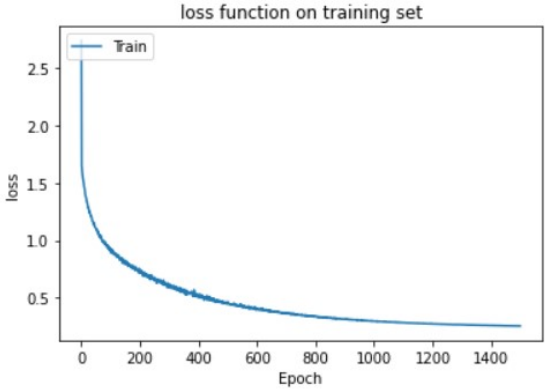

Figure 13: training loss with #epochs

The hyperparameters in neural network is shown in the following table:

Hyperparameters in neural network	Value
Number of layers	11
Number of neurons per layer	(64, 64, 64, 64, 64, 32, 32, 32, 32, 32, 7)
Activation function	Relu
Optimizer	SGD
Learning rate	1e-3
decay steps	1e4
decay rate	0.9
Batch size	64
Number of epochs	1500

4.5 Auto Machine Learning

The auto machine learning package **Autogluon** has been used to train models and fine tune hyperparameters. Autogluon contains fourteen common machine learning and deep learning algorithms, like Random Forest, Light GBM, XGBoost and neural network. AutoGluon will automatically choose a random training/validation split of the data. The data used for validation is seperated from the training data and is used to determine the models and hyperparameter-values that produce the best results. Rather than just a single model, AutoGluon trains multiple models and ensembles them together to ensure superior predictive performance. This process involves repeatedly training models under different hyperparameter settings and evaluating their performance. It can be computationally-intensive. In practice, the top 20 influential features have been used as input. It took about **2h21min** to train models and predict results.

Compared to the previous extra-tree model, autogluon obtained results with an accuracy improvement of nearly 1%, **reaching 86.91%, the best of all algorithms**.

4.6 Accuracy Table

The result of different models are summarized in the following table (the accuracy is obtained on test set).

Classifiers	Parameters	Accuracy
AutoGluon	presets='best_quality', sample_weight=1/weight of extratree result	86.91%
Extra_tree	n_estimators=2000, max_features=0.38, Iterate 5 times 1/class_weight	85.84%
LGBM	n_estimators=300, learning_rate=0.4	83.44%
Neural Network	shown in the table above	66.27%
Random Forest	n_estimators=2000, max_features=0.4	82.83%
Decision Tree	criterion="gini",max_depth=300	73.48%
AdaBoost	n_estimators=500	27.74%

5 Difficulties and Failures

There are mainly two difficulties we have found throughout the whole challenge.

1. A relatively small training set: the training set is only 2.5% of the test set. So we are lack of labeled data.

2. An imbalanced sampling: the training set is identically distributed over the 7 cover types while (from predictions we find) the test set is mainly composed of type 1 and 2.(86%) So there might be much more false predictions on the less frequent classes in test data because their proportions are exaggerated in training data.

Resampling seems to be a good idea to tackle the first difficulty. However, since we are not equipped with labeled test data, we cannot sample directly from the test data. An alternative is to simulate new data points from the train data. But we don't have any good idea to realize it since it contains

categorical features which we cannot interpolate. We may also conduct a semi-supervised approach according to [14].

For the second difficulty, we have tried to add weights to each class while performing the classification (mainly the extra tree and LGBM classifier). But there is no significant improvement.

However, we still come up with some ideas on explaining the good performance of the tree algorithm and the AutoML algorithm. Both the extra tree and the LGBM have implemented a resampling scheme by just taking a part of the given features into consideration during each iteration, which to some extent, tackles the first difficulty. And the AutoML combines different classifiers based on their performance on different class, adding weights to their predictions. This tackles the second difficulty.

6 Learning and Improvements

Through this project, we become more familiar with the pipeline of machine learning. In the beginning, we tended to focus on the algorithms. But the data preprocessing is also crucial for the performance of algorithms. In this project, the given data are relatively proper(in regardless of the imbalanced sampling issue). In other cases, we should possibly pay much more attention to treat the missing values and anomalies. After the treatments of these values, we need to analyze the distribution of the features. Based on that, there are several criterions to select features. For example, we can remove the features with little variance (An almost constant feature does not help a lot). Then, we could try to design new features using our knowledge on concrete problem. For instance, we have added a new feature, defined as "Square root of the sum of the squared horizontal and vertical distances.

The next step is to choose an appropriate algorithm. This choice is also relevant to the preprocessing. For example, for Support Vector Machine, it would be better to rescale the columns of data. But the rescaling is not necessary if we choose the decision tree. This process needs much theoretical basis and experience. After splitting the data into training set and validation set, we can fine tune hyperparameters of model and evaluate models with the validation data. The package `sklearn.model_selection.GridSearchCV` perfectly fullfills our need.

7 Conclusion

This report solves the classic multi-classification forest cover type prediction problem. The forest cover dataset is an extremely rich dataset for anyone who wishes to practice one's multi-classification skills. The three main methods used in this report, Extra-tree, LGBM, and Neural Network, are all widely used algorithms with good performance. With the development and innovation in the field of machine learning, more and more algorithms will be implemented to tackle this problem.

References

[1] Auto machine learning documentation. `https://auto.gluon.ai/stable/tutorials/tabular_prediction/tabular-quickstart.html`.

[2] Feature engineering notebook. `https://nbviewer.org/github/aguschin/kaggle/blob/master/forestCoverType_featuresEngineering.ipynb`.

[3] Feature engineering on forest cover type data. `https://www.slideshare.net/ChandanaTL/feature-engineering-on-forest-cover-type-data-with-decision-trees`.

[4] Feature selection avec sklearn. `https://www.youtube.com/watch?v=T4nZDuakYlU`.

[5] Forest cover type classification study. `https://rstudio-pubs-static.s3.amazonaws.com/160297_f7bcb8d140b74bd19b758eb328344908.html`.

[6] How should we deal with the lack of training data? `https://www.researchgate.net/post/How-should-we-deal-with-the-lack-of-training-data-in-a-machine-learning-task`.

[7] Lgbm features. `https://lightgbm.readthedocs.io/en/latest/Features.html`.

[8] One-hot encoding is making your tree-based ensembles worse. https://towardsdatascience.com/one-hot-encoding-is-making-your-tree-based-ensembles-worse-heres-why-d64b282b5769.

[9] Predicting forest cover types with ensemble learning. https://shankarmsy.github.io/posts/forest-cover-types.html.

[10] Predicting forest cover types with the machine learning workflow. https://towardsdatascience.com/predicting-forest-cover-types-with-the-machine-learning-workflow-1f6f049bf4df.

[11] Jock A. Blackard and Denis J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, 1999.

[12] Rahul R. Kishore, Shalvin S. Narayan, Sunil Pranit Lal, and Mahmood A. Rashid. Comparative accuracy of different classification algorithms for forest cover type prediction. *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, pages 116–123, 2016.

[13] Hugo Sjöqvist, Martin Längkvist, and Farrukh Javed. An analysis of fast learning methods for classifying forest cover types. *Applied Artificial Intelligence*, 34(10):691–709, 2020.

[14] Xiaojin Zhu. Semi-supervised learning literature survey. *Comput Sci, University of Wisconsin-Madison*, 2, 07 2008.