Cover Type Prediction

Sébastien Meyer
École polytechnique, France
sebastien.meyer@polytechnique.edu

This report details the different approaches and strategies that I have developed in order to get the best possible score I could on the cover type data set. We were given a dataset made up of a gathering of different numerical and qualitative features describing 30×30 meter forest cells. The main task was to predict the cover type of each of these cells. We had a training set and we were asked to submit a prediction containing a cover type associated to each cell, some of them being taken from the training set and others being brand-new data.

Despite the fact that this dataset is well-known since its donation to the UCI in 1998[1], and after past Kaggle competitions[2], it has been a really challenging project for me. In order to best describe how I tackled this problem, my report will follow my advances. Firstly, I will remind the reader with a brief description of the task and the data. Then, I will go on with my first attempts. Since my first tries were not satisfactory, I took inspiration from past Kaggle competitions - both on this dataset and others - to get new ideas. In the **Appendix**, I gathered my models with some details on the pipeline, as well as a description of the engineered features.

1 The dataset

The dataset is made up of 15120 training points and 581012 test points. It appears that the training points are also part of the test set, which is interesting and should ensure a certain accuracy of any model. However, we can already note that the test set is much larger than the training set and that it might contain different distributions of data. Therefore, we must be prepared to face overfitting. Our task is a classification task, corresponding to the *Cover_Type* feature which ranges between 1 and 7. The training set is quite imbalanced, as each class does not account for 2160 of the 15120 samples (see **Figure 1**).

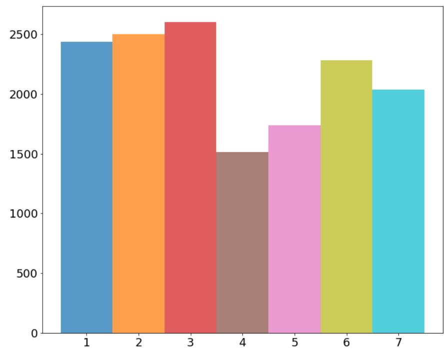

Figure 1: Repartition of the training labels.

Also, there are 54 available features. 40 of them correspond to *Soil_TypeX* with X ranging between 1 and 40. Each data point is assigned with one and only one *Soil_TypeX* feature, which gets the value 1. Same goes for 4 other features which are *Wilderness_AreaY* with Y ranging between 1 and 4. Finally, the ten last features are briefly summarized in **Table 1**. Only *Vertical_Distance_To_Hydrology* has negative values for some of the data points, while all the other features are nonnegative. The *Cover_TypeX* and *Wilderness_AreaY* features could have a huge impact on our predictions, because of them being uniquely assigned to the data points. Also, they are highly correlated with our target feature. In **Figure 2**, we show the four features that are the most correlated with our target feature. Overall, there is no correlation over 0.6 between two features, which does not lead us to exclude any feature here. However, the *Soil_TypeX* features are differently distributed, with a number of associated data points ranging from zero for *Soil_Type15* to thousands for some soil types.

[1] https://archive.ics.uci.edu/ml/datasets/covertype
[2] https://www.kaggle.com/c/forest-cover-type-prediction

Feature	Mean	Std	Min.	Max.
Elevation	2,749	419	1,877	3,850
Aspect	156	110	0	360
Slope	16.56	8.53	0	50
Horz Hyd	228	209	0	1,376
Vert Hyd	51.31	61.52	-135	570
Horz Road	1,718	1,330	0	6,803
Horz Fire	1,527	1,117	0	7,095
Hill 9AM	213	30.64	52	254
Hill Noon	219	22.80	99	254
Hill 3PM	134	46.07	0	251

Table 1: Description of numerical features.

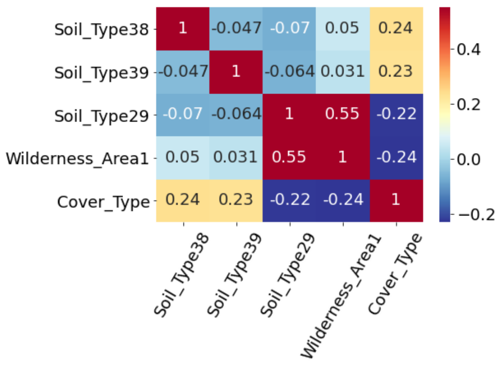

Figure 2: Simplified correlation matrix.

2 First attempts

After my first data exploration, I decided to try out some basic models that are widely used for multiclass classification. I did not do much feature engineering, apart from removing the *Cover_Type15* feature which is assigned to no data point. Moreover, I decided to merge the binary features into one feature, for instance *Soil_Type* with values ranging from 1 to 40 assigned to each data point and *Wilderness_Area* ranging from 1 to 4. Rapidly, I understood that this would not help building more accurate predictions. Indeed, when we do merge several binary features into only one, we induce a fictive ordering between values. Let X be a feature gathering binary features and thus having values from 1 to p. When splitting at a specific node, a decision tree can make a decision based on $X > a$ or $X \leq a$. We see that such decision implies that values between 1 and a have a relationship, which might not be the case. Therefore, we cannot expect accurate predictions by doing so.

2.1 Cross-validated gridsearch

I implemented an automated gridsearch pipeline using *optuna* package to select the parameters of any model I decided to use. *Optuna*[1] takes ranges of values for model parameters as entry, and performs an optimization to find the best parameters according to the cross-validation score. The cross-validation was 5 folds large, with accuracy as target. Typically, I would run a gridsearch for 15 to 50 trials, depending on the complexity of the model and the size of the dataset.

2.2 RandomForestClassifier and LightGBM

The first two models that I used were RandomForestClassifier and LightGBM[2] from *scikit-learn*[3] API. These models are widely used in the case of classifications with more than two possible labels. For instance, RandomForestClassifier relies on the averaging of several base decision trees. These decision trees are built in a top-bottom fashion. Each split builds two nodes or regions R and $\bar{R}$. Let us denote by p_k^R and $p_k^{\bar{R}}$ the proportions of samples from class k falling respectively in regions R and $\bar{R}$. Split is made by minimizing a criterion, the most used criterion being Gini index defined as:

$$C(R, \bar{R}) = \sum_k p_k^R(1 - p_k^R) + \sum_k p_k^{\bar{R}}(1 - p_k^{\bar{R}})$$

which is the sum of Gini impurities in both R and $\bar{R}$. Other criterions can be used and have been tried during gridsearch, namely entropy criterion, however Gini index yields best results in almost all runs. On the other hand, LightGBM is based on Boosting rather than Bagging. Although LightGBM is based on the same principle as other Boosting models, that is, updating a model by successively assigning weights to wrongly classified data points, it is designed to be much more faster on large datasets than its counterparts.

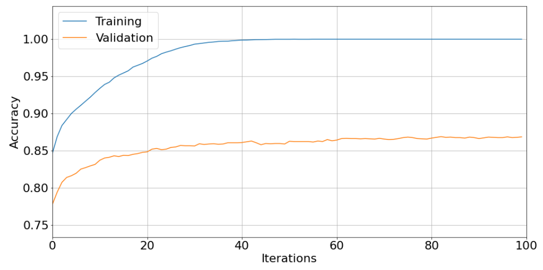

Figure 3: Learning curves for LightGBM.

A basic gridsearch for RandomForestClassifier lead to an accuracy of 0.7613, and a basic gridsearch for LightGBM lead to an accuracy of 0.78374. These scores were relatively good, considering that I did not started feature engineering and that my models were rather simple. However, tweaking these models did not bring much improvement. I had to start engineering new features.

3 Taking inspiration from past competitions

3.1 New features

As for most of the Kaggle competitions, feature engineering is mandatory for getting better scores. Looking at the available features, I computed some new features: *Distance_To_Hydrology* is built as the square root of the sum of *Horizontal_Distance_To_Hydrology*² and *Vertical_Distance_To_Hydrology*²; *Ratio_Distance_To_Hydrology* is built as *Vertical_Distance_To_Hydrology* over *Horizontal_Distance_To_Hydrology* (when dividing by zero, value is replaced by the median of finite values); *Horizontal_Distance_To_Point_Log* is built as $\log(1 + Horizontal\_Distance\_To\_Point)$ for *Point* being *Hydrology*, *Roadways* and *Fire_Points*; *Aspect_times_Hillshade_3pm* is built as the product of the features (relation derived from their closeness in a simple hierarchical clustering); all possible polynomial combination of some chosen numerical features.

Then, the base and engineered features were still not enough. After looking at some discussions on past Kaggle competitions, I started another data exploration. Simple statistics such as *Max*, *Min* or *Mean* can be added to the features. On top of that, new distances can be approximated from those in our possession. Indeed, the horizontal distance between hydrology and roadways is somewhere between *Horizontal_Distance_To_Hydrology* − *Horizontal_Distance_To_Roadways* in absolute value and *Horizontal_Distance_To_Hydrology* + *Horizontal_Distance_To_Roadways*. Thus, I added all the absolute differences and sums of these distances, as well as the mean distance to all three points of interest.

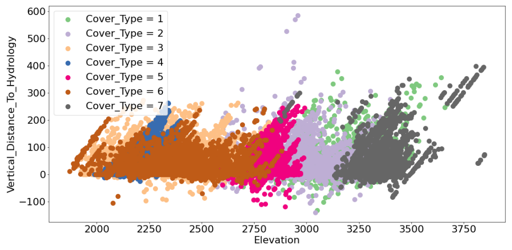

Figure 4: Relationship between *Elevation* and *Vertical_Distance_To_Hydrology*.

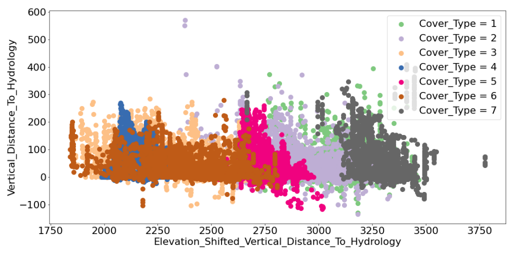

Figure 5: Relationship between *Elevation* and *Elevation_Shifted_Vertical_Distance_To_Hydrology*.

One of the most important features can be easily derived from the available data. As shown **Figure 4**, the relationship between *Elevation* and *Vertical_Distance_To_Hydrology* is affine. As a result, simple borders between cover types can be derived by a linear model such as a Random Forest if we manage to align a new feature with *Elevation*. As shown **Figure 5**, it is relatively easy to compute such a feature. Same can be done with *Horizontal_Distance_To_Hydrology* and *Horizontal_Distance_To_Roadways*. These new features are appended with the *Shifted* mark. An example of resulting repartition of training labels wrt a new feature is shown **Figure 6**.

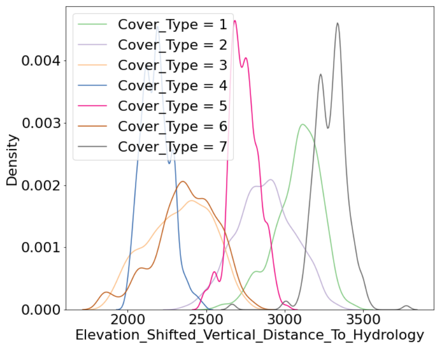

Figure 6: Repartition of training labels wrt *Elevation_Shifted_Vertical_Distance_To_Hydrology*.

3.2 ExtraTreesClassifier and OneVsRestClassifier

Thanks to these new features, RandomForestClassifier was able to outperform LightGBM and its accuracy approached 0.87 during cross-validation. Nevertheless, another great improvement for this dataset stems from the use of ExtraTreesClassifier. This classifier is based on trees that differ from usual decision trees in two points:

1. Bootstrapping is disabled by default for ExtraTreesClassifier
2. Splits are chosen among randomly drawn cuts given a subset of features

In some cases, such extremely randomized decisions can reduce overfitting, as it appears to be the case for this dataset. Indeed, the absolute difference in accuracy between training and validation was much smaller with ExtraTreesClassifier. My first trial with these new features and ExtraTreesClassifier obtained an accuracy of 0.82237. Going further, I started adding more features and manually tried to find the best subset of features. Therefore, I decided to compute polynomial features from a different subset of base and engineered features. Also, I used a OneVsRestClassifier from *scikit-learn*, which focuses on fitting one classifier per class in a "one-vs-all" fashion. All these slight improvements allow me to reach an accuracy of 0.82622.

3.3 Failed attempts

Once feature engineering was done, features would easily exceed 100 when counting polynomial features. I thought that such a number of features could lead to overfitting or that we could lose information due to correlation. In order to prevent this to happen, I tried two methods. First, I removed features that were too correlated (one per pair) with a fixed threshold. Second, I also tried to remove the features with a high variance inflation factor. Variance inflation factor is defined as $\mathrm{VIF}_i = 1/(1 - R_i^2)$, where R_i^2 is the R-squared value of a linear regression of feature i. This factor asseses the ability to estimate a feature thanks to the others and thus their correlation. However, removing features using these techniques always lead me to poorer accuracy. In addition, I ran PCA several times but the same problem occured. **Figure 7** shows the evolution of the cross-validation accuracy with ExtraTreesClassifier when PCA is applied. Features are typically those that I used in my last trials (see next section or **Appendix**).

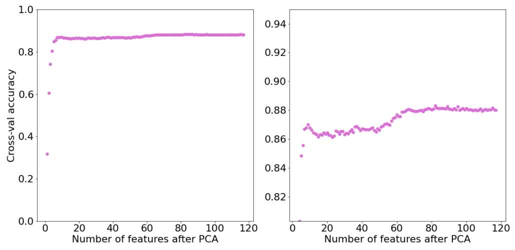

Figure 7: Evolution of cross-validation accuracy with number of PCA components.

Another model that I used is a simple feed-forward network. By tweaking some of the parameters, I was able to design a network with one hidden layer of approximately 50 neurons and sigmoid activation function, fitted with the RMSprop optimizer, which yielded relatively good results, nevertheless they were always below the scores of forest models. In my opinion, neural networks are more prone to overfitting in such context, moreover they often require lots of training data and features that we are in lack of in this data set.

4 Last improvements

4.1 More features and data cleaning

After that, I built some features and tried several combinations.They yielded only small improvements. It can be noted that some of the features that I have computed are the mean Hillshade index, binary features corresponding to soil types families (there are 7 families plus a category for neither of them, as detailed in the Data appendix) and soil types stonyness (either stony, rubly or neither). An interesting observation that I made only during the last days of the competition is about missing values for the *Hillshade_3pm* feature. As shown **Figure 8**, there are too much zeros for it to be natural.

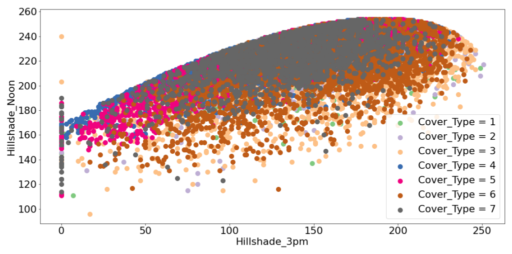

Figure 8: Training *Hillshade_3pm* values before estimation.

A simple technique to reduce the number of missing values is to estimate them thanks to known values. Thus, I trained an ExtraTreesRegressor model using all the other explanatory features of training data points where *Hillshade_3pm* is positive. Then, missing values are predicted both for training and test points. As shown **Figure 9**, we get a good estimation of at least a part of the missing values.

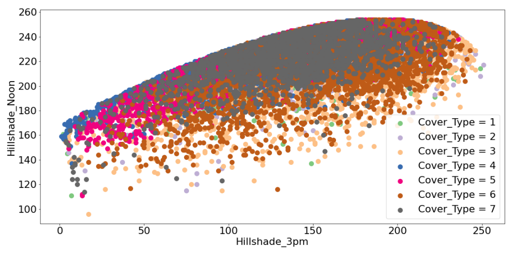

Figure 9: Training *Hillshade_3pm* values after estimation.

4.2 Stacking

Even though ExtraTreesClassifier appears to outperform other models, Boosting models such as LightGBM and XGBoost[4] perform relatively well on the dataset and RandomForestClassifier yields similar results to Boosting. One of the methods used to combine different models that can be good at predicting specific aspects of the data is through Stacking. In the *scikit-learn* API, several base models are fitted on the training dataset. Then, a final estimator - usually a LogisticRegression - is fitted on base estimators predictions over training data with cross-validation to avoid overfitting. A summary schema is shown **Figure 10**. In this model, I tried to combine different models such as RandomForestClassifier, ExtraTreesClassifier and LightGBM. Adding XGBoost yielded slight to no improvement.

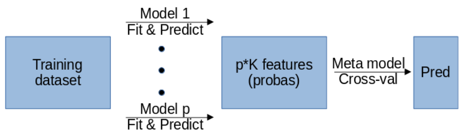

Figure 10: Stacking pipeline.

Since Stacking uses several models, its training is much more longer than for any base estimator. Therefore, running a gridsearch for the Stacking model would be computationally too expensive for my computer. I decided to fix the parameters of the base estimators using the best parameters of their own gridsearch. Together with some more features, Stacking allowed me to reach an accuracy of 0.83822.

4.3 Resampling the training dataset

One of the major issues that I observed in this challenge is that, there were huge differences in accuracy between cross-validation and final score. Clearly, one can expect a poorer accuracy between training and validation, mostly due to overfitting. However, I observed a difference between cross-validation accuracy and final score of approximately 0.10 for models such as RandomForestClassifier and almost 0.08 for ExtraTreesClassifier, which is better at preventing overfitting than the other models. These huge differences brought me to think that the distribution of labels in the test set was different than the one in the training set, which we recall is quite imbalanced. In **Figure 2**, we can see the predictions of an ExtraTreesClassifier model on the test set. It appears that classes 1 and 2, which are more difficult to differentiate (see **Figure 6**), are much more represented in the test set.

Cover_Type	Amount	Pct.
1	212,311	36.54%
2	249,455	42.93%
3	37,445	6.44%
4	4,079	0.70%
5	24,703	4.25%
6	24,325	4.19%
7	28,694	4.94%

Table 2: Predicted test set label repartition.

The first technique that I used consists in assigning a weight to training samples, as *scikit-learn* allows to do so for several models. Sample weights were computed thanks to our predicted repartition in the test set. Theoretically:

$$\sum_{j=1}^{n} \ell(Y_j, h(X_j)) \text{ becomes } \sum_{j=1}^{n} \beta_j \ell(Y_j, h(X_j))$$

where h is our predictor. My trials with this *scikit-learn* option were not conclusive and I tried another idea. In general, we are used to face imbalanced training dataset. For example, in the case of binary classification where the positive class accounts for 1:100 of the data, oversampling is a widely used technique to simulate more positive class data points. Our situation is the opposite, with a quite imbalanced training set and an imbalanced test set. Recall from **Figure 2** that we have an estimation of the test set labels repartition given by our models. Therefore, we can use this prediction to perform undersampling/oversampling operations to get an imbalanced training set. To do so, I used two state-of-the-art techniques implemented in the *imbalanced-learn*[5] Python package. At the end of the resampling process, I would like to keep the same amount of training points, which is 15120, in order to avoid expansive training. Firstly, ClusterCentroids is used for undersampling. For each class, we keep $\min(2160, \text{predicted repartition} \times 15120)$ points where $15120/7 = 2160$. Then, SMOTE[6] is used for oversampling. In SMOTE, new data points are generated along a convex line between two existing points. We want a final repartition of predicted repartition $\times$ 15120 points for each class. This process can be run several times until the estimated repartition of labels in the test set converges. Finally, we can manually fix a resampling repartition after the convergence. A typical manually-adjusted training repartition is given **Table 3** and can be used during data preprocessing.

Cover_Type	Amount	Pct.
1	5,500	36.30%
2	7,000	46.20%
3	1,000	6.60%
4	50	0.33%
5	450	2.97%
6	500	3.30%
7	650	4.29%

Table 3: Training dataset resampling.

Finally, ExtraTreesClassifier outperforms the other models as well as Stacking when the training set is imbalanced. My final trial reached an accuracy of 0.85095.

Conclusion

To conclude, I have learnt some practical skills from this project. Most of the time, it is difficult to get a precise idea of a model or even a pipeline to address a dataset. One has to try and search different combinations. All in all, we can keep in mind that once a baseline is found by using an usual model, other less "conventional" models can be tested out. In terms of the project itself, one of the most important aspects was to design pertinent features. This project allowed me to learn more about data analysis ideas and to be more careful about the models I use on a regular basis. Despite my satisfactory results, I think that things can still be done to improve my models. If I had more time, I would have tried to build more features, especially from the Hillshade indices which remained mysterious to me. In addition, more computational power would allow for more aggressive gridsearch. Furthermore, there must be other models or blending ideas that other teams have thought about and that I did not try!

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta and Masanori Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework.* 2019, in KDD.

[2] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye and Tie-Yan Liu. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay. *Scikit-learn: Machine Learning in Python.* 2011. Journal of Machine Learning Research, vol.12, pp. 2825-2830.

[4] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System.* San Francisco, California, USA. 2016. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785-794.

[5] Guilaume Lemaître, Fernando Nogueira and Christos K. Aridas. *Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning.* 2017. Journal of Machine Learning Research, vol. 18, n°18, pp 1-5. (Available at: `http://jmlr.org/papers/v18/16-365.html`)

[6] N. V. Chawla, K. W. Bowyer, L. O.Hall and W. P. Kegelmeyer, *SMOTE: synthetic minority oversampling technique.* 2002. Journal of artificial intelligence research, pp. 321-357.

Appendix

Models and results

The following tables sum up the different trials I made as well as corresponding features and tuned parameters. All runs and gridsearches were made with at least one of the following seeds: 42, 4586, 8005 and 13420.

Model	Preprocessing	Features	Tuned Parameters	Test accuracy
RandomForestClassifier	Merge binary: Yes Feature engineering: No Eliminate var.: *Soil_Type15* Eliminate correlated var.: No PCA: No Scaling: MinMax 0-1 Resampling: No	Base numerical + *Soil_Type*, *Wilderness_Area* **Total: 12**	*n_estimators* *criterion* *min_samples_split* *min_samples_leaf* *ccp_alpha*	0.76173
LightGBM	Merge binary: Yes Feature engineering: No Eliminate var.: *Soil_Type15* Eliminate correlated var.: No PCA: No Scaling: MinMax 0-1 Resampling: No	Base numerical + *Soil_Type*, *Wilderness_Area* **Total: 12**	*n_estimators* *num_leaves* *min_split_gain* *min_child_weight* *min_child_samples* *subsample* *subsample_freq* *reg_alpha* *reg_lambda*	0.78374
OneVsRestClassifier: - ExtraTreesClassifier	Merge binary: No Feature engineering: Yes Eliminate var.: *Soil_Type15* Eliminate correlated var.: No PCA: No Scaling: Standard Resampling: No	Base + *Horizontal_Distance_To_Point_Log* + *Distance_To_Hydrology* + *Ratio_Distance_To_Hydrology* + *Distance_PointA_{plus,minus}_Distance_PointB* + *Aspect_times_Hillshade_3pm* + *Elevation_Shifted_Distance_To_Point* + *Max* & *Std* of base numerical + degree 2 polynomial combination of some features[3] **Total: 76**	*n_estimators* *min_samples_split* *min_samples_leaf* *min_impurity_decrease* *ccp_alpha*	0.82622
Stacking: - ExtraTreesClassifier - RandomForestClassifier - LightGBM - XGBoost	Merge binary: No Feature engineering: Yes Eliminate var.: *Soil_Type15* Eliminate correlated var.: No PCA: No Scaling: Standard Resampling: No	Base (corrected *Hillshade_3pm*) + *Horizontal_Distance_To_Point_Log* + *Distance_To_Hydrology* + *Distance_PointA_{plus,minus}_Distance_PointB* + *Aspect_times_Hillshade_3pm* + *Elevation_Shifted_Distance_To_Point* + degree 2 polynomial combination of some features[3] **Total: 73**	(taken from base estimators)	0.83822
ExtraTreesClassifier	Merge binary: No Feature engineering: Yes Eliminate var.: *Soil_Type15* Eliminate correlated var.: No PCA: No Scaling: Standard Resampling: Yes	Base (corrected *Hillshade_3pm*) + *Horizontal_Distance_To_Point_Log* + *Distance_To_Hydrology* + *Distance_PointA_{plus,minus}_Distance_PointB* + *Aspect_times_Hillshade_3pm* + *Elevation_Shifted_Distance_To_Point* + *Max* & *Std* of base numerical features + *Mean_Distance_To_Points_Of_Interest* + *Mean_Hillshade* + *FamilyX_Soil_Type* + *{Stony,Rubly}Soil_Type* + degree 2 polynomial combination of some features[4] **Total: 117**	*n_estimators* *min_samples_split* *min_samples_leaf* *min_impurity_decrease* *ccp_alpha*	0.85095

[3] polynomial features: *Elevation*, *Horizontal_Distance_To_Roadways*, *Horizontal_Distance_To_Fire_Points*

[4] polynomial features: *Elevation*, *Horizontal_Distance_To_Roadways*, *Horizontal_Distance_To_Fire_Points*, *Wilderness_Area2*, *Horizontal_Distance_To_Roadways_Log*, *Horizontal_Distance_To_Fire_Points_Log*, *Elevation_Shifted_Vertical_Distance_To_Hydrology*, *Elevation_Shifted_Horizontal_Distance_To_Hydrology*

Description of the features

Here, we give an extensive description of all the features.

Feature	Formula	Remarks
Soil_Type	argmax of *Soil_TypeX* columns	The idea is to reduce the number of features. However, it implies an order between soil types that does not exist.
Wilderness_Area	argmax of *Wilderness_AreaY* columns	The idea is to reduce the number of features. However, it implies an order between wilderness areas that does not exist.
Horizontal_Distance_To_Point_Log	$\log(1 + \textit{Horizontal\_Distance\_To\_Point})$	Extract non-polynomial relation in distances. *Point* can be *Roadways*, *Hydrology* and *Fire_Points*.
Distance_To_Hydrology	$(\textit{Vertical\_Distance\_To\_Hydrology}^2 + \textit{Horizontal\_Distance\_To\_Hydrology}^2)^{1/2}$	True distance to hydrology.
Ratio_Distance_To_Hydrology	*Vertical_Distance_To_Hydrology* / *Horizontal_Distance_To_Hydrology*	Missing values due to zeros values in *Horizontal_Distance_To_Hydrology* are replaced by the median of finite values. The main idea behind this feature is to approximate a slope to hydrology.
Distance_To_PointA_{plus,minus}_Distance_To_PointB	*Horizontal_Distance_To_PointA* $\pm$ *Horizontal_Distance_To_PointB*	Estimation of minimal and maximal distances between two points of interest.
Aspect_times_Hillshade_3pm	*Aspect* $\times$ *Hillshade_3pm*	A hierarchical clusering shows that these features are close in distance.
Elevation_Shifted_Distance_To_Point	*Elevation* $-\alpha\times$*Distance_To_Point*	Creation of features that linearly separate classes. Repartition of α coefficients: - 1 for *Vertical_Distance_To_Hydrology* - 0.2 for *Horizontal_Distance_To_Hydrology* - 0.02 for *Horizontal_Distance_To_Roadways*
{Max,Std}	$\{\max, \mathrm{std}\}$(base numerical features)	Simple metric. Computed before scaling.
Mean_Distance_To_Points_Of_Interest	$\sum(\textit{Horizontal\_Distanc\_To\_Points})/3$	*Point* can be *Roadways*, *Hydrology* and *Fire_Points*.
Mean_Hillshade	$\sum(\textit{Hillshade\_Time})/3$	*Time* can be *9am*, *Noon* and *3pm*.
FamilyX_Soil_Type	Find corresponding families in the competition description.	A soil type can belong to none, one or several families. There are 7 families and one category for neither: Ratake, Vanet, Catamount, Leighan, Bullwark, Como, Moran.
{Stony,Rubly}_Soil_Type	Find corresponding stonyness in the competition description.	A soil type is either stony, rubly or neither.