FOREST COVER TYPE PREDICTION

ANQI CHENG

1. Part I : Compare Five Machine Learning Models

1.1. Introduction.

- In this part I explored and compared five machine learning models in predicting forest cover type basing on site attributes.
- The first ten continuous features are normalized to improve efficiency.
- For each model I performed k-fold cross validation (k = 3) for parameter selection.
- For each model after the parameters are chosen, I tested on the validation set and analyzed the confusion matrix.
- Since the main purpose in this part is to study each model and select the best one, to save computer time I used a quarter of the total training set (10,000 examples).

1.2. Logistic Regression (in Matlab).

1.2.1. *Regularized Logistic Regression.* Standard logistic regression only works with 2 classes, while in this problem we have 7 classes. Therefore I performed 7 one-vs-rest classifications, and output the sigmoid function value for each class (which can be interpreted as the probability). Then classification is made basing on the maximum probability.

The parameter in the model is the regularization parameter λ. I explored two different sizes of data set as well as a certain range of λ, see table 1.

TABLE 1. 3-fold cross validation accuracy of regularized logistic regression

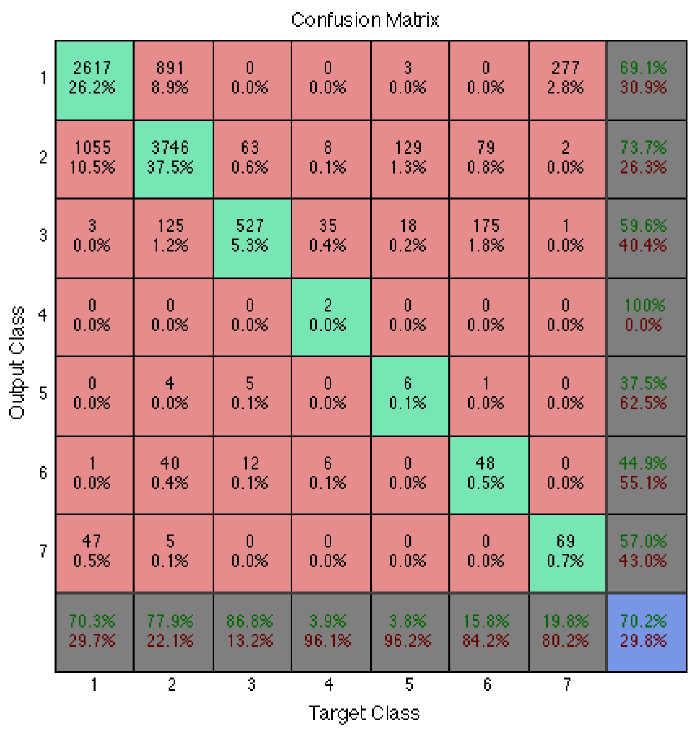
λ	0.1	1	10
m = 1000	69.90%	70.30%	66.60%
m = 10,000	70.37%	70.23%	69.79%

From the table we can see that when data set size increases an order of magnitude there is NO significant improvement of cross validation accuracy. This clearly indicates that we have reached the limitation of the model and increasing statistics will not help much.

To test the validation set I selected $\lambda = 0.1$ with training set size m = 10,000. The results are in Fig. 1. From the confusion matrix we can see that this model is **NOT good** at:

- separating class 1 and class 2,
- separating class 5 and 6 from class 2,
- separating class 7 from class 1.

The advantage of this model is that it is cheap and straight forward to implement.



Output Class \ Target Class	1	2	3	4	5	6	7	
1	2617 26.2%	891 8.9%	0 0.0%	0 0.0%	3 0.0%	0 0.0%	277 2.8%	69.1% 30.9%
2	1055 10.5%	3746 37.5%	63 0.6%	8 0.1%	129 1.3%	79 0.8%	2 0.0%	73.7% 26.3%
3	3 0.0%	125 1.2%	527 5.3%	35 0.4%	18 0.2%	175 1.8%	1 0.0%	59.6% 40.4%
4	0 0.0%	0 0.0%	0 0.0%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
5	0 0.0%	4 0.0%	5 0.1%	0 0.0%	6 0.1%	1 0.0%	0 0.0%	37.5% 62.5%
6	1 0.0%	40 0.4%	12 0.1%	6 0.1%	0 0.0%	48 0.5%	0 0.0%	44.9% 55.1%
7	47 0.5%	5 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	69 0.7%	57.0% 43.0%
	70.3% 29.7%	77.9% 22.1%	86.8% 13.2%	3.9% 96.1%	3.8% 96.2%	15.8% 84.2%	19.8% 80.2%	70.2% 29.8%

FIGURE 1. Confusion matrix on the validation set from regularized logistic regression.

1.2.2. *Regularized Kernel Logistic Regression.* One variation of above model is using Gaussian kernel for logistic regression, namely instead of using:

$$h_\theta(x) = g(\theta^T \cdot x),$$

I used

$$h_\alpha(x) = g(\alpha^T \cdot kernel),$$

where kernel I used is Gaussian with $\sigma = 1$:

$$kernel(a, b) = exp(-\frac{\|a - b\|^2}{2\sigma^2}).$$

This model is significantly more expensive than the naive logistic regression, because the number of internal parameters goes from 50 (number of features) to m (number of examples), and the input goes from $m \times 50$ to $m \times m$. A naive estimate will say the cost of this model scales with m^2. It will be extremely time consuming to implement this model on 10,000 examples, so instead I used 1000 examples for this model, and (external) parameter space I explored is in table 4.

TABLE 2. 3-fold cross validation accuracy of regularized **kernel** logistic regression

λ	0.1	1	10
m = 1000	67.10%	67.90%	64.70%

When compared with the normal logistic regression there seems no significant improvement but the cost is about two orders of magnitude higher (with m = 1000). Thus I will not bother to analysis this model.

1.3. Neutral Networks (in Matlab).

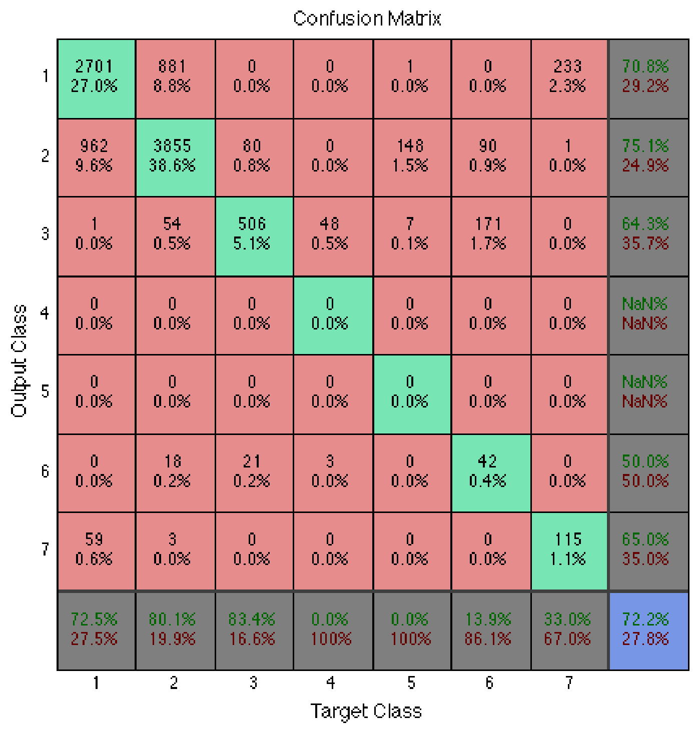
In this model I fixed number of layers to be 3 and the number of iterations to be 100, so there are only two external parameters of the neutral networks model left: the hidden layer size, and the regularization parameter λ. The parameter space I explored is:

TABLE 3. 3-fold cross validation accuracy of neutral networks with $\lambda = 1$

hidden layer size	10	15	20	30
m = 10,000	71.90%	72.24%	72.70%	72.26%

TABLE 4. 3-fold cross validation accuracy of neutral networks with hidden layer size = 20

λ	0.1	1	10
m = 10,000	72.62%	72.70%	70.85%



	1	2	3	4	5	6	7	
1	2701 27.0%	881 8.8%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	233 2.3%	70.8% 29.2%
2	962 9.6%	3855 38.6%	80 0.8%	0 0.0%	148 1.5%	90 0.9%	1 0.0%	75.1% 24.9%
3	1 0.0%	54 0.5%	506 5.1%	48 0.5%	7 0.1%	171 1.7%	0 0.0%	64.3% 35.7%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
6	0 0.0%	18 0.2%	21 0.2%	3 0.0%	0 0.0%	42 0.4%	0 0.0%	50.0% 50.0%
7	59 0.6%	3 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	115 1.1%	65.0% 35.0%
	72.5% 27.5%	80.1% 19.9%	83.4% 16.6%	0.0% 100%	0.0% 100%	13.9% 86.1%	33.0% 67.0%	72.2% 27.8%

FIGURE 2. Confusion matrix on the validation set from neutral networks.

Therefore the parameters I picked are {hidden layer size = 20, $\lambda = 1$}. The confusion matrix of the validation set is in Fig. 2. From the confusion matrix it is clear that this model:

- performs relative well on classifying the first three classes,
- completely misclassified class 4 to class 3, class 5 to class 2,
- cannot well separate class 6 from class 3 & 2, class 7 from class 1.

It seems that the neutral networks works well with the majority classes (1, 2, 3) but performs poorly on the minority classes(4, 5, 6, 7). I will attempt to conclude that the neutral networks model is not quite suitable for largely unevenly distributed data, so weighting the classes by inverse of their frequencies might be helpful.

1.4. **Support Vector Machines (SVM) (in Matlab and Python).** I implemented this model in Matlab and Python. In Matlab I modified the svmtrain package to fit for 7 classes in the following way:

- First I performed 7 one-vs-rest classifications and output the distance to the boundaries;
- Then I compared the distances and determined the class by the closest boundary.

In Python there are well developed tools in sklearn package to perform both one-vs-one and one-vs-rest classifications for multi-class classification. Since the Python code is more efficient and gives higher accuracy than the Matlab code, I will stick to Python for the following analysis.

I explored the following parameters in this model:

- C: penalty parameter of the error term;
- kernel: 'rbf' (Gaussian) or 'linear';
- gamma: kernel coefficient for 'rbf'.

Table 5, 6 shows the parameter space I explored:

TABLE 5. 3-fold cross validation accuracy of SVM with linear kernel (m=10,000)

C	0.05	0.1	0.5	1	5	10
m = 10,000	71.24%	71.35%	71.98%	72.04%	71.94%	71.96%

TABLE 6. 3-fold cross validation accuracy of SVM with Gaussian kernel (m=10,000)

(gamma, C)	1	5	7	10	30	50	100
0.02	71.24%	71.35%	–	71.98%	72.04%	71.94%	71.96%
0.1	–	–	–	–	76.22%	–	74.95%
0.2	–	76.69%	78.35%	76.34%	75.11%	–	–
0.3	75.96%	76.50%	–	75.88%	–	–	–

So the best parameters I will select from the cross validation are {C = 7, kernel = 'rbf', gamma = '0.2'}. The confusion matrix on the validation set is in Fig. 3.

I would say this model performs reasonably well except separating class 5 from class 2 and class 4 from class 3.

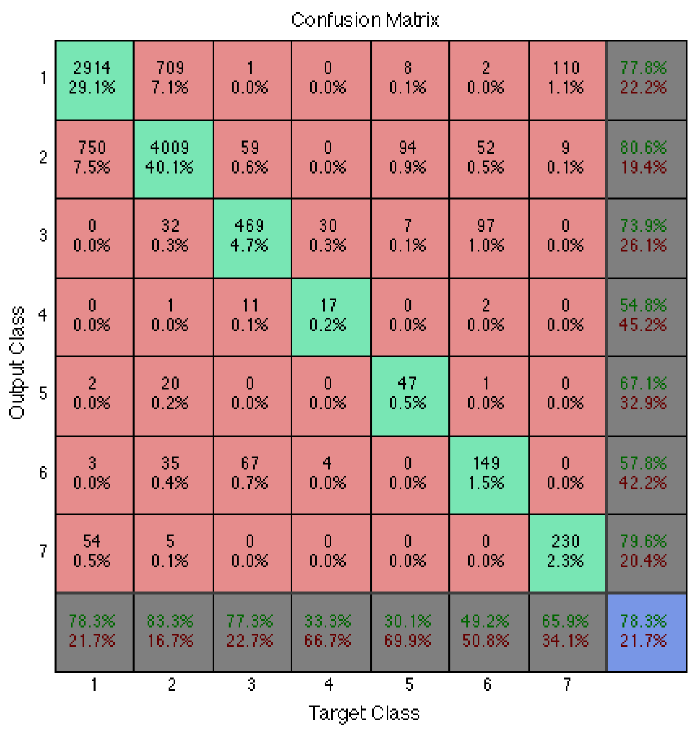


FIGURE 3. Confusion matrix on the validation set from SVM.

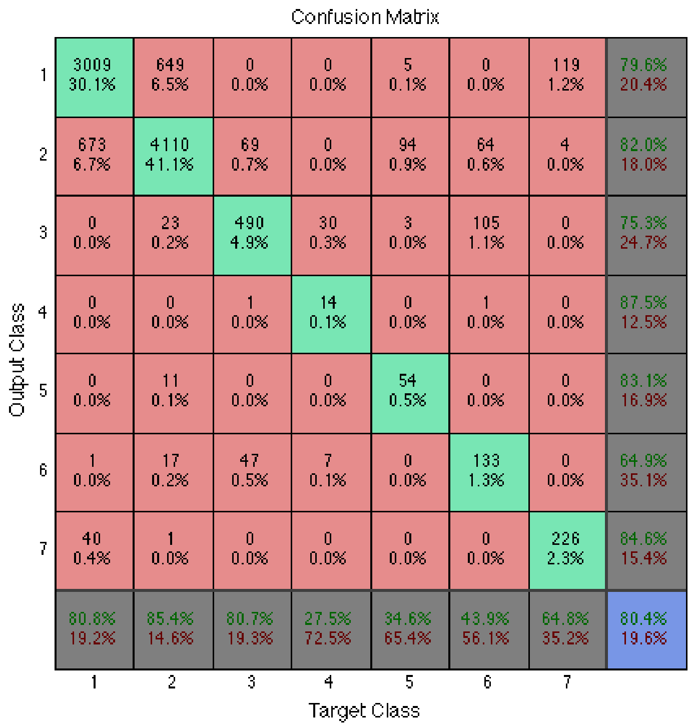
1.5. **Random Forests (in Python).** In this model the adjustable parameters are:

- n_estimators: the number of trees in the forest;
- max_features: the number of features to consider when looking for the best split;
- max_depth (default=None): the maximum depth of the tree.
- min_samples_split (default=2): the minimum number of samples required to split an internal node;
- min_samples_leaf (default=1): the minimum number of samples in newly created leaves;
- bootstrap (default=True): whether bootstrap samples are used when building trees.

Simple tests showed that the default values of the last 4 parameters are optimal for this problem, and therefore I concentrated on tuning the first 2 parameters. The parameter space I explored is in table 7.

TABLE 7. 3-fold cross validation accuracy of random forests (m=10,000)

(n_estimators, max_features)	5	7	10	15	20	30
30	78.05%	78.16%	77.98%	78.30%	78.89%	78.41%
50	–	–	–	78.89%	–	–
70	–	–	78.66%	78.96%	79.09%	79.05%
80	–	–	–	–	79.07%	–


Confusion Matrix

Output Class / Target Class

	1	2	3	4	5	6	7	
1	3009 30.1%	649 6.5%	0 0.0%	0 0.0%	5 0.1%	0 0.0%	119 1.2%	79.6% 20.4%
2	673 6.7%	4110 41.1%	69 0.7%	0 0.0%	94 0.9%	64 0.6%	4 0.0%	82.0% 18.0%
3	0 0.0%	23 0.2%	490 4.9%	30 0.3%	3 0.0%	105 1.1%	0 0.0%	75.3% 24.7%
4	0 0.0%	0 0.0%	1 0.0%	14 0.1%	0 0.0%	1 0.0%	0 0.0%	87.5% 12.5%
5	0 0.0%	11 0.1%	0 0.0%	0 0.0%	54 0.5%	0 0.0%	0 0.0%	83.1% 16.9%
6	1 0.0%	17 0.2%	47 0.5%	7 0.1%	0 0.0%	133 1.3%	0 0.0%	64.9% 35.1%
7	40 0.4%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	226 2.3%	84.6% 15.4%
	80.8% 19.2%	85.4% 14.6%	80.7% 19.3%	27.5% 72.5%	34.6% 65.4%	43.9% 56.1%	64.8% 35.2%	80.4% 19.6%

FIGURE 4. Confusion matrix on the validation set from random forest.

From cross validation the best set of parameters I selected is {n_estimators=70, max_features=20}. The confusion matrix of the validation set is Fig. 4. From the confusion matrix I found most of class 4 are misclassified to class 3, most of class 5 are misclassified to class 2, and there is a great portion of mixture of class 6 and class 3. However the overall performance of this model is the best among the others.

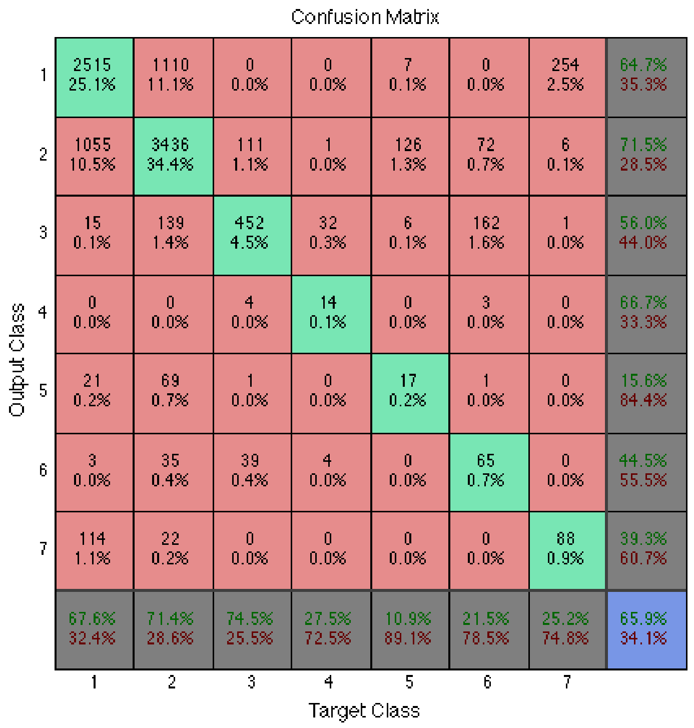
1.6. **Naive Bayes (in Python).**

In Python sklearn package Naive Bayes classifiers have 3 different types:

- GaussianNB is suitable for classification with continuous features;
- MultinomilaNB is suitable for classification with discrete features;
- BernoulliNB is suitable for classification with binary features.

In our problem we have 10 continuous features and 40 binary features, so I did Gaussian NB on the continuous features and BernoulliNB for the binary features, then made prediction by averaging over the probabilities from these two classifications. I am not aware of any suitable parameters to tune so the confusion matrix with all parameters default is Fig. 5.

This model performs quite poorly, especially in classifying the minor classes (class 4, 5, 6, 7). One reason might be this model is based on the 'naive' assumption of independence between every pair of features, which is clearly over simplified. Also the final classification is based on the probabilities from two early classifications, while naive Bayes is known to be a bad estimator (and


Confusion Matrix

Output Class

	1	2	3	4	5	6	7	
1	2515 25.1%	1110 11.1%	0 0.0%	0 0.0%	7 0.1%	0 0.0%	254 2.5%	64.7% 35.3%
2	1055 10.5%	3436 34.4%	111 1.1%	1 0.0%	126 1.3%	72 0.7%	6 0.1%	71.5% 28.5%
3	15 0.1%	139 1.4%	452 4.5%	32 0.3%	6 0.1%	162 1.6%	1 0.0%	56.0% 44.0%
4	0 0.0%	0 0.0%	4 0.0%	14 0.1%	0 0.0%	3 0.0%	0 0.0%	66.7% 33.3%
5	21 0.2%	69 0.7%	1 0.0%	0 0.0%	17 0.2%	1 0.0%	0 0.0%	15.6% 84.4%
6	3 0.0%	35 0.4%	39 0.4%	4 0.0%	0 0.0%	65 0.7%	0 0.0%	44.5% 55.5%
7	114 1.1%	22 0.2%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	88 0.9%	39.3% 60.7%
	67.6% 32.4%	71.4% 28.6%	74.5% 25.5%	27.5% 72.5%	10.9% 89.1%	21.5% 78.5%	25.2% 74.8%	65.9% 34.1%

Target Class

FIGURE 5. Confusion matrix on the validation set from naive Bayes.

it performs equally bad when I just use BernoulliNB over all features).

2. Part II: Improvement to Random Forests

2.1. **Introduction.** In part I we found the random forests model works the best, so in this part I will concentrate on the random forests model and implement improvements by adding one further step of randomness in the way splits are computed.

The high variance of tree-based models was acknowledged in mid nineties last century. In particular, it was shown empirically that the cut-point variance (for numerical inputs) is very high even for large sample sizes [1].

In reference [2] the authors proposed 'extra-trees algorithm' to explicitly randomize the cut-point of numerical features when splitting a node. Recall that in random forests the best split is picked among a random subset of features, and the averaged variance is reduced due tot this randomness. Now a further step of randomness is added by choosing cut-points for each candidate feature fully at random, and then the best among these randomly-generated cut-points is picked. Therefore the variance is usually reduced further by this extra step of randomness.

The Extra-Trees splitting algorithm (for numerical attributes) is the following [2]:

- **Split_a_node**(S)
  Input: the local learning subset S corresponding to the node we want to split
  Output: a split $[a < a_c]$ or nothing
  - If **Stop_split**(S) is TRUE then return nothing.
  - Otherwise select K attributes $\{a_1, ..., a_K\}$ among all non constant (in S) candidate attributes;
  - Draw K splits $\{s_1, ..., s_K\}$, where $s_i =$ **Pick_a_random_split**(S, a_i), $\forall i = 1, ..., K$;
  - Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1,...,K}\text{Score}(s_i, S)$.

- **Pick_a_random_split**(S, a)
  Inputs: a subset S and an attribute a
  Outputs: a split
  - Let a^S_{max} and a^S_{min} denote the maximal and minimal value of a in S;
  - Draw a random cut-point a_c uniformly in $[a^S_{\text{max}}, a^S_{\text{min}}]$;
  - Return the split $[a < a_c]$.

- **Stop_split**(S)
  Input: a subset S
  Output: a boolean
  - If $|S| < n_{\text{min}}$, then return TRUE;
  - If all attributes are constant in S, then return TRUE;
  - If the output is constant in S, then return TRUE;
  - Otherwise, return FALSE.

2.2. **Implementation.** This improvement is already adapted in Python sklearn package under the name 'ExtraTreesClassifier', so instead of implementing this improvement from scratch, I will directly use the existing and well-developed function in Python. It has similar parameters with the random forests classifier and the parameter space I explored is in table 8:

TABLE 8. 3-fold cross validation accuracy of extra random trees (m=10,000)

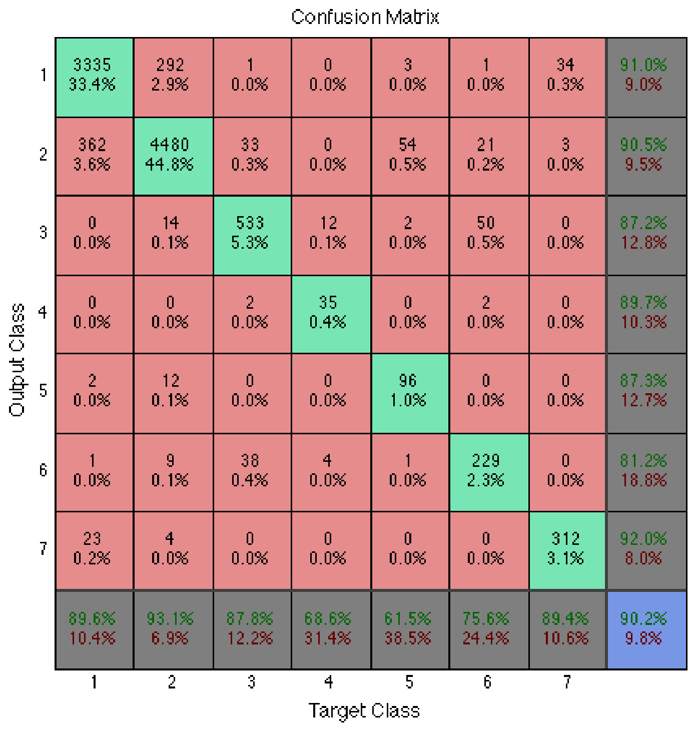
(n_estimators, max_features)	7	10	15
50	80.90%	80.94%	–
70	–	81.22%	–
80	–	81.34%	81.09%

Comparing the highest accuracy 81.34% in table 8 to the highest accuracy 79.09% in table 7, it is clear that the accuracy is significantly improved by adding one step of randomness.

2.3. **Final Model.** Now it is time to apply the final model to the full training set (m = 40,000). The parameter space I explored is in table 9.

TABLE 9. 3-fold cross validation accuracy of final system (m=40,000)

(n_estimators, max_features)	7	10	15
80	88.47%	88.81%	88.47%
100	–	88.73%	88.61



Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	
1	3335 33.4%	292 2.9%	1 0.0%	0 0.0%	3 0.0%	1 0.0%	34 0.3%	91.0% 9.0%
2	362 3.6%	4480 44.8%	33 0.3%	0 0.0%	54 0.5%	21 0.2%	3 0.0%	90.5% 9.5%
3	0 0.0%	14 0.1%	533 5.3%	12 0.1%	2 0.0%	50 0.5%	0 0.0%	87.2% 12.8%
4	0 0.0%	0 0.0%	2 0.0%	35 0.4%	0 0.0%	2 0.0%	0 0.0%	89.7% 10.3%
5	2 0.0%	12 0.1%	0 0.0%	0 0.0%	96 1.0%	0 0.0%	0 0.0%	87.3% 12.7%
6	1 0.0%	9 0.1%	38 0.4%	4 0.0%	1 0.0%	229 2.3%	0 0.0%	81.2% 18.8%
7	23 0.2%	4 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	312 3.1%	92.0% 8.0%
	89.6% 10.4%	93.1% 6.9%	87.8% 12.2%	68.6% 31.4%	61.5% 38.5%	75.6% 24.4%	89.4% 10.6%	90.2% 9.8%

FIGURE 6. Confusion matrix on the validation set from final model.

The confusion matrix on the validation set using parameters {n_estimators=80, max_features=10} is Fig. 6. This model works decent for all classes. When I train on the training + validation sets (m = 50,000) together to make predictions for the test set, I got an accuracy of 91.41%.

2.4. **Feature Importance Evaluation.** From Python Scikit Learn Documentation:“ The relative rank (i.e. depth) of a feature used as a decision node in a tree can be used to assess the relative importance of that feature with respect to the predictability of the target variable. Features used at the top of the tree are used contribute to the final prediction decision of a larger fraction of the input samples. The expected fraction of the samples they contribute to can thus be used as an estimate of the relative importance of the features.”

TABLE 10. Relative importance of features (m=40,000)

Feature	elevation	horiz. dist. to roads	horiz. dist. to fire points
Relative Importance	23.78%	11.97%	10.71%
Feature	horiz. dist. to hydrology	vert. dist. to hydrology	aspect
Relative Importance	6.46%	5.58%	4.67%
Feature	hillshade at noon	hillshade at 9am	hillshade at 3pm
Relative Importance	4.41%	4.05%	3.78%
Feature	slope	any soil type	–
Relative Importance	3.70%	less than 2.5%	–

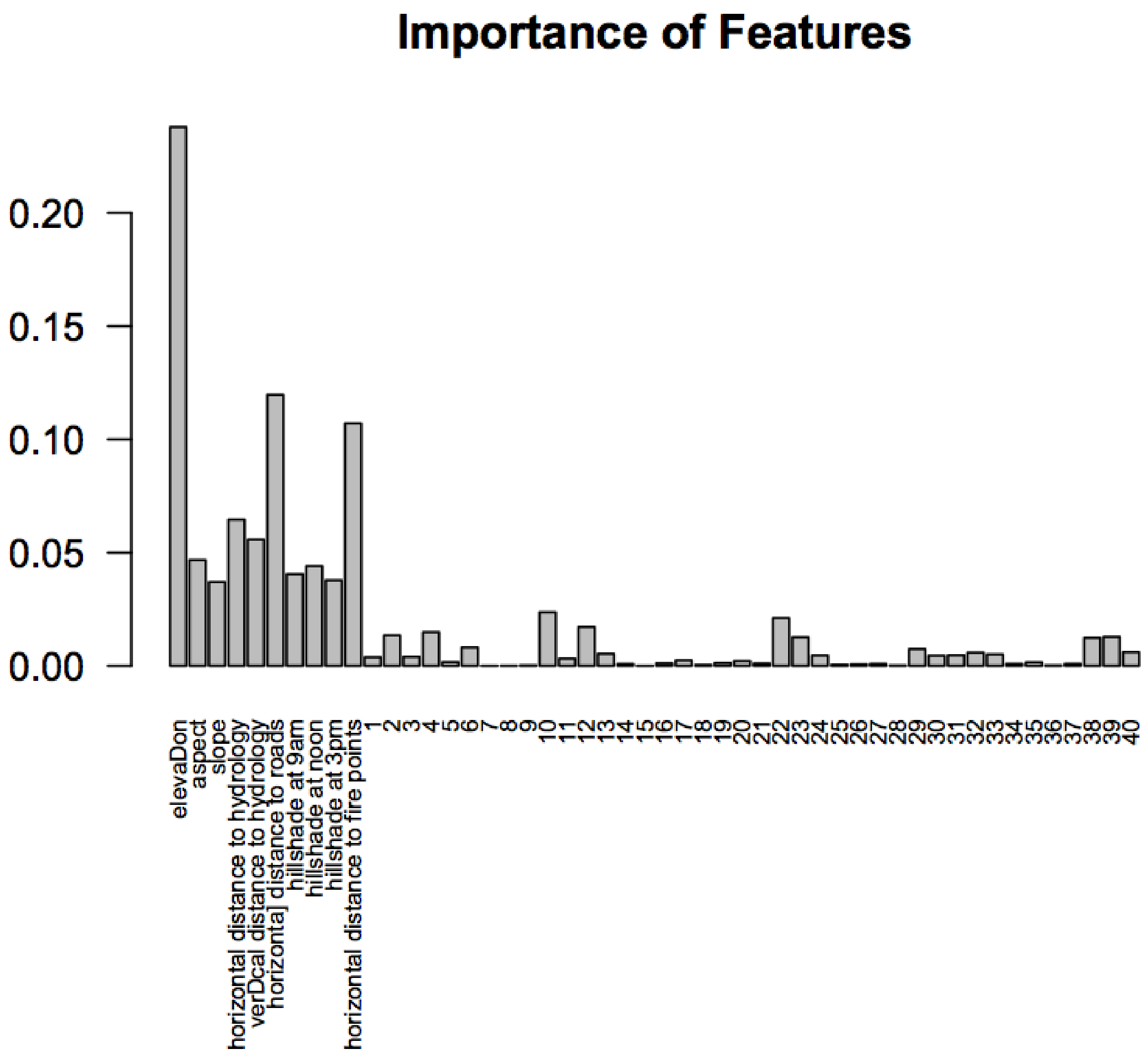


Figure 7. Evaluation of feature importance using their depth in the decision tree.

References

[1] Wehenkel, L. Discretization of continuous attributes for supervised learning: variance evaluation and variance reduction. *Proceedings of the International Fuzzy Systems Association World Congress, 381-388 (1997)*

[2] Pierre Geurts, Damien Ernst and Louis Wehenkel. Extremely randomized trees. *Mach Learn (2006) 63: 3-42*