



Random Forest Classification

Eric D. Stevens
June 13, 2019



Project Scope

What this project is:

- From scratch implementation of a known algorithm.
- Validation of work through replication of expected performance.
- Mimic performance of verified tools.



What this project is NOT:

- Invent a new machine learning algorithm.
- An effort to gain novel insight from some dataset.
- Tuning hyperparameters to get the best performance.



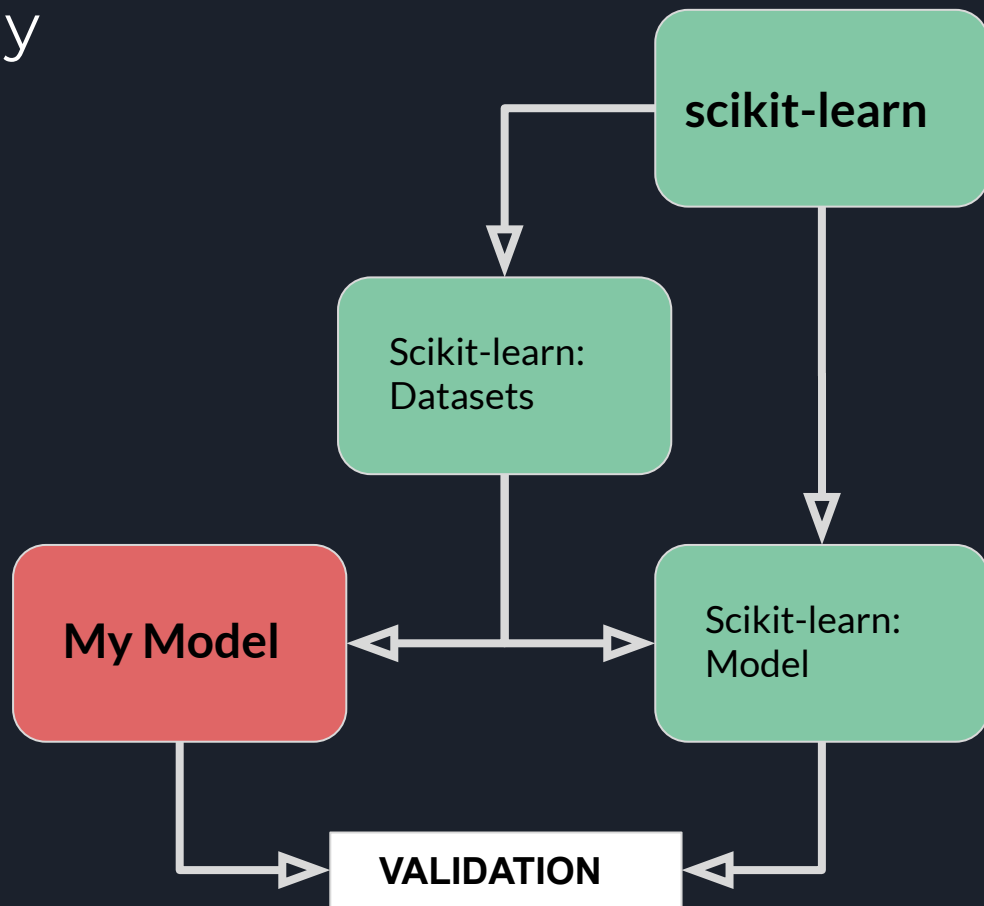
Execution Strategy

Building my model:

- Use only NumPy in my implementation.

Validating my performance

- Use scikit-learn data-sets.
- Validate against scikit-learn model implementation





The Random Forest Algorithm

Random Forest: Components

Decision Tree

- Recursive partitioning of dataset where partitions are more “pure”.

Bagging

- Bootstrap sampling of data instances selecting random subsets of features, resulting in a collection datasets that are random combinations of the original.

RANDOM FOREST

Building a decision tree for each bag and having them vote on a prediction





Decision Tree: Training

- Divide dataset into two partitions at each point for every dimension.
- At each one of these partitions, calculate the impurity.
- When the minimum impurity is found create the partitions of data and recursively call the function on each of these children.

Minimize the impurity equation:

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

k - dimension

t_k - threshold on dimension k

m - number of data points

$$G_i = 1 - \sum_{k=1}^n (p_{i,k})^2$$

p_{i,k} - proportion of points belonging to class k over total number of points.



Decision Tree: Regularization

Max_Depth: Stop branching after a tree reaches a certain depth. Prevents tree from making sure that every training point is correctly classified, thus overfitting to the training data.

Min_Leaf_Point: Prevents leaf nodes from being created if there are less than a certain number of training points that would fall in that leaf. Ensures that there is reasonable evidence that a leaf should be created and is generalizable.

Note: There are a huge number of regularization hyperparameters that could be implemented here but they generally accomplish similar goals. I mention the above because they are the ones I have implemented and are sufficient for high performance decision trees.



Bagging

Bagging is simply sampling training instances with replacement. In the context of Random Forest, the number of samples taken is usually the size of the training set.

For the training instances that are sampled from only a subset of features is selected.

This is repeated many times to obtain a collection (bag) of data sets that are bootstrapped instances with subsets of features.

Random Forest

Random forest is simply the decision tree algorithm applied to the data sets in the bag.

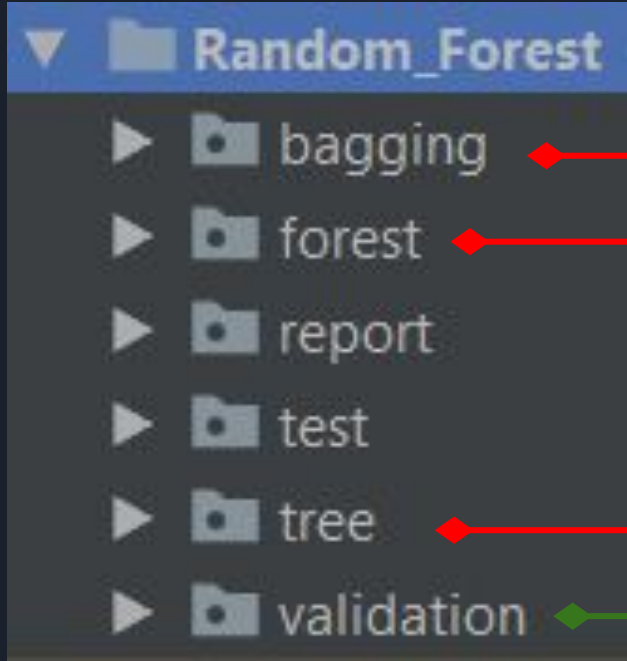
A voting scheme is used (in this project soft voting is used)



My Random Forest Implementation

File Structure

No implementation files contain any calls to libraries other than Numpy and each other.



Decision Tree implementation

Bagging implementation files

Random Forest implementation

Validation scripts contain data and model loading from scikit-learn.

Validation scripts

Decision Tree Implementation

```
class Tree:
```

```
    def __init__(self,
                  max_depth: int = 10,
                  min_node_points = 1):
```

```
    def train(self, x: np.ndarray, y: np.ndarray):
        assert(x.shape[0] == y.shape[0])
        assert(y.shape[1] == 1)

        self.head = Node(data=x,
                          labels=y,
                          max_depth=self.max_depth,
                          min_node_points=self.min_node_points)
```

- Tree class is essentially a container.
- Tree class holds on to decision tree regularization parameters and holds the head node of the decision tree.
- The train function makes a call to the Node class which does the heavy lifting of building a decision tree.



Decision Tree Implementation: Node

The node class does all the heavy lifting of the decision tree formation. **It is the most significant script in the project!** It would be impossible for me to take the time to explain all that is going on in such a short presentation. I do want to point out one important aspect though.

A brute force search for the proper feature and threshold to split on with 'm' features and 'n' data points would be:

$$O(m * n^2)$$

However, with some clever manipulation of the data we are able to dramatically reduce the time complexity of the algorithm to:

$$O(m * n \log_2 n)$$



Reducing Node split runtime

```
# get array of the size of the data but with values  
# corresponding to sorted indices by column.  
sorted_indices = np.argsort(self.data, axis=0)
```

$$O(m * n \log_2 n)$$

This numpy function takes in a matrix (in this case our training data) and returns the a matrix of the same size but containing indices of data points sorted by their value in in each column. This action is where the runtime complexity come from.



Linear from here on out ->

After this a linear sweep across each sorted dimension can occur, updating the best purity as it goes as seen below.

```
# iterate through each sorted index updating split membership
for i in range(1, self.n):
    left_val = self.data[indices[i-1, 0], dim]
    right_val = self.data[indices[i, 0], dim]
    left_label_counts[self.labels[indices[i-1, 0], 0]] += 1
    right_label_counts[self.labels[indices[i, 0], 0]] -= 1
    cost = mini_gini(left_label_counts, right_label_counts)

    # if split results in better purity, keep it
    if cost < best_impurity:
        best_impurity = cost
        best_threshold = (left_val+right_val)/2
```




Bagging

The Bagging class is simple and not worth spending time on.

It's a class whose main function is to create and hold a list of so called `_Bag` objects. The operation of the bag object can be seen here.

Bagging class is initialized with data and the bags hold onto index.

```
class _Bag:

    def __init__(self,
                  data_size: int,
                  num_features: int,
                  max_features: int,
                  bootstrap_features: bool):

        # determine how many features will be used
        self.n_features = np.random.randint(low=1, high=max_features+1)

        # get the features
        self.features = np.random.choice(range(num_features),
                                         size=self.n_features,
                                         replace=bootstrap_features)

        # sample index range randomly
        self.indices = np.random.choice(range(data_size),
                                         size=data_size,
                                         replace=True)
```



Random Forest Implementation


Steps:

1. Accept input training data and labels
2. Instantiate a Bagging() class on the data
3. Generate a list of trained decision trees based on the baglist in the Bagging class.

NOW YOU HAVE A TRAINED RANDOM FOREST CLASSIFIER!

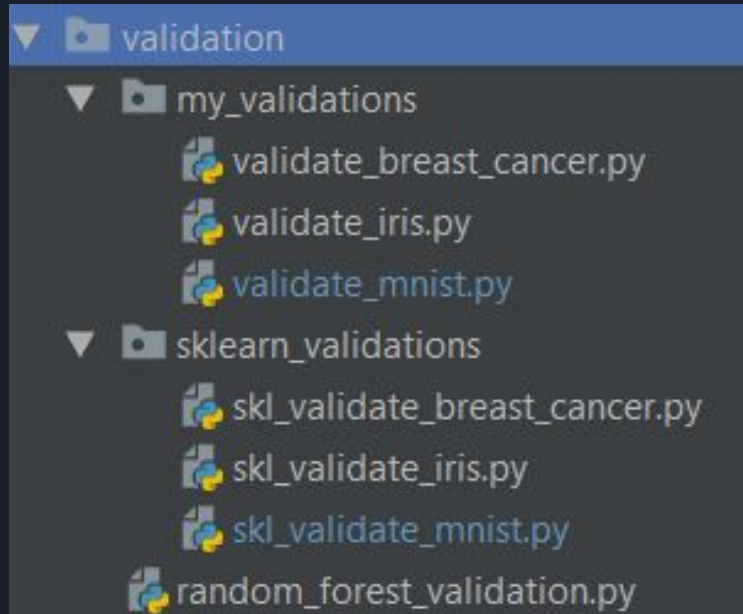
Notes:

- Soft voting is set by default but hard voting is implemented.
- Tree and Forest classes have predict methods that we did not have time to discuss the operation of.



Validation against scikit-learn

Validation Scripts



All the files in this section utilize scikit-learns data libraries for classification tasks.

The files in the my_validations directory are models trained and tested using only my code.

The files in the sklearn_validations directory are RandomForestClassifier() implementations with the declared hyperparameters that I have implemented in my code.

Effort was taken to maintain consistency between validation pipeline. Ideally, all that is different is the model used, mine or scikit-learns.



Results: Iris 1

(trees=30, max_depth=4, min_leaf_samples=3, max_features=2)

scikit-learn

scikit-learn

Training

```
[[39  0  0]
 [ 0 37  2]
 [ 0  0 34]]
```

Train Accuracy: 0.9821428571428571

Testing

```
[[11  0  0]
 [ 0 10  1]
 [ 0  1 15]]
```

Test Accuracy: 0.9473684210526315

my-forest

my-forest

Training

```
[[35.  0.  0.]
 [ 0. 38.  3.]
 [ 0.  2. 34.]]
```

train accuracy: 0.9553571428571429

Testing

```
[[15.  0.  0.]
 [ 0. 10.  1.]
 [ 0.  0. 12.]]
```

test accuracy: 0.9736842105263158



Results: Iris 2

(trees=30, max_depth=4, min_leaf_samples=3, max_features=2)

scikit-learn

```
scikit-learn
```

```
Training
```

```
[[37  0  0]
 [ 0 32  3]
 [ 0  1 39]]
```

```
Train Accuracy: 0.9642857142857143
```

```
Testing
```

```
[[13  0  0]
 [ 0 15  0]
 [ 0  0 10]]
```

```
Test Accuracy: 1.0
```

my-forest

```
my-forest
```

```
Training
```

```
[[37.  0.  0.]
 [ 0. 36.  2.]
 [ 0.  4. 33.]]
```

```
train accuracy: 0.9464285714285714
```

```
Testing
```

```
[[13.  0.  0.]
 [ 0.  9.  1.]
 [ 0.  1. 14.]]
```

```
test accuracy: 0.9473684210526315
```




Results: Breast Cancer

(trees=100, max_depth=8, min_leaf_samples=5, max_features=20)

scikit-learn

scikit-learn

Training

```
[[148  6]
 [ 2 270]]
```

Train Accuracy: 0.9812206572769953

Testing

```
[[51  7]
 [ 2 83]]
```

Test Accuracy: 0.9370629370629371

my-forest

my-forest

Training

```
[[154.  0.]
 [ 4. 268.]]
```

train accuracy: 0.9906103286384976

Testing

```
[[50.  3.]
 [ 4. 86.]]
```

test accuracy: 0.951048951048951

scikit-learn

Training

```
[[129  0  0  0  0  0  0  0  0  0]
 [  0 136  0  0  0  0  0  0  0  0]
 [  1  0 132  0  0  0  0  0  0  0]
 [  0  0  0 133  0  0  0  0  1  0]
 [  0  0  0  0 135  0  0  1  1  0]
 [  0  0  0  0  0 138  0  0  0  1]
 [  0  0  0  0  0  0 129  0  0  0]
 [  0  0  0  0  0  0  0 137  0  0]
 [  0  2  0  0  0  0  0  0  0 124]
 [  0  0  0  0  0  1  0  0  1 145]]
```

Train Accuracy: 0.9933184855233853

Testing

```
[[49  0  0  0  0  0  0  0  0  0]
 [  0 45  0  1  0  0  0  0  0  0]
 [  0  0 42  0  0  0  0  0  0  2]
 [  0  0  0 47  0  1  0  1  0  0]
 [  0  1  0  0 40  0  0  3  0  0]
 [  0  0  0  0  1 38  0  0  0  4]
 [  1  0  0  0  0  1 50  0  0  0]
 [  0  0  0  0  0  0  0 42  0  0]
 [  0  1  1  0  0  1  0  0 44  1]
 [  0  0  0  0  0  1  0  0  0 32]]
```

Test Accuracy: 0.9533333333333334

my-forest

Training

```
[[136.  0.  0.  1.  0.  1.  1.  0.  1.  0.]
 [  0. 128.  0.  1.  0.  0.  0.  0.  3.  0.]
 [  0.  1. 122.  2.  0.  0.  0.  0.  2.  0.]
 [  0.  0.  0. 134.  0.  1.  0.  0.  1.  1.]
 [  1.  0.  2.  0. 138.  1.  0.  1.  1.  0.]
 [  0.  0.  0.  0.  0. 132.  0.  0.  3.  2.]
 [  0.  0.  0.  1.  0.  1. 131.  0.  2.  0.]
 [  0.  0.  1.  3.  3.  0.  0. 133.  2.  8.]
 [  0.  0.  0.  0.  0.  0.  0.  0. 110.  0.]
 [  0.  0.  0.  1.  0.  3.  0.  0.  2. 129.]]
```

train accuracy: 0.9599109131403119

Testing

```
[[41.  0.  0.  1.  0.  0.  2.  0.  0.  0.]
 [  0. 49.  0.  0.  0.  0.  0.  0.  2.  0.]
 [  0.  1. 51.  2.  0.  0.  0.  0.  1.  1.]
 [  0.  0.  0. 35.  0.  1.  0.  0.  0.  2.]
 [  0.  1.  1.  0. 37.  0.  0.  1.  2.  0.]
 [  0.  1.  0.  1.  0. 42.  0.  1.  3.  2.]
 [  0.  1.  0.  0.  0.  0. 47.  0.  0.  0.]
 [  0.  0.  0.  0.  3.  0.  0. 43.  1.  1.]
 [  0.  0.  0.  1.  0.  0.  0.  0. 38.  0.]
 [  0.  0.  0.  0.  0.  0.  0.  0.  0. 34.]]
```

test accuracy: 0.9266666666666666

Digit
Recognition

trees=100,
max_depth=8,
min_leaf_samples=5,
max_features=10



Conclusion: SUCCESS!

It appears that the model implemented in this project is able to replicate the results of the same algorithms implemented in scikit-learn.

More extensive testing should be performed to ensure this, but I believe it is safe to say that the algorithm is functioning properly and the performance is good.

The random forest learning algorithm is an incredibly powerful classifier. I was surprised at the performance it was able to achieve, especially on the on the digit dataset.



Future for me

I want to learn more about the tradeoff between variance and bias in learning algorithms. This project leverages this concept heavily and my knowledge of the subject was not sufficient for me to truly grasp the ways the concepts were leveraged.

References

Title image:

<https://www.spotx.tv/resources/blog/developer-blog/exploring-random-forest-internal-knowledge-and-converting-the-model-to-table-form/>

Algorithm Development: <https://www.oreilly.com/library/view/hands-on-machine-learning/9781491962282/>



Thank you