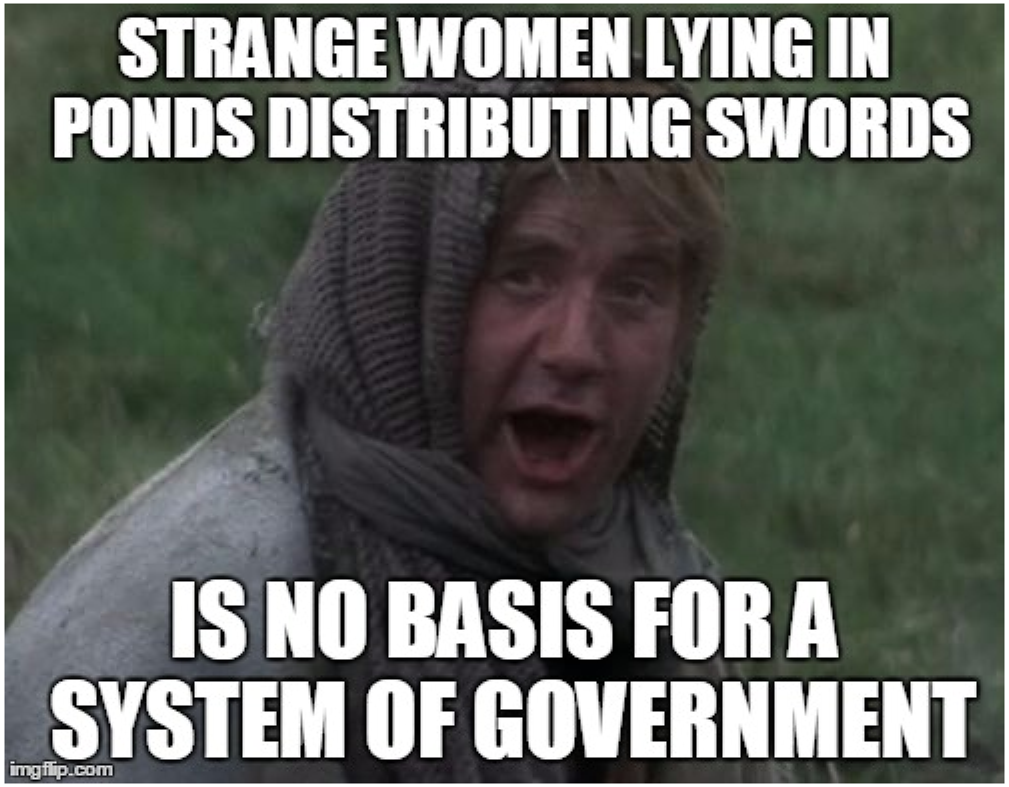
# Day 11: Random Forests, Other Ensembles

## Why Ensembles?

When we use single learning algorithms, we are very vulnerable to overfitting our model with respect to in-sample patterns, which reduces our ability to accurately model out-of-sample.

Who chose the next Roman emperor? Who chose the next king of France or England?

## Why Ensembles?


STRANGE WOMEN LYING IN PONDS DISTRIBUTING SWORDS
IS NO BASIS FOR A SYSTEM OF GOVERNMENT
imgflip.com

## Why Ensembles?

Who chose the next Roman emperor? Who chose the next king of France or England?

- The current emperor or king (mostly)

Why is this a bad idea?

- A single person (algorithm?) making a decision can easily make an error of judgement.
- If we choose a Nero, we end up with Rome in flames

## Why Ensembles?

How do most developed countries now choose leaders?

- They vote!

Why?

- NOT because we care if everyone has their opinion heard (see history of voting rights)
- Because large groups of people, when their opinions are averaged, make ~~good~~ better choices

## Why Ensembles?

I had a class of undergraduates who averaged 55% on their final exam.

On the other hand, a student who had chosen the most popular response to each question based on the responses of their classmates would have scored ~85%.

**In aggregate, poor students can select good answers**

## Why Ensembles?

This principle also applies to statistical learning algorithms. Aggregating "poor" algorithms can lead to a "good" algorithm.

Collections of learning algorithms are called **ensembles**.

## Bagging

Bagging (**B**ootstrap **Agg**regation) is a simple way to start creating an ensemble model.

Standard Model:

$$\hat{f}(x) = f^*(x)$$

All training data is used to generate our best estimate of the true functional form, $f(x)$.

## Bagging

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} f_b^*(x)$$

In bagging, each estimate utilizes a bootstrap (random) sample of the training data with each observation receiving probability $\frac{1}{N}$.

The bagged estimate is then based on the weighted average of all of the models.

## Boosting

If we boost an algorithm using $M$ stages, then we need to define $f_m(x)$ at each stage.

$$\hat{f}_0(x) = 0$$

At each subsequent stage, we solve for

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) + f_m^*(x)$$

So that each stage adds more information to our model.

## Boosting vs Bagging

**Bagging**:

- An averaged model utilizing bootstrapped samples of the complete dataset

**Boosting**:

- An additive model, where the predictions are incrementally improved

## Boosting vs Bagging

**Bagging**:

- Much easier to implement
- Less Overfitting

**Boosting**:

- Better Performance (generally)
- Still vulnerable to overfitting

## Tree Problems

One drawback to bagging can be illustrated by thinking about how decision trees are generated.

1. Find the biggest information gain
2. Split the tree
3. On each branch, find the next best information gain
4. Split again
5. Repeat 3 and 4 until stopping rule is reached (depth, purity, number of observations, etc.)

## Random Forests

Using bagging on decision trees in a situation where one variable is clearly superior to other inputs, the data itself will almost never allow a model to explore other inputs.

- The most informative input will mask the other options (always be chosen)
- Each tree in the bagging algorithm is highly correlated with the other trees
    - Permits overfitting, and reduces predictive power

## Random Forests

How can we alleviate this tendency?

- Restrict the inputs that the tree is allowed to choose from
- Bootstrap the sample
- Aggregate the forecasts to make a single, more accurate, prediction.

## Restricting Inputs

When a classification tree looks for maximum information gain, it searches across **all** available inputs.

Trees in a random forest are restricted to a random subset of inputs at each branching:

- Typically, $\sqrt{k}$ (where $k$ is the number of available parameters) inputs are provided at each branch
- When a new branch occurs, a new random subset of inputs is provided
- This is repeated for all branches on all trees

## Making a Classification

Once each tree in a random forest has been grown, we can use the trees to create a decision rule based on a vote by the classifiers:

- Each tree classifies an observation
- Whichever class receives the most votes (has highest predicted probability of being the true observed class) "wins," and is assigned as the predicted class for the observation

## MNIST Dataset

MNIST Handwriting Recognition Data

- Contains digits 0-9
- Full Dataset is 60,000 training observations, 10,000 testing observations
- 28 x 28 pixel images, (784 factors)
- Numbers are centered in the image
- Goal is to predict the written digit based on the image
- Great learning dataset

## Implementing Ensembles

We will implement Random Forests, Bagging, and Boosting using the `scikit-learn` module in Python, as we did for Decision Trees

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

## Prepare the Data

```python
# Read the handwriting MNIST dataset
data = pd.read_csv(
        '/home/dusty/DatasetsDA/mnist/trainingFull.csv')

# Separate the labels from the inputs
Y = data['Label']
X = data.drop('Label', axis=1)

# Randomly create train and test data
x, xt, y, yt = train_test_split(X, Y, test_size = 0.9,
        random_state=42)
```

## Baseline Decision Tree Classifier

```python
from sklearn.tree import DecisionTreeClassifier

# Generate the tree model
tree = DecisionTreeClassifier(max_depth=5,
        min_samples_leaf=10)
# Fit the tree to the training data
tclf = tree.fit(x, y)
# Make predictions
tpred = tclf.predict(xt)
# Print the accuracy score of the fitted model
print("\nThe decision tree has an accuracy of : %s\n"
        % str(accuracy_score(tpred, yt)))
```

Resulting in:

```
The decision tree has an accuracy of: 0.5795555555555556
```

## Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier

# Generate the random forest model
forest = RandomForestClassifier(n_estimators=100,
        n_jobs = -1, random_state=42)
# Fit the model to the training data
fclf = forest.fit(x, y)
# Make predictions
fpred = fclf.predict(xt)
# Print the accuracy score of the fitted model
print("The random forest has an accuracy of : %s\n"
        % str(accuracy_score(fpred, yt)))
```

Resulting in:

```
The random forest has an accuracy of: 0.8444444444444444
```

# Boosting Ensemble (of decision trees)

```python
from sklearn.ensemble import GradientBoostingClassifier

# Generate the boosting model
boost = GradientBoostingClassifier(n_estimators=100,
        max_depth=5, min_samples_leaf=10, random_state=42)
# Fit the model to the training data
boclf = boost.fit(x, y)
# Make predictions
bopred = boclf.predict(xt)
# Print the accuracy score of the fitted model
print("The boosting algorithm has an accuracy of : %s\n"
        % str(accuracy_score(bopred, yt)))
```

Resulting in:

```
The boosting algorithm has an accuracy of:
0.8095555555555556
```

## Bagging Ensemble (of decision trees)

```python
from sklearn.ensemble import BaggingClassifier

# Generate the bagging model
bag = BaggingClassifier(n_estimators=100, n_jobs = -1,
        random_state=42)
# Fit the model to the training data
baclf = bag.fit(x, y)
# Make predictions
bapred = baclf.predict(xt)
# Print the accuracy score of the fitted model
print("The bagging algorithm has an accuracy of : %s\n"
        % str(accuracy_score(bapred, yt)))
```

Resulting in:

```
The bagging algorithm has an accuracy of: 0.8011111111111
```

## Summary of Results

1. Decision Tree: 57.9%
2. Random Forest: 84.4%
3. Boosting Algorithm: 80.9%
4. Bagging Algorithm 80.1%

## Lab Time!