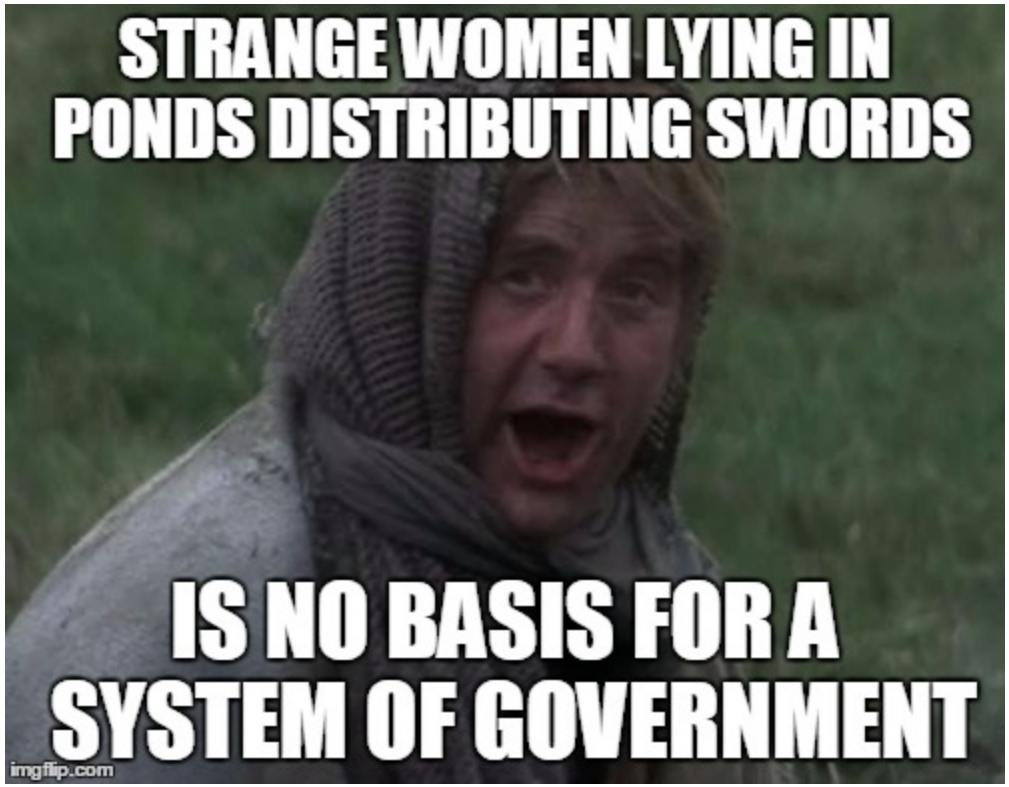
Random Forests, Ensembles

Why Ensembles?



Why Ensembles?

When we use single learning algorithms, we are very vulnerable to overfitting our model to in-sample variations, 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?

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 they care if everyone has their opinion heard (see history of voting rights for supporting evidence)
- Because large groups of people, when their opinions are averaged, make ~~good~~ better choices

Why Ensembles?

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

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

In aggregate, bad students can find 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:

$$f^*(x) = \frac{1}{n}\sum_{i=0}^{n} f_i(x)$$

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

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 with decision trees where one variable is clearly the best predictor, the data will 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
    - Leads to overfitting, and reduces predictive power

Random Forests

How can we alleviate this tendency?

- Restrict the variables that the tree is allowed to select
- 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 variables) 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

Implementing Ensembles

We will implement Random Forests (Bagging and Boosting are other tree-based ensemble methods, but we will just focus on a single model) using the `scikit-learn` module in Python, as we did for Decision Trees

```
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

```
# Read the handwriting MNIST dataset
data = pd.read_csv(
        'https://github.com/dustywhite7/pythonMikkeli/raw/'
        +'master/exampleData/passFailTrain.csv')

# Separate the labels from the inputs
Y = data['G3']
X = data.drop(['G3'], 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

```
from sklearn.tree import DecisionTreeClassifier

# Generate the tree model
tree = DecisionTreeClassifier()
# 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 : 0.6554307116104869
```

Random Forest Classifier

```
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 : 0.6816479400749064
```

Summary of Results

1. Decision Tree: 65.5%
2. Random Forest: 68.2%

This is a performance boost USING THE SAME DATA, meaning that we simply switch models and are likely to make better predictions!

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

Lab Time!