One Algorithm to Rule Them All

Jatin Mathur and Rohil Javeri

March 2020

Introduction

Intro i

Classification is one of the most prevalent types of machine learning problems out there. Currently, there is an implicit paradigm in solving any classification task: first try a variety of models (Logistic Regression, Support Vector Machine, Random Forest, Gradient Boosting, Artificial Neural Nets, etc.), and depending on your criteria (accuracy, speed, need for interpretability, etc.) you can pick one.

If that description left a sour taste in your mouth, that was the intention! The entire process seems bloated, arbitrary, and contrived. **Shouldn't there by a more theoretical, statistical way of determining which model would do best?**

Intro ii

We hypothesize that the aforementioned models have fundamental theoretical properties that result in stronger performance on some datasets. Our goal is to create an algorithm that can identify which model will perform best on a dataset prior to any model fitting. For the purpose of this hackathon, we contained the set of potential models to Logistic Regression (LR) and Random Forest (RF). We will be testing on binary classification datasets.

Why LR and RF? Both are relatively simple. LR has some great theoretical properties and makes very few assumptions about the data. RF can capture nonlinearities and interaction effects.

Theory intro - LR

LR has one assumption: that log-odds is a linear combination of the input feature space. Mathematically:

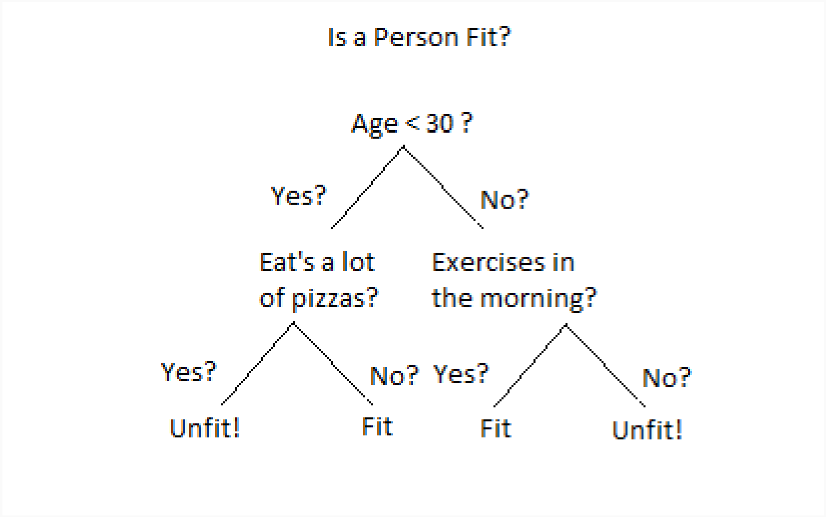
$$ln(\frac{p}{1-p}) = w^T x$$

where p represents $P(Y = 1|X)$, w the parameters to be trained, and x the input features.
This remarkably simple assumption leads to the following formulation:

$$p(y|x) = \frac{1}{1 + e^{-yw^T x}}$$

We can then frame our problem as maximizing the probability of the dataset (so picking the w that results in the largest product of the above expression for all y). **This is a convex optimization problem, meaning it can be solved to optimality using gradient descent.**

Theory intro - RF



Theory intro - RF (cont.)

Unlike LR, RF does not have any basis in probability. Decision trees are non-linear and can include interaction effects with other inputs, both of which LR cannot do.

RF typically minimizes Gini Impurity, which is a way to calculate how well the classifier has done at separating the target class. **Because this number is calculated at the node level it is a greedy algorithm and doesn't come with any theoretical guarentees.** We'll be forced to reason about how the algorithm proceeds, since the end result is difficult to know ahead of time.

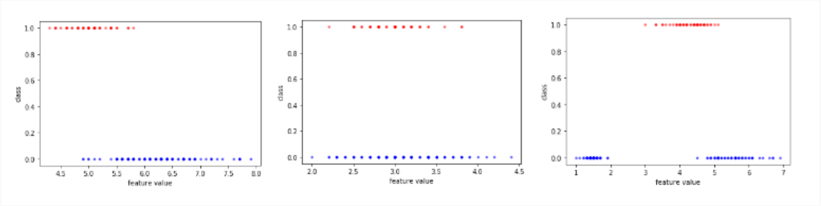
Existing Literature

There has been some research into this topic, and we've summarized their highlights below:
1. LR is more robust than RF when it comes to adding noise variables, adding explanatory variables, and increasing samples
2. RF outperforms LR usually, but when LR outperforms RF the difference is small.

While these are helpful, they do not help one know which model is best beforehand. Additionally, due to the nice theoretical properties of LR and the quick solve speed, we want to prioritize that option unless sufficient evidence indicates otherwise.

Methods

Feature-output distributions

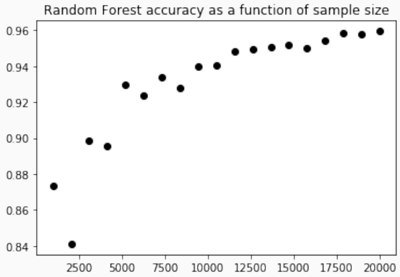


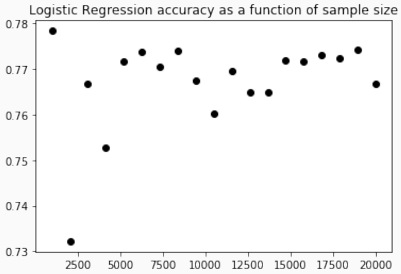
-left represents a *logit* situation, where as the feature value increases the class changes.
-middle represents a *half-logit* situation, where on one end of the feature value space one class dominates but on the other end both classes are present.
-right represents an *internally separable* situation, where one class dominates the interior, but not the edges, of a feature's distribution.

Last case is bad for LR, but all three are manageable by RF.

Runtime: $O(nf)$

n/f ratio

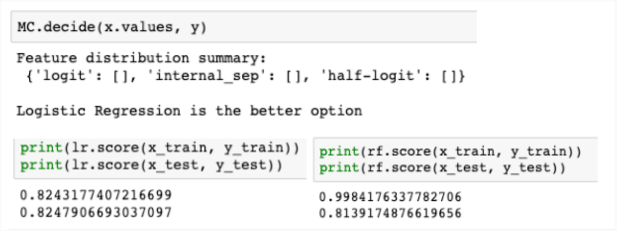




Random Forest is susceptible to overfit, as it lacks a guiding indication of when it has made enough splits or whether a split is significant. In the graph, you can see RF accuracy decreases when samples decreases. A paper we read suggested that an n/f ratio may be useful, where n is the number of samples and f the number of features.

Runtime: $O(1)$

Demo



```
MC.decide(x.values, y)
```

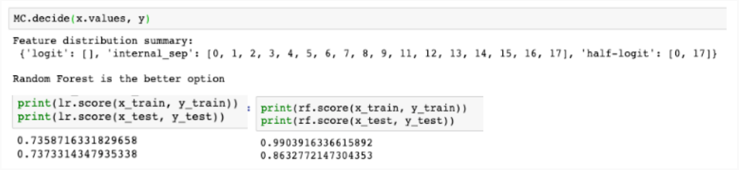
```
Feature distribution summary:
 {'logit': [], 'internal_sep': [], 'half-logit': []}

Logistic Regression is the better option
```

```
print(lr.score(x_train, y_train))
print(lr.score(x_test, y_test))
```

```
0.8243177407216699
0.8247906693037097
```

```
print(rf.score(x_train, y_train))
print(rf.score(x_test, y_test))
```

```
0.9984176337782706
0.8139174876619656
```



```
MC.decide(x.values, y)
```

```
Feature distribution summary:
 {'logit': [], 'internal_sep': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17], 'half-logit': [0, 17]}

Random Forest is the better option
```

```
print(lr.score(x_train, y_train))
print(lr.score(x_test, y_test))
```

```
0.7358716331829658
0.7373314347935338
```

```
print(rf.score(x_train, y_train))
print(rf.score(x_test, y_test))
```

```
0.9903916336615892
0.8632772147304353
```

Shortcomings

Our methods are not able to capture:
- information from interaction effects (RF can benefit from this)
- categorical variables (RF can use this directly)

Also:
- the scoring method is pretty arbitrary
- the n/f ratio could use more work

To be fair, modeling interaction effects would be difficult for anyone trying to do what we are, as the number of pairs of features to consider is $O(n^2)$. Additionally, categorical variables can be encoded using ordered data with respect to the output. The fact that LR can't (more like shouldn't) use these directly is kind of irrelevant.

References

1. https://scholar.smu.edu/cgi/viewcontent.cgi?article=1041&context=datasciencereview
2. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-018-2264-5