Google Play Store

And All Things Tree

The Goal

Model:

- Classify apps given certain information by how many stars they receive.
- Determine best modeling methods given modeling goals.

Why Trees?

- Trees do not model beyond a range of observed outcomes (no 6-star reviews).
- May reflect decision process users use to select apps.

Who is this for, and why?

Who

- App devs & Co.
- Investors

Why

- Predict the success of apps
- Help optimize resource allocation

The Data

- Data pulled directly from google play apps store [Kaggle]
- Contains various types of data, both categorical and numeric

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1	4.4 and up

All the Objects

Many of the columns in our dataset are categorical

We will need to create mutually exclusive ‘dummy variables’ for every category in each categorical feature

We are going to go from less than a dozen columns to over a hundred.

```
Index(['App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type',
       'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver',
       'Android Ver'],
      dtype='object')
App
8190
Category
33
Reviews
5990
Size
413
Installs
19
Type
2
Price
73
Content Rating
6
Genres
115
Last Updated
1299
Current Ver
2638
Android Ver
31
Number of unique ratings:  39
```

Prepare the Data

- Create dataframe (naturally)
- Assign numeric typing where applicable
- Create dummies for categorical variables

```
#Create Columns
data = gps_raw[['Category', 'Rating', 'Installs', 'Content Rating', 'Genres']]

#Cast columns w/numeric data to a numeric type
data['Reviews'] = gps_raw['Reviews'].astype(int)
data['Price'] = gps_raw['Price'].str.replace('$', '').astype(float)

#Create dummy variables for binary features
data = pd.get_dummies(data)

print('Final dataframe shape:\n{}'.format(data.shape))
```

```
Final dataframe shape:
(9360, 176)
```

On Decision Trees

$$S = -\sum_i P_i \log P_i$$

Asks a series of questions to travel down each ‘branch’ of the tree to arrive to the resultant ‘leaf node’

CART algorithm used, ID3 and C4.5

Splitting decisions based on informational entropy

Useful for dealing with dozens of mutually exclusive ‘dummy variables’

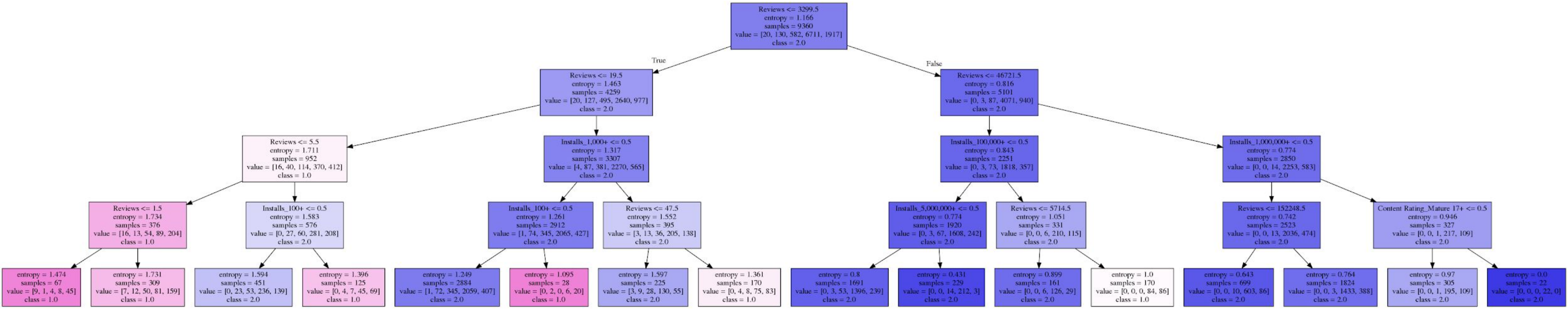
Tree Generating Algorithm

1. Check for the above base cases.
2. For each attribute *a*, find the normalized information gain ratio from splitting on *a*.
3. Let *a_best* be the attribute with the highest normalized information gain.
4. Create a decision *node* that splits on *a_best*.
5. Recur on the sublists obtained by splitting on *a_best*, and add those nodes as children of *node*.

ID3

```
ID3 (Examples, Target_Attribute, Attributes)
    Create a root node for the tree
    If all examples are positive, Return the single-node tree Root, with label = +.
    If all examples are negative, Return the single-node tree Root, with label = -.
    If number of predicting attributes is empty, then Return the single node tree Root,
    with label = most common value of the target attribute in the examples.
    Otherwise Begin
        A ← The Attribute that best classifies examples.
        Decision Tree attribute for Root = A.
        For each possible value, vᵢ, of A,
            Add a new tree branch below Root, corresponding to the test A = vᵢ.
            Let Examples(vᵢ) be the subset of examples that have the value vᵢ for A
            If Examples(vᵢ) is empty
                Then below this new branch add a leaf node with label = most common target value in the examples
            Else below this new branch add the subtree ID3 (Examples(vᵢ), Target_Attribute, Attributes – {A})
    End
    Return Root
```

The Decision Tree: Our Fundamental Building Block



Tree Validation

```
Model Score:  0.754059829059829

Fold Scores:
 [0.71878335 0.72183663 0.72795297 0.71779797 0.72688402]

Confusion Matrix:
 [[   8    0    0    3    9]
 [   2   12    0  104   12]
 [   0    2   35  501   44]
 [   1    0    9 6466  235]
 [   0    1    5 1374  537]]

--- 0.49192023277282715 seconds ---
```

Used a tree with max branch length of 8.

Error term not evenly distributed across classes.

Larger trees produce more accurate results, but become exponentially more computationally intensive.

Large trees are also prone to overfitting.

So how best do we use trees?

1st Ensemble Model: Random Forest

Ensemble Model: Model constructed from a series of smaller models

Random Forest: Many simple trees are generated from random samples (with replacement) of the data. The trees then vote on what class an observation belongs to.

Pros:
- Strong Performance
- Requires Minimal Setup and Feature Engineering

Cons:
- Large Forests may become memory intensive
- ‘Black Box’, Final model not ‘human readable’

Random Forest Validation

Terrible Overfitting

However, error term appears more evenly distributed across all classes than decision tree

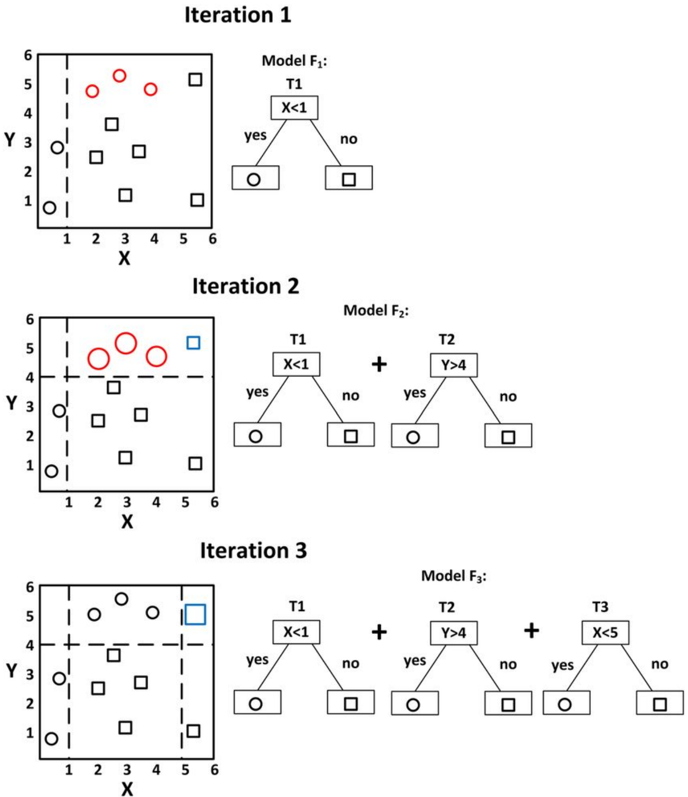
Parameters:

- n_trees = 100
- Max depth of 3

```
Model Score:  0.9842948717948717

Fold Scores:
 [0.67929562 0.72450614 0.6707643  0.63495457 0.64831641]

Confusion Matrix:
 [[  15    1    0    1    3]
 [   1  113    1    8    7]
 [   0    1  558   17    6]
 [   3    1   11 6652   44]
 [   1    1    9   31 1875]]


--- 10.244133949279785 seconds ---
```

2nd Ensemble Classifier: Gradient Boosting Classifier

Gradient Boosting: For each class in the outcome, a tree is fit to the training data to decide if a given observation belongs to its class. Once the model is complete, misclassed data is used to train subsequent trees.



Gradient Boosting Validation

- Model Score and fold scores are more consistent, implying less overfitting.
- Error terms fairly evenly distributed.
- Much slower than other algorithms

Parameters:

- 100 iterations
- Max tree depth of 3

```
Model Score:  0.7539529914529914

Fold Scores:
 [0.72145144 0.72023492 0.73597007 0.72421165 0.74184928]

Confusion Matrix:
 [[   9    0    0    4    7]
 [   1   13    0  102   14]
 [   0    0   16  516   50]
 [   0    0    2 6595  114]
 [   0    0    0 1493  424]]

--- 61.81561470031738 seconds ---
```

Regression, a Quick Note

Wow, these scores are terrible!

- Terrible Accuracy
- Rampant Overfitting

How to fix:

- Feature Selection
- Hyper Parameters

```
Depth 8 Decision Tree Regressor:

Model Score:  0.20128952739572614

Fold Scores:
 [-0.06282406 -0.13975333  0.08539844 -0.17601475 -0.00258484]

Random Forest Regressor:

Model Score:  0.7785501924240102

Fold Scores:
 [-0.12155964 -0.0482767   0.0051703  -0.10011518 -0.16049563]

Gradient Boosting Regressor:

Model Score:  0.18909181643942818

Fold Scores:
 [0.04558991 0.04076426 0.12334975 0.05618297 0.06923814]
```

Future Work

Focus on Gradient Boosting

- Less overfitting than RF.
- More accurate than D-tree

More Feature Selection

- K best features (Chi squared, ANOVA)

Model Implementation

- Iterate through (or sample) combinations of features to find the best result.