Random Forests & XGBoost

Fartash Faghri
University of Toronto
CSC2515, Fall 2019
Decision Trees

- Handles tabular data
- Features can be of any type (discrete, categorical, raw text, etc)
- Features can be of different types
- No need to “normalize” features
- Too many features? DTs can be efficient by looking at only a few.
- Easy to interpret
Decision Trees

Input: age, gender, occupation, ...

Does the person like computer games

age < 15

is male?

Y

N

Y

N

prediction score in each leaf

+2

+0.1

-1

(XGBoost slides)
Random Forests

Average multiple decision trees

- **tree1**
  - age < 15
    - Y: is male?
      - Y: +2
      - N: +0.1
    - N: -1

- **tree2**
  - Use Computer Daily
    - Y: +0.9
    - N: -0.9

\[ f(\text{Y, is male?}) = 2 + 0.9 = 2.9 \]
\[ f(\text{N, is male?}) = -1 - 0.9 = -1.9 \]

(XGBoost slides)
Random Forests

Scikit-learn ipynb

Scikit-learn official docs

Tabular data example and ipynb
One DT Overfits

```python
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```
Averaging DTs in a Random Forest

```python
>>> from sklearn.ensemble import RandomForestClassifier
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = RandomForestClassifier(n_estimators=10)
>>> clf = clf.fit(X, Y)
```
Train Model

```python
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor

# Instantiate model
rf = RandomForestRegressor(n_estimators=1000, random_state=42)

# Train the model on training data
rf.fit(train_features, train_labels);
```

max_depth : The maximum depth of the tree.
Make Predictions on Test Data

In [14]:

# Use the forest's predict method on the test data
predictions = rf.predict(test_features)

# Calculate the absolute errors
errors = abs(predictions - test_labels)

# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees."

Mean Absolute Error: 3.83 degrees.
Randomization methods

Each DT is trained on:

- Random subset of training data (sklearn.ensemble.RandomForestClassifier)
- Random subset of features (sklearn.ensemble.RandomForestClassifier)
- Noisy thresholds (sklearn.ensemble.ExtraTreesClassifier)

*Parallel construction and prediction*
Tabular Data

Ipython notebook example end post
In [1]: # Pandas is used for data manipulation
import pandas as pd

# Read in data as pandas dataframe and display first 5 rows
features = pd.read_csv('data/temps.csv')
features.head(5)

Out[1]:
<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>day</th>
<th>week</th>
<th>temp_2</th>
<th>temp_1</th>
<th>average</th>
<th>actual</th>
<th>friend</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016</td>
<td>1</td>
<td>1</td>
<td>Fri</td>
<td>45</td>
<td>45</td>
<td>45.6</td>
<td>45</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>2016</td>
<td>2</td>
<td>Sat</td>
<td>44</td>
<td>45</td>
<td>45.7</td>
<td>44</td>
<td>61</td>
</tr>
<tr>
<td>2</td>
<td>2016</td>
<td>3</td>
<td>Sun</td>
<td>45</td>
<td>44</td>
<td>45.8</td>
<td>41</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>2016</td>
<td>4</td>
<td>Mon</td>
<td>44</td>
<td>41</td>
<td>45.9</td>
<td>40</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>2016</td>
<td>5</td>
<td>Tues</td>
<td>41</td>
<td>40</td>
<td>46.0</td>
<td>44</td>
<td>41</td>
</tr>
</tbody>
</table>
In [2]: print('The shape of our features is:', features.shape)

The shape of our features is: (348, 9)

In [3]: # Descriptive statistics for each column
features.describe()

<table>
<thead>
<tr>
<th></th>
<th>year</th>
<th>month</th>
<th>day</th>
<th>temp_2</th>
<th>temp_1</th>
<th>average</th>
<th>actual</th>
<th>friend</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>348.0</td>
<td>348.0</td>
<td>348.0</td>
<td>348.0</td>
<td>348.0</td>
<td>348.0</td>
<td>348.0</td>
<td>348.0</td>
</tr>
<tr>
<td>mean</td>
<td>2016.0</td>
<td>6.477011</td>
<td>15.514368</td>
<td>62.511494</td>
<td>62.560345</td>
<td>59.760632</td>
<td>62.543103</td>
<td>60.034483</td>
</tr>
<tr>
<td>std</td>
<td>0.0</td>
<td>3.498380</td>
<td>8.772982</td>
<td>11.813019</td>
<td>11.767406</td>
<td>10.527306</td>
<td>11.794146</td>
<td>15.626179</td>
</tr>
<tr>
<td>min</td>
<td>2016.0</td>
<td>1.000000</td>
<td>1.000000</td>
<td>35.000000</td>
<td>35.000000</td>
<td>45.100000</td>
<td>35.000000</td>
<td>28.000000</td>
</tr>
<tr>
<td>25%</td>
<td>2016.0</td>
<td>3.000000</td>
<td>8.000000</td>
<td>54.000000</td>
<td>54.000000</td>
<td>49.975000</td>
<td>54.000000</td>
<td>47.750000</td>
</tr>
<tr>
<td>50%</td>
<td>2016.0</td>
<td>6.000000</td>
<td>15.000000</td>
<td>62.500000</td>
<td>62.500000</td>
<td>58.200000</td>
<td>62.500000</td>
<td>60.000000</td>
</tr>
<tr>
<td>75%</td>
<td>2016.0</td>
<td>10.000000</td>
<td>23.000000</td>
<td>71.000000</td>
<td>71.000000</td>
<td>69.025000</td>
<td>71.000000</td>
<td>71.000000</td>
</tr>
<tr>
<td>max</td>
<td>2016.0</td>
<td>12.000000</td>
<td>31.000000</td>
<td>92.000000</td>
<td>92.000000</td>
<td>77.400000</td>
<td>92.000000</td>
<td>95.000000</td>
</tr>
</tbody>
</table>
Data preparation

One-hot encoding:

<table>
<thead>
<tr>
<th>week</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tue</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wed</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Thu</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Fri</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

# One-hot encode categorical features
features = pd.get_dummies(features)
Data preparation

Why should be prepare data?

Any other encoding?

We also use one-hot encoding in cross-entropy, remember why? (Tutorial 2)
Feature importance

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node.

# Get numerical feature importances
importances = list(rf.feature_importances_)
Model with Two Most Important Features

```python
# New random forest with only the two most important variables
rf_most_important = RandomForestRegressor(n_estimators= 1000, random_state=42)

# Extract the two most important features
important_indices = [feature_list.index('temp_1'), feature_list.index('average')]
train_important = train_features[:, important_indices]
test_important = test_features[:, important_indices]

# Train the random forest
rf_most_important.fit(train_important, train_labels)
```
AdaBoost

Lecture 4 (Slides 43-70)

- Additive models
- Exponential loss \( L(y, \hat{y}) = \exp(-\hat{y}y) \)

Weak-learners can be any function, e.g. decision trees and decision stumps.
Gradient Boosting

Additive model with loss $L$:

$$\min_{\alpha_{n=1:N}, \beta_{n=1:N}} L \left( y, \sum_{n=1}^{N} \alpha_n f(x, \beta_n) \right)$$

GB approximately solves this objective iteratively and greedily:

$$\min_{\alpha_n, \beta_n} L \left( y, f_{n-1}((x) + \alpha_n f_n(x, \beta_n)) \right)$$

Elements of Statistical learning (Chapter 10.10)
XGBoost (A Scalable Tree Boosting System)

Gradient Tree Boosting with Regularization

Parallelization construction on CPU cores

Distributed training on a cluster of machines (large models)

Out-of-Core computing (large datasets that do not fit in memory)
import xgboost as xgb

# read in data
dtrain = xgb.DMatrix('demo/data/agaricus.txt.train')
dtest = xgb.DMatrix('demo/data/agaricus.txt.test')

# specify parameters via map
param = {'max_depth':2, 'eta':1, 'silent':1, 'objective':'binary:logistic'}
num_round = 2
bst = xgb.train(param, dtrain, num_round)

# make prediction
preds = bst.predict(dtest)
XGBoost

Slides (formulation on slides 17-26)

Recorded Presentation

Official Examples

XGBoost Official Documentation

Basic walk-through