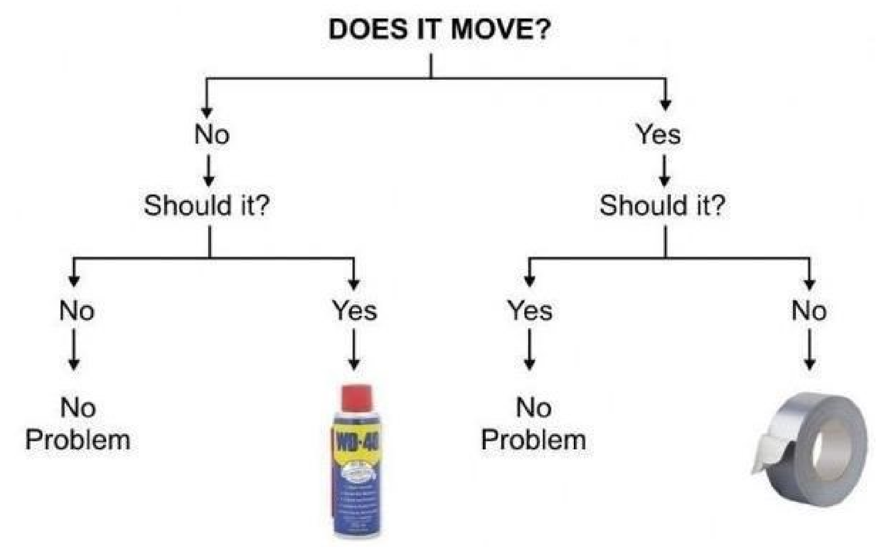
InDepth: Parameter tuning for Decision Tree

Mohtadi Ben Fraj Follow
Dec 20, 2017 · 5 min read



In this post we will explore the most important parameters of Decision tree model and how they impact our model in term of over-fitting and under-fitting.

We will use the Titanic Data from kaggle. For the sake of this post, we will perform as little feature engineering as possible as it is not the purpose of this post.

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Load train data

```
# get titanic & test csv files as a DataFrame
train = pd.read_csv(“input/train.csv”)
print train.shape
 > (891, 12)
```

Check for missing values

```
#Checking for missing data
NAs = pd.concat([train.isnull().sum()], axis=1, keys=
['Train'])
NAs[NAs.sum(axis=1) > 0]
```

```
>Age 177
Cabin 687
Embarked 2
```

We will remove 'Cabin', 'Name' and 'Ticket' columns as they require some processing to extract useful features

```
# At this point we will drop the Cabin feature since it is
missing a lot of the data
train.pop('Cabin')
```

```
train.pop('Name')
```

```
train.pop('Ticket')
```

```
train.shape
> (891, 9)
```

Fill the missing age values by the mean value

```
# Filling missing Age values with mean
train['Age'] = train['Age'].fillna(train['Age'].mean())
```

Fill the missing 'Embarked' values by the most frequent value

```
# Filling missing Embarked values with most common value
train['Embarked'] =
train['Embarked'].fillna(train['Embarked'].mode()[0])
```

'Pclass' is a categorical feature so we convert its values to strings

```
train['Pclass'] = train['Pclass'].apply(str)
```

Let’s perform a basic one hot encoding of categorical features

```
# Getting Dummies from all other categorical vars
for col in train.dtypes[train.dtypes == 'object'].index:
 for_dummy = train.pop(col)
 train = pd.concat([train, pd.get_dummies(for_dummy,
prefix=col)], axis=1)
```

```
# Prepare data for training models
labels = train.pop('Survived')
```

For testing, we choose to split our data to 75% train and 25% for test

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(train,
labels, test_size=0.25)
```

Let’s first fit a decision tree with default parameters to get a baseline idea of the performance

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
```

```
dt.fit(x_train, y_train)

> DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=None,
 max_features=None, max_leaf_nodes=None,
 min_impurity_split=1e-07, min_samples_leaf=1,
 min_samples_split=2, min_weight_fraction_leaf=0.0,
 presort=False, random_state=None, splitter='best')

y_pred = dt.predict(x_test)
```

We will use AUC (Area Under Curve) as the evaluation metric. Our target value is binary so it's a binary classification problem. AUC is a good way for evaluation for this type of problems.

```
from sklearn.metrics import roc_curve, auc

false_positive_rate, true_positive_rate, thresholds =
roc_curve(y_test, y_pred)

roc_auc = auc(false_positive_rate, true_positive_rate)

roc_auc
> 0.72657348804500699
```

max_depth

The first parameter to tune is *max_depth*. This indicates how deep the tree can be. The deeper the tree, the more splits it has and it captures more information about the data. We fit a decision tree with depths ranging from 1 to 32 and plot the training and test auc scores.

```
max_depths = np.linspace(1, 32, 32, endpoint=True)

train_results = []
test_results = []
for max_depth in max_depths:
   dt = DecisionTreeClassifier(max_depth=max_depth)
   dt.fit(x_train, y_train)

   train_pred = dt.predict(x_train)

   false_positive_rate, true_positive_rate, thresholds =
roc_curve(y_train, train_pred)
```

```
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous train results
   train_results.append(roc_auc)

   y_pred = dt.predict(x_test)

   false_positive_rate, true_positive_rate, thresholds =
roc_curve(y_test, y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   # Add auc score to previous test results
   test_results.append(roc_auc)

from matplotlib.legend_handler import HandlerLine2D

line1, = plt.plot(max_depths, train_results, ‘b’,
label=”Train AUC”)
line2, = plt.plot(max_depths, test_results, ‘r’, label=”Test
AUC”)

plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})

plt.ylabel(‘AUC score’)
plt.xlabel(‘Tree depth’)
plt.show()
```

We see that our model overfits for large depth values. The tree perfectly predicts all of the train data, however, it fails to generalize the findings for new data

min_samples_split

min_samples_split represents the minimum number of samples required to split an internal node. This can vary between considering at least one sample at each node to considering all of the samples at each node. When we increase this parameter, the tree becomes more constrained as it has to consider more samples at each node. Here we will vary the parameter from 10% to 100% of the samples

```
min_samples_splits = np.linspace(0.1, 1.0, 10, 
endpoint=True)

train_results = []
test_results = []
for min_samples_split in min_samples_splits:
   dt = 
DecisionTreeClassifier(min_samples_split=min_samples_split)
   dt.fit(x_train, y_train)

   train_pred = dt.predict(x_train)

   false_positive_rate, true_positive_rate, thresholds = 
roc_curve(y_train, train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)

   y_pred = dt.predict(x_test)

   false_positive_rate, true_positive_rate, thresholds = 
roc_curve(y_test, y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

from matplotlib.legend_handler import HandlerLine2D

line1, = plt.plot(min_samples_splits, train_results, ‘b’, 
label=”Train AUC”)
line2, = plt.plot(min_samples_splits, test_results, ‘r’, 
label=”Test AUC”)

plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})

plt.ylabel(‘AUC score’)
plt.xlabel(‘min samples split’)
plt.show()
```

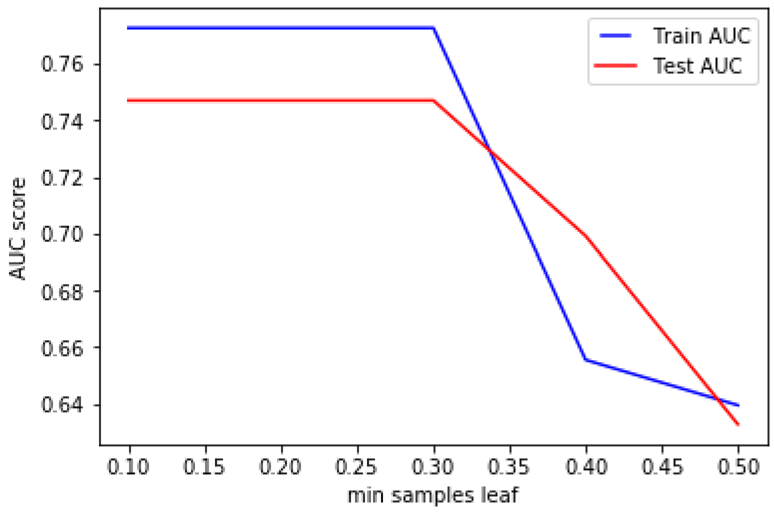
We can clearly see that when we consider 100% of the samples at each node, the model cannot learn enough about the data. This is an underfitting case.

min_samples_leaf

min_samples_leaf is The minimum number of samples required to be at a leaf node. This parameter is similar to *min_samples_splits*, however, this describe the minimum number of samples of samples at the leafs, the base of the tree.

```
min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint=True)

train_results = []
test_results = []
for min_samples_leaf in min_samples_leafs:
   dt = 
DecisionTreeClassifier(min_samples_leaf=min_samples_leaf)
   dt.fit(x_train, y_train)

   train_pred = dt.predict(x_train)

   false_positive_rate, true_positive_rate, thresholds = 
roc_curve(y_train, train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)

   y_pred = dt.predict(x_test)

   false_positive_rate, true_positive_rate, thresholds = 
roc_curve(y_test, y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)
```

```
from matplotlib.legend_handler import HandlerLine2D

line1, = plt.plot(min_samples_leafs, train_results, 'b',
label="Train AUC")
line2, = plt.plot(min_samples_leafs, test_results, 'r',
label="Test AUC")

plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})

plt.ylabel('AUC score')
plt.xlabel('min samples leaf')
plt.show()
```

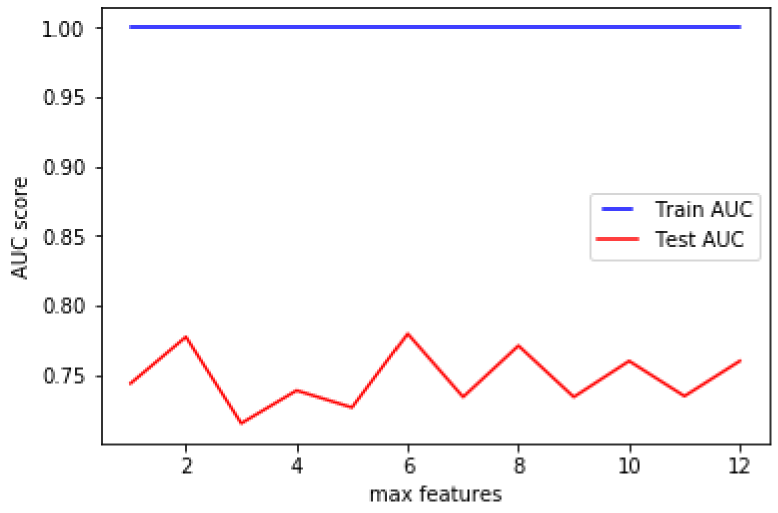


Same conclusion as to previous parameter. Increasing this value may cause underfitting.

max_features

max_features represents the number of features to consider when looking for the best split.

```
max_features = list(range(1,train.shape[1]))

train_results = []
test_results = []
for max_feature in max_features:
   dt = DecisionTreeClassifier(max_features=max_feature)
   dt.fit(x_train, y_train)

   train_pred = dt.predict(x_train)
   false_positive_rate, true_positive_rate, thresholds = 
roc_curve(y_train, train_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   train_results.append(roc_auc)

   y_pred = dt.predict(x_test)

   false_positive_rate, true_positive_rate, thresholds = 
roc_curve(y_test, y_pred)
   roc_auc = auc(false_positive_rate, true_positive_rate)
   test_results.append(roc_auc)

from matplotlib.legend_handler import HandlerLine2D

line1, = plt.plot(max_features, train_results, ‘b’, 
label=”Train AUC”)
line2, = plt.plot(max_features, test_results, ‘r’, 
label=”Test AUC”)

plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})

plt.ylabel(‘AUC score’)
plt.xlabel(‘max features’)
plt.show()
```



This is also an overfitting case. It’s unexpected to get overfitting for all values of max_features. However, according to sklearn documentation for decision tree, the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.

· · ·

The inDepth series investigates how model parameters affect performance in term of overfitting and underfitting. This is the first post in the series. Next one will be for Random Forrest.