Boosted trees are better than trees generated by random forest algorithm in some cases.

Boosted Tree Intuition

Given training set of size m

For $b = 1$ to B:

Use sampling with replacement to create a new training set of size m

Train a decision tree on the new dataset

Ear shape	Face shape	Whiskers	Prediction	
Pointy	Round	Present	Cat	✓
Floppy	Not Round	Present	Not cat	✗
Floppy	Round	Absent	Not cat	✓
Pointy	Not Round	Present	Not cat	✓
Pointy	Round	Present	Cat	✓
Pointy	Round	Absent	Not cat	✗
Floppy	Not Round	Absent	Not cat	✓
Pointy	Round	Absent	Not cat	✗
Floppy	Round	Absent	Not cat	✓
Floppy	Round	Absent	Not cat	✓

← misclassified

← misclassified

← misclassified

Instead of picking up from all examples the algorithm focuses more on samples that were "misclassified" by the previous trees

Boosted trees can be hard to implement because they are very susceptible to overfitting because each tree is a new tree is linked to the previous one because it tries to solve its mistake.

Since Random Forest Algorithm didn't have its trees linked to the previous one, it was better at preventing overfitting.

Luckily, an algorithm :- XGBoost helps with this.

XGBoost (eXtreme Gradient Boosting)

- Open source implementation of boosted trees
- Fast efficient implementation
- Good choice of default splitting criteria and criteria for when to stop splitting
- Built in regularization to prevent overfitting } prevents overfitting
- Highly competitive algorithm for machine learning competitions (eg: Kaggle competitions)

Rather than bootstrap sampling, XGBoost assigns different weights to different examples so that it doesn't need to generate a new training set and makes it even more efficient.

But the overall intuition discussed previously remains correct in terms of "how XGBoost is choosing examples to focus on"

Using XGBoost

Classification

```
from XGBoost import XGBClassifier
model = XGBClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

initializing the model

Regression

```
from XGBoost import XGBRegressor
model = XGBRegressor()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```