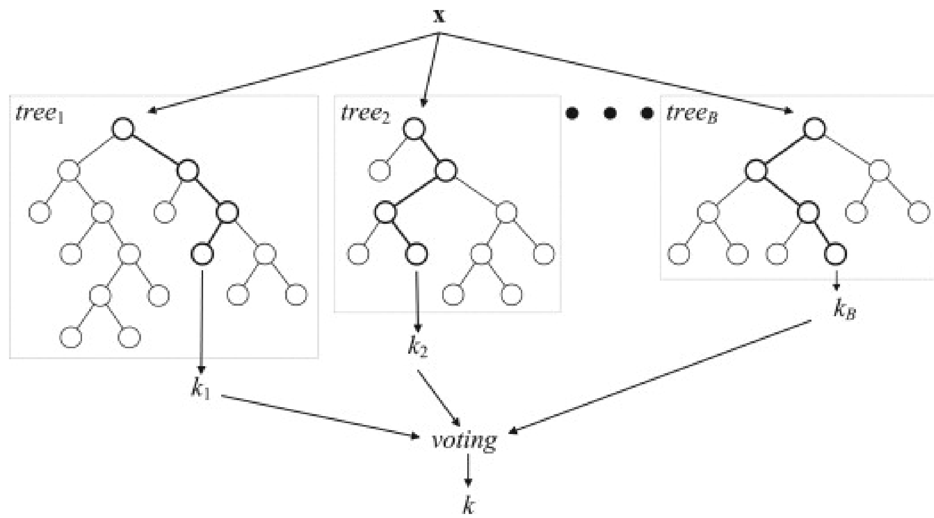
Introduction to Data Science

(Decision Trees and) Random Forests



Random Forests

- Train an "**ensemble**" of decisions trees
- RF decision is
  - the mean of decisions by individual trees (regression)
  - the majority vote of decisions by individual trees (classification)

Pros

- Reduces variance of individual decision tree models
- Can rank the importance of features in a natural way
- Can operate on many (1000s) of features without dimensionality reduction
- Can train well with relatively few samples

Cons

- Loss of interpretability, compared to individual decision trees
- Not as good at regression:
  - Output is not truly continuous valued, but rather, discretized (less problematic than for individual trees, however)

Random Forest trees are different from regular decision trees

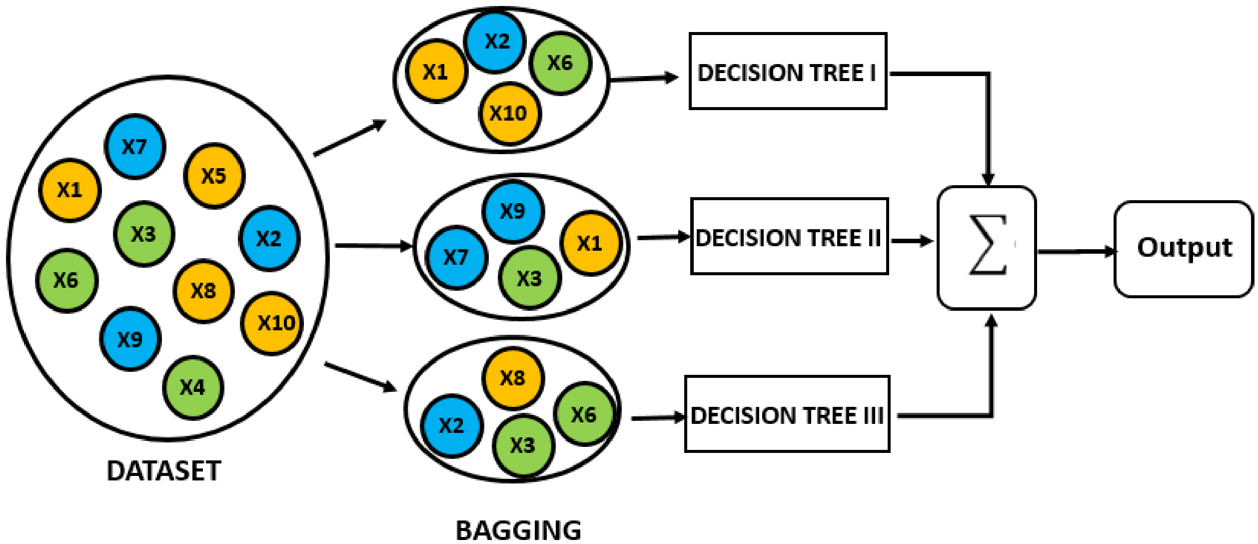
If the same training set and method is used for each tree, all the trees will be correlated (produce nearly identical outputs) and nothing will be gained by having an ensemble.

Use two forms of **randomization** when training each tree, to **reduce tree correlation**.

1. **Bagging**
2. **Feature randomization** (**feature bagging**)

Bagging (bootstrap aggregating)

The training set for each tree is B samples randomly drawn from the full training set, **with replacement**.



Bonus: We can get an **"out-of-bag" (OOB)** score from our training set:
For a given sample, we make a prediction using subset of trees that were not trained with that sample. This is a from of cross-validation.

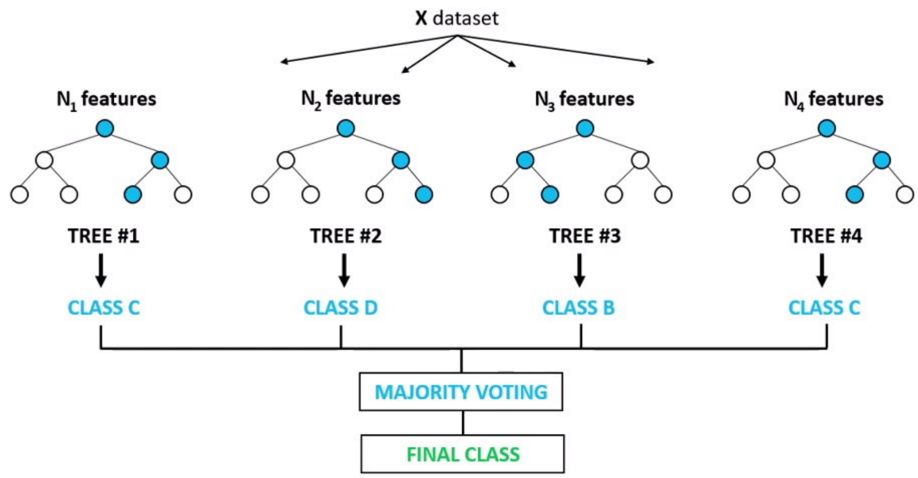
Feature subset randomization

During training of **individual RF trees**, a random subset of **features** is selected at each branch split in the tree. In regular decision trees, all features are considered.

The figure below doesn't quite do this justice. It shows a subset of features being using for each tree.

However, in RF training of a single tree, **a random subset of features is selected at each split, and from those features one is then selected as the feature on which the split the data** (based on Gini impurity or some other metric).

Random Forest Classifier



We'll work with the Titanic data set again

In [5]:

```python
# Get necessary packages
import sklearn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import graphviz # needed to visualize trained decision tree
from sklearn.tree import export_graphviz # needed to visualize trained decision tr
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import tree

np.random.seed(1000)

# Set up for plotting
plt.style.use("ggplot")
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# Read in our data and glance at its formatting
df = pd.read_csv('titanic_data.csv')
df.head()
```

Out[5]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500

In [6]:

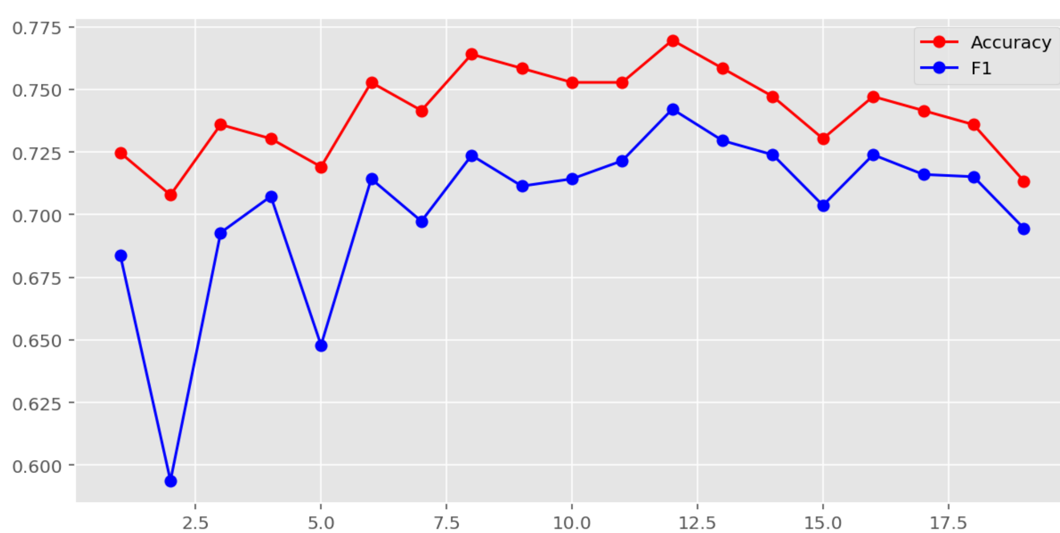
```python
## Prepare our data for viable input to the sklearn decision tree model

# Convert Sex feature into a pair of Boolean/binary dummy features (male and femal
df_dummy = pd.get_dummies(df, columns=['Sex'])

# Remove redundant male or female feature, and unneeded Name feature
df_dummy = df_dummy.drop(columns=['Sex_male', 'Name'])
df_dummy.head()

# Create train/test data sets
X = df_dummy.drop(columns=['Survived'])
y = df_dummy['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Let's repeat our work on decision trees, for eventual comparison with random forest models

In [7]:

```python
## Let's train models over a range of depths, and score them with the test set
from sklearn.metrics import f1_score

acc_scores_dt = []
f1_scores_dt = []
max_max = 20
max_depth = np.arange(1, max_max)

for depth in max_depth:
    # Build model and train
    titanic_tree = DecisionTreeClassifier(max_depth=depth, random_state=0)
    titanic_tree.fit(X_train, y_train)

    # Test
    acc_scores_dt.append(titanic_tree.score(X_test, y_test))
    y_test_hat = titanic_tree.predict(X_test)
    f1_scores_dt.append(f1_score(y_test, y_test_hat))

# Plot results
plt.figure(figsize=(10, 5))
plt.plot(max_depth, acc_scores_dt, 'ro-', label='Accuracy')
plt.plot(max_depth, f1_scores_dt, 'bo-', label='F1')
plt.legend()

# Print values for best test score
ix_best = np.argmax(acc_scores_dt)
print('Best accuracy score is %0.3f, for max_depth=%d' % (acc_scores_dt[ix_best], m
ix_best = np.argmax(f1_scores_dt)
print('Best F1-score is %0.3f, for max_depth=%d' % (f1_scores_dt[ix_best], max_dep
```

```
Best accuracy score is 0.770, for max_depth=12
Best F1-score is 0.742, for max_depth=12
```



Now we'll use sklearn to train random forests on the same data set

In addition to scores for the test set, we'll measure the *out-of-bag* error from the training set

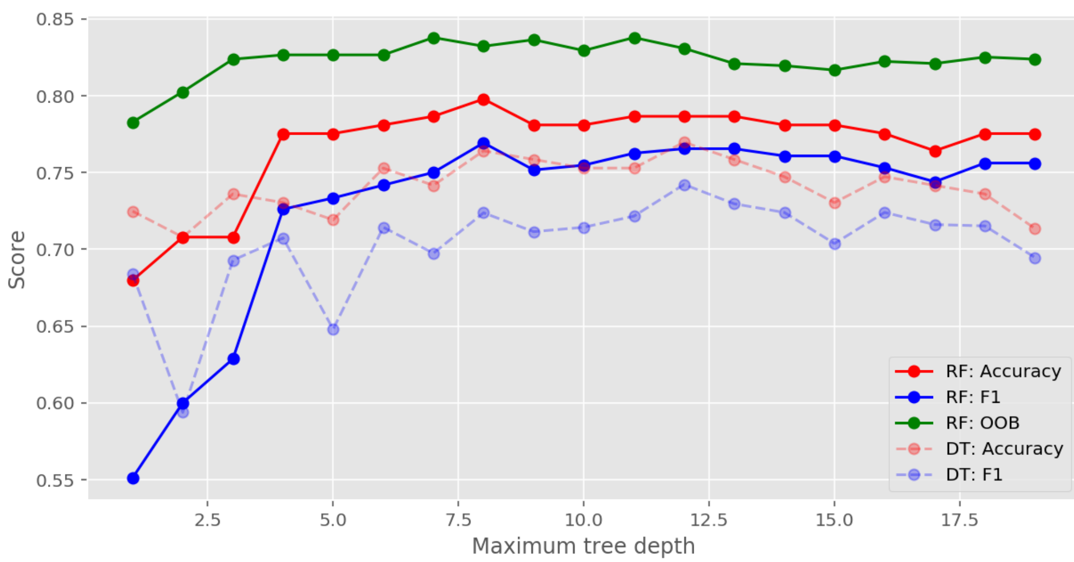
In [8]:

```
## We'll train models over a range of tree depths

from sklearn.ensemble import RandomForestClassifier

n_estimators = 100
max_depth = np.arange(1, max_max)
acc_scores_rf = []
f1_scores_rf = []
oob_scores_rf = []
for depth in max_depth:
    # Build model and train
    titanic_forest = RandomForestClassifier(n_estimators=n_estimators,
                                            max_depth=depth,
                                            oob_score=True,
                                            random_state=0) # call the RandomFores
    titanic_forest.fit(X_train, y_train)

    # Save out-of-bag (OOB) score
    oob_scores_rf.append(titanic_forest.oob_score_)

    # Test
    acc_scores_rf.append(titanic_forest.score(X_test, y_test))
    y_test_hat = titanic_forest.predict(X_test)
    f1_scores_rf.append(f1_score(y_test, y_test_hat))

# Print values for best test score
ix_best = np.argmax(acc_scores_rf)
print('Best accuracy score is %0.3f, for max_depth=%d' % (acc_scores_rf[ix_best], 
ix_best = np.argmax(f1_scores_rf)
print('Best F1-score is %0.3f, for max_depth=%d' % (f1_scores_rf[ix_best], max_dep
ix_best = np.argmax(oob_scores_rf)
print('Best OOB score is %0.3f, for max_depth=%d' % (oob_scores_rf[ix_best], max_d
```

```
Best accuracy score is 0.798, for max_depth=8
Best F1-score is 0.769, for max_depth=8
Best OOB score is 0.838, for max_depth=7
```

In [9]:

```python
# Plot results
plt.figure(figsize=(10, 5))
plt.plot(max_depth, acc_scores_rf, 'ro-', label='RF: Accuracy')
plt.plot(max_depth, f1_scores_rf, 'bo-', label='RF: F1')
plt.plot(max_depth, oob_scores_rf, 'go-', label='RF: OOB')
plt.plot(max_depth, acc_scores_dt, 'ro--', alpha=0.3, label='DT: Accuracy')
plt.plot(max_depth, f1_scores_dt, 'bo--', alpha=0.3, label='DT: F1')
plt.xlabel('Maximum tree depth')
plt.ylabel('Score')
_ = plt.legend()
```



Let's see which features have the highest "importance" for an RF with the best max_depth.

Importance, in sklearn, is calculated as the **Mean Decrease in Impurity (MDI)**.
That is, the **averge decrease in Gini impurity** across nodes that use the feature for splitting, weighted by the number of samples that reach that node.

In [10]:

```python
ix_best = np.argmax(acc_scores_rf)
max_depth_best = max_depth[ix_best]

# Build model and train
titanic_forest = RandomForestClassifier(n_estimators=n_estimators, random_state=0,
titanic_forest.fit(X_train, y_train)

plt.figure(figsize=(10,5))
column_names = X_train.columns
x_tick = np.array(range(len(column_names)))
plt.bar(x_tick, titanic_forest.feature_importances_)
plt.xticks(x_tick, column_names, rotation=90, fontsize=18)
_ = plt.ylabel('Importance', fontsize=18)
```