In [ ]:
```python
# Import.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris, load_wine
import importlib
rf = importlib.import_module("random-forests")

# Load datasets.
X_iris, y_iris = load_iris(return_X_y=True)  # feature_type="continuous"
X_wine, y_wine = load_wine(return_X_y=True)  # feature_type="continuous"
mushroom = np.genfromtxt("agaricus-lepiota.data", delimiter=",", dtype=str)
X_mushroom = mushroom[:, 1:]  # feature_type="categorical"
y_mushroom = mushroom[:, 0]
breast_cancer = np.genfromtxt("breast-cancer.data", delimiter=",", dtype=str)
breast_cancer = breast_cancer[(breast_cancer != "?").all(axis=1), :]
X_breast_cancer = breast_cancer[:, 1:]  # feature_type="categorical"
y_breast_cancer = breast_cancer[:, 0]
heart_disease = np.genfromtxt("processed.cleveland.data", delimiter=",")
heart_disease = heart_disease[~np.isnan(heart_disease).any(axis=1), :]
X_heart_disease = heart_disease[:, :13]  # feature_type="categorical"
y_heart_disease = heart_disease[:, 13]
y_heart_disease[np.where(y_heart_disease >= 1)] = 1
titanic = pd.read_csv("titanic.csv").drop(columns=["PassengerId", "Name", "Ticket"
X_titanic = titanic.loc[:, titanic.columns!="Survived"].to_numpy()  # feature_type
y_titanic = titanic["Survived"].to_numpy()
```

In [ ]:
```python
# train_test_split(X, y) -> X_train, X_test, y_train, y_test
ftype_h_d = np.array([0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1])
feature_type_titanic = np.ones(7)
feature_type_titanic[2] = 0  # Age
feature_type_titanic[5] = 0  # Fare
datasets = {
    "iris (continuous)": (*train_test_split(X_iris, y_iris, stratify=y_iris), "con
    "wine (continuous)": (*train_test_split(X_wine, y_wine, stratify=y_wine), "con
    # "mushroom (categorical)": (*train_test_split(X_mushroom, y_mushroom, stratif
    "breast_cancer (categorical)": (*train_test_split(X_breast_cancer, y_breast_ca
    "heart_disease (complex)": (*train_test_split(X_heart_disease, y_heart_disease
    "titanic (complex)": (*train_test_split(X_titanic, y_titanic, stratify=y_titan
}
```

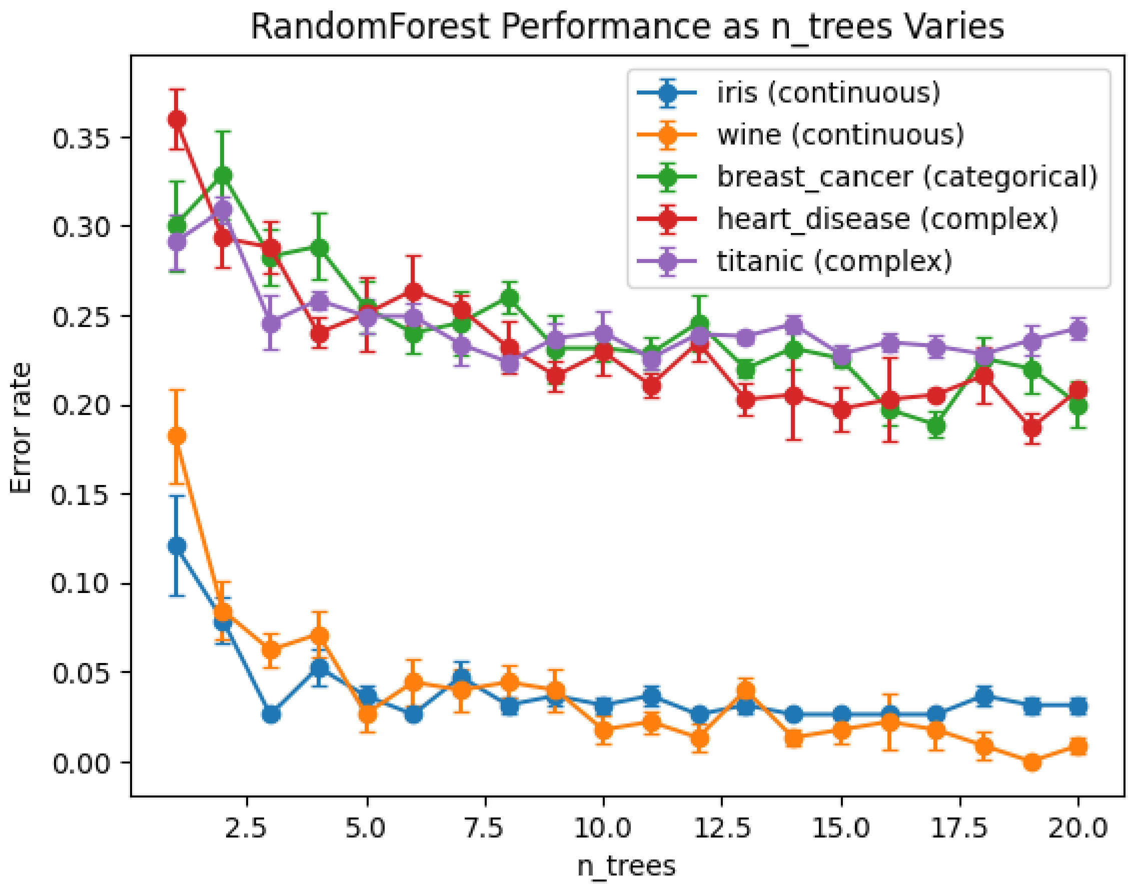
Breiman

- RF accuracy depends on individual tree **strength** and **correlation**
- **LLN** guarantees the generalisation error converges a.s. as `n_trees` increases, so that overfitting is not a problem

In [ ]:
```python
# Test varying `n_trees`.
n_trees = np.arange(1, 21)
n_exp=5  # Number of experiments.
error_rates_mean = np.ones((len(datasets.keys()), len(n_trees)))
error_rates_std = np.zeros((len(datasets.keys()), len(n_trees)))
for dataset_idx, dataset in enumerate(datasets.keys()):
    X_train, X_test, y_train, y_test, feature_type= datasets[dataset]
    n_features = X_train.shape[1]
    n_c = int(np.sqrt(n_features))
    # m = int(np.sqrt(len(np.unique(y_train))))
    for n in n_trees:
        random_forest = rf.RandomForest(
            n_trees=n, max_depth=100, min_leaf_size=1, n_candidates=n_c, criterion
        error_rate = np.ones(n_exp)
        for i in range(n_exp):
            random_forest.fit(X_train, y_train, feature_type, m_features=2)
            y_predicted = random_forest.predict(X_test)
            error_rate[i] = rf.random_forests.misclassification_rate(y_predicted, 
        error_rates_mean[dataset_idx, n-1] = error_rate.mean()
        error_rates_std[dataset_idx, n-1] = error_rate.std()

np.savez("test_n.npz", mean=error_rates_mean, std=error_rates_std)
```

```
In [ ]: error_rates = np.load("test_n.npz")
        error_rates_mean = error_rates["mean"]
        error_rates_std = error_rates["std"]
        n_exp=5
        for dataset_idx, dataset in enumerate(datasets.keys()):
            plt.errorbar(n_trees, error_rates_mean[dataset_idx, :],
                         yerr=error_rates_std[dataset_idx, :]/np.sqrt(n_exp),
                         label=dataset, capsize=3, fmt="o-"
                         )
        plt.xlabel("n_trees")
        plt.ylabel("Error rate")
        plt.title("RandomForest Performance as n_trees Varies")
        plt.legend()
```

```
Out[ ]: <matplotlib.legend.Legend at 0x21d69280850>
```



```
In [ ]: # Test varying `n_trees` using ImprovedRandomForest.
        n_trees = np.arange(1, 21)
        n_exp=5  # Number of experiments.
        error_rates_mean = np.ones((len(datasets.keys()), len(n_trees)))
        error_rates_std = np.zeros((len(datasets.keys()), len(n_trees)))
        for dataset_idx, dataset in enumerate(datasets.keys()):
            X_train, X_test, y_train, y_test, feature_type= datasets[dataset]
            n_features = X_train.shape[1]
```

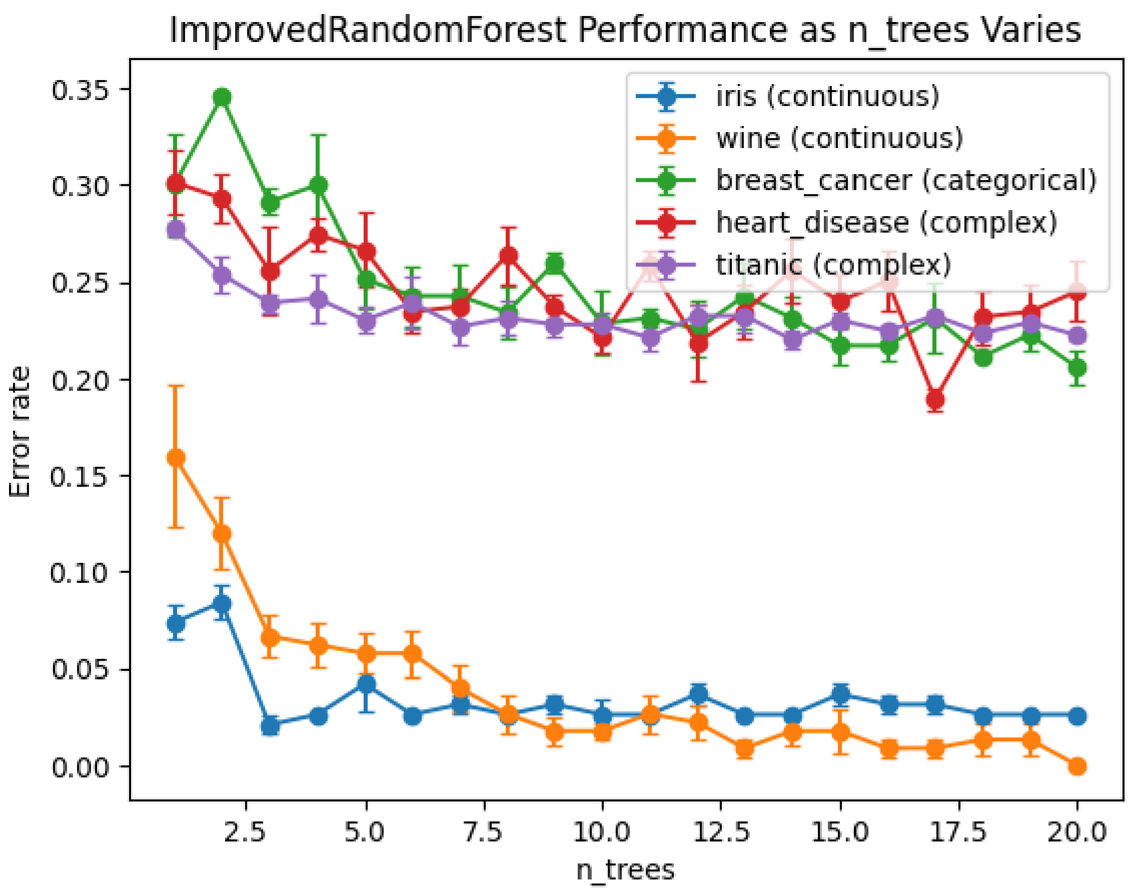
```
    n_c = int(np.sqrt(n_features))
    # m = int(np.sqrt(len(np.unique(y_train))))
    for n in n_trees:
        random_forest = rf.ImprovedRandomForest(
            n_trees=n, max_depth=100, min_leaf_size=1, n_candidates=n_c, criterion
        error_rate = np.ones(n_exp)
        for i in range(n_exp):
            random_forest.fit(X_train, y_train, feature_type, m_features=2)
            y_predicted = random_forest.predict(X_test)
            error_rate[i] = rf.random_forests.misclassification_rate(y_predicted, 
        error_rates_mean[dataset_idx, n-1] = error_rate.mean()
        error_rates_std[dataset_idx, n-1] = error_rate.std()

np.savez("test_n_improved.npz", mean=error_rates_mean, std=error_rates_std)
```

In [ ]:

```
error_rates = np.load("test_n_improved.npz")
error_rates_mean = error_rates["mean"]
error_rates_std = error_rates["std"]
n_trees = np.arange(1, 21)
n_exp=5
for dataset_idx, dataset in enumerate(datasets.keys()):
    plt.errorbar(n_trees, error_rates_mean[dataset_idx, :],
                 yerr=error_rates_std[dataset_idx, :]/np.sqrt(n_exp),
                 label=dataset, capsize=3, fmt="o-"
                 )
plt.xlabel("n_trees")
plt.ylabel("Error rate")
plt.title("ImprovedRandomForest Performance as n_trees Varies")
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x21d1a543d00>
```



In [ ]:

```
# Test varing `n_trees` with different `m_features` on the Titanic dataset.
X_train, X_test, y_train, y_test, feature_type= datasets["titanic (complex)"]
n_trees = np.arange(1, 31)
```

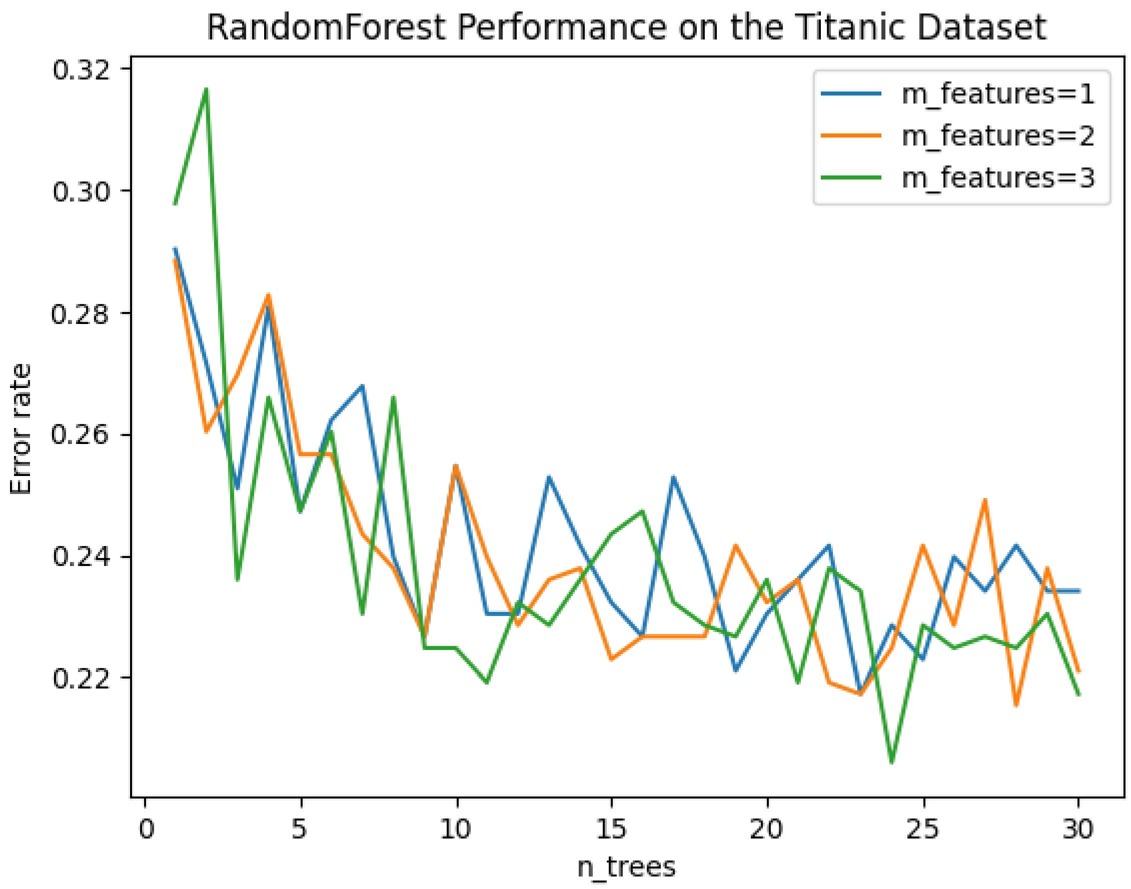
```python
m_features = [1, 2, 3]
n_exp=3
error_rates_mean = np.ones((len(m_features), len(n_trees)))
error_rates_std = np.zeros((len(m_features), len(n_trees)))
for n in n_trees:
    for m in m_features:
        random_forest = rf.RandomForest(
            n_trees=n, max_depth=100, min_leaf_size=1, n_candidates=2, criterion="
        error_rate = np.ones(n_exp)
        for i in range(n_exp):
            random_forest.fit(X_train, y_train, feature_type, m_features=m)
            y_predicted = random_forest.predict(X_test)
            error_rate[i] = rf.random_forests.misclassification_rate(y_predicted, 
        error_rates_mean[m_features.index(m), n-1] = error_rate.mean()
        error_rates_std[m_features.index(m), n-1] = error_rate.std()

np.savez("test_titanic_n30_m.npz", mean=error_rates_mean, std=error_rates_std)
```

In [ ]:

```python
error_rates = np.load("test_titanic_n30_m.npz")
error_rates_mean = error_rates["mean"]
error_rates_std = error_rates["std"]
n_trees = np.arange(1, 31)
m_features = [1, 2, 3]
n_exp=3
for m in m_features:
    # plt.errorbar(n_trees, error_rates_mean[m_features.index(m), :],
    #                 yerr=error_rates_std[m_features.index(m), :]/np.sqrt(n_exp),
    #                 label=f"m_features={m}", capsize=3, fmt=".-"
    #                 )
    plt.plot(n_trees, error_rates_mean[m_features.index(m), :],
             label=f"m_features={m}")
plt.xlabel("n_trees")
plt.ylabel("Error rate")
plt.title("RandomForest Performance on the Titanic Dataset")
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x23aca0c31f0>
```



Breiman

- RF performance is *insensitive* to the number of features selected to split each node.
- Usually `m_features=1` or `m_features=2` gives near optimum results

In [ ]:
```
# Test varing `n_trees` with different `m_features` on the Breast Cancer dataset.
X_train, X_test, y_train, y_test, feature_type= datasets["breast_cancer (categorica
n_trees = np.arange(1, 31)
m_features = [1, 2, 3]
n_exp=3
error_rates_mean = np.ones((len(m_features), len(n_trees)))
error_rates_std = np.zeros((len(m_features), len(n_trees)))
for n in n_trees:
    for m in m_features:
        random_forest = rf.RandomForest(
            n_trees=n, max_depth=100, min_leaf_size=1, n_candidates=3, criterion="
        error_rate = np.ones(n_exp)
        for i in range(n_exp):
            random_forest.fit(X_train, y_train, feature_type, m_features=m)
            y_predicted = random_forest.predict(X_test)
            error_rate[i] = rf.random_forests.misclassification_rate(y_predicted, 
        error_rates_mean[m_features.index(m), n-1] = error_rate.mean()
        error_rates_std[m_features.index(m), n-1] = error_rate.std()

np.savez("test_breast_cancer_n30_m.npz", mean=error_rates_mean, std=error_rates_st
```

In [ ]:
```
# Test varing `n_trees` with different `m_features` on the Breast Cancer dataset
# using ImprovedRandomForest
X_train, X_test, y_train, y_test, feature_type= datasets["breast_cancer (categorica
n_trees = np.arange(1, 31)
m_features = [1, 2, 3]
n_exp=3
error_rates_mean = np.ones((len(m_features), len(n_trees)))
```

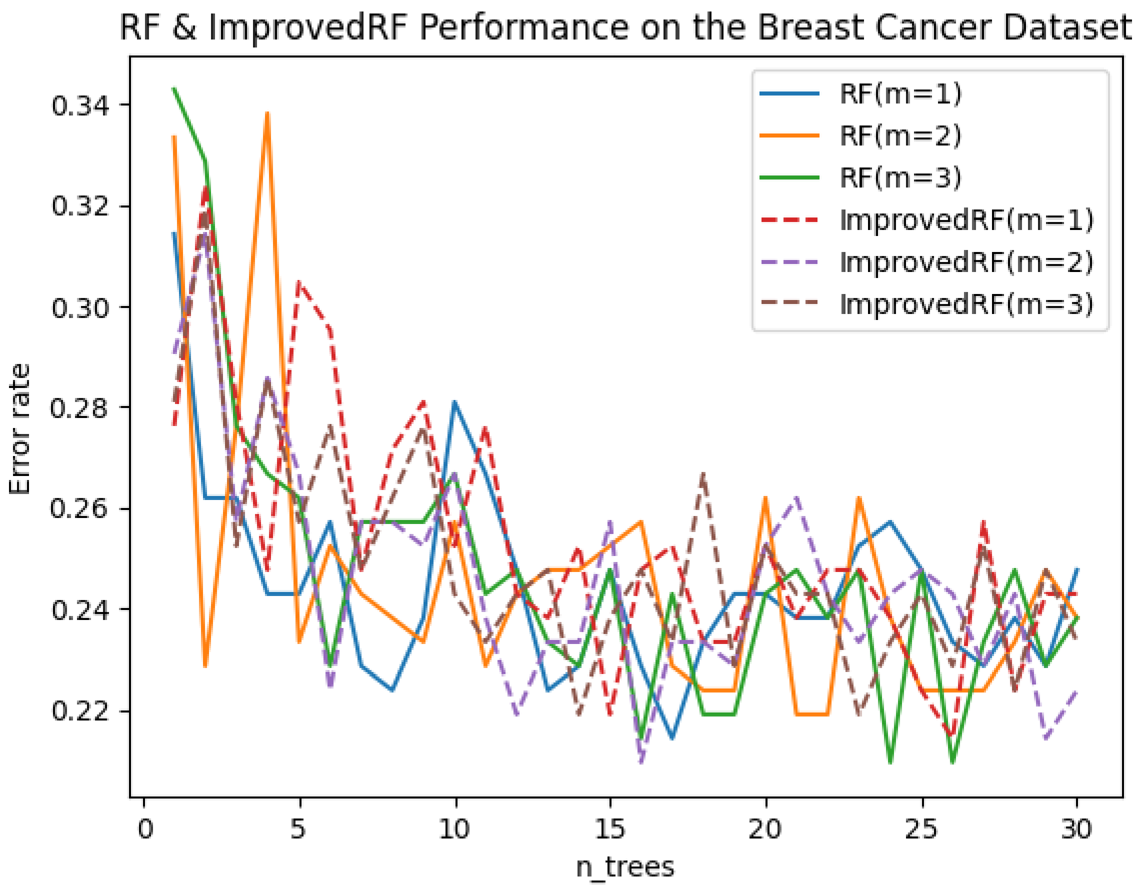
```
error_rates_std = np.zeros((len(m_features), len(n_trees)))
for n in n_trees:
    for m in m_features:
        random_forest = rf.ImprovedRandomForest(
            n_trees=n, max_depth=100, min_leaf_size=1, n_candidates=3, criterion="
        error_rate = np.ones(n_exp)
        for i in range(n_exp):
            random_forest.fit(X_train, y_train, feature_type, m_features=m)
            y_predicted = random_forest.predict(X_test)
            error_rate[i] = rf.random_forests.misclassification_rate(y_predicted, 
        error_rates_mean[m_features.index(m), n-1] = error_rate.mean()
        error_rates_std[m_features.index(m), n-1] = error_rate.std()

np.savez("test_breast_cancer_n30_m_improved.npz", mean=error_rates_mean, std=error_
```

In [ ]:

```
n_trees = np.arange(1, 31)
m_features = [1, 2, 3]
n_exp=3

error_rates = np.load("test_breast_cancer_n30_m.npz")
error_rates_mean = error_rates["mean"]
error_rates_std = error_rates["std"]
for m in m_features:
    plt.plot(n_trees, error_rates_mean[m_features.index(m), :],
             label=f"RF(m={m})")

error_rates = np.load("test_breast_cancer_n30_m_improved.npz")
error_rates_mean = error_rates["mean"]
error_rates_std = error_rates["std"]
for m in m_features:
    plt.plot(n_trees, error_rates_mean[m_features.index(m), :],
             "--", label=f"ImprovedRF(m={m})")

plt.xlabel("n_trees")
plt.ylabel("Error rate")
plt.title("RF & ImprovedRF Performance on the Breast Cancer Dataset")
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x21d1b929960>
```


RF & ImprovedRF Performance on the Breast Cancer Dataset
RF(m=1)
RF(m=2)
RF(m=3)
ImprovedRF(m=1)
ImprovedRF(m=2)
ImprovedRF(m=3)
Error rate
n_trees
0.34
0.32
0.30
0.28
0.26
0.24
0.22
0
5
10
15
20
25
30

```
In [ ]:
```