In [21]:
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

# Load the dataset
crops = pd.read_csv("data/soil_measures.csv")
```

Check for missing values.

In [3]:
```
crops.isna().sum()
```

Out[3]:
```
N       0
P       0
K       0
ph      0
crop    0
dtype: int64
```

In [4]:
```
crops.dtypes
```

Out[4]:
```
N         int64
P         int64
K         int64
ph      float64
crop     object
dtype: object
```

In [5]:
```
crops["crop"].value_counts()
```

```
Out[5]: crop
        rice           100
        maize          100
        chickpea       100
        kidneybeans    100
        pigeonpeas     100
        mothbeans      100
        mungbean       100
        blackgram      100
        lentil         100
        pomegranate    100
        banana         100
        mango          100
        grapes         100
        watermelon     100
        muskmelon      100
        apple          100
        orange         100
        papaya         100
        coconut        100
        cotton         100
        jute           100
        coffee         100
        Name: count, dtype: int64
```

```
In [6]: features = ["N","P","K","ph"]
```

```
In [7]: X = crops[features]
```

```
In [8]: y = crops[["crop"]]
```

```
In [9]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
```

Trying out logistic regression.

```
In [10]: log_reg = LogisticRegression()
```

```
In [ ]: log_reg.fit(X_train, y_train)
```

```
In [12]: y_pred = log_reg.predict(X_test)
```

```
In [13]: display(metrics.accuracy_score(y_test,y_pred))
```

```
0.5484848484848485
```

Not the best results. Now lets see how a random forest performs.

```
In [14]: forest = RandomForestClassifier()
```

```
In [ ]: forest.fit(X_train,y_train)
```

In [16]:
```
y_pred = forest.predict(X_test)
display(metrics.accuracy_score(y_test,y_pred))
```

```
0.7636363636363637
```

Much better!

Let's find the most relevant feature.

In [ ]:
```
feature_scores = {}
for feature in features:
    new_X=X[feature].values.reshape(-1,1)
    X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(new_X, y)
    new_forest = RandomForestClassifier()
    new_forest.fit(X_train_new, y_train_new)
    y_pred_new = new_forest.predict(X_test_new)
    score = metrics.accuracy_score(y_test_new, y_pred_new)
    feature_scores[feature] = score
```

In [18]:
```
print(feature_scores)
```

```
{'N': 0.12, 'P': 0.19090909090909092, 'K': 0.3054545454545455, 'ph': 0.1327272727272
7272}
```

On its own, the content ratio of potassium is the best indicator for the optimal crop choice.

Let's find the best set of parameters.

In [22]:
```
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}

forest = RandomForestClassifier()
grid = GridSearchCV(estimator=forest, param_grid=param_grid, cv=3, n_jobs=-1)
```

In [ ]:
```
grid.fit(X_train, y_train)
```

In [28]:
```
display(grid.best_params_)
```

```
{'bootstrap': True,
 'max_depth': 80,
 'max_features': 2,
 'min_samples_leaf': 3,
 'min_samples_split': 12,
 'n_estimators': 100}
```

Now that we have our best params, we select the estimator with the best results.

```
In [25]: best_forest = grid.best_estimator_
```

```
In [ ]: best_forest.fit(X_train,y_train)
```

```
In [27]: y_pred = best_forest.predict(X_test)
         accuracy = metrics.accuracy_score(y_test, y_pred)

         display(accuracy)
```

```
0.7757575757575758
```

A tiny improvement but an improvement nonetheless.