In [17]:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

Data Loading

In [18]:

```
car_df = pd.read_csv("car.data", header=None)
car_df.head()
```

Out[18]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

In [19]:

```
wine_df = pd.read_csv("wine.data", header=None)
wine_df = wine_df[[1,2,3,4,5,6,7,8,9,10,11,12,13,0]]
wine_df.head()
```

Out[19]:

	1	2	3	4	5	6	7	8	9	10	11	12	13	0
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065	1
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	1
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	1
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	1
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	1

In [20]:

```
iris_df = pd.read_csv("iris.data", header=None)
iris_df.head()
```

Out[20]:

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Data Cleaning

In [21]:

```
#Adult data frame
#Use on hot coding to change qualitative data to quantitative data
#<=50k changes to 0; >=50k changes to 1
enc = LabelEncoder()
car_df = car_df.apply(enc.fit_transform)
car_df[6] = np.where(car_df[6]%2==0, 1, 0)
car_XandY = car_df.values
car_df.head()
```

Out[21]:

	0	1	2	3	4	5	6
0	3	3	0	0	2	1	1
1	3	3	0	0	2	2	1
2	3	3	0	0	2	0	1
3	3	3	0	0	1	1	1
4	3	3	0	0	1	2	1

In [22]:

```
#Wine data frame
#if the first column has value 1, encode it as 1, otherwise, encode it as 0
wine_df[0] = np.where(wine_df[0]==1, 1, 0)
wine_XandY = wine_df.values
```

In [23]:

```
#Iris data frame
iris_df[4] = np.where(iris_df[4]=='Iris-setosa', 1, 0)
iris_XandY = iris_df.values
```

Parameter Tuning

In [24]:

```
def knn(X, Y):
    parameters = {'n_neighbors':[i+1 for i in range(18)]}
    clf = GridSearchCV(KNeighborsClassifier(), parameters, cv=3)
    clf.fit(X,Y)
    return clf.best_estimator_, clf.best_params_
```

In [25]:

```
def svm_new(X, Y):
    parameters = {'C': [0.001,0.005,0.01,0.05,0.1,0.5,1,2]}
    clf = GridSearchCV(svm.SVC(kernel="rbf"), parameters, cv=3)
    clf.fit(X,Y)
    return clf.best_estimator_, clf.best_params_
```

In [26]:

```
def rf(X, Y):
    parameters = {'n_estimators': [1,2,4,6,8,12,16,20]}
    clf = GridSearchCV(RandomForestClassifier(), parameters, cv=3)
    clf.fit(X,Y)
    return clf.best_estimator_, clf.best_params_
```

Training

In [27]:

```
def train(X_train, Y_train, X_test, Y_test, model):
    train_acc = -1
    val_acc = -1
    test_acc = -1
    if model == "knn":
        clf, param = knn(X_train, Y_train)
        train_acc = clf.score(X_train, Y_train)
        val_acc = cross_val_score(clf, X_train, Y_train, cv=3)
        test_acc = clf.score(X_test, Y_test)
        print(model)
        print(param)
    elif model == "svm":
        clf, param = svm_new(X_train, Y_train)
        train_acc = clf.score(X_train, Y_train)
        val_acc = cross_val_score(clf, X_train, Y_train, cv=3)
        test_acc = clf.score(X_test, Y_test)
        print(model)
        print(param)
    else:
        clf, param = rf(X_train, Y_train)
        train_acc = clf.score(X_train, Y_train)
        val_acc = cross_val_score(clf, X_train, Y_train, cv=3)
        test_acc = clf.score(X_test, Y_test)
        print(model)
        print(param)
    print("Training accuracy is: {:.3f}".format(train_acc))
    print("Validation accuracy is: {:3f}".format(val_acc.mean()))
    print("Testing accuracy is: {:.3f}".format(test_acc))
    return (train_acc, val_acc.mean(), test_acc)
```

In [28]:

```
def produce(X_and_Y):
    X = X_and_Y[:, :-1]
    Y = X_and_Y[:,-1]
    for partition in [0.2,0.5,0.8]:
        knn_acc = np.zeros([3,3])
        svm_acc = np.zeros([3,3])
        rf_acc = np.zeros([3,3])
        for i in range(3):
            X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=part
            knn_acc[i,:] = train(X_train, Y_train, X_test, Y_test,"knn")
            svm_acc[i,:] = train(X_train, Y_train, X_test, Y_test,"svm")
            rf_acc[i,:] = train(X_train, Y_train, X_test, Y_test,"rf")
        knn_avg = [np.mean(knn_acc[:,i]) for i in range(3)]
        svm_avg = [np.mean(svm_acc[:,i]) for i in range(3)]
        rf_avg = [np.mean(rf_acc[:,i]) for i in range(3)]
        print("Partition is:" + str(partition))
        print("For knn, train_acc is {:.3f}; val_acc is {:.3f}; test_acc is {:.3f}"
              format(knn_avg[0], knn_avg[1], knn_avg[2]))
        print("For svm, train_acc is {:.3f}; val_acc is {:.3f}; test_acc is {:.3f}"
              format(svm_avg[0], svm_avg[1], svm_avg[2]))
        print("For rf, train_acc is {:.3f}; val_acc is {:.3f}; test_acc is {:.3f}".
              format(rf_avg[0], rf_avg[1], rf_avg[2]))
        print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

In [29]:

```
produce(car_XandY)
```

```
Training accuracy is: 0.988
Validation accuracy is: 0.950725
Testing accuracy is: 0.948
knn
{'n_neighbors': 5}
Training accuracy is: 0.959
Validation accuracy is: 0.956522
Testing accuracy is: 0.928
svm
{'C': 0.001}
Training accuracy is: 0.948
Validation accuracy is: 0.947826
Testing accuracy is: 0.916
rf
{'n_estimators': 12}
Training accuracy is: 1.000
Validation accuracy is: 0.942029
Testing accuracy is: 0.940
Partition is:0.8
For knn, train_acc is 0.959; val_acc is 0.957; test_acc is 0.928
```

In [30]:

```
produce(wine_XandY)
```

```
{'n_estimators': 20}
Training accuracy is: 1.000
Validation accuracy is: 0.979019
Testing accuracy is: 1.000
knn
{'n_neighbors': 1}
Training accuracy is: 1.000
Validation accuracy is: 0.922725
Testing accuracy is: 0.917
svm
{'C': 0.001}
Training accuracy is: 0.683
Validation accuracy is: 0.683067
Testing accuracy is: 0.611
rf
{'n_estimators': 16}
Training accuracy is: 1.000
Validation accuracy is: 0.986111
Testing accuracy is: 1.000
knn
{'n_neighbors': 1}
```

In [31]:

```
produce(iris_XandY)
```

```
knn
{'n_neighbors': 1}
Training accuracy is: 1.000
Validation accuracy is: 1.000000
Testing accuracy is: 1.000
svm
{'C': 0.05}
Training accuracy is: 1.000
Validation accuracy is: 1.000000
Testing accuracy is: 1.000
rf
{'n_estimators': 1}
Training accuracy is: 1.000
Validation accuracy is: 1.000000
Testing accuracy is: 1.000
knn
{'n_neighbors': 1}
Training accuracy is: 1.000
Validation accuracy is: 1.000000
```

In [ ]:

```

```