In [ ]:
```
#Question 1
```

In [1]:
```
from __future__ import division
import time
import numpy as np
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
```

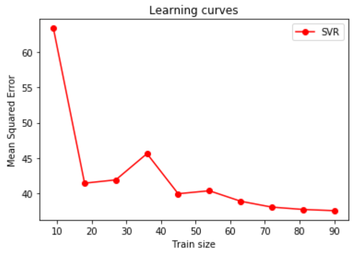
In [2]:
```
#load dataset
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.data.shape)
X=boston.data
y=boston.target

print (X.shape)
print (y.shape)
```

```
(506, 13)
(506, 13)
(506,)
```

In [3]:
```
train_size = 100
# Parameters are already given in the question, no grid search required.
```

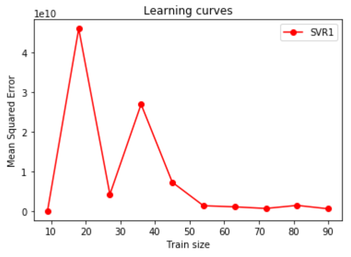
In [4]:
```
#Implementing RBF kernel
svr_rbf = SVR(kernel='rbf', C=100, degree=2, gamma=0.1)
train_sizes, train_scores_svr, test_scores_svr = \
    learning_curve(svr_rbf, X[:100], y[:100], train_sizes=np.linspace(0.1, 1, 10),
                   scoring="neg_mean_squared_error", cv=10)
print (test_scores_svr.shape)
plt.plot(train_sizes, -test_scores_svr.mean(1), 'o-', color="r",
         label="SVR")
plt.xlabel("Train size")
plt.ylabel("Mean Squared Error")
plt.title('Learning curves')
plt.legend(loc="best")
#Plotting learning curves
plt.show()
```

```
(10, 10)
```



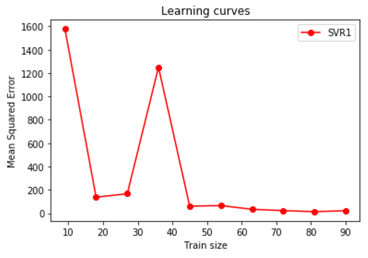
In [5]:
```
#Implementing Poly Kernel
svr_poly = SVR(kernel='poly', C=100, degree=2, gamma=0.1)
train_sizes, train_scores_svr, test_scores_svr = \
    learning_curve(svr_poly, X[:100], y[:100], train_sizes=np.linspace(0.1, 1, 10),
                   scoring="neg_mean_squared_error", cv=10)
print (test_scores_svr.shape)
plt.plot(train_sizes, -test_scores_svr.mean(1), 'o-', color="r",
         label="SVR1")
plt.xlabel("Train size")
plt.ylabel("Mean Squared Error")
plt.title('Learning curves')
plt.legend(loc="best")
#Plotting learning curves
plt.show()
```

```
(10, 10)
```



In [6]:
```
#Implementing Linear kernel
svr_linear = SVR(kernel='linear', C=100, degree=2, gamma=0.1)
train_sizes, train_scores_svr, test_scores_svr = \
    learning_curve(svr_linear, X[:100], y[:100], train_sizes=np.linspace(0.1, 1, 10),
                   scoring="neg_mean_squared_error", cv=10)
print (test_scores_svr.shape)
plt.plot(train_sizes, -test_scores_svr.mean(1), 'o-', color="r",
         label="SVR1")
plt.xlabel("Train size")
plt.ylabel("Mean Squared Error")
plt.title('Learning curves')
plt.legend(loc="best")
#Plotting learning curves
plt.show()
```

```
(10, 10)
```



In [7]:
```
#Fitting the model
svr_rbf.fit(X[:train_size], y[:train_size])
svr_poly.fit(X[:train_size], y[:train_size])
svr_linear.fit(X[:train_size], y[:train_size])
```

Out[7]:
```
SVR(C=100, cache_size=200, coef0=0.0, degree=2, epsilon=0.1, gamma=0.1,
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

In [8]:
```
#Accuracy for SVR
from sklearn.metrics import mean_squared_error
prediction = svr_linear.predict(X[train_size:])
mean_squared_error(y[train_size:], prediction)
```

Out[8]:
```
6646.9704301698221
```

In [9]:
```
#Accuracy for Poly
prediction = svr_poly.predict(X[train_size:])
mean_squared_error(y[train_size:], prediction)
```

Out[9]:
```
93652818478.405212
```

In [10]:
```
#Accuracy for RBF
prediction = svr_rbf.predict(X[train_size:])
mean_squared_error(y[train_size:], prediction)
```

Out[10]:
```
96.691254372778118
```

In [11]:
```
# Lowest error obtained for rbf model
```

In [ ]:
```
#Question 2
```

In [1]:
```
from sklearn import tree
from sklearn import datasets
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import classification_report
from sklearn.grid_search import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
```

```
C:\Users\Renuka\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarni
ng: This module was deprecated in version 0.18 in favor of the model_selection module into w
hich all the refactored classes and functions are moved. Also note that the interface of the
new CV iterators are different from that of this module. This module will be removed in 0.2
0.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\Renuka\Anaconda3\lib\site-packages\sklearn\grid_search.py:42: DeprecationWarning: T
his module was deprecated in version 0.18 in favor of the model_selection module into which
all the refactored classes and functions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [2]:
```
#Load breast cancer dataset
from sklearn.datasets import load_breast_cancer
bc = load_breast_cancer()
X = bc.data
y = bc.target
print (X.max())
print (y.shape)
#Classification using train-test split
X, y = shuffle(X, y, random_state=0)
```

```
4254.0
(569,)
```

In [3]:
```
#Optimizing for different values of n_estimators and max_depth for Gradient Boost implementatio
n
cv_params = {'n_estimators':[10,100,200,500],'max_depth': [2,3,5,7]}
optimized_GBM = GridSearchCV(estimator=GradientBoostingClassifier(n_estimators=100, learning_ra
te=1.0,random_state=0),param_grid=cv_params,scoring = 'accuracy', cv = 5)
```

In [4]:
```
optimized_GBM.fit(X, y)
```

Out[4]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1.0, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=0, subsample=1.0, verbose=0,
              warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [10, 100, 200, 500], 'max_depth': [2, 3, 5, 7]},
       pre_dispatch='2*n_jobs', refit=True, scoring='accuracy', verbose=0)
```

In [5]:
```
optimized_GBM.grid_scores_
```

Out[5]:
```
[mean: 0.95431, std: 0.02197, params: {'max_depth': 2, 'n_estimators': 10},
 mean: 0.96309, std: 0.02041, params: {'max_depth': 2, 'n_estimators': 100},
 mean: 0.96309, std: 0.02041, params: {'max_depth': 2, 'n_estimators': 200},
 mean: 0.96309, std: 0.02041, params: {'max_depth': 2, 'n_estimators': 500},
 mean: 0.93673, std: 0.02208, params: {'max_depth': 3, 'n_estimators': 10},
 mean: 0.95255, std: 0.01752, params: {'max_depth': 3, 'n_estimators': 100},
 mean: 0.95255, std: 0.01752, params: {'max_depth': 3, 'n_estimators': 200},
 mean: 0.95255, std: 0.01752, params: {'max_depth': 3, 'n_estimators': 500},
 mean: 0.95255, std: 0.02370, params: {'max_depth': 5, 'n_estimators': 10},
 mean: 0.95606, std: 0.01830, params: {'max_depth': 5, 'n_estimators': 100},
 mean: 0.95606, std: 0.01830, params: {'max_depth': 5, 'n_estimators': 200},
 mean: 0.95606, std: 0.01830, params: {'max_depth': 5, 'n_estimators': 500},
 mean: 0.92794, std: 0.02627, params: {'max_depth': 7, 'n_estimators': 10},
 mean: 0.92794, std: 0.02627, params: {'max_depth': 7, 'n_estimators': 100},
 mean: 0.92794, std: 0.02627, params: {'max_depth': 7, 'n_estimators': 200},
 mean: 0.92794, std: 0.02627, params: {'max_depth': 7, 'n_estimators': 500}]
```

In [6]:
```
#Best parameters obtained
optimized_GBM.best_params_
```

Out[6]:
```
{'max_depth': 2, 'n_estimators': 100}
```

In [7]:
```
optimized_GBM.best_score_
```

Out[7]:
```
0.9630931458699473
```

In [9]:
```
#Splitting dataset into equal parts
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
gbc=GradientBoostingClassifier(n_estimators=100, max_depth =2, learning_rate=1.0,random_state=0
).fit(X_train,y_train)
Results=gbc.predict(X_test)
#Classification accuracy report
print(classification_report(Results,y_test))
```

```
             precision    recall  f1-score   support

          0       0.94      0.98      0.96        46
          1       0.99      0.97      0.98        97
```

In [ ]:
```
#Question 3
```

In [1]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
from sklearn import datasets
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import classification_report
```

In [2]:
```
#Load breast cancer dataset
from sklearn.datasets import load_breast_cancer
bc2 = load_breast_cancer()
X = bc2.data
y = bc2.target
print (X.max())
print (y.shape)
#Classification using train-test split
X, y = shuffle(X, y, random_state=0)
```

```
4254.0
(569,)
```

In [7]:
```
#Optimizing for different values of n_estimators for Bagging Implementation
from sklearn.grid_search import GridSearchCV
cv_params = {'n_estimators':[10,100,200,500]}
optimized_bag = GridSearchCV(BaggingClassifier(tree.DecisionTreeClassifier(), n_estimators=100,
 random_state=0),param_grid=cv_params,scoring = 'accuracy', cv = 5)
```

In [8]:
```
optimized_bag.fit(X, y)
```

Out[8]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            ..., n_estimators=100, n_jobs=1, oob_score=False,
          random_state=0, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [10, 100, 200, 500]},
       pre_dispatch='2*n_jobs', refit=True, scoring='accuracy', verbose=0)
```

In [9]:
```
optimized_bag.grid_scores_
```

Out[9]:
```
[mean: 0.94200, std: 0.02219, params: {'n_estimators': 10},
 mean: 0.95782, std: 0.01799, params: {'n_estimators': 100},
 mean: 0.95782, std: 0.01799, params: {'n_estimators': 200},
 mean: 0.95958, std: 0.01533, params: {'n_estimators': 500}]
```

In [10]:
```
#Best parameters
optimized_bag.best_params_
```

Out[10]:
```
{'n_estimators': 500}
```

In [11]:
```
optimized_bag.best_score_
```

Out[11]:
```
0.9595782073813708
```

In [13]:
```
#Splitting dataset into equal parts
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
bagging = BaggingClassifier(tree.DecisionTreeClassifier(),n_estimators=200,oob_score=True,max_s
amples=0.5, max_features=0.5).fit(X_train,y_train)
Results=bagging.predict(X_test)
#Classification accuracy report
print (classification_report(Results,y_test))
```

```
             precision    recall  f1-score   support

          0       0.90      0.96      0.93        49
          1       0.98      0.95      0.96        94

avg / total       0.95      0.95      0.95       143
```

In [14]:
```
#Classification using Cross-validation
X, y = shuffle(X, y, random_state=0)
bg = BaggingClassifier(tree.DecisionTreeClassifier(),n_estimators=500,oob_score=True,max_sample
s=0.5, max_features=0.5).fit(X_train,y_train)
results = model_selection.cross_val_score(bg, X, y, cv=5)
#Average score
print (results.mean())
```

```
0.957799153521
```

In [ ]:
```
#Question 4
```

In [1]:
```
from sklearn import tree
from sklearn import datasets
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
```

In [3]:
```
#Load Breast Cancer dataset
from sklearn.datasets import load_breast_cancer
bc3 = load_breast_cancer()
X = bc3.data
y = bc3.target
print (X.max())
print (y.shape)
####perform classification with train-test split####
X, y = shuffle(X, y, random_state=0)
```

```
4254.0
(569,)
```

In [6]:
```
#Optimizing for different values of n_estimators and max_depth for Random Forest Implementation

from sklearn.grid_search import GridSearchCV
cv_params = {'n_estimators':[10,100,200,500],'max_depth': [2,3,5,7]}
optimized_rf = GridSearchCV(estimator=RandomForestClassifier(n_estimators=100,random_state=0),p
aram_grid=cv_params,scoring = 'accuracy', cv = 5)
```

In [8]:
```
optimized_rf.fit(X, y)
```

Out[8]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_estimators': [10, 100, 200, 500], 'max_depth': [2, 3, 5, 7]},
       pre_dispatch='2*n_jobs', refit=True, scoring='accuracy', verbose=0)
```

In [9]:
```
#Best parameters
optimized_rf.best_params_
```

Out[9]:
```
{'max_depth': 7, 'n_estimators': 100}
```

In [10]:
```
optimized_rf.best_score_
```

Out[10]:
```
0.9595782073813708
```

In [11]:
```
#Splitting dataset into equal parts
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
clf = RandomForestClassifier(n_estimators=100, max_depth=7,min_samples_split=2, random_state=0)
.fit(X_train,y_train)
Results=clf.predict(X_test)
#Classification accuracy report
print (classification_report(Results,y_test))
```

```
             precision    recall  f1-score   support

          0       0.90      0.96      0.92        45
          1       0.98      0.95      0.96        98

avg / total       0.95      0.95      0.95       143
```

In [12]:
```
#Classification using Cross-validation
clf = RandomForestClassifier(n_estimators=10, max_depth=None,min_samples_split=2, random_state=
0).fit(X_train,y_train)
results = model_selection.cross_val_score(clf, X, y, cv=5)
#Average score
print (results.mean())
```