Coding Exercise, Week 5, Dec 22

CSML1010 Fall 2019

Pete Gray [ptgray@my.yorku.ca (mailto:ptgray@my.yorku.ca)](mailto:ptgray@my.yorku.ca) YorkU #217653247

Using code from [https://blog.statsbot.co/ensemble-learning-d1dcd548e936 (https://blog.statsbot.co/ensemble-learning-d1dcd548e936)](https://blog.statsbot.co/ensemble-learning-d1dcd548e936)

Ensemble Methods

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking). Ensemble methods can be divided into two groups: *sequential* ensemble methods where the base learners are generated sequentially (e.g. AdaBoost) and *parallel* ensemble methods where the base learners are generated in parallel (e.g. Random Forest). The basic motivation of sequential methods is to exploit the dependence between the base learners since the overall performance can be boosted by weighing previously mislabeled examples with higher weight. The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging.

Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type leading to *homogeneous ensembles*. There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to *heterogeneous ensembles*. In order for ensemble methods to be more accurate than any of its individual members the base learners have to be as accurate as possible and as diverse as possible.

Bagging

Bagging stands for bootstrap aggregation. One way to reduce the variance of an estimate is to average together multiple estimates. For example, we can train M different trees f_m on different subsets of the data (chosen randomly with replacement) and compute the ensemble:

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x)$$

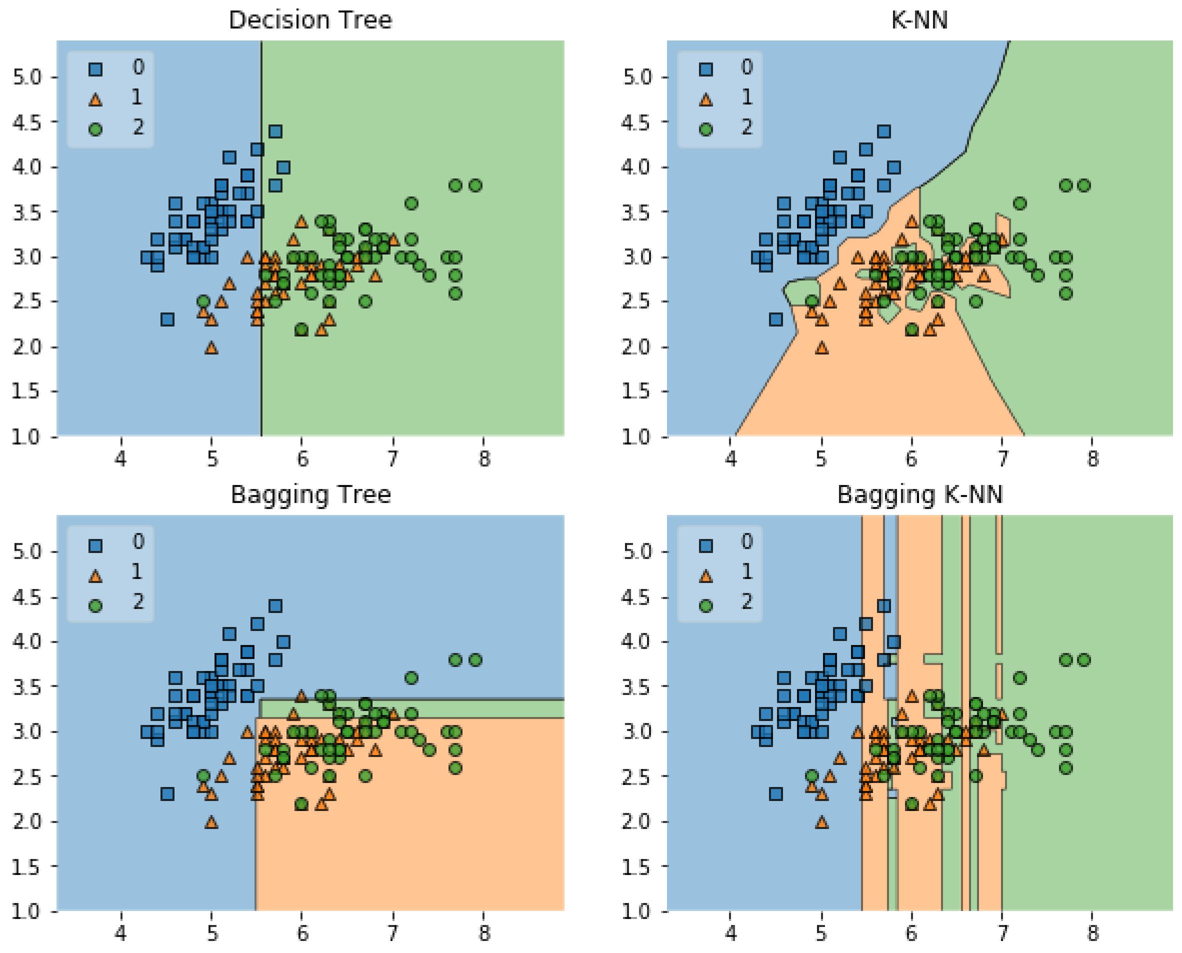
Bagging uses bootstrap sampling to obtain the data subsets for training the base learners. For aggregating the outputs of base learners, bagging uses voting for classification and averaging for regression.

In [2]:
```
%matplotlib inline

import itertools
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import cross_val_score, train_test_split

from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions

np.random.seed(0)
```

In [3]:
```
iris = datasets.load_iris()
X, y = iris.data[:, 0:2], iris.target

clf1 = DecisionTreeClassifier(criterion='entropy', max_depth=1)
clf2 = KNeighborsClassifier(n_neighbors=1)

bagging1 = BaggingClassifier(base_estimator=clf1, n_estimators=10, max_samples=0
bagging2 = BaggingClassifier(base_estimator=clf2, n_estimators=10, max_samples=0
```

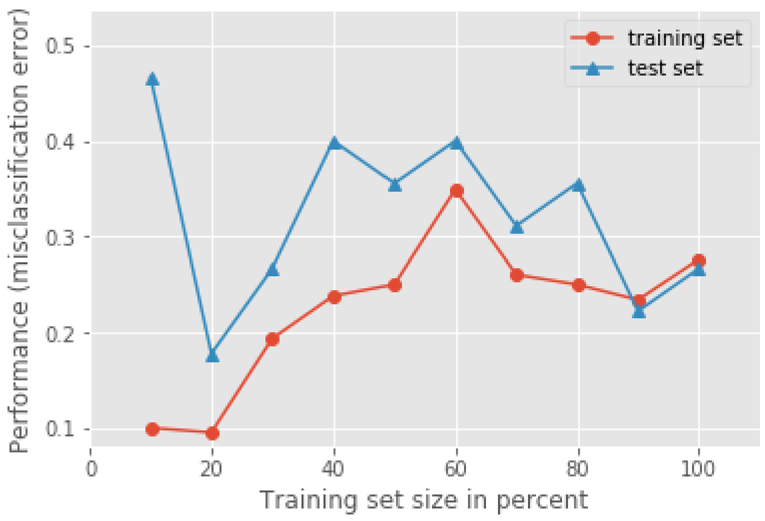
In [4]:
```python
label = ['Decision Tree', 'K-NN', 'Bagging Tree', 'Bagging K-NN']
clf_list = [clf1, clf2, bagging1, bagging2]

fig = plt.figure(figsize=(10, 8))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for clf, label, grd in zip(clf_list, label, grid):
    scores = cross_val_score(clf, X, y, cv=3, scoring='accuracy')


    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label)

    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y, clf=clf, legend=2)
    plt.title(label)

plt.show()
```

```
Accuracy: 0.63 (+/- 0.02) [Decision Tree]
Accuracy: 0.70 (+/- 0.02) [K-NN]
Accuracy: 0.67 (+/- 0.07) [Bagging Tree]
Accuracy: 0.68 (+/- 0.06) [Bagging K-NN]
```



The figure above shows the decision boundary of a decision tree and k-NN classifiers along with their bagging ensembles applied to the Iris dataset. The decision tree shows axes parallel boundaries while the $k = 1$ nearest neighbors fits closely to the data points. The bagging ensembles were trained using 10 base estimators with 0.8 subsampling of training data and 0.8 subsampling of features. The decision tree bagging ensemble achieved higher accuracy in comparison to k-NN bagging ensemble because k-NN are less sensitive to perturbation on training samples and therefore they are called *stable learners*. Combining stable learners is less advantageous since the ensemble will not help improve generalization performance.

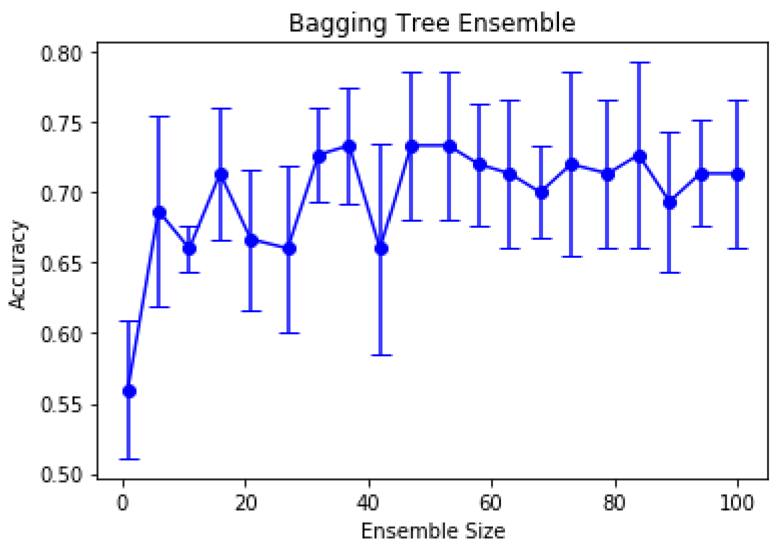
In [5]:
```
#plot learning curves
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

plt.figure()
plot_learning_curves(X_train, y_train, X_test, y_test, bagging1, print_model=Fal
plt.show()
```



The figure above shows learning curves for the bagging tree ensemble. We can see an average error of 0.3 on the training data and a U-shaped error curve for the testing data. The smallest gap between training and test errors occurs at around 80% of the training set size.

In [6]:
```
#Ensemble Size
# num_est = map(int, np.linspace(1,100,20))
num_est = np.linspace(1,100,20).astype(int)
bg_clf_cv_mean = []
bg_clf_cv_std = []
for n_est in num_est:
    bg_clf = BaggingClassifier(base_estimator=clf1, n_estimators=n_est, max_samp
    scores = cross_val_score(bg_clf, X, y, cv=3, scoring='accuracy')
    bg_clf_cv_mean.append(scores.mean())
    bg_clf_cv_std.append(scores.std())
```

In [7]:
```
plt.figure()
(_, caps, _) = plt.errorbar(num_est, bg_clf_cv_mean, yerr=bg_clf_cv_std, c='blue
# caps = plt.errorbar(num_est, bg_clf_cv_mean, yerr=bg_clf_cv_std, c='blue', fmt=
for cap in caps:
    cap.set_markeredgewidth(1)
plt.ylabel('Accuracy'); plt.xlabel('Ensemble Size'); plt.title('Bagging Tree Ense
plt.show()
```



The figure above shows how the test accuracy improves with the size of the ensemble. Based on cross-validation results, we can see the accuracy increases until approximately 10 base estimators and then plateaus afterwards. Thus, adding base estimators beyond 10 only increases computational complexity without accuracy gains for the Iris dataset.

A commonly used class of ensemble algorithms are forests of randomized trees. In **random forests**, each tree in the ensemble is built from a sample drawn with replacement (i.e. a bootstrap sample) from the training set. In addition, instead of using all the features, a random subset of features is selected further randomizing the tree. As a result, the bias of the forest increases slightly but due to averaging of less correlated trees, its variance decreases resulting in an overall better model.

In **extremely randomized trees** algorithm randomness goes one step further: the splitting thresholds are randomized. Instead of looking for the most discriminative threshold, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule. This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias.

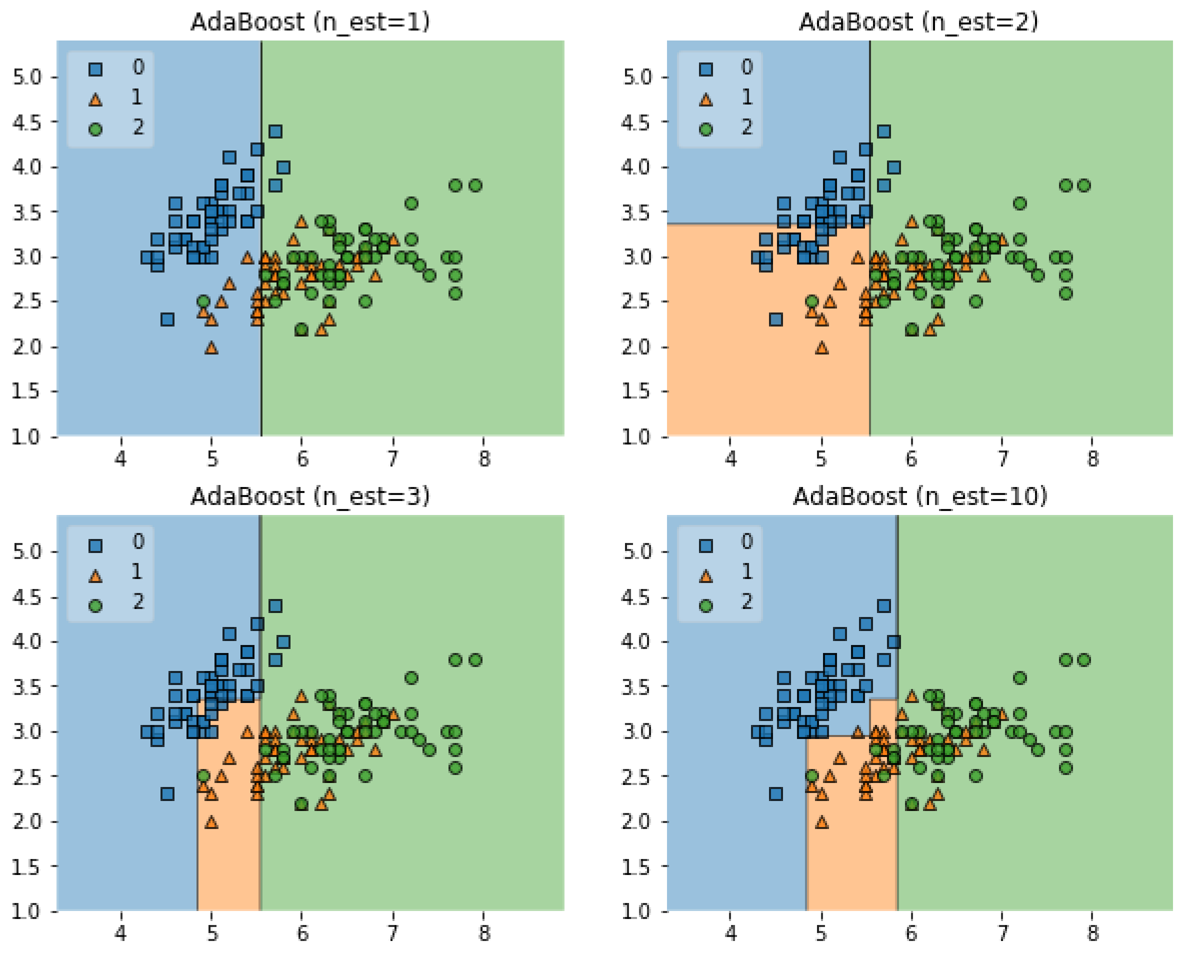
Boosting

Boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners (models that are only slightly better than random guessing, such as small decision trees) to weighted versions of the data, where more weight is given to examples that were mis-classified by earlier rounds. The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression) to produce the final prediction. The principal difference between boosting and the committee methods such as bagging is that base learners are trained in sequence on a weighted version of the data.

In [8]:
```
import itertools
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score, train_test_split

from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions
```

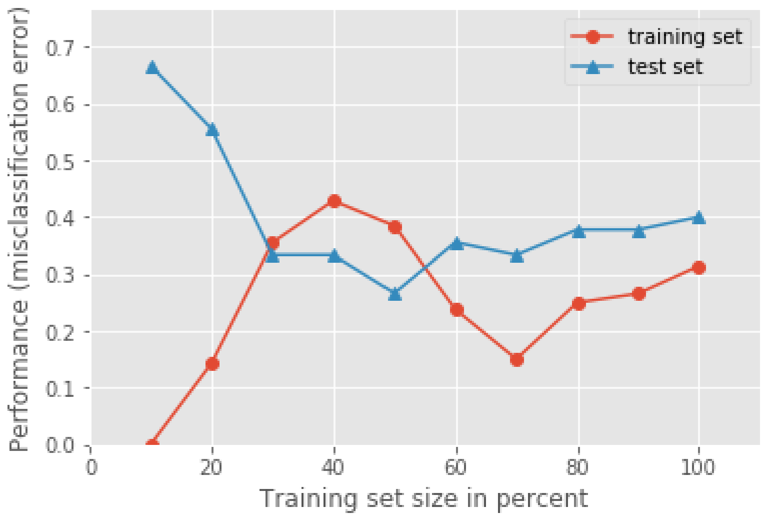
In [9]:
```
iris = datasets.load_iris()
X, y = iris.data[:, 0:2], iris.target

#XOR dataset
#X = np.random.randn(200, 2)
#y = np.array(map(int,np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)))

clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)

num_est = [1, 2, 3, 10]
label = ['AdaBoost (n_est=1)', 'AdaBoost (n_est=2)', 'AdaBoost (n_est=3)', 'AdaBo
```

In [10]:
```
fig = plt.figure(figsize=(10, 8))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for n_est, label, grd in zip(num_est, label, grid):
    boosting = AdaBoostClassifier(base_estimator=clf, n_estimators=n_est)
    boosting.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y, clf=boosting, legend=2)
    plt.title(label)

plt.show()
```



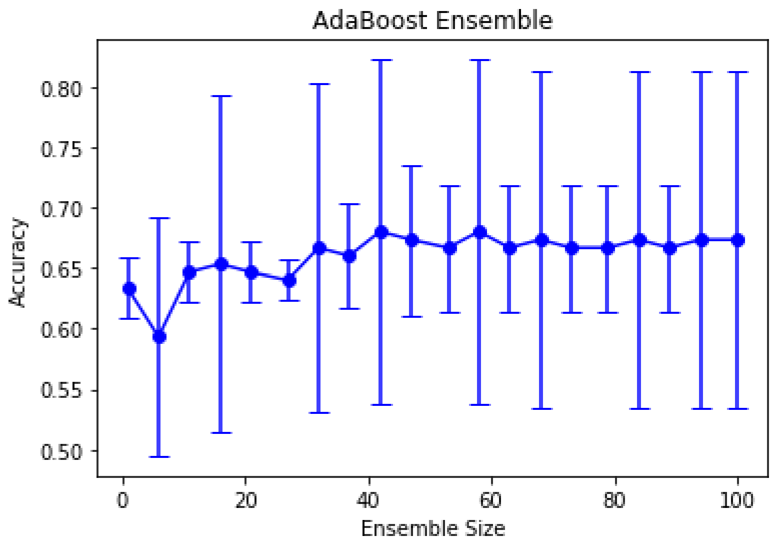
The AdaBoost algorithm is illustrated in the figure above. Each base learner consists of a decision tree with depth 1, thus classifying the data based on a feature threshold that partitions the space into two regions separated by a linear decision surface that is parallel to one of the axes.

In [11]:
```
#plot learning curves
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

boosting = AdaBoostClassifier(base_estimator=clf, n_estimators=10)

plt.figure()
plot_learning_curves(X_train, y_train, X_test, y_test, boosting, print_model=Fals
plt.show()
```



In [12]:
```
#Ensemble Size
#num_est = map(int, np.linspace(1,100,20))
num_est = np.linspace(1,100,20).astype(int)
bg_clf_cv_mean = []
bg_clf_cv_std = []
for n_est in num_est:
    ada_clf = AdaBoostClassifier(base_estimator=clf, n_estimators=n_est)
    scores = cross_val_score(ada_clf, X, y, cv=3, scoring='accuracy')
    bg_clf_cv_mean.append(scores.mean())
    bg_clf_cv_std.append(scores.std())
```

In [13]:
```
plt.figure()
(_, caps, _) = plt.errorbar(num_est, bg_clf_cv_mean, yerr=bg_clf_cv_std, c='blue
for cap in caps:
    cap.set_markeredgewidth(1)
plt.ylabel('Accuracy'); plt.xlabel('Ensemble Size'); plt.title('AdaBoost Ensembl
plt.show()
```



The figure above shows how the test accuracy improves with the size of the ensemble.

Gradient Tree Boosting is a generalization of boosting to arbitrary differentiable loss functions. It can be used for both regression and classification problems.

Stacking

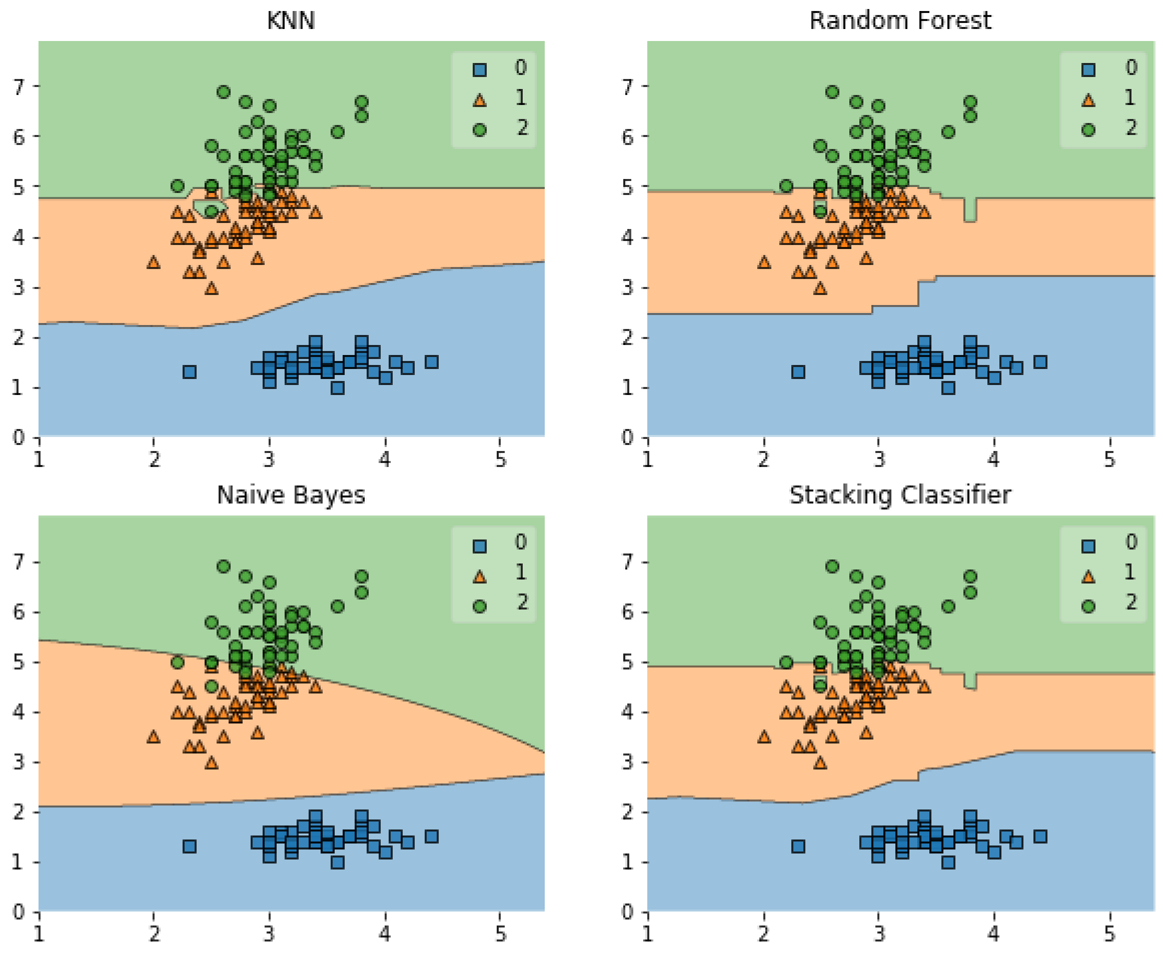
Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on complete training set then the meta-model is trained on the outputs of base level model as features. The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous.

In [14]:
```
import itertools
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from sklearn import datasets

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier

from sklearn.model_selection import cross_val_score, train_test_split

from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions
```

In [15]:
```
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target

clf1 = KNeighborsClassifier(n_neighbors=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],
                          meta_classifier=lr)
```

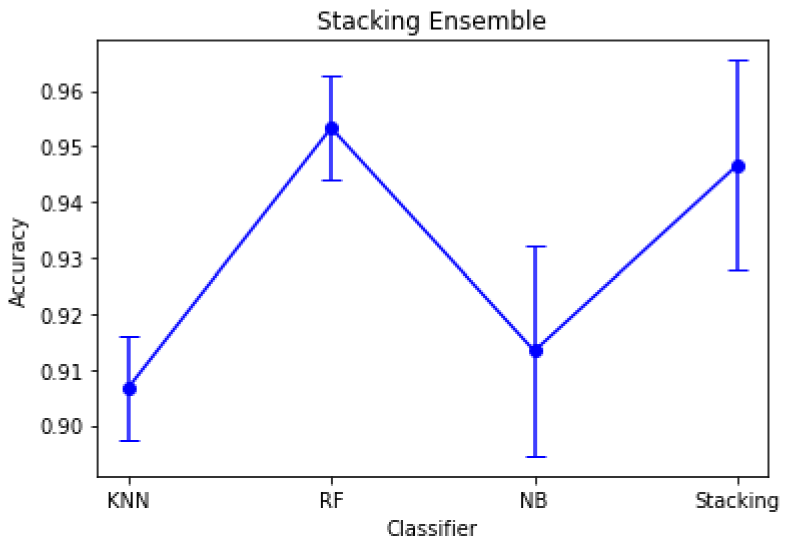
```
In [16]: label = ['KNN', 'Random Forest', 'Naive Bayes', 'Stacking Classifier']
         clf_list = [clf1, clf2, clf3, sclf]

         fig = plt.figure(figsize=(10,8))
         gs = gridspec.GridSpec(2, 2)
         grid = itertools.product([0,1],repeat=2)

         clf_cv_mean = []
         clf_cv_std = []
         for clf, label, grd in zip(clf_list, label, grid):

             scores = cross_val_score(clf, X, y, cv=3, scoring='accuracy')
             print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
             clf_cv_mean.append(scores.mean())
             clf_cv_std.append(scores.std())

             clf.fit(X, y)
             ax = plt.subplot(gs[grd[0], grd[1]])
             fig = plot_decision_regions(X=X, y=y, clf=clf)
             plt.title(label)

         plt.show()
```

```
Accuracy: 0.91 (+/- 0.01) [KNN]
Accuracy: 0.95 (+/- 0.01) [Random Forest]
Accuracy: 0.91 (+/- 0.02) [Naive Bayes]
Accuracy: 0.95 (+/- 0.02) [Stacking Classifier]
```

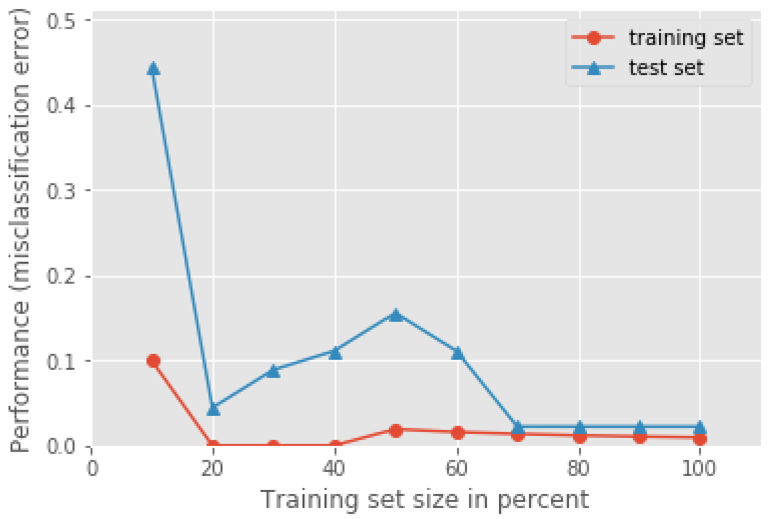


The stacking ensemble is illustrated int the figure above. It consists of k-NN, Random Forest and Naive Bayes base classifiers whose predictions are combined by Lostic Regression as a meta-classifier. We can see the blending of decision boundaries achieved by the stacking classifier.

In [17]:
```
#plot classifier accuracy
plt.figure()
(_, caps, _) = plt.errorbar(range(4), clf_cv_mean, yerr=clf_cv_std, c='blue', fm
for cap in caps:
    cap.set_markeredgewidth(1)
plt.xticks(range(4), ['KNN', 'RF', 'NB', 'Stacking'])
plt.ylabel('Accuracy'); plt.xlabel('Classifier'); plt.title('Stacking Ensemble')
plt.show()
```



In [18]:
```
#plot learning curves
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

plt.figure()
plot_learning_curves(X_train, y_train, X_test, y_test, sclf, print_model=False, 
plt.show()
```



We can see that stacking achieves higher accuracy than individual classifiers and based on learning curves, it shows no signs of overfitting.

===

END of Coding Exercise, Week 5, Dec 22

CSML1010 Fall 2019

Pete Gray ptgray@my.yorku.ca (mailto:ptgray@my.yorku.ca) YorkU #217653247

Using code from https://blog.statsbot.co/ensemble-learning-d1dcd548e936 (https://blog.statsbot.co/ensemble-learning-d1dcd548e936)