AdaBoost Classification of climate data

In [6]:
```
import numpy as np
import pandas as pa
import matplotlib.pyplot as plt
import random
print(__doc__)
from matplotlib.colors import Normalize
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.grid_search import GridSearchCV
import timeit
from sklearn.ensemble import AdaBoostClassifier
from sklearn import cross_validation
```

```
Automatically created module for IPython interactive environment
```

In [2]:
```
# split the data into training set and test set randomly
climate=np.array(pa.read_fwf('C:\Users\Christina\Desktop\climate.txt'))
random.shuffle(climate)
train_x=climate[:378,:20]
train_y=climate[:378:,-1]
test_x=climate[378:,:20]
test_y=climate[378:,-1]
```

Since main parameter is the n_estimators, I apply cross validation to find the best parameter through the training data

In [19]:
```
ab_model=AdaBoostClassifier()
estimator=[10,20,30,50,70,80,90,100,110,120,150,200,250,300,400,500]
scores=[]
scores_std=[]
for n_estimators in estimator:
    ab_model.n_estimators=n_estimators
    this_scores = cross_validation.cross_val_score(ab_model,train_x,train_y, n_jobs=1)
    scores.append(np.mean(this_scores))
    scores_std.append(np.std(this_scores))
```

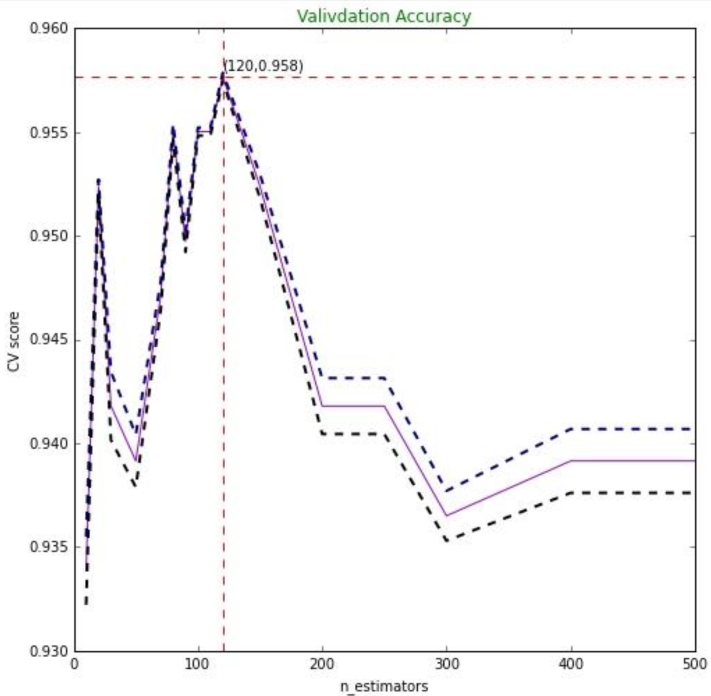
In [23]:
```
scores
```

Out[23]:
```
[0.93386243386243384,
 0.95238095238095222,
 0.94179894179894186,
 0.93915343915343907,
 0.94708994708994698,
 0.955026455026455,
 0.94973544973544965,
 0.95502645502645489,
 0.95502645502645489,
 0.95767195767195767,
 0.95238095238095244,
 0.94179894179894186,
 0.94179894179894186,
 0.9365079365079364,
 0.93915343915343918,
 0.93915343915343918]
```

In [24]:
```
scores_std
```

Out[24]:
```
[0.031965730088874551,
 0.0064801315946645139,
```

In [41]:
```
plt.figure(figsize=(8, 8))
plt.plot(estimator, scores,color="darkviolet")
# plot error lines showing +/- std. errors of the scores
plt.plot(estimator, np.array(scores) + np.array(scores_std) / np.sqrt(len(train_x)),
             '--',color="navy",linewidth=2)
plt.plot(estimator, np.array(scores) - np.array(scores_std) / np.sqrt(len(train_x)),
             '--',color="black",linewidth=2)
plt.ylabel('CV score')
plt.xlabel('n_estimators')
plt.axhline(np.max(scores), linestyle='--', color='red')
plt.axvline(estimator[argmax(scores)],linestyle='--',color='red')
plt.title("Valivdation Accuracy",color="green")
plt.text(120,0.958,"(120,0.958)",color='black')
```



From the above we can see that the optimal n_estimators is 120 with the accuracy 0.958. Then we train the whole training data with the optimal parameters and test on the test data set

In [43]:
```
ada_model=AdaBoostClassifier(n_estimators=120).fit(train_x,train_y)
ada_model
```

Out[43]:
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=120, random_state=None)
```

In [45]:
```
start=timeit.default_timer()
ada_predict=ada_model.predict(test_x)
stop=timeit.default_timer()
count=0
for i in range(162):
  if ada_predict[i]!=test_y[i]:
        count=count+1
print ("error rate:",count/162.)
print ("running time per stream:",(stop-start)/162.)
```

```
('error rate:', 0.024691358024691357)
('running time per stream:', 0.0005403633198163132)
```

Error Rate: 2.47%

Running Time per Stream: 5.4e(-4)

Show the importance of the features

In [48]:
```
importances=ada_model.feature_importances_
importances
```

Out[48]:
```
array([ 0.        ,  0.025     ,  0.125     ,  0.125     ,  0.00833333,
        0.05833333,  0.05833333,  0.03333333,  0.01666667,  0.04166667,
        0.00833333,  0.04166667,  0.06666667,  0.04166667,  0.08333333,
        0.06666667,  0.025     ,  0.075     ,  0.06666667,  0.03333333])
```

In [56]:
```
std = np.std([tree.feature_importances_ for tree in ada_model.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(20):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest
plt.figure(figsize=(8,8))
plt.title("Feature importances")
plt.bar(range(20), importances[indices],
       color="darkcyan",ecolor="blueviolet",align="center")
plt.xticks(range(20), indices)
plt.xlim([-1, 20])
plt.ylim([-0,0.15])
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.show()
```

```
Feature ranking:
1. feature 2 (0.125000)
2. feature 3 (0.125000)
3. feature 14 (0.083333)
4. feature 17 (0.075000)
5. feature 15 (0.066667)
6. feature 12 (0.066667)
7. feature 18 (0.066667)
8. feature 6 (0.058333)
9. feature 5 (0.058333)
10. feature 9 (0.041667)
11. feature 13 (0.041667)
12. feature 11 (0.041667)
13. feature 19 (0.033333)
14. feature 7 (0.033333)
15. feature 16 (0.025000)
16. feature 1 (0.025000)
17. feature 8 (0.016667)
18. feature 4 (0.008333)
19. feature 10 (0.008333)
20. feature 0 (0.000000)
```