RandomForest classification of climate data by Christina Fan (2015/08/05)

In [7]:
```
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 RandomForestClassifier
```

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

Have a look at the data format

In [3]:
```
pa.read_fwf('C:\Users\Christina\Desktop\climate.txt')#show the format of the climate data
```

Out[3]:

	Study	Run	vconst_corr	vconst_2	vconst_3	vconst_4	vconst_5	vconst_7	ah_corr	ah_bolus	...	efficiency_
0	1	1	0.859036	0.927825	0.252866	0.298838	0.170521	0.735936	0.428325	0.567947	...	0.245675
1	1	2	0.606041	0.457728	0.359448	0.306957	0.843331	0.934851	0.444572	0.828015	...	0.616870
2	1	3	0.997600	0.373238	0.517399	0.504993	0.618903	0.605571	0.746225	0.195928	...	0.679355
3	1	4	0.783408	0.104055	0.197533	0.421837	0.742056	0.490828	0.005525	0.392123	...	0.471463
4	1	5	0.406250	0.513199	0.061812	0.635837	0.844798	0.441502	0.191926	0.487546	...	0.551543

Split the data into taining set(0.7) and test set(0.3) randomly

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

train the training set with respect to the parameter "n_estimators" and "max_features" and plot the 2D grid to visiualize the result

In [57]:
```
# Train classifiers
# For an initial search, we set the feature is from 2 to 18 and the n_estimators is from 10 to 2000
param_grid = {"max_features":[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18],
              "n_estimators":[10,15,20,25,30,35,40,50,100,200]}
cv = StratifiedShuffleSplit(train_y, n_iter=5, test_size=0.2, random_state=42)
grid = GridSearchCV(RandomForestClassifier(), param_grid=param_grid, cv=cv)
grid.fit(train_x, train_y)

print("The best parameters are %s with a score of %0.2f"
      % (grid.best_params_, grid.best_score_))
```

```
The best parameters are {'max_features': 17, 'n_estimators': 20} with a score of 0.98
```

```
In [58]: grid
```

```
Out[58]: GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  1. ...,  1.  1.], n_iter=5, test_size=0.2, random
_state=42),
       error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'max_features': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18], '
n_estimators': [10, 15, 20, 25, 30, 35, 40, 50, 100, 200]},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring=None,
       verbose=0)
```

```
In [60]: feature_range=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]
estimator_range=[10,15,20,25,30,35,40,50,100,200]
scores = [x[1] for x in grid.grid_scores_]
scores = np.array(scores).reshape(len(feature_range), len(estimator_range))
scores
```

```
Out[60]: array([[ 0.96842105,  0.96315789,  0.96842105,  0.96842105,  0.96842105,
          0.96842105,  0.96842105,  0.96842105,  0.96842105,  0.96842105],
        [ 0.96052632,  0.96578947,  0.96842105,  0.96842105,  0.96842105,
          0.96842105,  0.96842105,  0.96842105,  0.96842105,  0.96842105],
        [ 0.96842105,  0.96842105,  0.97105263,  0.96842105,  0.96842105,
          0.96842105,  0.96842105,  0.96842105,  0.96842105,  0.96842105],
        [ 0.96842105,  0.97105263,  0.97105263,  0.96578947,  0.97105263,
          0.96842105,  0.96842105,  0.96842105,  0.96842105,  0.96842105],
        [ 0.97105263,  0.96842105,  0.96842105,  0.97105263,  0.96842105,
          0.97105263,  0.97105263,  0.96842105,  0.97105263,  0.96842105],
```

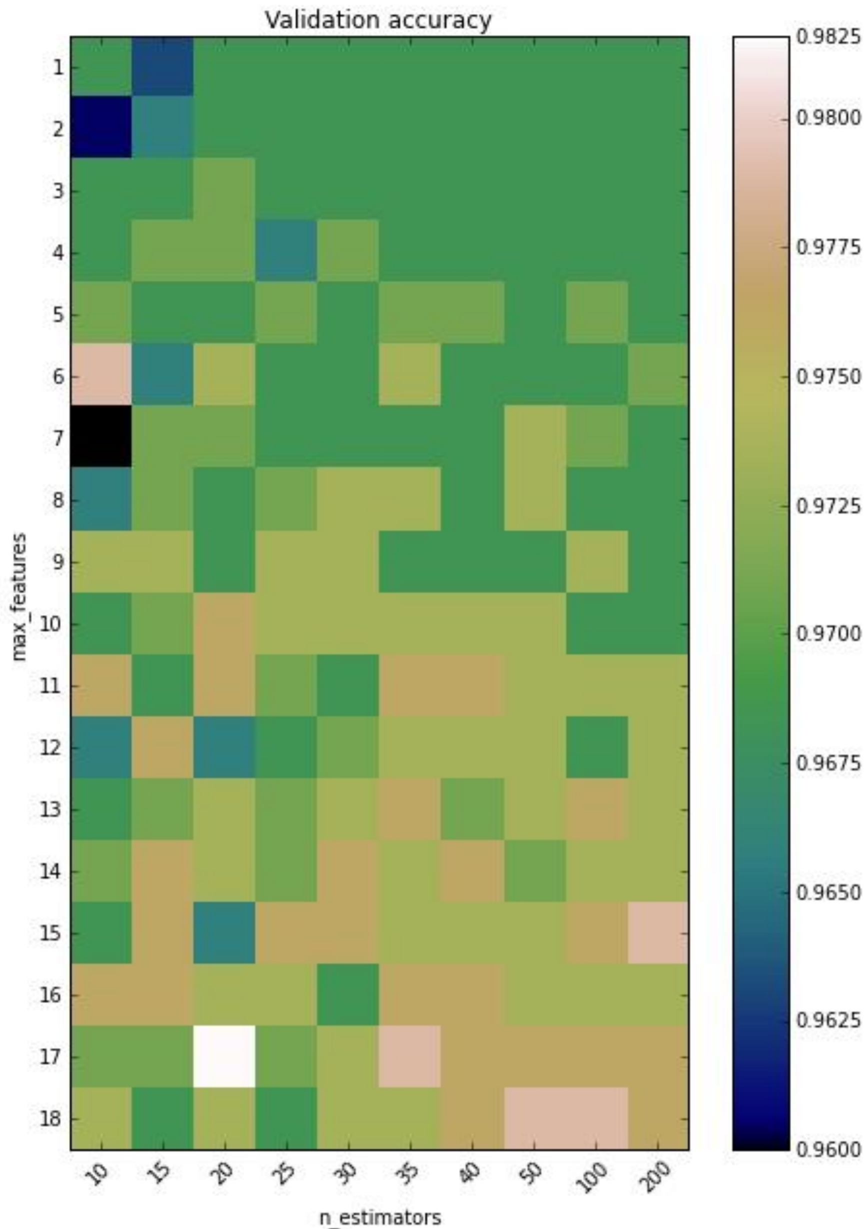
```
In [36]: # Utility function to move the midpoint of a colormap to be around
# the values of interest.

class MidpointNormalize(Normalize):

    def __init__(self, vmin=None, vmax=None, midpoint=None, clip=False):
        self.midpoint = midpoint
        Normalize.__init__(self, vmin, vmax, clip)

    def __call__(self, value, clip=None):
        x, y = [self.vmin, self.midpoint, self.vmax], [0, 0.5, 1]
        return np.ma.masked_array(np.interp(value, x, y))
```

```
In [80]: # Draw heatmap of the validation accuracy as a function of gamma and C
# The score are encoded as colors with the hot colormap which varies from dark
# red to bright yellow. As the most interesting scores are all located in the
# 0.95 to 0.97 range we use a custom normalizer to set the mid-point to 0.96so
# as to make it easier to visualize the small variations of score values in the
# interesting range while not brutally collapsing all the low score values to
# the same color.
plt.figure(figsize=(8, 8))
plt.subplots_adjust(left=0.2, right=1, bottom=0, top=1)
plt.imshow(scores, interpolation='nearest', cmap=plt.cm.gist_earth,
           norm=MidpointNormalize(vmin=0.96, midpoint=0.97))
plt.ylabel('max_features')
plt.xlabel('n_estimators')
plt.colorbar()
plt.xticks(np.arange(len(estimator_range)),estimator_range, rotation=45)
plt.yticks(np.arange(len(feature_range)),feature_range)
plt.title('Validation accuracy')
plt.show()
```


Validation accuracy
max_features
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
n_estimators
10
15
20
25
30
35
40
50
100
200
0.9825
0.9800
0.9775
0.9750
0.9725
0.9700
0.9675
0.9650
0.9625
0.9600

From the above we can see that the optimal n_estimators is 20 and max_features is 17 with the accuracy 0.97. Then we train the whole training data with the optimal parameters and test on the test data set ¶

In [62]:
```
rf_model=RandomForestClassifier(n_estimators=20,max_features=17).fit(train_x,train_y)
rf_model
```

Out[62]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features=17, max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [64]:
```
start=timeit.default_timer()
rf_predict=rf_model.predict(test_x)
stop=timeit.default_timer()
count=0
for i in range(162):
  if rf_predict[i]!=test_y[i]:
        count=count+1
print ("error rate:",count/162.)
print ("running time per stream:",(stop-start)/162.)
```

```
('error rate:', 0.04938271604938271)
('running time per stream:', 1.083128899683185e-05)
```

Error Rate: 0.049

Running Time per Stream: 1.08e-05

Show the importance of the features

In [65]:
```
importances=rf_model.feature_importances_
```

In [66]:
```
importances
```

Out[66]:
```
array([ 0.        ,  0.01812199,  0.32025041,  0.28756058,  0.00726225,
        0.02977199,  0.00223333,  0.00840308,  0.02247806,  0.00660003,
        0.00673337,  0.0168447 ,  0.03216375,  0.03053713,  0.10295076,
        0.0166147 ,  0.0227347 ,  0.01100816,  0.0093623 ,  0.04836868])
```

In [90]:
```
std = np.std([tree.feature_importances_ for tree in rf_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", yerr=std[indices],ecolor="blueviolet",align="center")
plt.xticks(range(20), indices)
plt.xlim([-1, 20])
plt.ylim([-0.03,0.45])
plt.show()
```

```
Feature ranking:
1. feature 2 (0.320250)
2. feature 3 (0.287561)
3. feature 14 (0.102951)
4. feature 19 (0.048369)
5. feature 12 (0.032164)
6. feature 13 (0.030537)
7. feature 5 (0.029772)
8. feature 16 (0.022735)
9. feature 8 (0.022478)
10. feature 1 (0.018122)
11. feature 11 (0.016845)
12. feature 15 (0.016615)
13. feature 17 (0.011008)
14. feature 18 (0.009362)
15. feature 7 (0.008403)
16. feature 4 (0.007262)
17. feature 10 (0.006733)
18. feature 9 (0.006600)
19. feature 6 (0.002233)
20. feature 0 (0.000000)
```