SVM to classify the climate data

In [37]:
```
from lxml.html import parse
from urllib import urlopen
import numpy as np
import pandas as pa
import matplotlib.pyplot as plt
from sklearn import svm
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
```

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

Have a look at the format of the data

In [38]:
```
pa.read_fwf('C:\Users\Christina\Desktop\climate.txt')#show the format of the climate data
```

Out[38]:

cay_scale	convect_corr	bckgrnd_vdc1	bckgrnd_vdc_ban	bckgrnd_vdc_eq	bckgrnd_vdc_psim	Prandtl	outcome
	0.997518	0.448620	0.307522	0.858310	0.796997	0.869893	0
	0.845247	0.864152	0.346713	0.356573	0.438447	0.512256	1
	0.718441	0.924775	0.315371	0.250642	0.285636	0.365858	1
	0.362751	0.912819	0.977971	0.845921	0.699431	0.475987	1
	0.650223	0.522261	0.043545	0.376660	0.280098	0.132283	1
	0.017487	0.932320	0.329318	0.954123	0.135379	0.294805	1
	0.698107	0.467359	0.637078	0.011251	0.147325	0.213814	1
	0.886522	0.411673	0.481108	0.926546	0.026431	0.092740	1
	0.254944	0.488400	0.053684	0.862226	0.415055	0.487126	1
	0.374270	0.100291	0.213290	0.222860	0.007286	0.420027	1
	0.926424	0.295426	0.804212	0.870840	0.546295	0.884871	1
	0.948095	0.999616	0.728459	0.285888	0.210890	0.833590	1
	0.530425	0.175170	0.544458	0.081392	0.733015	0.531369	0
	0.597281	0.428806	0.401370	0.820446	0.599584	0.135681	1
	0.499516	0.589648	0.014998	0.893355	0.562122	0.028449	1

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

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]
```

In [83]:
```
train_x
```

Out[83]:
```
array([[  1.00000000e+00,   1.51000000e+02,   2.12629184e-01, ...,
          4.19111547e-01,   9.94421327e-01,   4.24921258e-01],
       [  1.00000000e+00,   1.22000000e+02,   2.47917166e-01, ...,
          3.36783675e-01,   8.10902066e-01,   4.12556592e-02],
       [  3.00000000e+00,   1.16000000e+02,   8.69088735e-01, ...,
          1.91352455e-01,   1.64485152e-01,   6.17243327e-02],
       ...,
       [  2.00000000e+00,   1.08000000e+02,   4.36968107e-01, ...,
          5.74838330e-01,   5.84562067e-01,   4.52357342e-01],
       [  3.00000000e+00,   5.60000000e+01,   3.45904385e-01, ...,
          8.33705546e-01,   8.73343656e-01,   9.53905657e-01],
       [  3.00000000e+00,   1.72000000e+02,   5.71211299e-01, ...,
          2.48115920e-01,   4.15337021e-01,   2.06783775e-01]])
```

In [86]:
```
train_y
```

Out[86]:
```
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  0.,  1.,  1.,  1.,  0.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  0.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  0.,  1.,  0.,  1.,  1.,  0.,  1.,  1.,  1.,  1.,
```

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

In [3]:
```
# 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))
```

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

In [4]:
```
x_2d=train_x
y_2d=train_y
```

In [10]:
```
grid
```

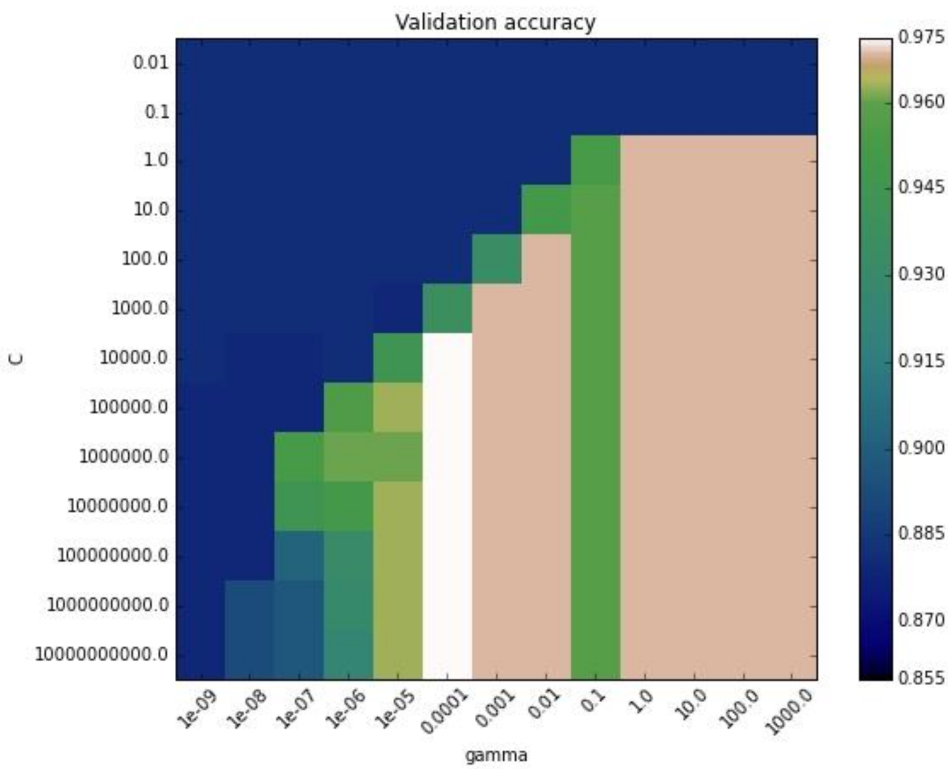
Out[10]:
```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  1. ...,  1.  1.], n_iter=5, test_size=0.2, random
_state=42),
       error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'C': array([  1.00000e-02,   1.00000e-01,   1.00000e+00,   1.00000e+01,
         1.00000e+02,   1.00000e+03,   1.00000e+04,   1.00000e+05,
         1.00000e+06,   1.00000e+07,   1.00000e+08,   1.00000e+09,
         1.00000e+10]), 'gamma': array([  1.00000e-09,   1.00000e-08,   1.00000e-07,   1.00000e-06,
         1.00000e-05,   1.00000e-04,   1.00000e-03,   1.00000e-02,
         1.00000e-01,   1.00000e+00,   1.00000e+01,   1.00000e+02,
         1.00000e+03])},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring=None,
       verbose=0)
```

In [12]:
```
scores = [x[1] for x in grid.grid_scores_]
scores = np.array(scores).reshape(len(C_range), len(gamma_range))
```

In [13]:
```
scores
```

Out[13]:
```
array([[ 0.88157895,  0.88157895,  0.88157895,  0.88157895,  0.88157895,
         0.88157895,  0.88157895,  0.88157895,  0.88157895,  0.88157895,
         0.88157895,  0.88157895,  0.88157895],
       [ 0.88157895,  0.88157895,  0.88157895,  0.88157895,  0.88157895,
         0.88157895,  0.88157895,  0.88157895,  0.88157895,  0.88157895,
         0.88157895,  0.88157895,  0.88157895],
       [ 0.88157895,  0.88157895,  0.88157895,  0.88157895,  0.88157895,
         0.88157895,  0.88157895,  0.88157895,  0.95263158,  0.96842105,
         0.96842105,  0.96842105,  0.96842105],
       [ 0.88157895,  0.88157895,  0.88157895,  0.88157895,  0.88157895,
```

In [30]:
```
# 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, 6))
plt.subplots_adjust(left=.1, right=0.97, bottom=0.1, top=0.97)
plt.imshow(scores, interpolation='nearest', cmap=plt.cm.gist_earth,
           norm=MidpointNormalize(vmin=0.86, midpoint=0.96))
plt.xlabel('gamma')
plt.ylabel('C')
plt.colorbar()
plt.xticks(np.arange(len(gamma_range)), gamma_range, rotation=45)
plt.yticks(np.arange(len(C_range)), C_range)
plt.title('Validation accuracy')
plt.show()
```



From the picture above, we can see that the best gamma and C is 0.0001 and 10000 with the accuracy rate 0.97. Then, we test on the testing data with the optimal parameters

In [34]:

```
#train the whole training data
svm_model=svm.SVC(C=10000,gamma=0.0001).fit(train_x, train_y)
svm_model
```

Out[34]:

```
SVC(C=10000, cache_size=200, class_weight=None, coef0=0.0, degree=3,
  gamma=0.0001, kernel='rbf', max_iter=-1, probability=False,
  random_state=None, shrinking=True, tol=0.001, verbose=False)
```

In [39]:

```
# do the test on the testing data
start=timeit.default_timer()
svm_predict=svm_model.predict(test_x)
stop=timeit.default_timer()
count=0
for i in range(162):
  if svm_predict[i]!=test_y[i]:
        count=count+1
print ("error rate:",count/162.)
print ("running time per stream:",(stop-start)/162.)
```

```
('error rate:', 0.06172839506172839)
('running time per stream:', 5.306090880737624e-06)
```

Error Rate: 6.17%, Running Time per Stream:5.3e-06