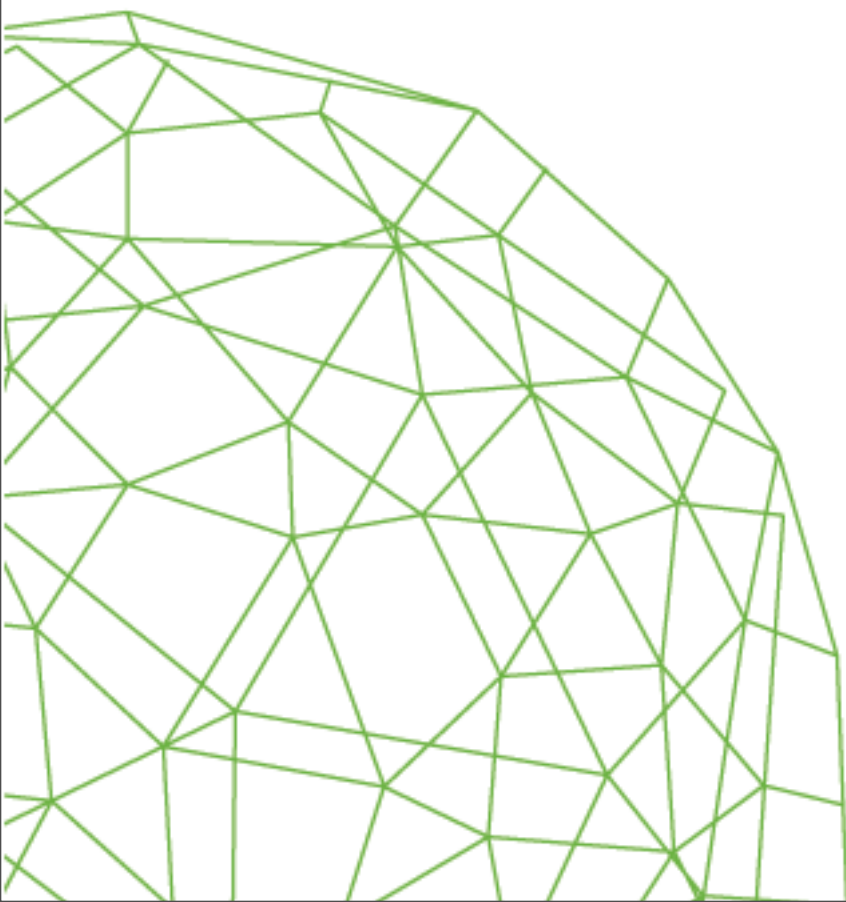


Demo: Machine Learning

Guillermo Cabrera
CMM, Universidad de Chile



Our Data

- We want to classify stars/QSOs/White dwarfs in terms of their colors: SDSS_test.csv
- But, we have a selection of SDSS point sources, along with training sets for three spectroscopically confirmed classes: SDSS_all_classes_21.csv

Lets plot our training data

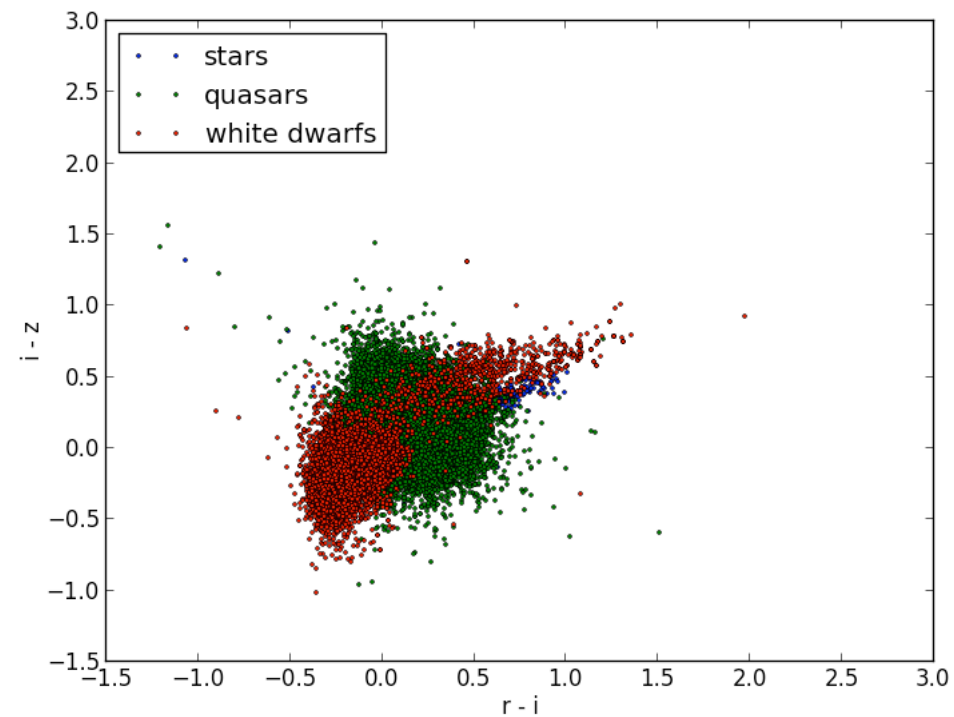
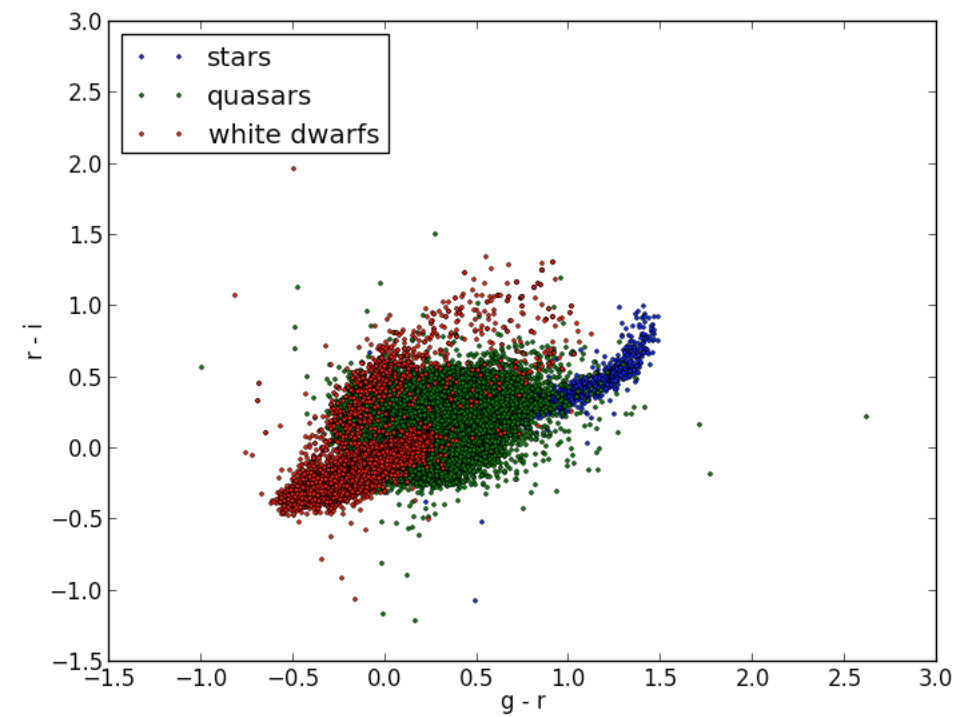
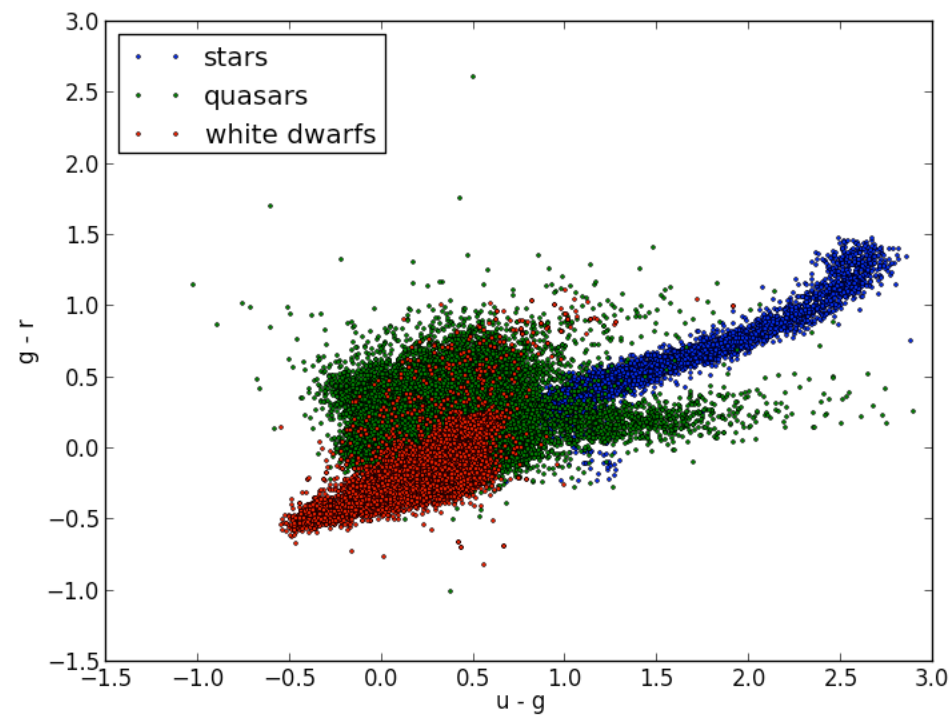
```
# python 01_supervised_plots.py data.dat

import sys
import numpy as np
import pylab as pl
import sklearn as skl

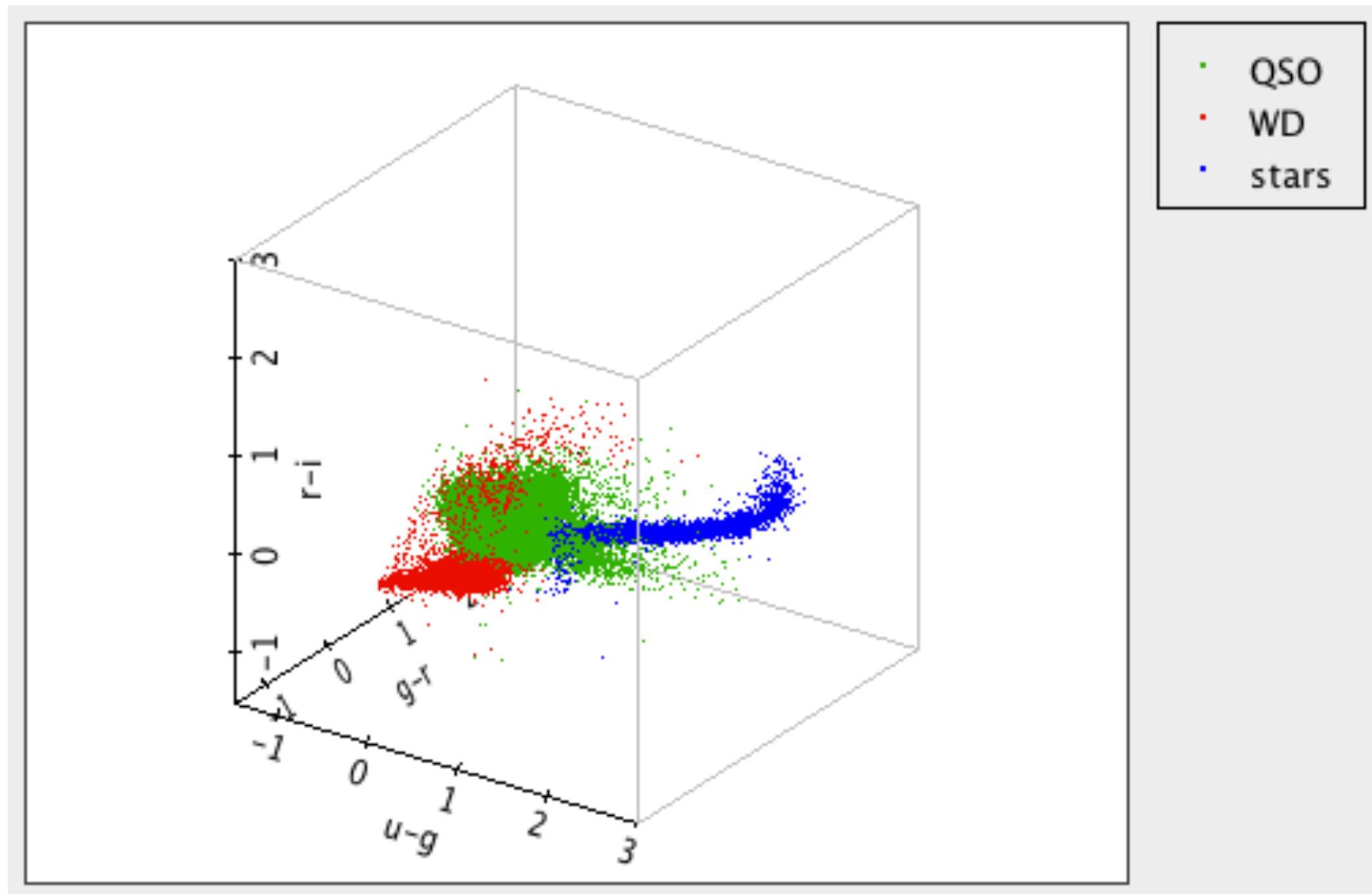
data = np.genfromtxt(sys.argv[1], delimiter=',', skip_header = 1)
classes, u, g, r, i, z = data[:, 0], data[:, 1], data[:, 2], data[:, 3],
data[:, 4], data[:, 5]
colors = u - g, g - r, r - i, i - z
names = np.array(["u - g", "g - r", "r - i", "i - z"])
cl_names = ["stars", "quasars", "white dwarfs"]

for i in range (3):
    pl.clf()
    for cl in np.arange(1, 4):
        criteria = (classes == cl)
        pl.plot (colors[i][criteria], colors[i+1][criteria], "o",
markersize=2, label = cl_names[cl-1])
    #pl.plot (colors[i], colors[i+1], "o", markersize=2)
    pl.xlabel (names[i])
    pl.ylabel (names[i+1])
    pl.legend (loc = "upper left")
    pl.xlim([-1.5, 3])
    pl.ylim([-1.5, 3])
    pl.savefig ("plot_sup_" + str(i) + "_" + names[i] + "-" + names[i+1])
```

Lets plot our training data



Lets plot our training data



Using scikit-learn

scikit-learn.org

```
# python 02_supervised_train.py data.dat
# Basic SVM using the whole training set for testing.

import sys
import numpy as np
import pylab as pl
from sklearn import svm

data = np.genfromtxt(sys.argv[1], delimiter=',', skip_header = 1)
classes, u, g, r, i, z = data[:, 0], data[:, 1], data[:, 2], data[:, 3], data[:, 4],
data[:, 5]
colors = u - g, g - r, r - i, i - z
names = np.array(["u - g", "g - r", "r - i", "i - z"])
cl_names = ["stars", "quasars", "white dwarfs"]

data_svm = np.array(colors).transpose()
clf = svm.SVC(kernel = "linear")
print len(classes)
clf.fit(data_svm, classes)
print clf

pred_class = clf.predict(data_svm)
N_match = (pred_class == classes).sum()
print "N_match = ", N_match
acc = 1. * N_match / len(classes)
print "Accuracy = ", acc
```

Using scikit-learn

scikit-learn.org

```
$ python 02_supervised_train.py SDSS_all_classes_21.csv
83102
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
gamma=0.0, kernel=linear, max_iter=-1, probability=False,
shrinking=True, tol=0.001, verbose=False)
N_match = 79212
Accuracy = 0.953190055594
```

But...are we using it correctly?

Holdout validation

```
# python 03_supervised_test.py data.dat
# Basic SVM using the holdout method

import sys
import numpy as np
import pylab as pl
from sklearn import svm, cross_validation

data = np.genfromtxt(sys.argv[1], delimiter=',', skip_header = 1)
...

print N
X_train, X_test, y_train, y_test = cross_validation.train_test_split (data_svm,
classes, test_size=1./3., random_state=0)
print X_train.shape, y_train.shape
print X_test.shape, y_test.shape

clf = svm.SVC(kernel = "linear")
clf.fit(X_train, y_train)

pred_class = clf.predict(X_test)
N_match = (pred_class == y_test).sum()
print "N_match = ", N_match
acc = 1. * N_match / len(pred_class)
print "Accuracy = ", acc
```


Holdout validation

```
$ python 03_supervised_test.py SDSS_all_classes_21.csv
83102
(55401, 4) (55401,)
(27701, 4) (27701,)
N_match = 26444
Accuracy = 0.954622576802
```

Much better, isn't it?
**But is it possible that we obtained
these results by chance?**

Getting the standard deviation

```
# python 04_supervised_randomSS.py data.dat
# Basic SVM using the random subsampling method

import sys
import numpy as np
import pylab as pl
from sklearn import svm, cross_validation

data = np.genfromtxt(sys.argv[1], delimiter=',', skip_header = 1)
classes, u, g, r, i, z = data[:, 0], data[:, 1], data[:, 2], data[:, 3], data[:, 4], data[:, 5]
colors = u - g, g - r, r - i, i - z
names = np.array(["u - g", "g - r", "r - i", "i - z"])
cl_names = ["stars", "quasars", "white dwarfs"]
N = len(classes)
data_svm = np.array(colors).transpose()

print N

clf = svm.SVC(kernel = "linear")
scores = cross_validation.cross_val_score(clf, data_svm, classes, cv=5)

print "Accuracy = ", scores.mean(), "+-", scores.std()
```

Getting the standard deviation

```
$ python 04_supervised_randomSS.py SDSS_all_classes_21.csv  
83102  
Accuracy = 0.953250216061 +- 0.00163561716782
```

That's great!!

But maybe we're using more data than needed.

Accuracy vs N

```
# python 05_supervised_accvsN.py data.dat  
# Plot acc vs N
```

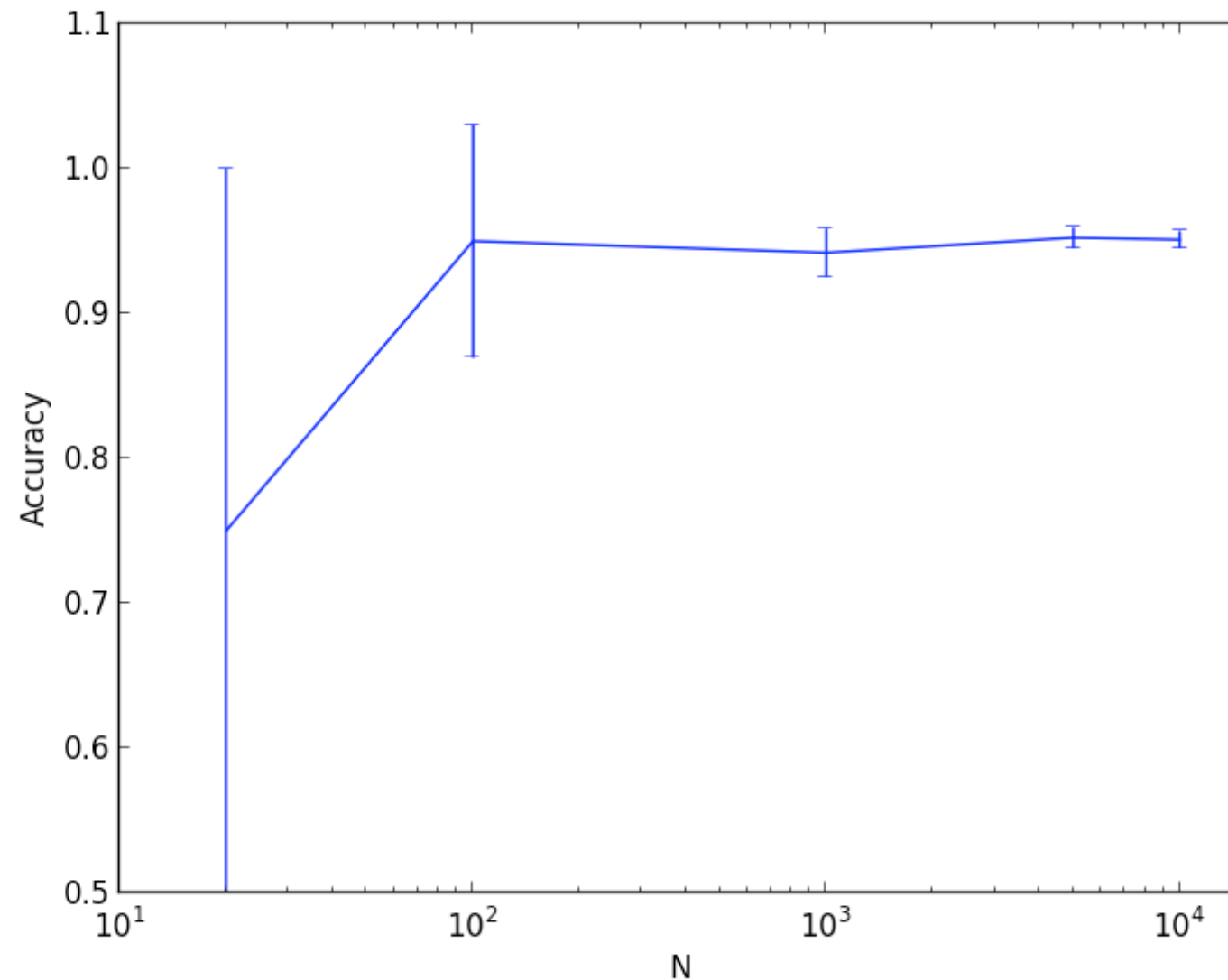
```
import sys
```

```
...
```

```
Ns = np.array([20, 100, 1000, 5000, 10000])  
scores = np.zeros(len(Ns))  
stds = np.zeros(len(Ns))  
i_s = np.arange(N)  
np.random.shuffle(i_s)  
for i in range(len(Ns)):  
    N = Ns[i]  
    i1 = i_s[:N]  
    scores_i = cross_validation.cross_val_score(clf, data_svm[i1], classes[i1], cv=10)  
    scores[i] = scores_i.mean()  
    stds[i] = scores_i.std()
```

```
pl.clf()  
fig = pl.figure()  
ax = fig.add_subplot(1,1,1)  
ax.errorbar (Ns, scores, yerr = stds)  
ax.set_xscale("log")  
ax.set_xlabel("N")  
ax.set_ylabel("Accuracy")  
ax.set_xlim([10, 15000])  
pl.savefig ("accvsN")
```

Accuracy vs N



Hmmm...I think 1000 objects is good enough.
Can I do better?

Choosing SVM parameters

$$\min_{\mathbf{w}, \xi, b} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \right\}$$

subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0$

What value for C shall we use?

The default is 1

Accuracy vs N

```
# python 06_supervised_choosingC.py data.dat
# Plot acc vs N
```

```
...
from sklearn.grid_search import GridSearchCV
...
N = 1000
i_s = np.arange(len(classes))
np.random.shuffle(i_s)
i_s = i_s[:N]
data_svm = np.array(colors).transpose()[i_s]
classes = classes[i_s]

C_range = 10. ** np.arange(-5, 5)
param_grid = dict(C=C_range)
grid = GridSearchCV(svm.SVC(kernel = "linear"), param_grid=param_grid, cv=10)
grid.fit (data_svm, classes)

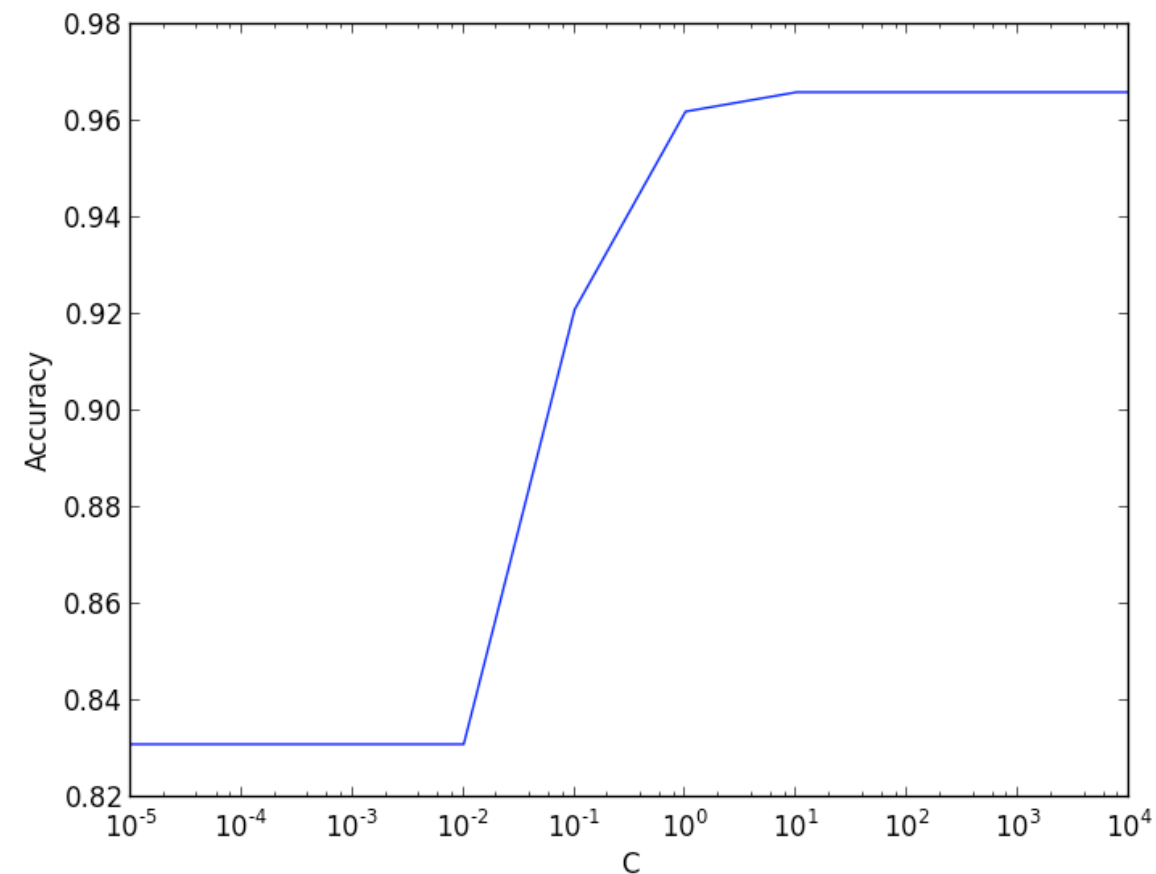
print "The best classifier is: ", grid.best_estimator_
# plot the scores of the grid grid_scores_ contains parameter settings and scores
score_dict = grid.grid_scores_
# We extract just the scores
scores = [x[1] for x in score_dict]

pl.clf()
fig = pl.figure()
ax = fig.add_subplot(1,1,1)
ax.plot (C_range, scores)
...
```

Accuracy vs C

```
$ python 06_supervised_choosingC.py SDSS_all_classes_21.csv
The best classifier is: SVC(C=10.0, cache_size=200,
class_weight=None, coef0=0.0, degree=3, gamma=0.0,
    kernel=linear, max_iter=-1, probability=False, shrinking=True,
tol=0.001,
    verbose=False)
[0.830999999999999996, 0.830999999999999996, 0.830999999999999996,
0.830999999999999996, 0.921000000000000004, 0.961999999999999997,
0.965999999999999997, 0.965999999999999997, 0.965999999999999997,
0.965999999999999997]
```


Accuracy vs C



What if we use a different kernel?

Using an RBF Kernel

$$K(x, x') = \exp(-\gamma|x - x'|^2)$$

Now we have to choose C and gamma!!!

Using a RBF Kernel

```
# python 07_supervised_RBF.py data.dat
# Plot acc vs C and gamma

...

C_range = 10. ** np.arange(-5, 5)
gamma_range = 10. ** np.arange(-5, 5)
param_grid = dict(gamma=gamma_range, C=C_range)
grid = GridSearchCV(svm.SVC(kernel = "rbf"), param_grid=param_grid, cv=10)
grid.fit (data_svm, classes)

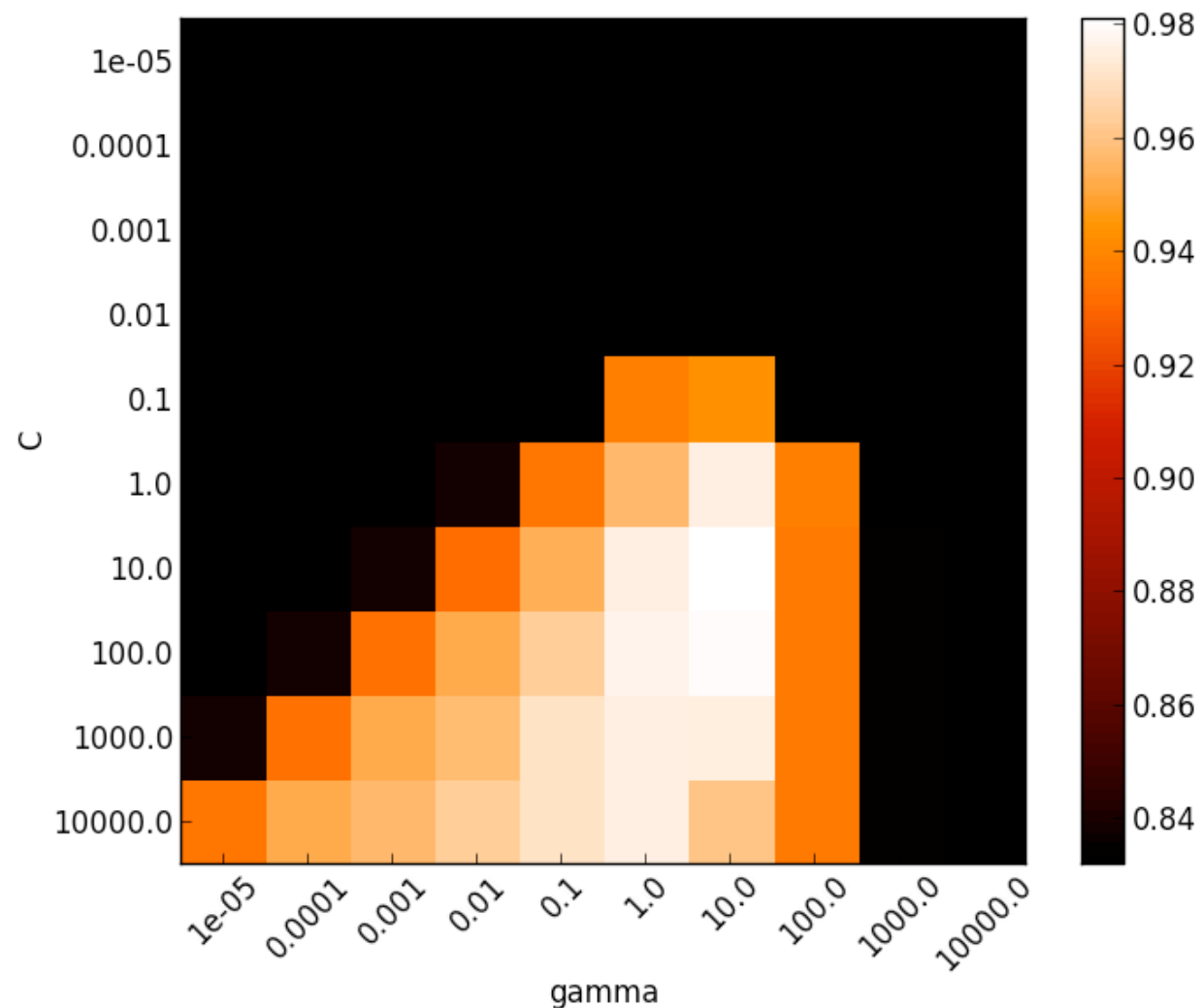
...

# We extract just the scores
scores = [x[1] for x in score_dict]
scores = np.array(scores).reshape(len(C_range), len(gamma_range))

# Make a nice figure
pl.figure(figsize=(8, 6))
pl.subplots_adjust(left=0.15, right=0.95, bottom=0.15, top=0.95)
pl.imshow(scores, interpolation='nearest', cmap=pl.cm.gist_heat)
pl.xlabel('gamma')
pl.ylabel('C')
pl.colorbar()
pl.xticks(np.arange(len(gamma_range)), gamma_range, rotation=45)
pl.yticks(np.arange(len(C_range)), C_range)
pl.savefig("RBF_C_gamma")
```

Using a RBF Kernel

```
$ python 07_supervised_RBF.py SDSS_all_classes_21.csv  
The best classifier is: SVC(C=10.0, cache_size=200, class_weight=None,  
coef0=0.0, degree=3, gamma=10.0, kernel=rbf, max_iter=-1, probability=False,  
shrinking=True, tol=0.001, verbose=False)
```



Now we have a good classifier,
lets go to the unlabeled data.

```

# python 08_supervised_final.py training_data.csv unlabeled_data.csv classes.csv
# Classify for real

import sys
import numpy as np
import pylab as pl
from sklearn import svm

# load training data
data = np.genfromtxt(sys.argv[1], delimiter=',', skip_header = 1)
classes, u, g, r, i, z = data[:, 0], data[:, 1], data[:, 2], data[:, 3], data[:, 4], data[:, 5]
colors = u - g, g - r, r - i, i - z
names = np.array(["u - g", "g - r", "r - i", "i - z"])
cl_names = ["stars", "quasars", "white dwarfs"]

N = 1000
i_s = np.arange(len(classes))
np.random.shuffle(i_s)
i_s = i_s[:N]
data_svm = np.array(colors).transpose()[i_s]
classes = classes[i_s]

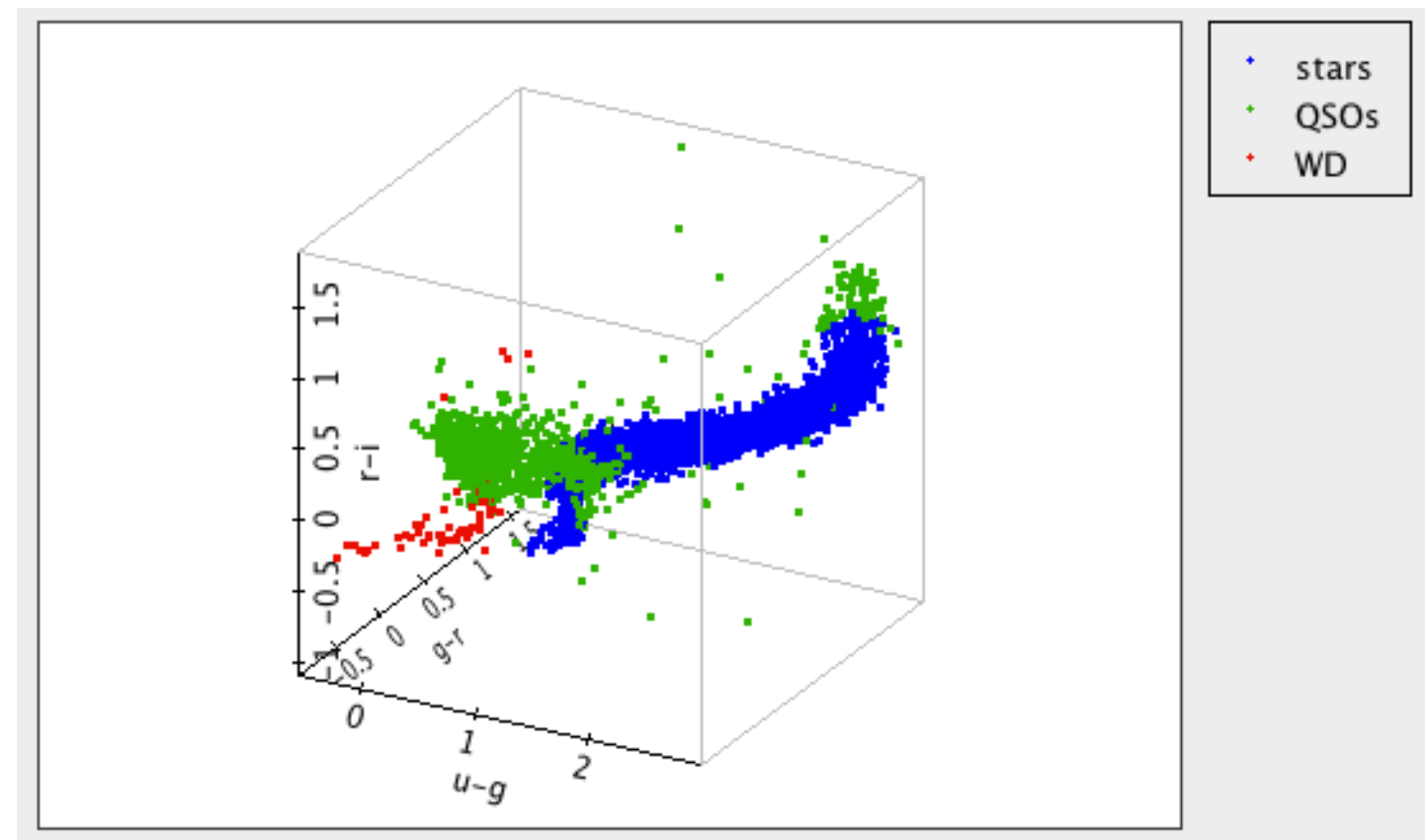
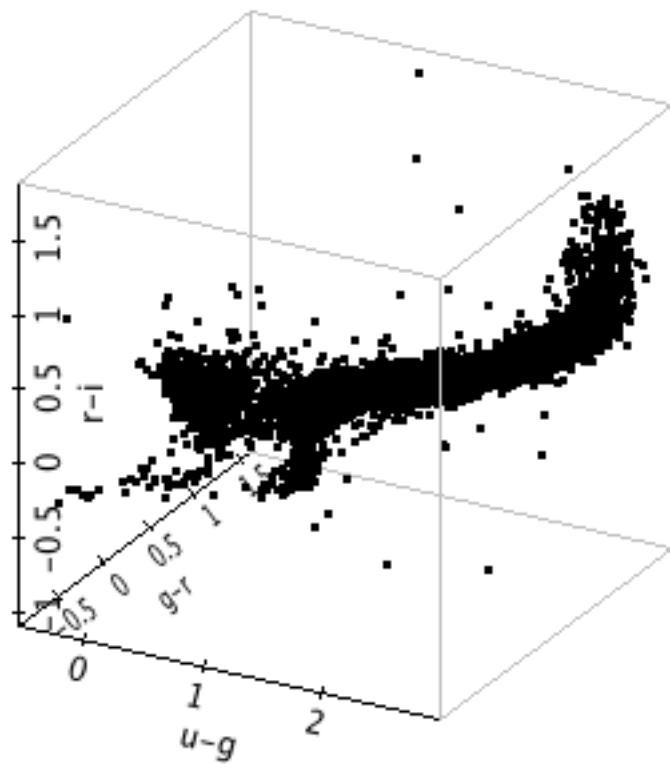
# load unlabeled data
data_ul = np.genfromtxt(sys.argv[2], delimiter=',', skip_header = 1)
u, g, r, i, z = data_ul[:, 0], data_ul[:, 1], data_ul[:, 2], data_ul[:, 3], data_ul[:, 4]
colors_ul = u - g, g - r, r - i, i - z
data_ul = np.array(colors_ul).transpose()

clf = svm.SVC (kernel = "rbf", C = 10., gamma = 10.)
clf.fit (data_svm, classes)
pred_class = clf.predict (data_ul)

out = np.zeros ((len(pred_class), data_ul.shape[1] + 1))
out[:, :data_ul.shape[1]] = data_ul[:, :]
out[:, -1] = pred_class [:]
np.savetxt(sys.argv[3], out, delimiter=",")

```

Is it good enough?



Summary

- Get training data (and hopefully visualize it).
- Train classifier but validate with unseen data.
- Determine classification parameters.
- Once all that is done: train with the whole dataset.
- Classify your new objects.

THANKS!

