Xe1T_BandStuff

August 28, 2017

1 ER/NR Band Discrimination

In [1]:
```
!rm -Rf ~/.cache ./pax_*

import logging
logging.getLogger('rootpy.stl').setLevel(logging.CRITICAL)
logging.getLogger('hax').setLevel(logging.CRITICAL)
logging.getLogger('requests').setLevel(logging.CRITICAL)
logging.getLogger('ROOT').setLevel(logging.CRITICAL)
logging.basicConfig(level=logging.CRITICAL)

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Set name of notebook for folder to save plots in
notebook_name = 'Xe1T_BandStuff'

# run imports
%run "../helpers/initialize_midway.ipynb"

# get all plotfunctions
%run "../helpers/plot_functions.ipynb"
```

```
Initialization done, Notebook was last run on: 28/08/2017
```

In [3]:
```
hax.misc.code_hider()
```

Out[3]:
```
<IPython.core.display.HTML object>
```

In [4]:
```
# Initialize pax/hax

pax_version = '6.6.5'

# pax configuration
from pax import units, configuration, datastructure
pax_config = configuration.load_configuration('XENON1T')
tpc_radius = pax_config['DEFAULT']['tpc_radius']
```

```
tpc_height = pax_config['DEFAULT']['tpc_length']

# hax configuration
import hax
from lax.lichen import Lichen, RangeLichen, ManyLichen, StringLichen
from lax import __version__ as lax_version

hax.init(minitree_paths = ['/home/danielfm/',
                           '/scratch/midway2/berget2/minitrees/',
                           '/scratch/midway2/breur/miniforest/',
                           '/project/lgrandi/xenon1t/minitrees/pax_v' + pax_version+'/',
                             '/project2/lgrandi/xenon1t/minitrees/pax_v' + pax_version+'/'
         pax_version_policy = pax_version)
#make_minitrees = False
```

In [5]:
```
# Parse Datasets
dsets = hax.runs.datasets

# select the latest versions
dsets = dsets[(dsets.pax_version == '6.6.5')]

# Select tags
dsets = hax.runs.tags_selection(dsets, include=['sciencerun1'], exclude=['MVoff,blinded'
       'blinded,MVoff', 'blinded,earthquake', 'blinded,flash',
       'blinded,messy,PMTtrip,MVoff,flash', 'blinded,messy,flash', 'messy',
       'messy,flash', 'messy,flash,pmttrip', 'messy,pmttrip,flash',
       'messy,pmttrip,flash,ramping', 'noise', 'test', 'trip,messy'])

# Select with a processed location
dsets = dsets[(dsets.location != '')]
print('We start with %i processed SR1 datasets' % len(dsets))

dsets_list = []
sources = ['Rn220','AmBe','neutron_generator']
for i, source in enumerate(sources):
    dsets_list.append(dsets[(dsets.source__type == source) ])
    print('%s Datasets: %i' % (source, len(dsets_list[i])) )
```

```
We start with 3140 processed SR1 datasets
Rn220 Datasets: 238
AmBe Datasets: 332
neutron_generator Datasets: 47
```

In [6]:
```
import lax
lax_version = lax.__version__
from lax.lichens import sciencerun1
```

```
list_of_treemakers = ['Corrections','Basics','Extended','Fundamentals', 'TotalProperties
preselection_cuts = ['cs1 < 1000','cs2 < 300000','x**2 + y**2 < 45**2']

dfs = []
for i, source in enumerate(sources):
    #dfs.append(hax.minitrees.load(dsets_list[i].name,
    #                              treemakers=['Corrections','Basics','Extended','Fundam
    #                              preselection = 'cs1<500', num_workers=20))
    dfs.append(hax.minitrees.load(dsets_list[i].name, list_of_treemakers, preselection=p
```

```
cs1 < 1000 selection: 16649682 rows removed (5.92% passed)
cs2 < 300000 selection: 25671 rows removed (97.55% passed)
x**2 + y**2 < 45**2 selection: 354784 rows removed (65.26% passed)
cs1 < 1000 selection: 10379332 rows removed (6.08% passed)
cs2 < 300000 selection: 149598 rows removed (77.72% passed)
x**2 + y**2 < 45**2 selection: 178014 rows removed (65.89% passed)
cs1 < 1000 selection: 3750426 rows removed (5.55% passed)
cs2 < 300000 selection: 23166 rows removed (89.48% passed)
x**2 + y**2 < 45**2 selection: 47035 rows removed (76.13% passed)
```

```
In [7]: # get low energy er/nr cuts
        cut_list = [sciencerun1.LowEnergyRn220(), sciencerun1.LowEnergyAmBe(), sciencerun1.LowEn
```

```
In [8]: excluded_cuts = ['CutS1LowEnergyRange']
        for i in range(len(dfs)):
            print('%s Source Cut Summary:' % sources[i])
            dfs[i] = cut_list[i].process(dfs[i])
            for cut_name in cut_list[i].get_cut_names():
                if cut_name in excluded_cuts:
                    continue
                else:
                    dfs[i] = cuts.selection(dfs[i], dfs[i][cut_name], desc=cut_name)
            dfs[i] = cuts.below(dfs[i], 's1_range_50p_area', 400, desc='s1Width400')
            cuts.history(dfs[i])
            print('\n')
```

```
Rn220 Source Cut Summary:
CutFiducialCylinder1T selection: 582483 rows removed (12.59% passed)
CutS2Threshold selection: 24877 rows removed (70.36% passed)
CutS2AreaFractionTop selection: 1747 rows removed (97.04% passed)
CutS2SingleScatterSimple selection: 4912 rows removed (91.43% passed)
CutDAQVeto selection: 1903 rows removed (96.37% passed)
CutS1SingleScatter selection: 716 rows removed (98.58% passed)
CutS1AreaFractionTop selection: 692 rows removed (98.61% passed)
CutS2PatternLikelihood selection: 35418 rows removed (27.84% passed)
```

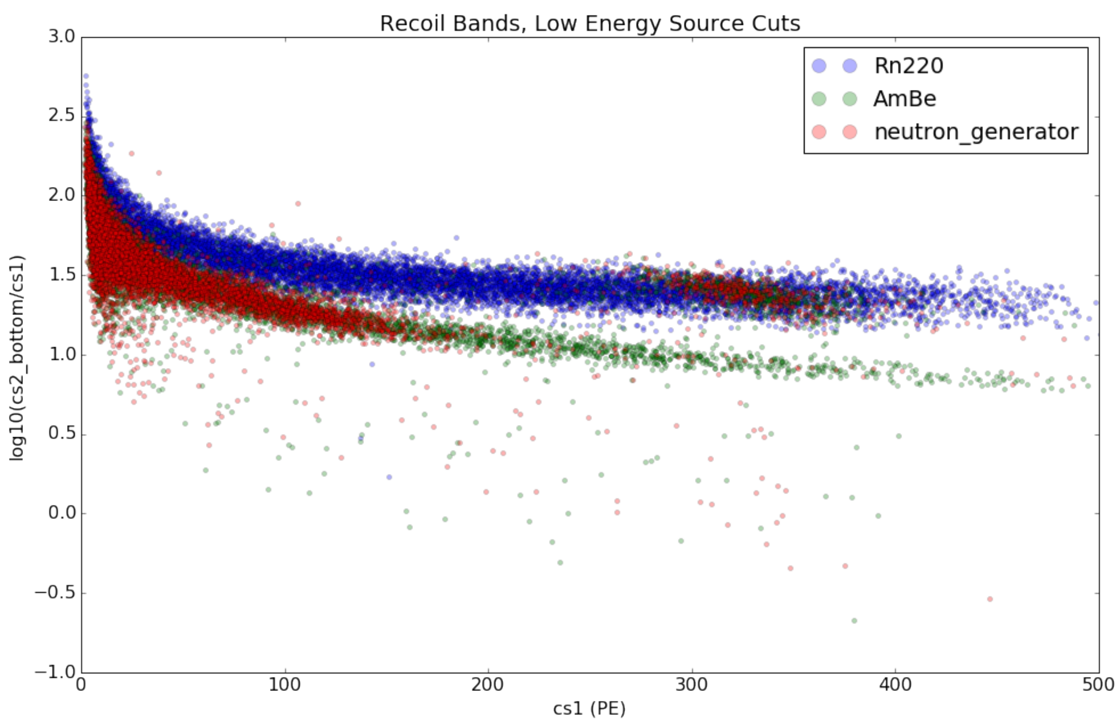
```
CutS2Tails selection: 753 rows removed (94.49% passed)
CutInteractionPeaksBiggest selection: 269 rows removed (97.92% passed)
CutS1PatternLikelihood selection: 2341 rows removed (81.48% passed)
CutS2Width selection: 243 rows removed (97.64% passed)
CutS1MaxPMT selection: 185 rows removed (98.16% passed)
s1Width400 selection: 11 rows removed (99.89% passed)


AmBe Source Cut Summary:
CutAmBeFiducial selection: 298695 rows removed (13.12% passed)
CutS2Threshold selection: 4682 rows removed (89.62% passed)
CutS2AreaFractionTop selection: 332 rows removed (99.18% passed)
CutS2SingleScatterSimple selection: 29230 rows removed (27.13% passed)
CutDAQVeto selection: 105 rows removed (99.03% passed)
CutS1SingleScatter selection: 93 rows removed (99.14% passed)
CutS1AreaFractionTop selection: 103 rows removed (99.04% passed)
CutS2PatternLikelihood selection: 1879 rows removed (82.24% passed)
CutS2Tails selection: 606 rows removed (93.03% passed)
CutInteractionPeaksBiggest selection: 47 rows removed (99.42% passed)
CutS1PatternLikelihood selection: 617 rows removed (92.33% passed)
CutS2Width selection: 549 rows removed (92.61% passed)
CutS1MaxPMT selection: 173 rows removed (97.49% passed)
s1Width400 selection: 12 rows removed (99.82% passed)


neutron_generator Source Cut Summary:
CutNGFiducial selection: 76837 rows removed (48.79% passed)
CutS2Threshold selection: 7254 rows removed (90.09% passed)
CutS2AreaFractionTop selection: 516 rows removed (99.22% passed)
CutS2SingleScatterSimple selection: 51318 rows removed (21.57% passed)
CutDAQVeto selection: 413 rows removed (97.07% passed)
CutS1SingleScatter selection: 269 rows removed (98.04% passed)
CutS1AreaFractionTop selection: 181 rows removed (98.65% passed)
CutS2PatternLikelihood selection: 3226 rows removed (75.66% passed)
CutS2Tails selection: 2049 rows removed (79.57% passed)
CutInteractionPeaksBiggest selection: 97 rows removed (98.78% passed)
CutS1PatternLikelihood selection: 521 rows removed (93.39% passed)
CutS2Width selection: 621 rows removed (91.56% passed)
CutS1MaxPMT selection: 192 rows removed (97.15% passed)
s1Width400 selection: 49 rows removed (99.25% passed)
```

In [9]:
```python
#consts for CES
g2=11.463   # pe/e-
g1=0.1471 # pe/ph
w_value=0.0137
```

```
for i in range(len(dfs)):
    dfs[i]["CES"] = (dfs[i].cs2_bottom/g2 + dfs[i].cs1/g1)*w_value
    dfs[i]["DISC"] = np.log10(dfs[i].cs2_bottom/dfs[i].cs1)
```

1.0.1 Recoil Bands, ER Discrimination, and Leakage

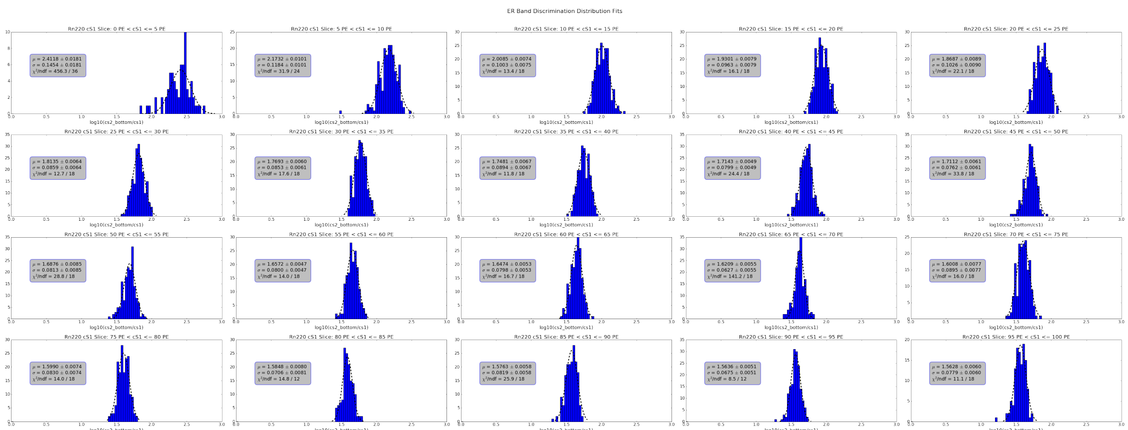
```
In [10]: fig = plt.figure(figsize = (16,10))

         for i in range(len(dfs)):
             plt.plot(dfs[i].cs1, np.log10(dfs[i].cs2_bottom/dfs[i].cs1),'o',label=sources[i],al

         plt.xlim(0,500)
         plt.xlabel('cs1 (PE)')
         plt.ylabel('log10(cs2_bottom/cs1)')
         plt.title('Recoil Bands, Low Energy Source Cuts')
         plt.legend(markerscale=3)

         plt.show()
```




```
In [11]: # Develop ER Discrimination Line

         df_rn = dfs[0]
```

```
cs1_lim, cs1_width = 100, 5
cs1_bins = int(cs1_lim/cs1_width)
hist_range, hist_bins = [0,3], 100
hist_width = (hist_range[1]-hist_range[0])/hist_bins

fig = plt.figure(figsize = (12*5,6*4))
er_disc = []

for i in range(cs1_bins):

    # slice data in cs1
    cs1_min, cs1_max = i*cs1_width, (i+1)*cs1_width
    df = cuts.selection(df_rn, (df_rn['cs1'] > cs1_min) & (df_rn['cs1'] <= cs1_max), de


    # fill and fit discrimination hist

    x = df.DISC
    mean = np.mean(x)
    i_mean = int((mean/hist_width)+0.5)
    std = np.std(x)
    i_std = int((std/hist_width)+0.5)
    i_min, i_max = max(i_mean-3*i_std,0), min(i_mean+3*i_std,hist_bins)

    plt.subplot(4,5,i+1)
    ax = plt.gca()

    n, bins, patches = plt.hist(x, bins=hist_bins, range=hist_range)
    bin_centers = bins[:-1] + 0.5 * (bins[1:] - bins[:-1])
    x_fit = bin_centers[i_min:i_max]
    y_fit = n[i_min:i_max]
    guess = (mean, std, max(y_fit))
    popt, pcov = curve_fit(gaussian, x_fit, y_fit, p0=guess)
    fit = gaussian(x_fit, *popt)
    perr = np.sqrt(np.diag(pcov))
    chi2, ndf = chisquare_ndf(y_fit,fit)

    s = ('$\mu$ = %.4f $\pm$ %.4f  \n$\sigma$ = %.4f $\pm$ %.4f \n$\chi ^2$/ndf = %.1f
         % (popt[0], perr[0], popt[1], perr[1], chi2, ndf))
    bbox_props = dict(boxstyle="Round,pad=0.5", fc="gray", ec="blue", lw=2, alpha = 0.5
    plt.text(0.1, 0.5, s,  transform=ax.transAxes, bbox=bbox_props, size=18)
    x_plot = np.linspace(mean-3*std,mean+3*std,50)
    y_plot = gaussian(x_plot, *popt)
    plt.plot(x_plot, y_plot, c='k', linewidth=3, linestyle='dashed')
    plt.xlabel('log10(cs2_bottom/cs1)', fontsize=18)
    plt.title('Rn220 cS1 Slice: %i PE < cS1 <= %i PE' % (cs1_min,cs1_max), fontsize=20)

    er_disc.append([np.mean([cs1_min,cs1_max]),popt[0],popt[1]])
```

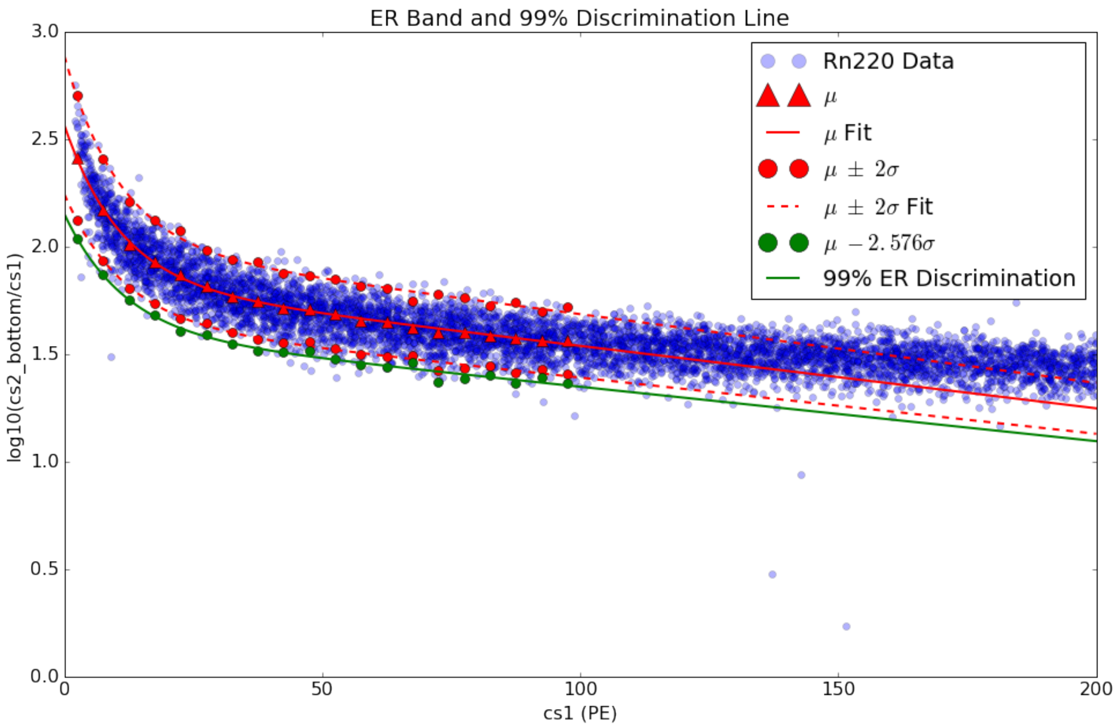
```
er_disc = np.array(er_disc)

plt.suptitle('ER Band Discrimination Distribution Fits', fontsize=22)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



```
In [12]: # Discrimination line
         def disc_exp(cs1, p0, p1, p2, p3):
             return np.exp(p0+p1*cs1) + p2 + p3*cs1

         # popt for SR0 99%
         sr0_99_popt = (-0.645351,-0.0544212,1.52071,-0.00108844)

         # Discrimination parameters for specific exclusion
         def disc_popt(er_disc, sigmas):
             x = er_disc[:,0]
             y = er_disc[:,1] + sigmas*er_disc[:,2]

             # guess from SR0 99%
             guess = sr0_99_popt

             popt, pcov = curve_fit(disc_exp, x, y, p0=guess)
             return popt
```

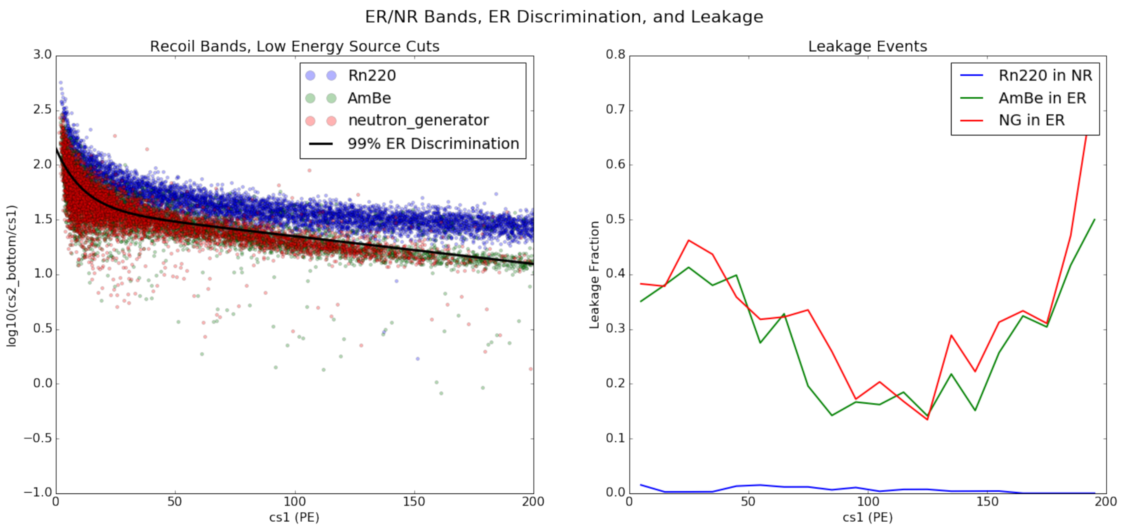
```
In [13]: fig = plt.figure(figsize = (16,10))


         plt.plot(df_rn.cs1, df_rn.DISC,'bo',label='Rn220 Data',alpha=0.3,markersize=6)


         x = np.linspace(0,200,400)
```

```
plt.plot(er_disc[:,0], er_disc[:,1],'r^',label='$\mu$',markersize=10)
mean_popt = disc_popt(er_disc, 0)
mean_fit = disc_exp(x,*mean_popt)
plt.plot(x,mean_fit,'r-',lw=2,label='$\mu$ Fit')

plt.plot(er_disc[:,0], er_disc[:,1]+2*er_disc[:,2],'ro',label='$\mu\ \pm\ 2\sigma$',mar
p2s_popt = disc_popt(er_disc, 2)
p2s_fit = disc_exp(x,*p2s_popt)
plt.plot(x,p2s_fit,'r-',lw=2,linestyle='dashed',label='$\mu\ \pm\ 2\sigma$ Fit')

plt.plot(er_disc[:,0], er_disc[:,1]-2*er_disc[:,2],'ro',markersize=8)
n2s_popt = disc_popt(er_disc, -2)
n2s_fit = disc_exp(x,*n2s_popt)
plt.plot(x,n2s_fit,'r-',lw=2,linestyle='dashed')

plt.plot(er_disc[:,0], er_disc[:,1]-2.576*er_disc[:,2],'go',markersize=8,label='$\mu\ -
disc_99_popt = disc_popt(er_disc, -2.576)
disc_99_fit = disc_exp(x,*disc_99_popt)
plt.plot(x,disc_99_fit,'g-',lw=2,label='99% ER Discrimination')

plt.xlim(0,200)
plt.xlabel('cs1 (PE)')
plt.ylabel('log10(cs2_bottom/cs1)')
plt.title('ER Band and 99% Discrimination Line')
plt.legend(markerscale=2)

plt.show()
```



In [14]:
```
fig = plt.figure(figsize = (24,10))
plt.suptitle('ER/NR Bands, ER Discrimination, and Leakage', fontsize=22)

plt.subplot(121)
for i in range(len(dfs)):
    plt.plot(dfs[i].cs1, np.log10(dfs[i].cs2_bottom/dfs[i].cs1),'o',label=sources[i],al

plt.plot(x,disc_99_fit,'k-',lw=3,label='99% ER Discrimination')

plt.xlim(0,200)
plt.xlabel('cs1 (PE)')
plt.ylabel('log10(cs2_bottom/cs1)')
plt.title('Recoil Bands, Low Energy Source Cuts')
plt.legend(markerscale=3)

plt.subplot(122)

# Calculate leakage ratios
cs1_lim, cs1_width = 200, 10
cs1_bins = int(cs1_lim/cs1_width)
leakage_ratios = []
for i in range(cs1_bins):

    # slice data in cs1
```

```
    cs1_min, cs1_max = i*cs1_width, (i+1)*cs1_width
    ratios = []
    for i in range(len(sources)):
        df_slice = cuts.selection(dfs[i], (dfs[i]['cs1'] > cs1_min) & (dfs[i]['cs1'] <=
        if i==0:
            df_slice_disc = cuts.selection(df_slice,df_slice['DISC'] < disc_exp(df_slic
        else:
            df_slice_disc = cuts.selection(df_slice,df_slice['DISC'] > disc_exp(df_slic
        ratios.append(len(df_slice_disc)/len(df_slice))
    leakage_ratios.append([np.mean([cs1_min,cs1_max]),ratios[0],ratios[1],ratios[2]])
leakage_ratios = np.array(leakage_ratios)

plt.plot(leakage_ratios[:,0],leakage_ratios[:,1],'-',lw=2,label='Rn220 in NR')
plt.plot(leakage_ratios[:,0],leakage_ratios[:,2],'-',lw=2,label='AmBe in ER')
plt.plot(leakage_ratios[:,0],leakage_ratios[:,3],'-',lw=2,label='NG in ER')

plt.xlabel('cs1 (PE)')
plt.ylabel('Leakage Fraction')
plt.legend(markerscale=3)
plt.title('Leakage Events')

plt.show()
```



1.0.2 How Else Can We Discriminate ER/NR? ML!

```
In [15]: # ML imports
         from matplotlib.colors import ListedColormap
         from pandas.tools.plotting import scatter_matrix
         from sklearn import model_selection
         from sklearn.metrics import classification_report
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

In [16]:
```
# prepare data
dfs_lowE = []
for i in range(len(dfs)):
    dfs_lowE.append(cuts.below(dfs[i],'cs1',200,quiet=True))
    if sources[i] == 'Rn220':
        dfs_lowE[i]['recoil'] = 'er'
    else:
        dfs_lowE[i]['recoil'] = 'nr'

df_ml = pd.concat(dfs_lowE)
df_ml_min = df_ml[['cs1','DISC','recoil']]

# Count recoil events
er_tot = len(df_ml_min[df_ml_min.recoil=='er'])
print('%i total ER events (Rn220)' % er_tot)
nr_tot = len(df_ml_min[df_ml_min.recoil=='nr'])
print('%i total NR events (AmBe and NG)' % nr_tot)
```

```
6030 total ER events (Rn220)
11243 total NR events (AmBe and NG)
```

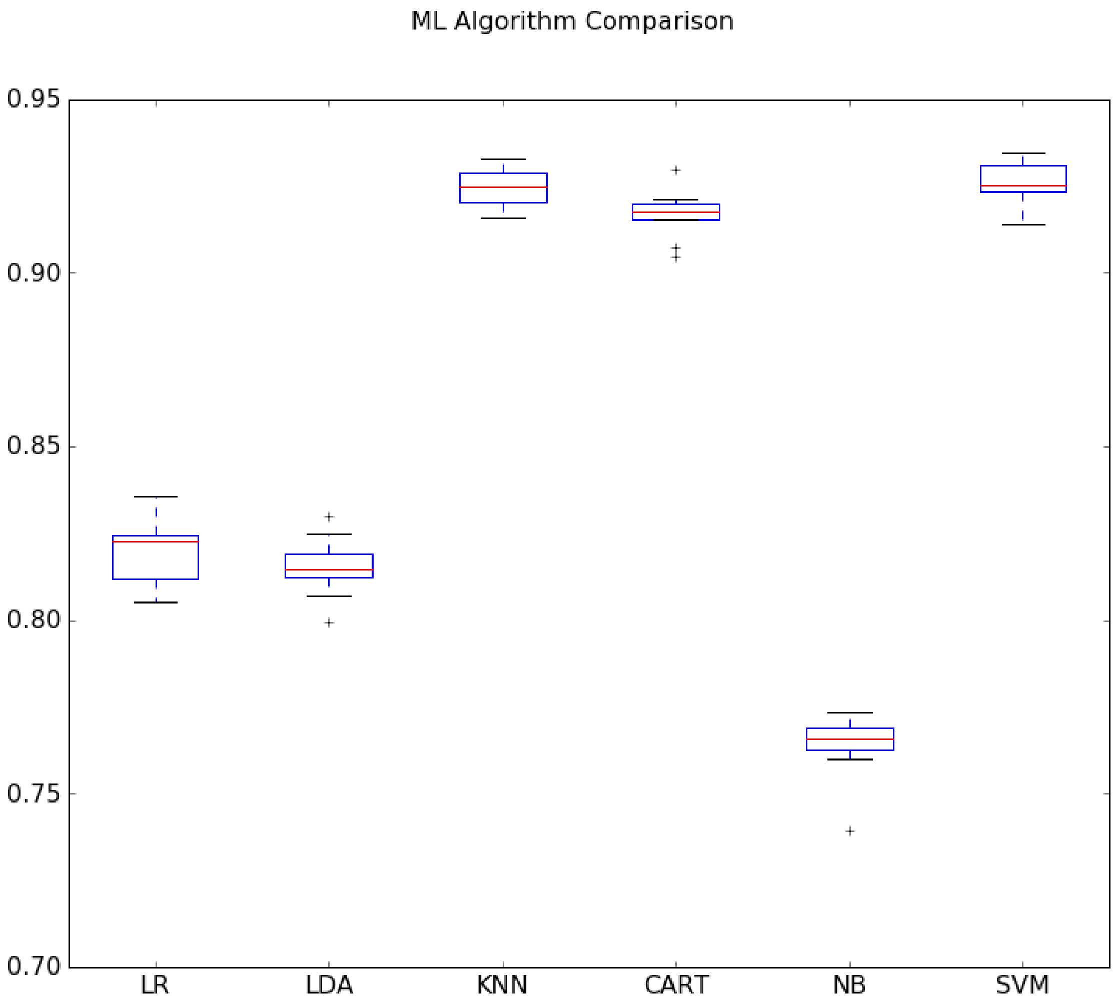
In [17]:
```
# Split-out validation dataset
array = df_ml_min.values
X = array[:,0:len(df_ml_min.keys())-1]
Y = array[:,len(df_ml_min.keys())-1]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, t
```

In [18]:
```
# Spot Check Algorithms
scoring = 'accuracy'

models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
```

```
# evaluate each model in turn
print('ML Algorithm Accuracy Summary; Mean (Std) \n')
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, sco
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

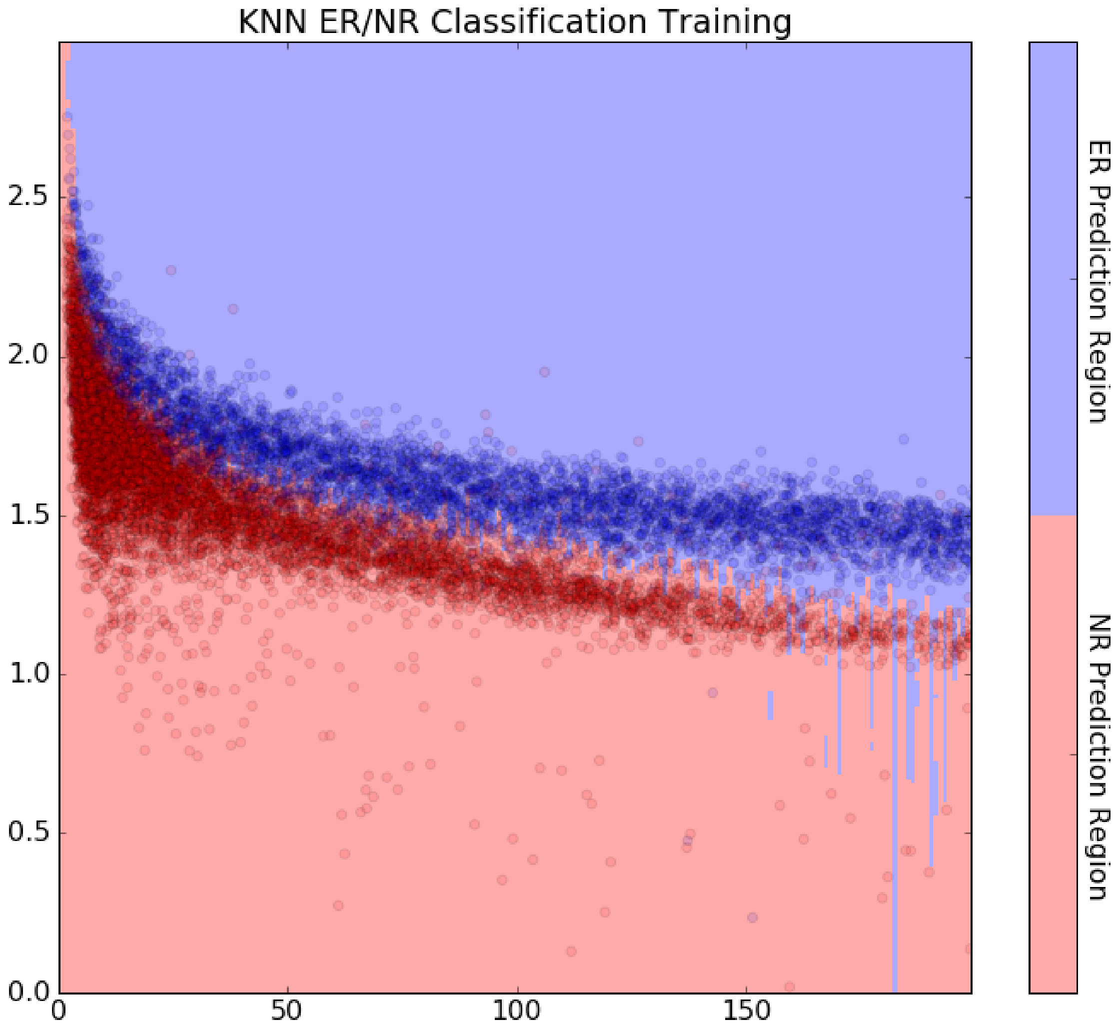
```
ML Algorithm Accuracy Summary; Mean (Std)

LR: 0.819655 (0.008570)
LDA: 0.815312 (0.008217)
KNN: 0.924809 (0.005524)
CART: 0.916775 (0.006711)
NB: 0.763785 (0.009051)
SVM: 0.925749 (0.006283)
```

In [19]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('ML Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



In [20]:
```
n_neighbors=15

cmap_light = ListedColormap(['#AAAAFF', '#FFAAAA'])
cmap_bold = ListedColormap(['#0000FF', '#FF0000'])

knn = KNeighborsClassifier(n_neighbors, weights='distance')
knn.fit(X_train,Y_train)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X_train[:, 0].min(), X_train[:, 0].max()
y_min, y_max = X_train[:, 1].min(), X_train[:, 1].max()
xx, yy = np.meshgrid(np.arange(0, 200, 1),
                     np.arange(0, 3, 0.01))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
lookupTable, Z = np.unique(np.array(Z),return_inverse=True)
```

```
Z = Z.reshape(xx.shape)

fig, ax = plt.subplots()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
cbar = plt.colorbar()
cbar.set_ticks([0.25,0.75])
cbar.set_ticklabels(['ER Prediction Region','NR Prediction Region'])
cbar.ax.set_yticklabels(cbar.ax.get_yticklabels(),rotation=-90)
cbar.ax.invert_yaxis()

# Plot also the training points
lookupTable, Y_colors = np.unique(np.array(Y_train),return_inverse=True)
ax.scatter(X_train[:, 0], X_train[:, 1], color=Y_colors, cmap=cmap_bold,
            edgecolor='k', s=30, alpha=0.1)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("KNN ER/NR Classification Training")

plt.show()
```

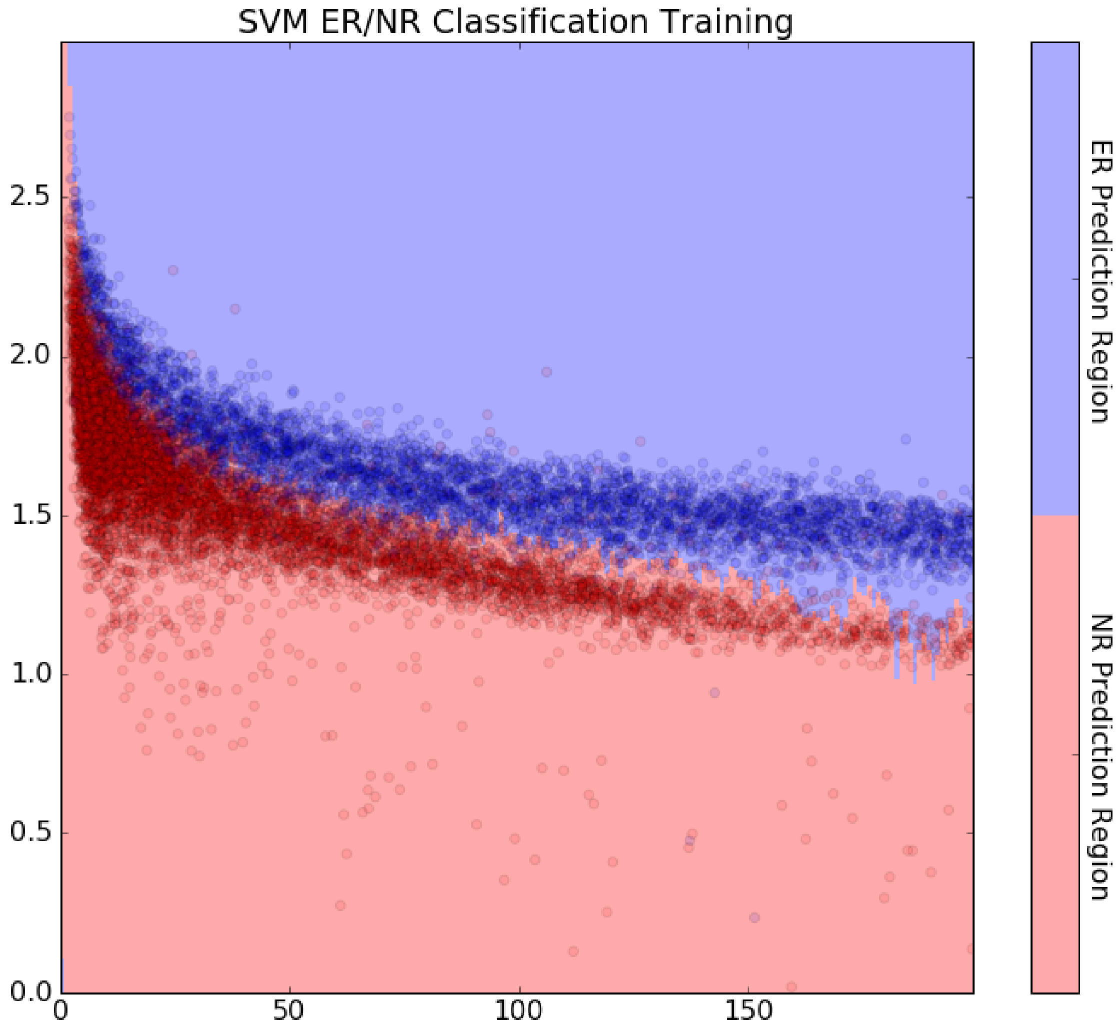



```
In [21]: print( list( df.keys() ) )
```

```
['run_number', 'event_number', 's2_area_tailcut_set_by', 's2_over_tdiff', 'tailcut_set_by', 'nea
```

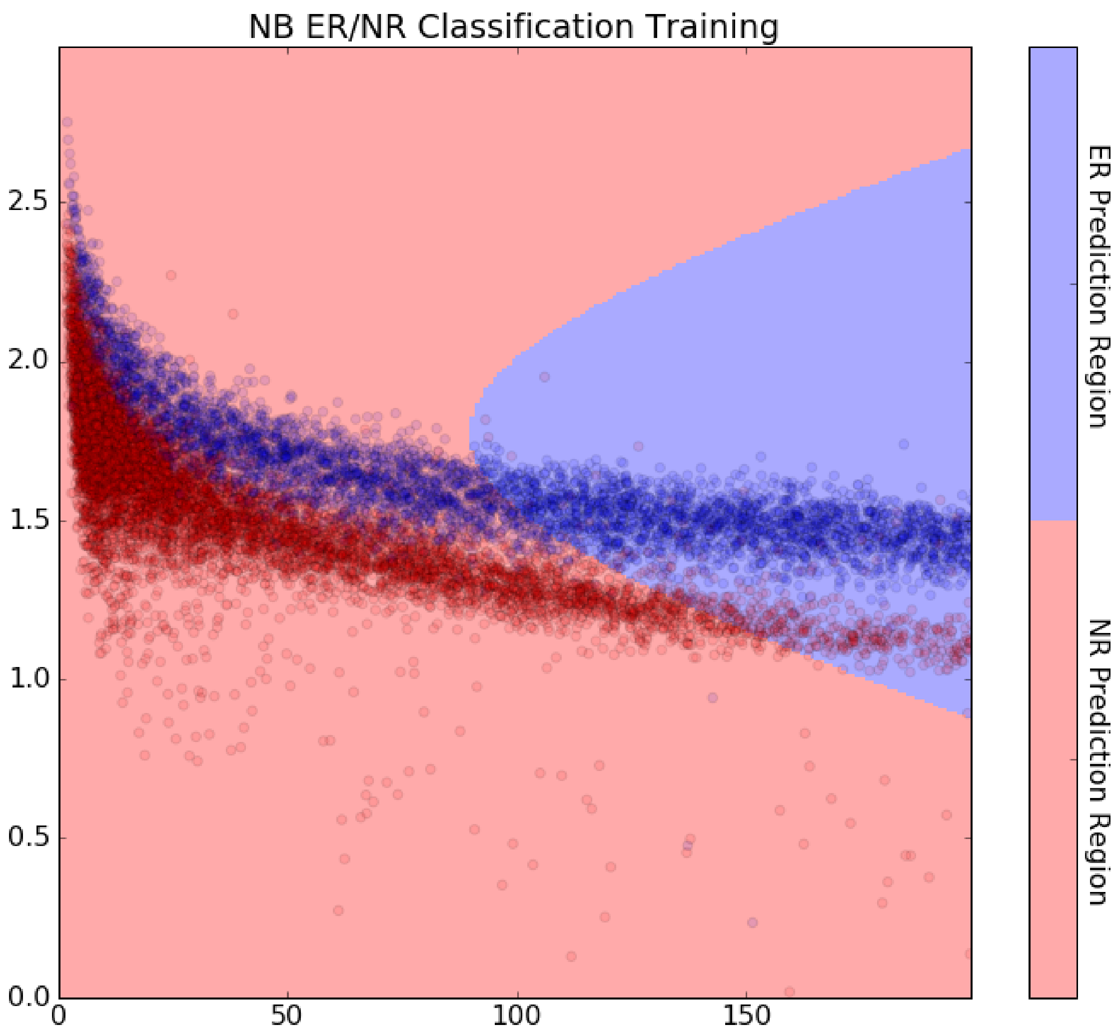
```
In [30]: from sklearn import svm

         cmap_light = ListedColormap(['#AAAAFF', '#FFAAAA'])
         cmap_bold = ListedColormap(['#0000FF', '#FF0000'])

         svm_clf= svm.SVC()

         SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)
```

```python
svm_clf.fit(X_train,Y_train)


# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X_train[:, 0].min(), X_train[:, 0].max()
y_min, y_max = X_train[:, 1].min(), X_train[:, 1].max()
xx, yy = np.meshgrid(np.arange(0, 200, 1),
                     np.arange(0, 3, 0.01))
Z = svm_clf.predict(np.c_[xx.ravel(), yy.ravel()])
lookupTable, Z = np.unique(np.array(Z),return_inverse=True)
Z = Z.reshape(xx.shape)

fig, ax = plt.subplots()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
cbar = plt.colorbar()
cbar.set_ticks([0.25,0.75])
cbar.set_ticklabels(['ER Prediction Region','NR Prediction Region'])
cbar.ax.set_yticklabels(cbar.ax.get_yticklabels(),rotation=-90)
cbar.ax.invert_yaxis()

# Plot also the training points
lookupTable, Y_colors = np.unique(np.array(Y_train),return_inverse=True)
ax.scatter(X_train[:, 0], X_train[:, 1], color=Y_colors, cmap=cmap_bold,
            edgecolor='k', s=30, alpha=0.1)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("SVM ER/NR Classification Training")

plt.show()
```



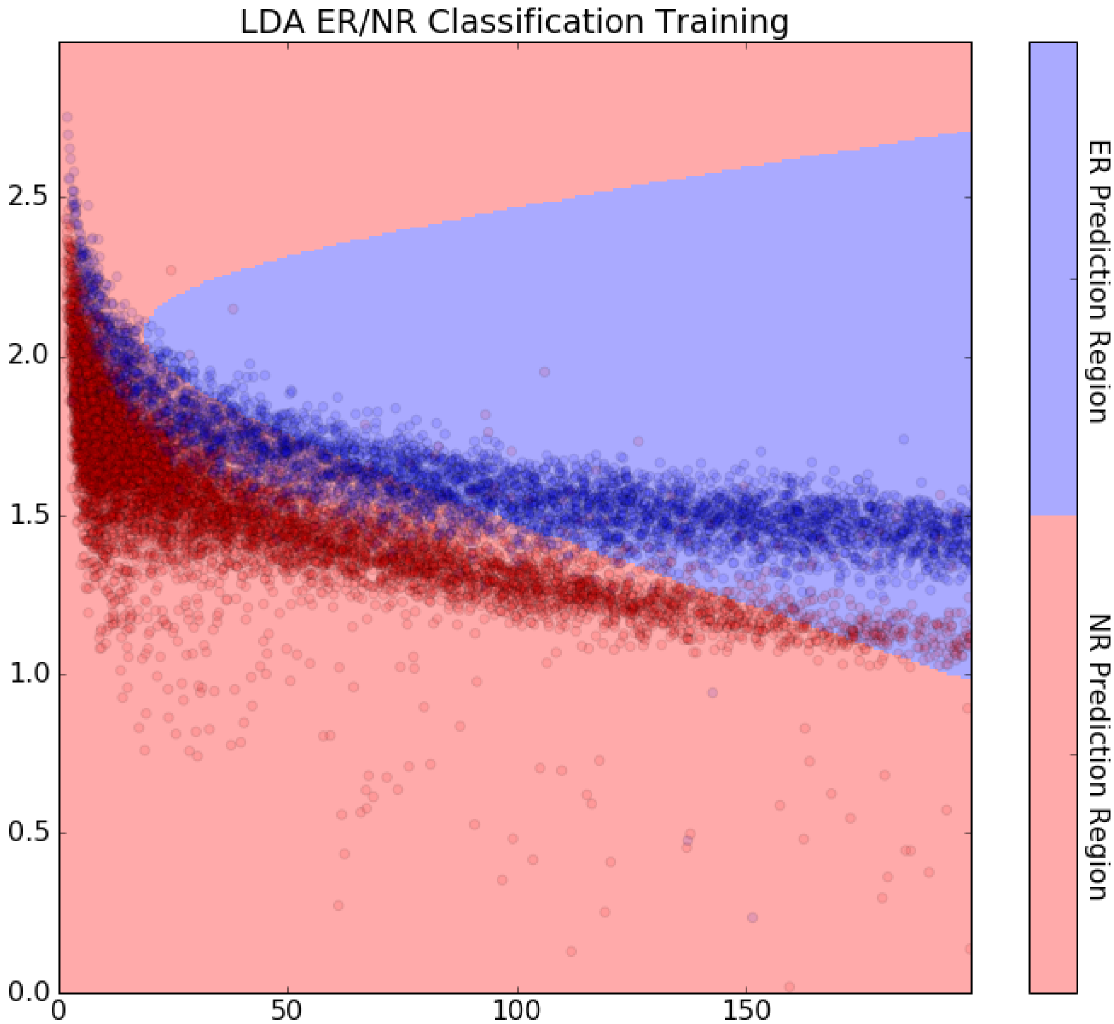

```
In [36]: from sklearn.naive_bayes import GaussianNB

         nb_clf = GaussianNB()

         nb_clf.fit(X_train,Y_train)
         GaussianNB(priors=None)

         # Plot the decision boundary. For that, we will assign a color to each
         # point in the mesh [x_min, x_max]x[y_min, y_max].
         x_min, x_max = X_train[:, 0].min(), X_train[:, 0].max()
         y_min, y_max = X_train[:, 1].min(), X_train[:, 1].max()
         xx, yy = np.meshgrid(np.arange(0, 200, 1),
                              np.arange(0, 3, 0.01))
         Z = nb_clf.predict(np.c_[xx.ravel(), yy.ravel()])
         lookupTable, Z = np.unique(np.array(Z),return_inverse=True)
```

```python
Z = Z.reshape(xx.shape)

fig, ax = plt.subplots()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
cbar = plt.colorbar()
cbar.set_ticks([0.25,0.75])
cbar.set_ticklabels(['ER Prediction Region','NR Prediction Region'])
cbar.ax.set_yticklabels(cbar.ax.get_yticklabels(),rotation=-90)
cbar.ax.invert_yaxis()

# Plot also the training points
lookupTable, Y_colors = np.unique(np.array(Y_train),return_inverse=True)
ax.scatter(X_train[:, 0], X_train[:, 1], color=Y_colors, cmap=cmap_bold,
            edgecolor='k', s=30, alpha=0.1)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("NB ER/NR Classification Training")

plt.show()
```




```
In [42]: from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
         import numpy as np

         lda_clf = QuadraticDiscriminantAnalysis()
         lda_clf.fit(X_train,Y_train)

         LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                       solver='svd', store_covariance=False, tol=0.0001)



         # Plot the decision boundary. For that, we will assign a color to each
         # point in the mesh [x_min, x_max]x[y_min, y_max].
         x_min, x_max = X_train[:, 0].min(), X_train[:, 0].max()
         y_min, y_max = X_train[:, 1].min(), X_train[:, 1].max()
```

```python
xx, yy = np.meshgrid(np.arange(0, 200, 1),
                     np.arange(0, 3, 0.01))
Z = lda_clf.predict(np.c_[xx.ravel(), yy.ravel()])
lookupTable, Z = np.unique(np.array(Z),return_inverse=True)
Z = Z.reshape(xx.shape)

fig, ax = plt.subplots()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
cbar = plt.colorbar()
cbar.set_ticks([0.25,0.75])
cbar.set_ticklabels(['ER Prediction Region','NR Prediction Region'])
cbar.ax.set_yticklabels(cbar.ax.get_yticklabels(),rotation=-90)
cbar.ax.invert_yaxis()

# Plot also the training points
lookupTable, Y_colors = np.unique(np.array(Y_train),return_inverse=True)
ax.scatter(X_train[:, 0], X_train[:, 1], color=Y_colors, cmap=cmap_bold,
            edgecolor='k', s=30, alpha=0.1)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("LDA ER/NR Classification Training")

plt.show()
```


LDA ER/NR Classification Training
2.5
2.0
1.5
1.0
0.5
0.0
0
50
100
150
ER Prediction Region
NR Prediction Region

```
In [ ]:
```