9.2

In [36]:
```
from sklearn.datasets import fetch_openml
x,y= fetch_openml('mnist_784', version=1,return_X_y=True)
```

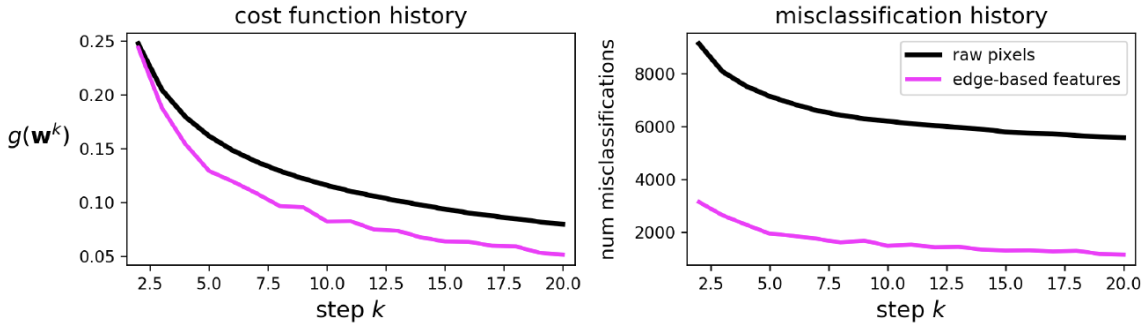
In [37]:
```
x = x.T
y = np.array([int(v) for v in y])[np.newaxis,:]
```

In [18]:
```
num_sample = 50000
inds = np.random.permutation(y.shape[1])[:num_sample]
x_sample = x.iloc[:,inds].values
y_sample = y[:,inds]
```

In [19]:
```
def standard_normalizer(x):
    x_means = np.nanmean(x,axis = 1)[:,np.newaxis]
    x_stds = np.nanstd(x,axis = 1)[:,np.newaxis]
    ind = np.argwhere(x_stds < 10**(-2))
    if len(ind) > 0:
        ind = [v[0] for v in ind]
        adjust = np.zeros((x_stds.shape))
        adjust[ind] = 1.0
        x_stds += adjust
    ind = np.argwhere(np.isnan(x) == True)
    for i in ind:
        x[i[0],i[1]] = x_means[i[0]]
    normalizer = lambda data: (data - x_means)/x_stds
    inverse_normalizer = lambda data: data*x_stds + x_means
    return normalizer,inverse_normalizer
```

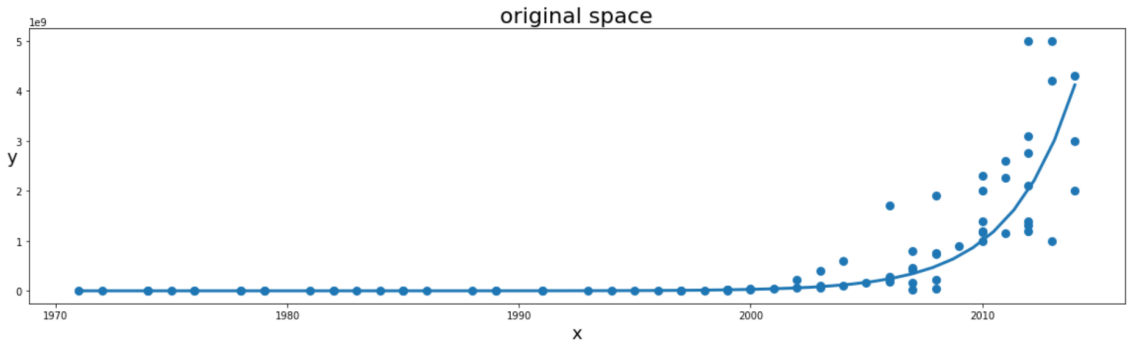
In [21]:
```
normalizer,inverse_normalizer = standard_normalizer(x_sample.T)
x_sample = normalizer(x_sample.T).T
```

In [25]:
```
from otherSources import data_transformer
x_sample_edgebased_features = data_transformer.edge_transformer(x_sample)
print('shape of original input ', x_sample.shape)
print('shape of transformed input ', x_sample_edgebased_features.shape)
```



10.4

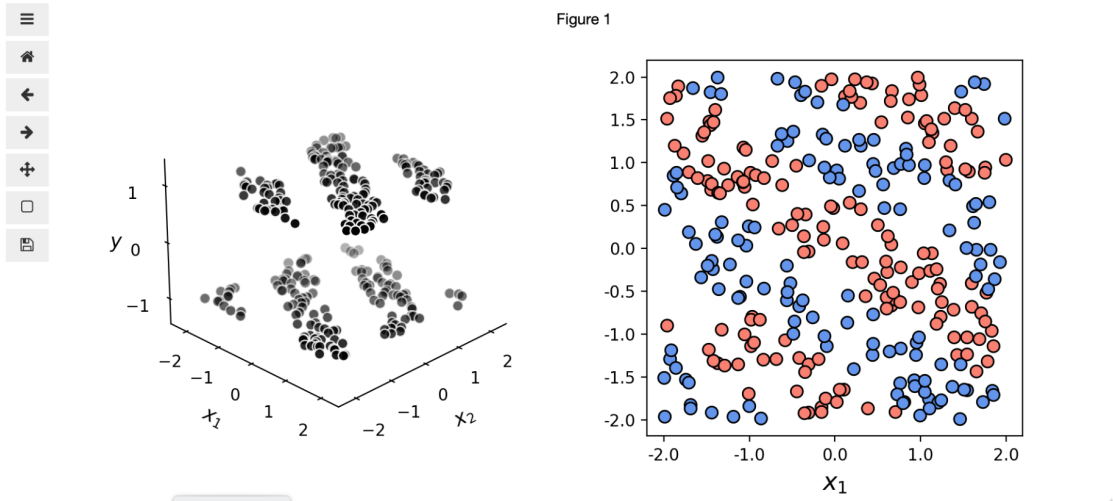
In [71]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

csvname = '../machine_learning_refined/machine_learning_refined/mlrefined_exercises/ed_2/mlrefined_datasets/nonlinea
data = np.asarray(pd.read_csv(csvname,header = None))
x = data[:,0]
x.shape = (len(x),1)
y = data[:,1]
y.shape = (len(y),1)
y_logged = np.log(y)
o = np.ones((len(x),1))
x_new = np.concatenate((o,x),axis = 1)
A = 0
b = 0
for i in range(len(x)):
    A += np.outer(x_new[i,:],x_new[i,:].T)
    b += y_logged[i]*x_new[i,:].T
w = np.linalg.solve(A,b)
fig = plt.figure(figsize = (16,5))
ax1 = fig.add_subplot(1,1,1) # panel for original space
ax1.scatter(x,y,linewidth = 3)
s = np.linspace(np.min(x),np.max(x))
t = np.exp(w[0] + w[1]*s)
ax1.plot(s,t,linewidth = 3)
ax1.set_xlabel('x',fontsize =18)
ax1.set_ylabel('y',rotation = 0,fontsize = 18)
ax1.set_title('original space',fontsize = 22);
```



10.8

In [25]:
```
demo = nonlib.nonlinear_classification_visualizer.Visualizer('../mlrefined/mlrefined_datasets/nonlinear_superlearn_d
x = demo.x.T
y = demo.y[np.newaxis,:]
demo.plot_data();
```

Figure 1

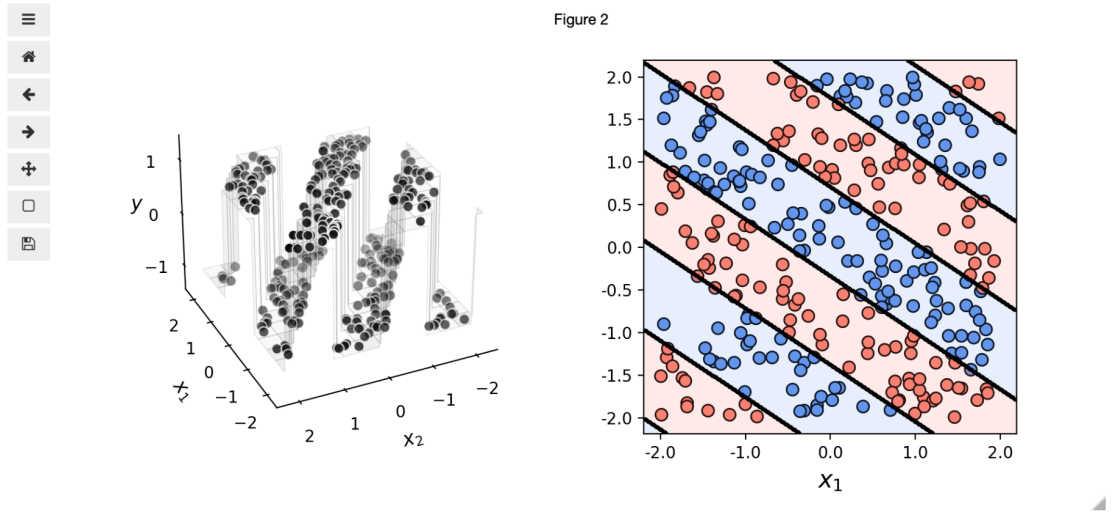


In [26]:
```
def feature_transforms(x,w):
    f = np.sin(w[0] + np.dot((x).T,w[1:])).T
    return f
```

In [27]:
```
scale = 2
w = [scale*np.random.randn(3,1),scale*np.random.randn(2,1)]
maxx = 1000
a = 10**(-1)
run = nonlib.basic_runner.Setup(x,y,feature_transforms,'softmax',normalize = 'standard')
run.fit(w=w,alpha_choice = a,max_its = maxx)
ind = np.argmin(run.cost_history)
wh = run.weight_history[ind]
demo.static_N2_simple(wh,run,view = [30,155])
```

Figure 2

Figure 2



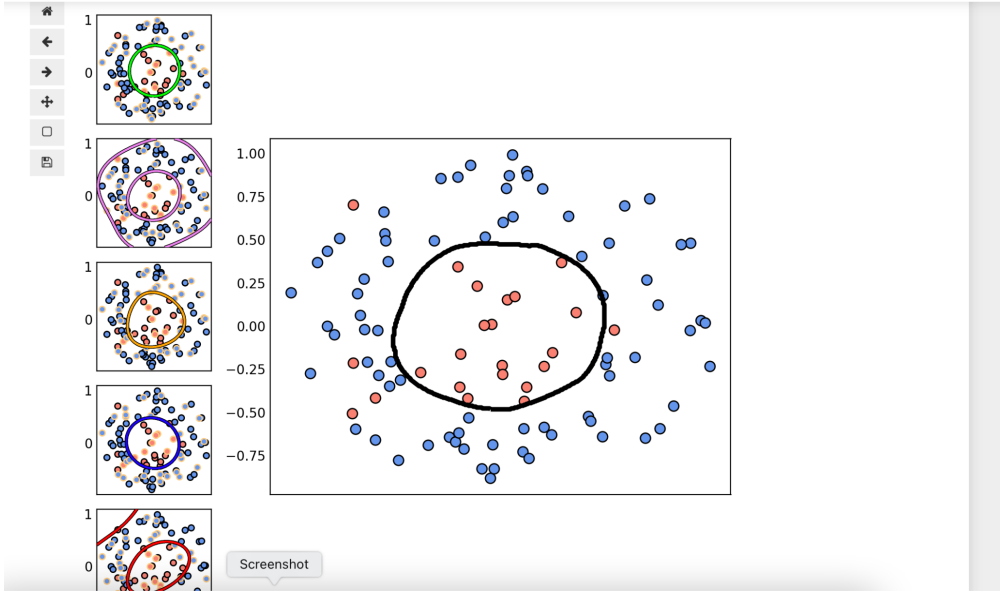
11.7

In [31]:
```
import sys
sys.path.append('../')
datapath = '../mlrefined/mlrefined_datasets/nonlinear_superlearn_datasets/'
import autograd.numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import math
import copy
from mlrefined_libraries import nonlinear_superlearn_library as nonlib
from mlrefined_libraries import math_optimization_library as optlib
regress_plotter = nonlib.nonlinear_regression_demos_multiple_panels
classif_plotter = nonlib.nonlinear_classification_visualizer_multiple_panels
%matplotlib notebook
from matplotlib import rcParams
rcParams['figure.autolayout'] = True
```

```
Warning: Cannot change to a different GUI toolkit: notebook. Using widget instead.
```

In [32]:
```
csvname = datapath + 'new_circle_data.csv'
data = np.loadtxt(csvname,delimiter = ',')
x = data[:-1,:]
y = data[-1:,:]
degree = 8
num_bags = 5
train = 0.66
best = []
runs = []
```

```
for j in range(num_bags):
    lib = nonlib.reg_lib.super_setup.Setup(x,y)
    lib.preprocessing_steps(normalizer = 'none')
    lib.make_train_val_split(train_portion = train)
    for d in range(1,degree+1):
        lib.choose_cost(name = 'softmax')
        lib.choose_features(name = 'polys',degree = d)
        lib.fit(algo = 'newtons_method',max_its = 10,verbose = False,lam = 10**(-8))
    costs = [np.min(lib.valid_count_histories[i]) for i in range(degree)]
    mini = np.argmin(costs)
    minv = val_costs[mini]
    smallest_ind = np.argmin(lib.valid_count_histories[mini])
    lib.train_cost_histories = lib.train_cost_histories[mini][smallest_ind]
    lib.valid_cost_histories = lib.valid_cost_histories[mini][smallest_ind]
    lib.train_count_histories = lib.train_count_histories[mini][smallest_ind]
    lib.valid_count_histories = lib.valid_count_histories[mini][smallest_ind]
    lib.weight_histories = lib.weight_histories[mini][smallest_ind]
    lib.choose_features(name = 'polys',degree = mini + 1)
    best.append(copy.deepcopy(lib))

demo = nonlib.classification_bagging_visualizers_v2.Visualizer(csvname)
demo.show_runs(best)
```



11.10

In [87]:
```
def assign_to_folds(L,K):
# split data into k equal (as possible) sized sets
    order = np.random.permutation(L)
    c = np.ones((L,1))
    L = int(np.round((1/K)*L))
    for s in np.arange(0,K-2):
        c[order[s*L:(s+1)*L]] = s + 20
    c[order[(K-1)*L:]] = K
    return c


csvname = datapath + 'new_gene_data.csv'
data = np.loadtxt(csvname,delimiter = ',')
x = data[:-1,:]
y = data[-1:,:]
# assign data to K folds
K = 10
num_pts = y.size
fold_nums = assign_to_folds(num_pts,K)
lams = np.linspace(0,20,100)
# loop over each fold and complete calculations
all_train_counts = []
all_valid_counts = []
for k in range(K):
    lib = nonlib.kfolds_reg_lib.superlearn_setup.Setup(x,y)
    lib.choose_normalizer(name = 'standard')
    train_inds = np.argwhere(fold_nums != k)
    train_inds = [v[0] for v in train_inds]
    valid_inds = np.argwhere(fold_nums == k)
    valid_inds = [v[0] for v in valid_inds]
```

```
    lib.train_inds = train_inds
    lib.x_train = lib.x[:,train_inds]
    lib.y_train = lib.y[:,train_inds]
    lib.valid_inds = valid_inds
    lib.x_valid = lib.x[:,valid_inds]
    lib.y_valid = lib.y[:,valid_inds]
    # choose cost
    lib.choose_cost(cost_name = 'softmax',reg_name = 'L1')
    # choose optimizer
    lib.choose_optimizer('gradient_descent',max_its=100,alpha_choice='diminishing')
    # run regularization
    lib.tryout_lams(lams)
    # record counts
    all_train_counts.append(copy.deepcopy(mylib.train_count_vals))
```