HW 5

- Robert 'Quinn' Hull
- I acknowledge that this exam is soltely my effort. I have done this work by myself. I have noted when and how I have used resources to help me arrive at my conclusions

In [1]:
```
## Modules
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_gaussian_quantiles
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB

from sklearn.svm.libsvm import predict_proba
```

```
/Users/roberthull/opt/miniconda3/envs/Res1/lib/python3.8/site-packages/sklearn/u
tils/deprecation.py:143: FutureWarning: The sklearn.svm.libsvm module is  deprec
ated in version 0.22 and will be removed in version 0.24. The corresponding clas
ses / functions should instead be imported from sklearn.svm. Anything that canno
t be imported from sklearn.svm is now part of the private API.
  warnings.warn(message, FutureWarning)
```

Semi - Supervised Learning

- Implement a self-training algorithm
- Resources:
  - I used this article to double check my workflow, but didn't end up using any of the code from it. I think the main functionality I learned was the sklearn predict_proba function, super helpful https://towardsdatascience.com/a-gentle-introduction-to-self-training-and-semi-supervised-learning-ceee73178b38

Self Training

Self training uses unlabeld data through an iterative process to "try" to generalize better.

Algorithm

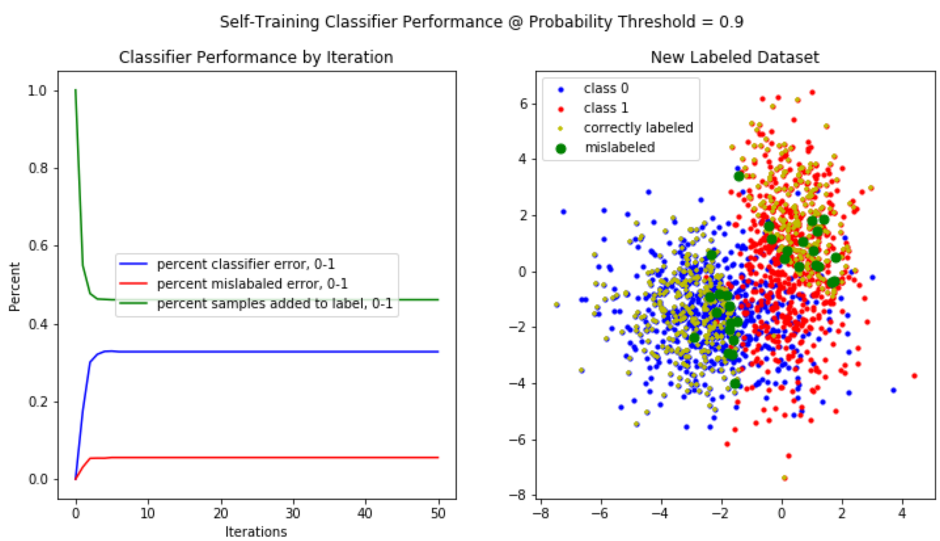
1. Train f on (X_ℓ, Y_ℓ)
2. Make predictions on X_u with $f(x)$. $x \in X_u$
3. Choose the samples in X_u w/ high confidence and add them into the labeled data set X_ℓ
   $(x, f(x))$
   - make it a hard label
   - use the posterior
4. Repeat

1. An experiment on Synthetic Data

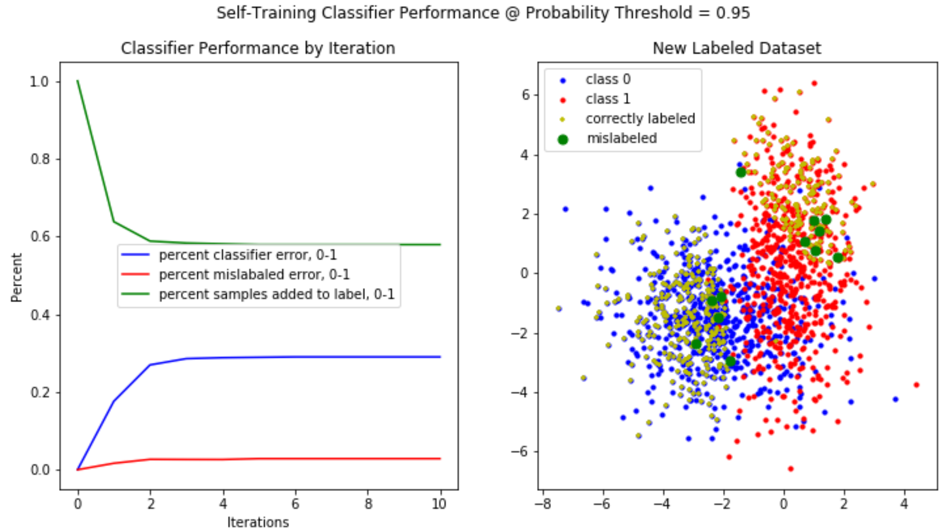
- Generate 2D Gaussian dataset
- Training Set: 1000 -> (500 samples from each class)
- Testing Set: 1000 -> (500 samples form each class)
- Requirements:
  - Self-Training Must Use a Classifier that can you probabilities to select the data points that will have pseudo labels
  - Choose a suitable threshold to determine the data samples that will be labeled for the next round of self-training
  - Report the error of the self-training algorithm on the testing data at:
    1. The first time a classifier is training using only labled data
    2. At least one time point during the self-training process
    3. After self-training is complete
    - Comment on the results
  - Peform an experiment reporting the above requirements with 10% and then 25% of the training data are labeled

Experiment Writeup

- My model uses a quadratic discriminant classifier (from the first homework).
- Some orienting notes on the figures below. The left figure shows the several percentages measuring the performance of the model at each iteration
    - Percent Classifier Error (blue line) shows how well the quadratic discriminant classifier is performing on the unlabeled (in this case, also 'test') data. Higher values mean the classifier is mis-identifying more readily in what's left of the unlabeled dataset.
    - Percent Mislabeled Error (red line) shows the percent of the data moved from unlabeled to labeled sets that have been mislabeled. This is a cumulative function (once mislabeled, always mislabeled). Higher percentages mean
    - Percent Samples Added to Label (green line) shows the percent of the unlabeled dataset that has been moved into the labeled dataset. This is a cumulative function (once moved, always moved).
- After generating two gaussian datasets with overlapping distributions (see below), I started testing the model.
    - First, I began by setting the probability threshold (required to decide whether or not to add a data point to the labeled set) at 0.9 and iterated through this process 50 times

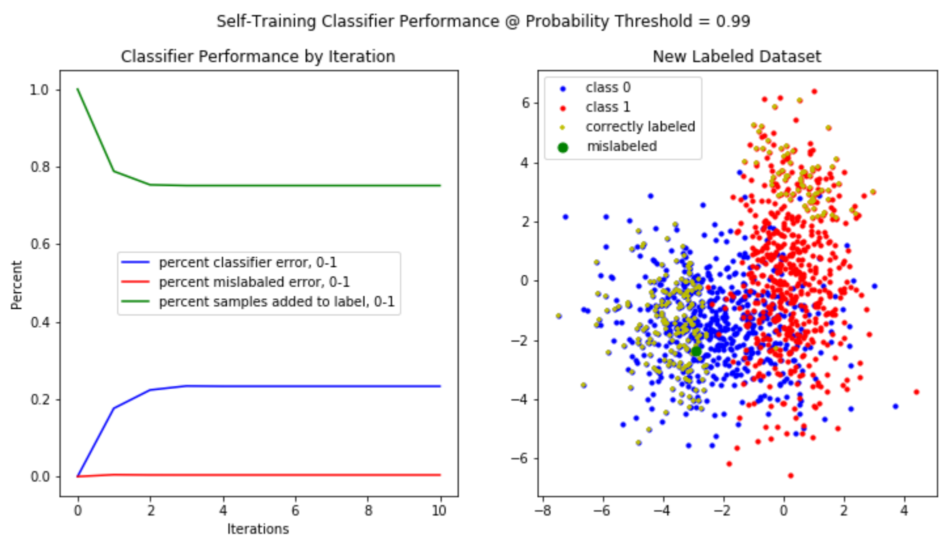


        - This reclassified nearly half of the dataset to labeled data, but made a decent number of errors (on the order of 20)
        - This showed me that I had selected a tolerance that was too low, and probably was running for longer than necessary
    - Then, I set the probability threshold at 0.95 and decreased the number of iterations to 10



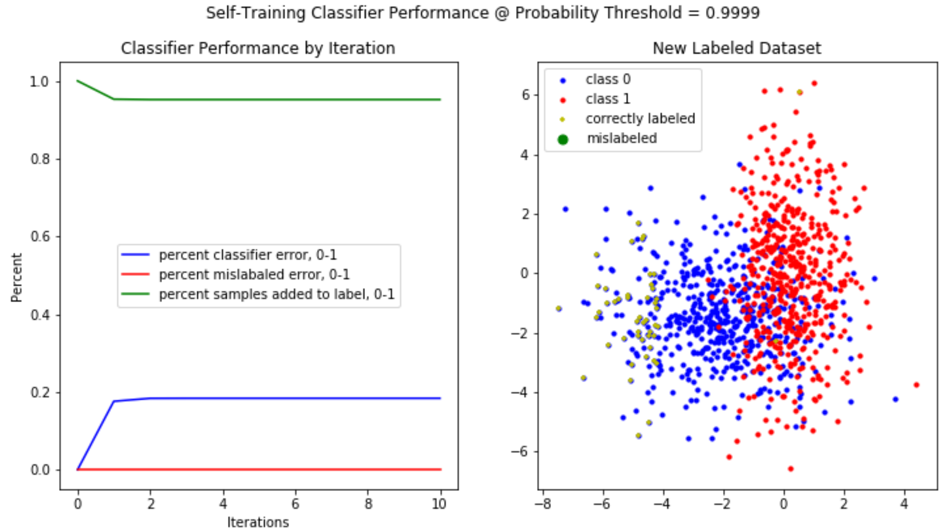
- This reclassified roughly 40% of the dataset to labeled data, and made fewer errors (on the order of <10)
- This showed me that increasing the tolerance was the surest way to make fewer classification errors, however it comes with the cost of reclassifying less data.

- To see if I could get rid of all classification errors, I set the tolerance to 0.99 (keeping the number of iterations running constant)



- This resulted in nearly perfect classification during the relabeling process. However, only 20% of the unlabeled dataset was able to be confidently put into the labeled set

- I then set the strictest tolerance I could think of, 0.9999. Thus the model would only reclassify when it was super sure



- This resulted in zero reclassification errors, however only labeled roughly 5% of the dataset.

- My takeaways:
  - Labeling data using this semi-supervised technique is a tradeoff between accuracy and the amount of labelling you are able to do. I.E. - setting a loose tolerance can get you to relabel most of the data, but you find more and more mislabeled datapoints creeping into your new labeled dataset.
  - Percentages only tell part of the story. Although the percent mislabeled error can be helpful, it's really important to visualize the dataset as we have done to (hopefully) get a better idea of how the classifier has performed.
  - Its interesting that this approach only relabels data that is relatively far from the apparent decision boundary. This makes sense (given that the model would be lesson confident in this area), but I'd like to retry this by shuffling my dataset.
  - I'd like to try this again with a different classifier sometime. I think the quadratic discriminant is probably the roughest one we can use. But it is interesting that it
  - Really intersting (but predictable) that the percent classifier error (blue line) increases over time. This makes sense in that the classifier is only left trying to make predictions on the unlabeled part of the dataset which is hardest to understand. I.E. You've preferentially taken all the low-hanging fruit and moved it into the labeled section, and so you are left making your predictions on data that are more uncertain. The more you move to the labeled dataset, the more classification errors you will have on the remaining samples (by percentage)

2D Gaussian dataset

- Training Set: 1000 -> (500 samples from each class)
- Testing Set: 1000 -> (500 samples form each class)

In [99]:
```
# *NOTE What is covariance matrix, really?
def randomsamples(d, size, up=1, down=-1, u=False, sig=False, condin = True, ret
    """A function to generate random samples
```

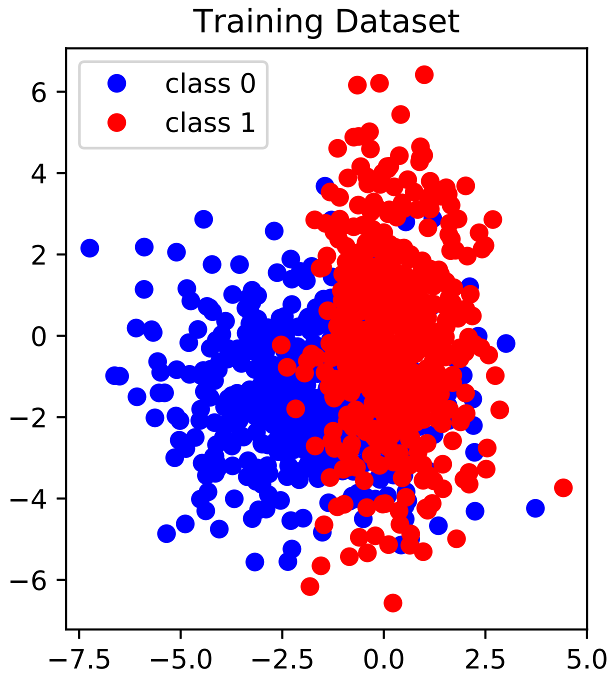
```
        inputs:
        d -> dimensions (int)
        size -> the size of the sample desired
        up -> the max of range of numbers to generate random
            (default 1)
        down -> the min of range of numbers to generate random
            (default -1)
        u -> optional input mean, a vector of size d
            (if not added, script will generate randomly)
        sig -> optional input covariance matrix, a matrix
            of dimensions d*d
            (if not added, script will generate randomly)
        condin -> conditional independence boolean
            if True (default) then off-diagonal
            values of sigma are zero
            if False, then any values in sigma
            may be a real number
        retall -> boolean for returning u and sigma
            True -> returns distribution, u, sig
            False -> returns distribution
            (default False)


        returns:
        a multivariate matrix sample with gaussian distribution

        and optionally u and sig
        """

    if u is False:
        ## means of dimensions 'd' [0, 1)
        u = np.random.uniform(down, up, size=(d,))

    if sig is False:
        ## covariance matrix of dimension 'd*d' [0,1)
        sig = np.random.uniform(down, up, size=(d,d))
        ## test for conditional independence
        if condin:
            sig = sig*np.identity(d)

    if retall:
        return np.random.multivariate_normal(u, sig, size).T, u, sig

    else:
        return np.random.multivariate_normal(u, sig, size).T
```

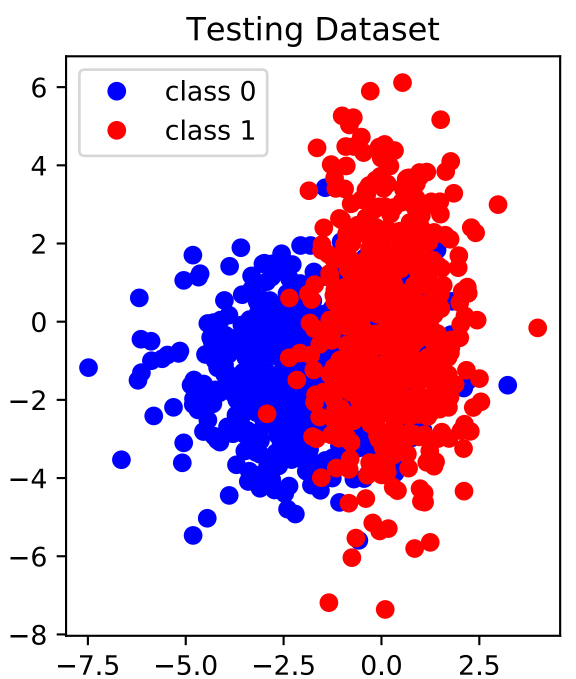
In [138…

```
# 1. globals
d = 2 # dimensions
k = 1000 # size of input
n = 500 # size of test subset
class_num = 2 # number of classes
up_in = 6 # upper bound of input data
down_in = -6 # lower bound of input data

# 2. create bivariate gaussian data with 3 known classes, that are conditionally
x_1, u_true1, sig_true1 = randomsamples(d,k,up=up_in,down=down_in,retall=True)
x_2, u_true2, sig_true2 = randomsamples(d,k,up=up_in,down=down_in,retall=True)


# 3. reserve some as a test set (n number)
```

```
x_1_train, x_1_test = x_1[:,0:-n], x_1[:,-n:]
x_2_train, x_2_test = x_2[:,0:-n], x_2[:,-n:]
x_train_list = [x_1_train, x_2_train]
x_test_list = [x_1_test, x_2_test]

# 4. calculate mean and sigma from other (train) set for each of three classes
u_1, u_2 = np.mean(x_1_train, axis=1), np.mean(x_2_train, axis=1)
sig_1, sig_2 = np.var(x_1_train, axis=1), np.var(x_2_train, axis=1)
u_list = [u_1, u_2]
sig_list = [sig_1, sig_2]

# 5. list containing priors for each class_c
# assume equal priors (because there are the same
#   numbers in each class, they have a 1/c chance
#   of occuring, 1/2)
pri_list = [(1/2), (1/2)]

# 6. plot training
# color list for graphing
color_list = ['b', 'r']
for cla in range(class_num):
    plt.plot(x_train_list[cla][0], x_train_list[cla][1], 'o', c=color_list[cla],

plt.legend()
plt.gca().set_aspect('equal', adjustable='box')
plt.title('Training Dataset')
plt.show()

# 6. plot testing
# color list for graphing
color_list = ['b', 'r']
for cla in range(class_num):
    plt.plot(x_test_list[cla][0], x_test_list[cla][1], 'o', c=color_list[cla],la

plt.legend()
plt.gca().set_aspect('equal')
plt.title('Testing Dataset')
plt.show()
```

```
<ipython-input-99-ce2ff57db36f>:45: RuntimeWarning: covariance is not positive-s
emidefinite.
  return np.random.multivariate_normal(u, sig, size).T, u, sig
```





Algorthm

- for t in T:
    - 1. Train f on (xl, yl)
    - 1. Make predictions on Xu with f(x) x<-Xu
    - 1. Choose the samples in Xu w/ high confidence and add them into the labeled dataset xl
        - Make it a hard label
        - Use the posterior to make the decision
    - NOTE: Report Error

- The first time a classifier is trained
- at least one point during self-training
- after self-training

In [161…

```
# original dataset
X_l = np.concatenate(x_train_list,axis=1).T # training data, predictors
y_l = np.array([np.full((k-n),0), np.full((k-n),1)]).flatten() # training data,
X_u = np.concatenate(x_test_list,axis=1).T # testing data, predictors - in this
y_u = np.array([np.full((n),0), np.full((n),1)]).flatten() # testing data, targe
y_real = y_l # for comparing after the fact to the labels added

# create classifier
clf = QuadraticDiscriminantAnalysis()

# name
name = 'QuadraticDiscriminant_1'

# set run time
T = 10

# set posterior probability threshold for appending predicted unlabeled data int
th = 0.9999

# keep track of loss and num unlabeled
loss = [0] # prediction error (regardless of whether or not unlabeled data are a
num = [1] # the number of samples that have been added to the newest dataset
hcloss = [0] # the number of high-confidence mistakes added to labeled data
```

In [162…

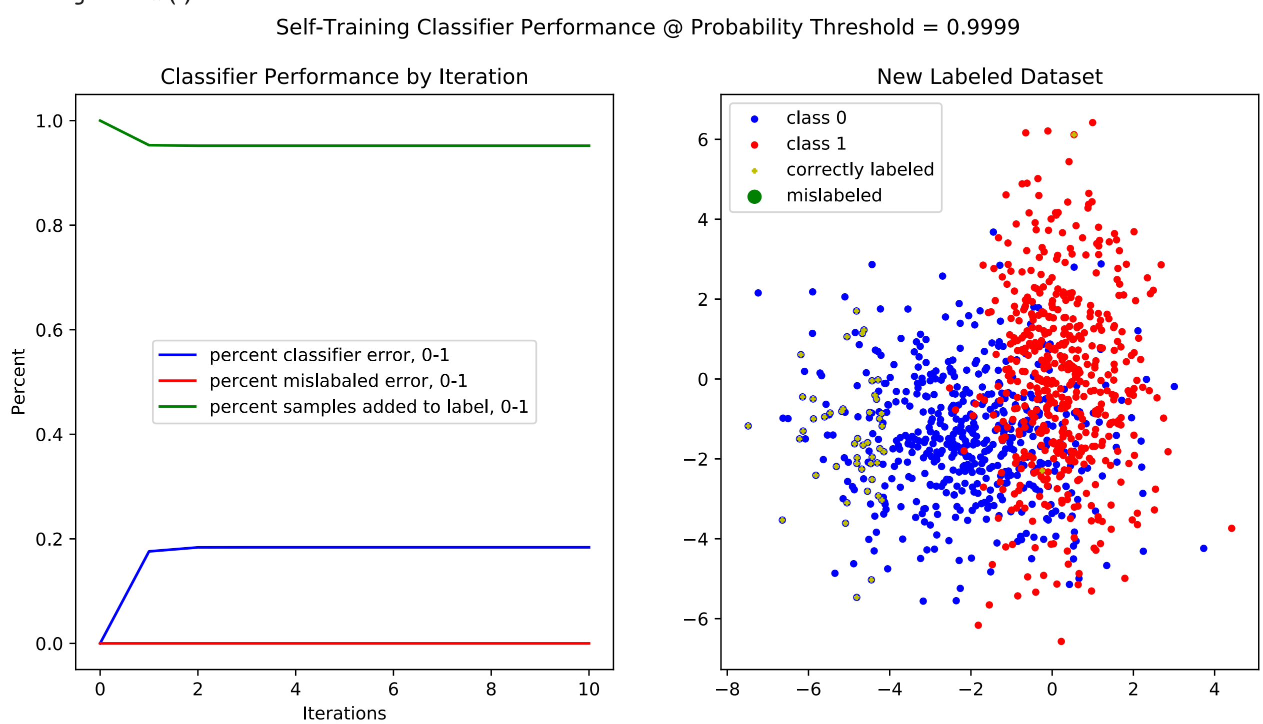
```
# algorithm
for t in range(T):
    # print('iteration', t)
    # 1 . Train f on (xl, yl)
    clf.fit(X_l, y_l)

    # 2. Make predictions on Xu with f(x) x<-Xu
    y_hat = clf.predict(X_u)

    # 3. Choose the samples in Xu w/ high confidence and add them into the label
    # predict probabilities for each prediction
    X_u_prob = clf.predict_proba(X_u)
    # find indices of those that make threshold
    idx = np.where(X_u_prob >= th)
    # print('number of high probability ids', len(idx[0]))
    in_arr, in_y = np.array(X_u[idx[0]]), idx[1] # for later use adding and dele
    y_real = np.append(y_real, y_u[idx[0]]) # to remember the true values for ev

    # report prediction error (regardless of whether or not unlabeled data are a
    totalwrong = len(np.where(y_hat != y_u)[0])
    total = y_u.shape[0]
    err = (totalwrong / total)
    loss.append(err)

    # add x data, and y data to labeled
    X_l, y_l = np.vstack([X_l, in_arr]), np.append(y_l, in_y)
    # print('shapes of new labeled sets x and y', X_l.shape, y_l.shape)

    # remove x data, and y data from unlabeled
    X_u, y_u = np.delete(X_u,idx[0],axis=0), np.delete(y_u,idx[0])
    # print('shapes of new unlabeled sets x and y', X_u.shape, y_u.shape, '\n')
```

```
    # the number of new samples added
    num.append(y_u.shape[0]/k)

    # the number of samples added from the wrong class to the labeled data
    labeledwrong = len(np.where(y_l != y_real)[0])
    totallabeled = y_l.shape[0] - n*2
    hcloss.append(labeledwrong / totallabeled)
```

In [163…

```
fig, ax = plt.subplots(1,2,figsize=(12,6))
# ax2 = ax.twinx()
ax[0].plot(loss, label='percent classifier error, 0-1', c='b')
ax[0].plot(hcloss, label='percent mislabaled error, 0-1', c='r')
ax[0].plot(num, label='percent samples added to label, 0-1', c='g')
ax[0].set_xlabel('Iterations')
ax[0].set_ylabel('Percent')
ax[0].legend(loc='center')
ax[0].set_title('Classifier Performance by Iteration')

ax[1].scatter(X_l[:,0][y_l == 0],X_l[:,1][y_l == 0], label='class 0', s=10, c='b
ax[1].scatter(X_l[:,0][y_l == 1],X_l[:,1][y_l == 1], label='class 1', s=10, c='r
ax[1].scatter(X_l[n*2-1:-1,0][y_l[n*2-1:-1] == y_real[n*2-1:-1]],X_l[n*2-1:-1,1]
ax[1].scatter(X_l[:,0][y_l != y_real],X_l[:,1][y_l != y_real], label='mislabeled
ax[1].legend()
ax[1].set_title('New Labeled Dataset')
fig.suptitle('Self-Training Classifier Performance @ Probability Threshold = '+s
fig.show()
plt.savefig('assets/'+name+str(th)+'.png')
```

```
<ipython-input-163-3d805ab23129>:18: UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot s
how the figure.
  fig.show()
```



In [1]:
```
## Modules
import numpy as np
import matplotlib.pyplot as plt
from numpy import genfromtxt
import pandas as pd

from sklearn.datasets import make_gaussian_quantiles
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold

from sklearn.svm.libsvm import predict_proba

from copy import deepcopy

import warnings
```

```
/Users/roberthull/opt/miniconda3/envs/Res1/lib/python3.8/site-packages/sklearn/u
tils/deprecation.py:143: FutureWarning: The sklearn.svm.libsvm module is  deprec
ated in version 0.22 and will be removed in version 0.24. The corresponding clas
ses / functions should instead be imported from sklearn.svm. Anything that canno
t be imported from sklearn.svm is now part of the private API.
  warnings.warn(message, FutureWarning)
```

2. An Experiment on Real World Data

- Implement the self-training algorithm using ten datasets available on the course Github repo
- Requirements:
  - Report your results using 5-fold cross validation.
    - In each cross-validation step you can use only 15% of data as labeled
- Write a brief discussion on whether semi-supervised helped on real-world data sets

Reference: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html for some more discussion about comparing classifiers in sklearn

Brief Discussion

In general, the self-training approach for incorporating unlabeled data into the labeled dataset yielded improved accuracy for some classifiers in the right conditions, but not for others. In general, gains were modest. I would say that better classifiers are more likely to benefit from self-training than poor classifiers. In fact, poor classifiers that get self-trained are likely to decrease in their performance due to over-confidence in datasets filled with more and more mis-labeled data. This seems to hold true even for very strict probability thresholds re: relabeling

1. In our simplest classifier, for all 10 datasets (shown and discussed below) trained using a *QuadraticDiscriminant* classifier at a probability threshold of 0.999 trained up to 5 iterations, **the performance of the classifier (when measured as % error performance on a fixed test dataset) stayed constant or decreased** after incorporating data 'labeled' by self-training.

2. In our moderate classifer, for 9 datasets (shown and discussed below) trained using a *K Nearest Neighbors* classifier at a probability threshold of 0.999 trained up to 5 iterations, **the performance of the classifier (when measured as % error performance on a fixed test dataset) stayed constant or improved** after incorporating data 'labeled' by self-training. It's worth noting that for some of these datasets, the baseline (not self-trained) model was actually sufficient to classify the test data without any additionally labelling at all.

3. In our 'nuclear option', for 9 datasets (shown and discussed below) trained using a *multi-level perceptron* classifier at a probability threshold of 0.999 trained up to 5 iterations, **the performance of the classifier (when measured as % error performance on a fixed test dataset) was *unaffected or slightly decreased*** after incorporating data 'labeled' by self-training **even as in general it had a lower error both before and after pre-training**. This was true even though this method required a lot more compute time! This makes me think that with really effective classifiers like MLP perhaps what they benefit from is not just more data points, but in fact bigger more comprehensive datasets with more features that better characterize the datasets.

A broad brushed review of the performance for each (10) datasets before (`baseline`) and after (`end`) self-training is discussed below:

- **abalone**
  - Quadratic

    ```
    The baseline (not self-trained) error on test dataset abalone-is 38.57
    The end (self-trained) error on test dataset abaloneis 39.21
    ```
  - KNN

    ```
    The baseline (not self-trained) error on test dataset abaloneis 40.27
    The end (self-trained) error on test dataset abaloneis 40.2
    ```
  - MLP

    ```
    The baseline (not self-trained) error on test dataset abaloneis 34.12
    The end (self-trained) error on test dataset abaloneis 33.92
    ```
- **acute-inflammation**

- Quadratic

```
      The baseline (not self-trained) error on test dataset acute-inflammationis 5.83
      The end (self-trained) error on test dataset acute-inflammationis 5.83
```

- KNN

```
      The baseline (not self-trained) error on test dataset acute-inflammationis 0.0
      The end (self-trained) error on test dataset acute-inflammationis 0.0
```

- MLP

```
      The baseline (not self-trained) error on test dataset acute-inflammationis 0.0
      The end (self-trained) error on test dataset acute-inflammationis 0.0
```

- **acute-nephritis**
  - Quadratic

```
      The baseline (not self-trained) error on test dataset acute-nephritisis 14.17
      The end (self-trained) error on test dataset acute-nephritisis 16.67
```

  - KNN

```
      The baseline (not self-trained) error on test dataset acute-nephritisis 0.0
      The end (self-trained) error on test dataset acute-nephritisis 0.0
```

  - MLP

```
      The baseline (not self-trained) error on test dataset acute-nephritisis 0.0
      The end (self-trained) error on test dataset acute-nephritisis 0.0
```

- **miniboone**
  - Quadratic
    - The baseline (not self-trained) error on test dataset miniblooneis 26.53
    - The end (self-trained) error on test dataset miniblooneis 38.1
  - KNN
    - miniboone is too big for this
  - MLP
    - too big for this
- **balance-scale**
  - Quadratic

```
      The baseline (not self-trained) error on test dataset
balance-scaleis 8.48
      The end (self-trained) error on test dataset balance-scaleis
8.48
```

- KNN

```
      The baseline (not self-trained) error on test dataset
balance-scaleis 20.32
      The end (self-trained) error on test dataset balance-scaleis
19.68
```

- MLP

```
      The baseline (not self-trained) error on test dataset
balance-scaleis 6.72
      The end (self-trained) error on test dataset balance-scaleis
6.08
```

- **bank**
  - Quadratic

```
      The baseline (not self-trained) error on test dataset bankis
14.97
      The end (self-trained) error on test dataset bankis 16.17
```

  - KNN

```
      The baseline (not self-trained) error on test dataset bankis
11.46
      The end (self-trained) error on test dataset bankis 11.28
```

  - MLP

```
      The baseline (not self-trained) error on test dataset bankis
10.59
      The end (self-trained) error on test dataset bankis 10.46
```

- **blood**
  - Quadratic

```
      The baseline (not self-trained) error on test dataset bloodis
54.71
      The end (self-trained) error on test dataset bloodis 43.04
```

  - KNN

```
      The baseline (not self-trained) error on test dataset bloodis
24.2
      The end (self-trained) error on test dataset bloodis 24.59
```

  - MLP

```
      The baseline (not self-trained) error on test dataset bloodis
20.31
      The end (self-trained) error on test dataset bloodis 20.58
```

- **breast-cancer**
  - Quadratic

```
      The baseline (not self-trained) error on test dataset breast-canceris 29.36
      The end (self-trained) error on test dataset breast-canceris 31.46
```

- KNN

```
      The baseline (not self-trained) error on test dataset breast-canceris 36.69
      The end (self-trained) error on test dataset breast-canceris 37.04
```

- MLP

```
      The baseline (not self-trained) error on test dataset breast-canceris 30.04
      The end (self-trained) error on test dataset breast-canceris 27.6
```

- **car**
  - Quadratic

```
      The baseline (not self-trained) error on test dataset caris 34.96
      The end (self-trained) error on test dataset caris 35.59
```

  - KNN

```
      The baseline (not self-trained) error on test dataset caris 6.71
      The end (self-trained) error on test dataset caris 6.19
```

  - MLP

```
      The baseline (not self-trained) error on test dataset caris 3.65
      The end (self-trained) error on test dataset caris 4.05
```

- **chess-krvk**
  - Quadratic

```
      The baseline (not self-trained) error on test dataset chess-krvkis 78.02
      The end (self-trained) error on test dataset chess-krvkis 79.57
```

  - KNN

```
      The baseline (not self-trained) error on test dataset chess-krvkis 36.2
      The end (self-trained) error on test dataset chess-krvkis 36.72
```

  - MLP

```
      The baseline (not self-trained) error on test dataset chess-krvkis 53.97
      The end (self-trained) error on test dataset chess-krvkis 53.95
```

Real Datasets

In [9]:
```
# 1. read in data

nms = ['abalone', 'acute-inflammation', 'acute-nephritis', 'balance-scale',
       'bank', 'blood', 'breast-cancer', 'car', 'chess-krvk'] # removed miniboo

path = '../UA-ECE-523-Sp2018/data/'

data_db = pd.DataFrame(columns=['nm','fold','x_train','y_train','x_test','y_test

n_splits = 5

for nm in nms:
    # temporarily read-in the data
    temp = genfromtxt(path+nm+'.csv',delimiter=',')
    temp_x = temp[:,:-1]
    temp_y = temp[:,-1]

    # set up folds

    kf = KFold(n_splits=n_splits, shuffle=True)
    kf.get_n_splits(temp_x)
    i = 1

    # read in test and train data using K-Fold
    for train_index, test_index in kf.split(temp_x):
       X_train, X_test = temp_x[train_index], temp_x[test_index]
       y_train, y_test = temp_y[train_index], temp_y[test_index]

       data_dict = {'nm': [nm],
                    'fold' : [i],
                    'x_train': [X_train],
                    'y_train': [y_train],
                    'x_test': [X_test],
                    'y_test': [y_test]}

       temp_df = pd.DataFrame(data_dict)
       data_db = data_db.append(temp_df)
       i = i + 1

       del temp_df, data_dict, X_train, X_test, y_train, y_test

    del temp, temp_x, temp_y
```

Algorthm

- for t in T:
    - 1. Train f on (xl, yl)
    - 1. Make predictions on Xu with f(x) x<-Xu
    - 1. Choose the samples in Xu w/ high confidence and add them into the labeled dataset xl
        - Make it a hard label
        - Use the posterior to make the decision
    - NOTE: Report Error

- The first time a classifier is trained
- at least one point during self-training
- after self-training

In [11]:

```
def _calcError(y_true, y_predicted):
    '''
    calculating percent error, 0 to 1
    y_true : y_hat - true target values
    y_predicted : y_pred - target values predicted using classifier
    '''
    totalwrong = len(np.where(y_true != y_predicted)[0])
    total = y_predicted.shape[0]
    err = (totalwrong / total)
    return err

def selfTraining(df_in, clf=None, th=0.99, T=10):
    '''
    algorithm for self training and calculating losses after adding a self-train
    df_in : the dataframe for a single dataset (formatted in a predicted way - c
    clf : classifier, by default the model is QuadraticDiscriminantAnalysis() if
    th : threshold for posterior probability for appending predicted unlabeled d
    T : number of iterations
    '''

    # original dataset
    dataset_name = df_in['nm'][0]
    dataset_fold = df_in['fold'][0]
    print(dataset_name)
    X_l = df_in['x_train'][0] # training data, predictors
    y_l = df_in['y_train'][0] # training data, target
    X_u = df_in['x_test'][0] # testing data, predictors - in this case unlabeled
    y_u = df_in['y_test'][0]# testing data, target - in this case treated as unl
    X_u_orig = deepcopy(X_u) # testing data, predictors - a dataset that doesn't
    y_u_orig = deepcopy(y_u) # testing data, target - a datset that doesn't get
    y_real = y_l # for comparing after the fact to the labels added

    # characteristics of original dataset
    n = y_l.shape[0] # size of original labeled dataset
    k = y_u.shape[0] # size of original unlabeled dataset

    if clf is None:
        # create classifier
        clf = QuadraticDiscriminantAnalysis()

    # keep track of loss and num unlabeled
    loss = [0] # prediction error (the error against the remaining test dataset)
    loss_gross = [] # prediction error (the error against the original dataset)
    num = [1] # the number of samples that have been added to the newest dataset
    hcloss = [0] # the number of high-confidence mistakes added to labeled data

    # initialize gross loss (the prediction error against the original test data
    # print(X_l.shape)
    # print(y_l.shape)
    clf.fit(X_l, y_l)
    y_hat_orig = clf.predict(X_u_orig)
    err = _calcError(y_true=y_hat_orig, y_predicted=y_u_orig)
    loss_gross.append(err)
    del err, y_hat_orig
```

```
    # algorithm
    for t in range(T):
        # print('iteration', t)
        # 1 . Train f on (xl, yl)
        clf.fit(X_l, y_l)

        # 2. Make predictions on Xu with f(x) x<-Xu
        try:
            y_hat = clf.predict(X_u)
        except:
            break

        # 3. Choose the samples in Xu w/ high confidence and add them into the l
        # predict probabilities for each prediction
        X_u_prob = clf.predict_proba(X_u)
        # find indices of those that make threshold
        idx = np.where(X_u_prob >= th)
        # print('number of high probability ids', len(idx[0]))
        in_arr, in_y = np.array(X_u[idx[0]]), idx[1] # for later use adding and
        y_real = np.append(y_real, y_u[idx[0]]) # to remember the true values fo

        # report prediction error (the error against the remaining test dataset)
        err = _calcError(y_true=y_hat, y_predicted=y_u)
        loss.append(err)
        del err

        # report prediction error (the error against the original test dataset)
        y_hat_orig = clf.predict(X_u_orig)
        err = _calcError(y_true=y_hat_orig, y_predicted=y_u_orig)
        loss_gross.append(err)
        del err

        # add x data, and y data to labeled
        X_l, y_l = np.vstack([X_l, in_arr]), np.append(y_l, in_y)
        # print('shapes of new labeled sets x and y', X_l.shape, y_l.shape)

        # remove x data, and y data from unlabeled
        X_u, y_u = np.delete(X_u,idx[0],axis=0), np.delete(y_u,idx[0])
        # print('shapes of new unlabeled sets x and y', X_u.shape, y_u.shape, '\

        # the number of new samples added
        num.append(y_u.shape[0]/k)

        # the number of samples added from the wrong class to the labeled data
        labeledwrong = len(np.where(y_l != y_real)[0])
        totallabeled = y_l.shape[0] - n
        if totallabeled == 0:
            hcloss.append(0)
        else:
            hcloss.append(labeledwrong / totallabeled)


    fig, ax = plt.subplots(figsize=(6,6))
    # ax2 = ax.twinx()
    ax.plot(loss, label='percent classifier error - - reduced testing set, 0-1',
    ax.plot(loss_gross, marker='|', label='percent classifier error - - full tes
    ax.plot(hcloss, label='percent mislabaled error, 0-1', c='r')
    ax.plot(num, label='percent samples added to label, 0-1', c='g')
    ax.set_xlabel('Iterations')
    ax.set_ylabel('Percent')
    ax.legend()
```

```
    ax.set_title('Classifier Performance by Iteration - '+dataset_name+' dataset
    fig.suptitle('Self-Training Classifier Performance @ Probability Threshold =
    plt.show()
    fig.savefig('assets/'+dataset_name+'_'+str(clf)+'_'+str(dataset_fold)+'_'+st

    # returns the percent error on all testing data before and after
    return loss_gross[0], loss_gross[-1]
```

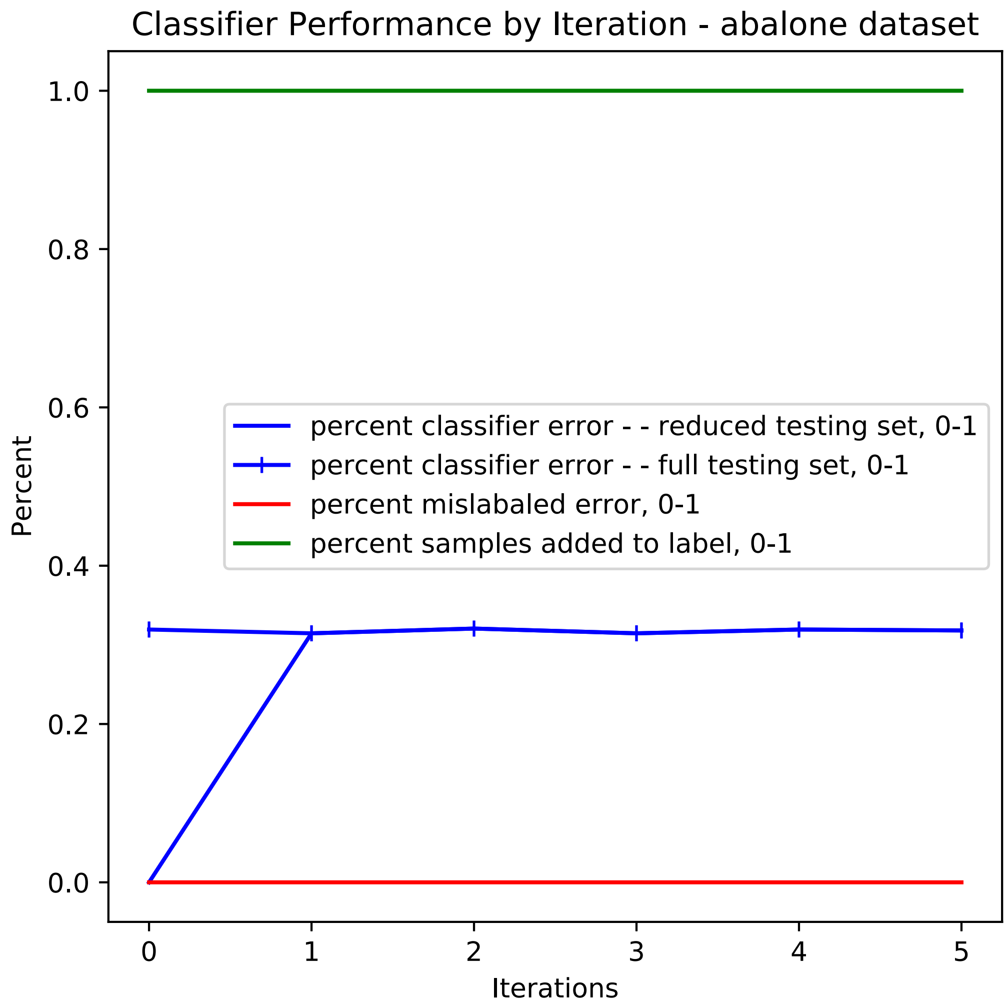
Comparison of algorithm on different datasets

In [14]:

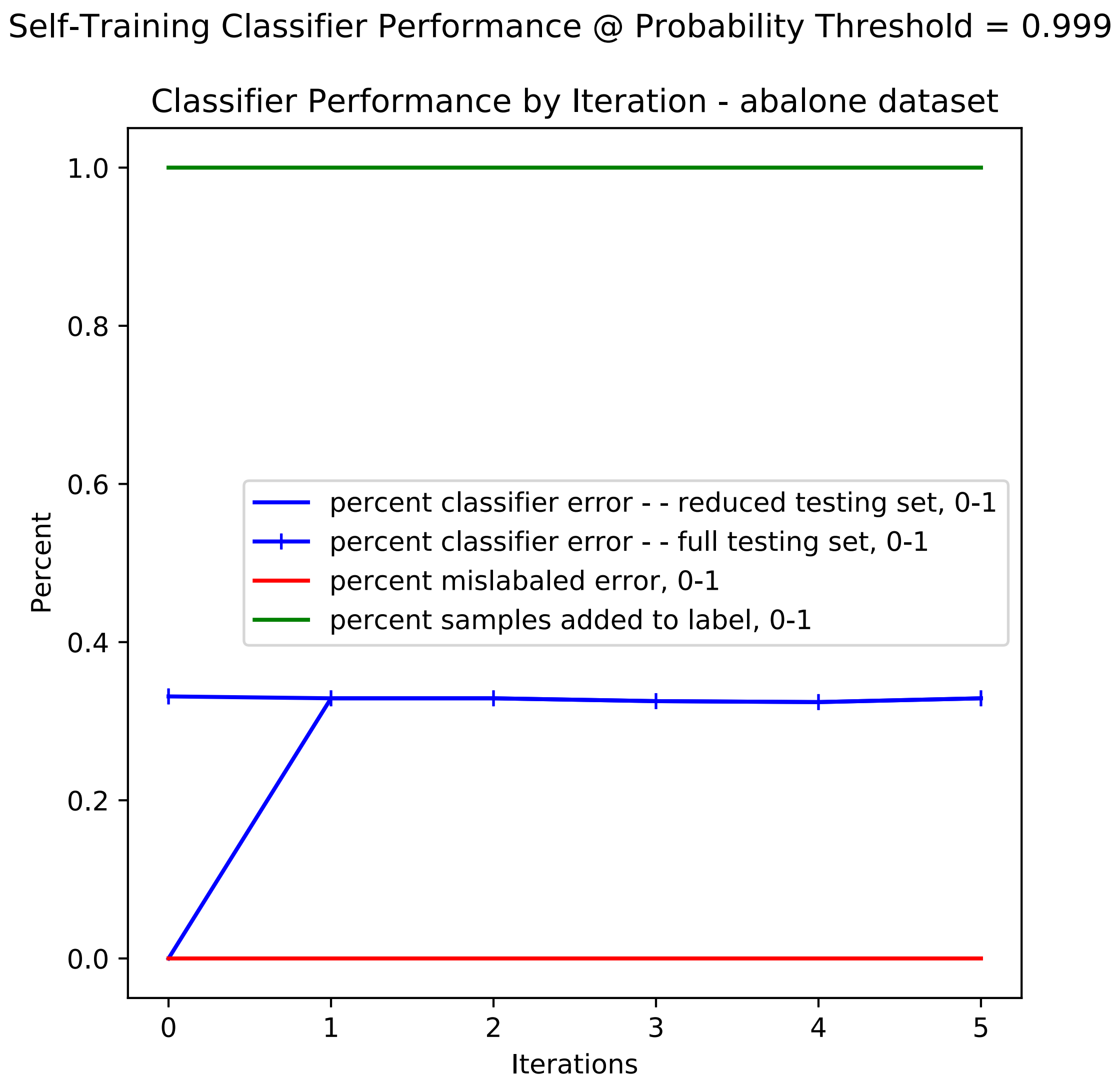
```
classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]


# set globals
clf_in = classifiers[6]
th_in = 0.999 # set posterior probability threshold for appending predicted unla
T_in = 5 # set the number of iterations


# suppress errors
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    # loop through all names
    for nm in nms:
        # keep track of the error associated with each classifier
        baseline_err = []
        ending_err = []
        # loop through each split
        for j in range(1, n_splits+1,1):
            df_in_in = data_db[(data_db['nm'] == nm) & (data_db['fold'] == j)] #
            # run each model keeping track of self-training
            berr, eerr = selfTraining(df_in_in, clf=clf_in, th=th_in, T=T_in)
            baseline_err.append(berr)
            ending_err.append(eerr)


        file1 = open('assets/HW5_model_out'+str(clf_in)+str(th_in)+str(T_in)+'.t
        text_base = '\n The baseline (not self-trained) error on test dataset '+
        text_err = '\n The end (self-trained) error on test dataset '+nm+'is '+s
        file1.write(text_base)
        file1.write(text_err)
        file1.close()
```

```
abalone
```

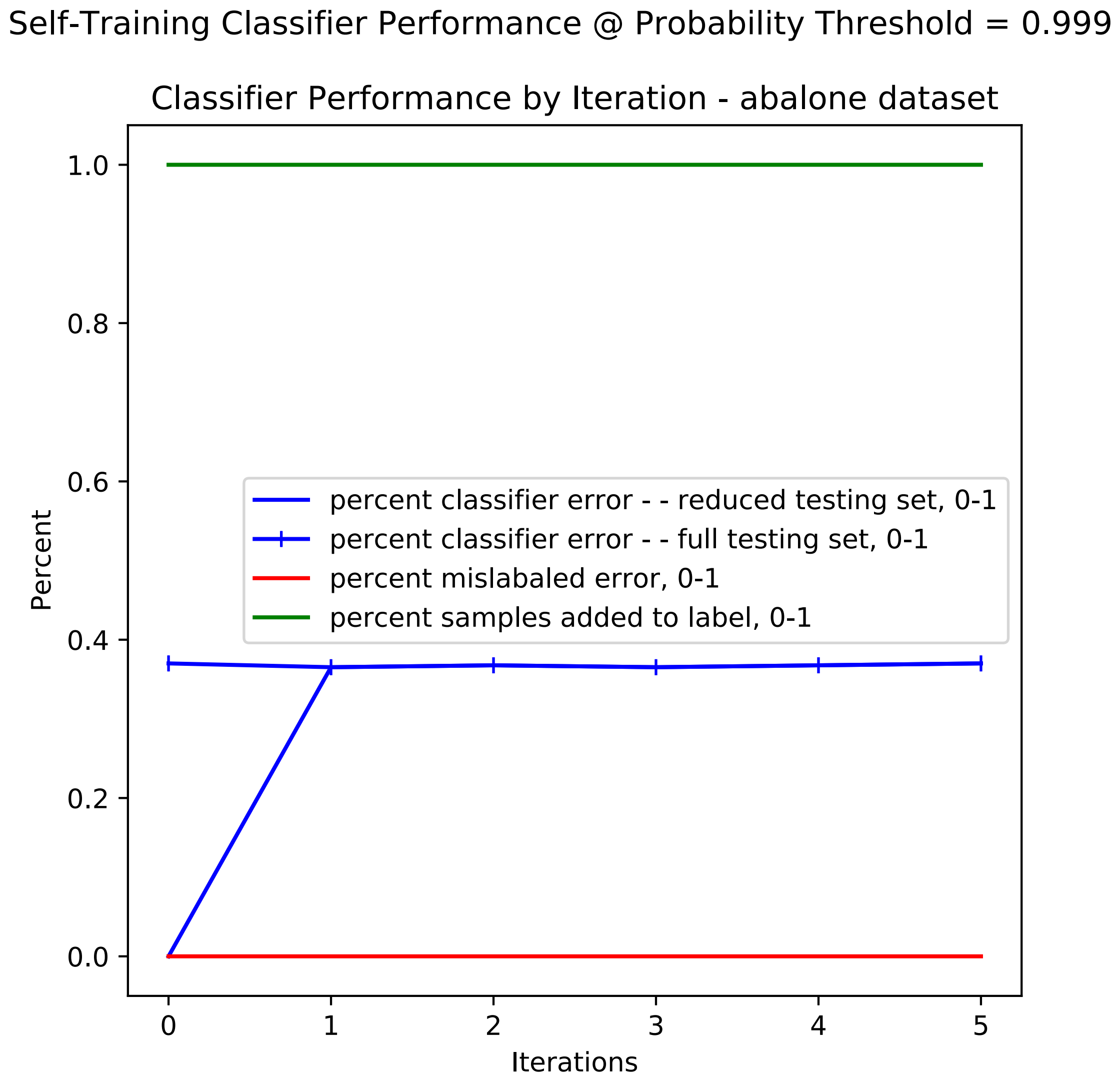
Self-Training Classifier Performance @ Probability Threshold = 0.999




```
abalone
```

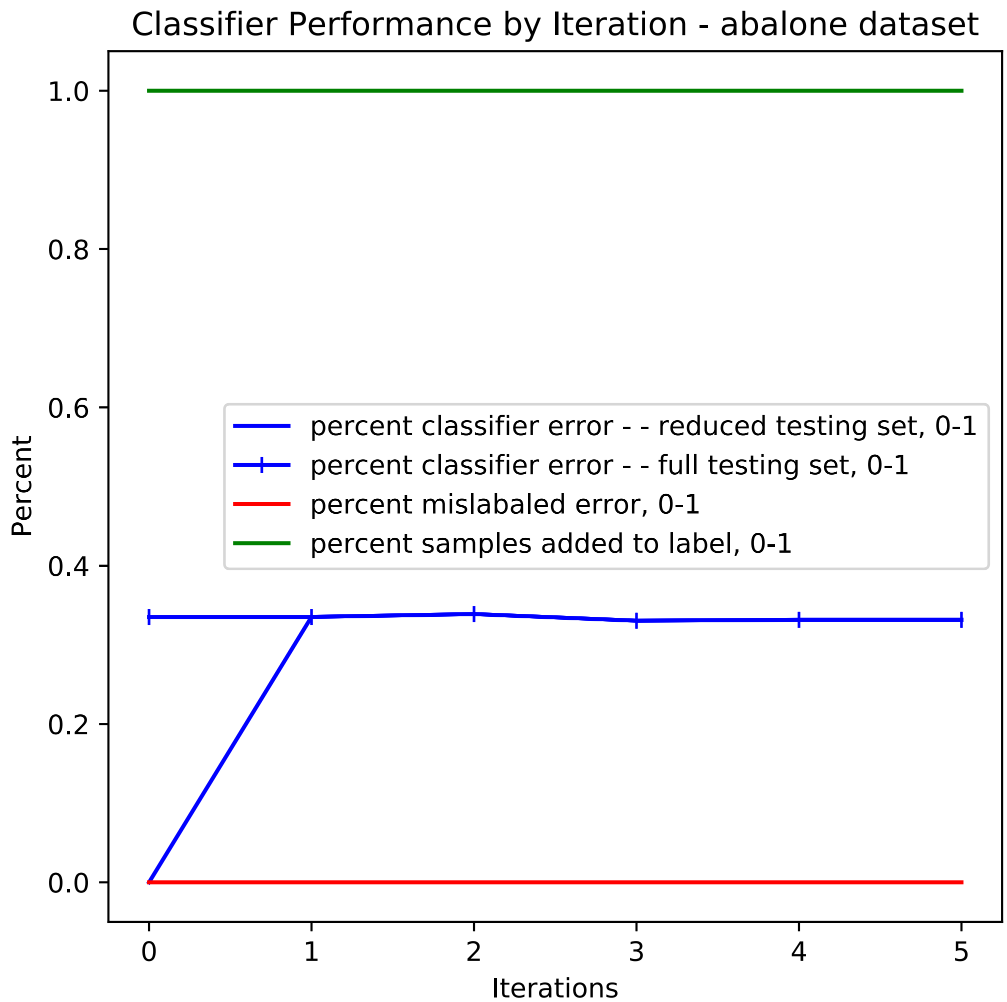

Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - abalone dataset
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
Percent
Iterations
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5

```
abalone
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - abalone dataset
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
Percent
Iterations
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5

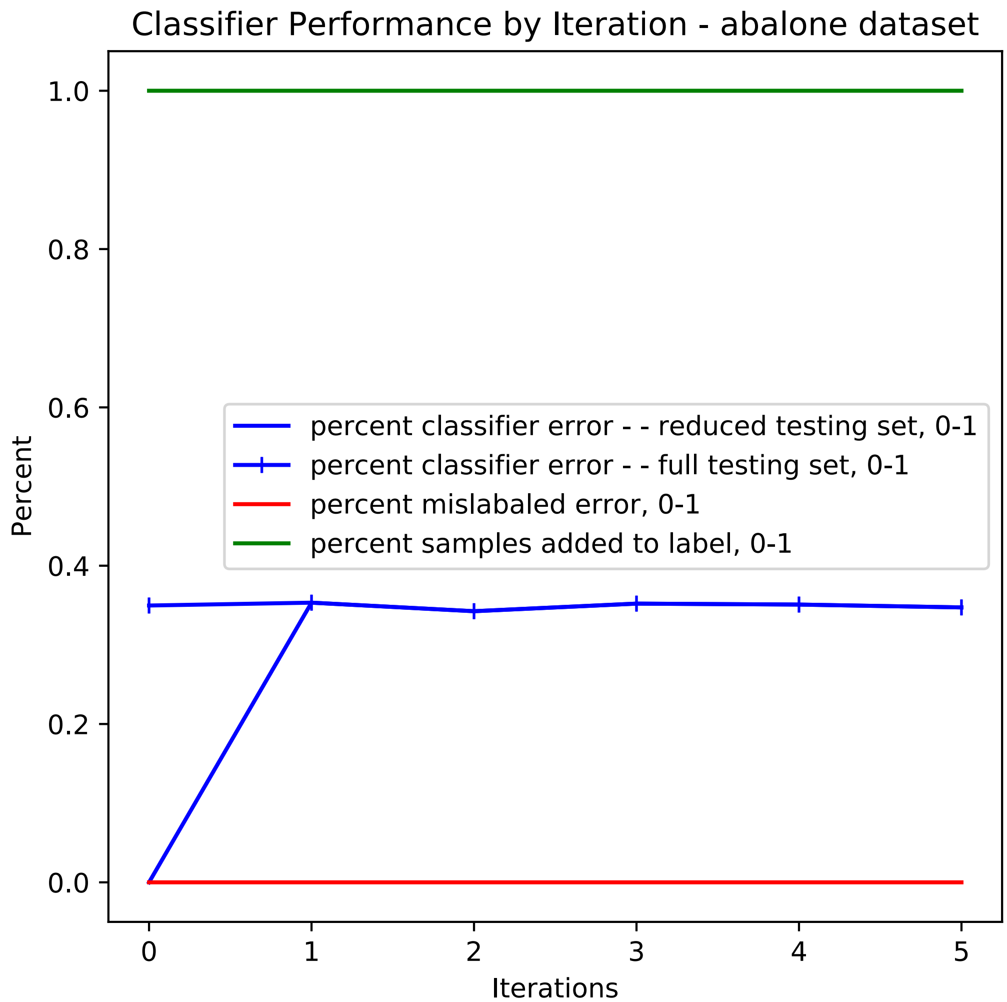
```
abalone
```

Self-Training Classifier Performance @ Probability Threshold = 0.999

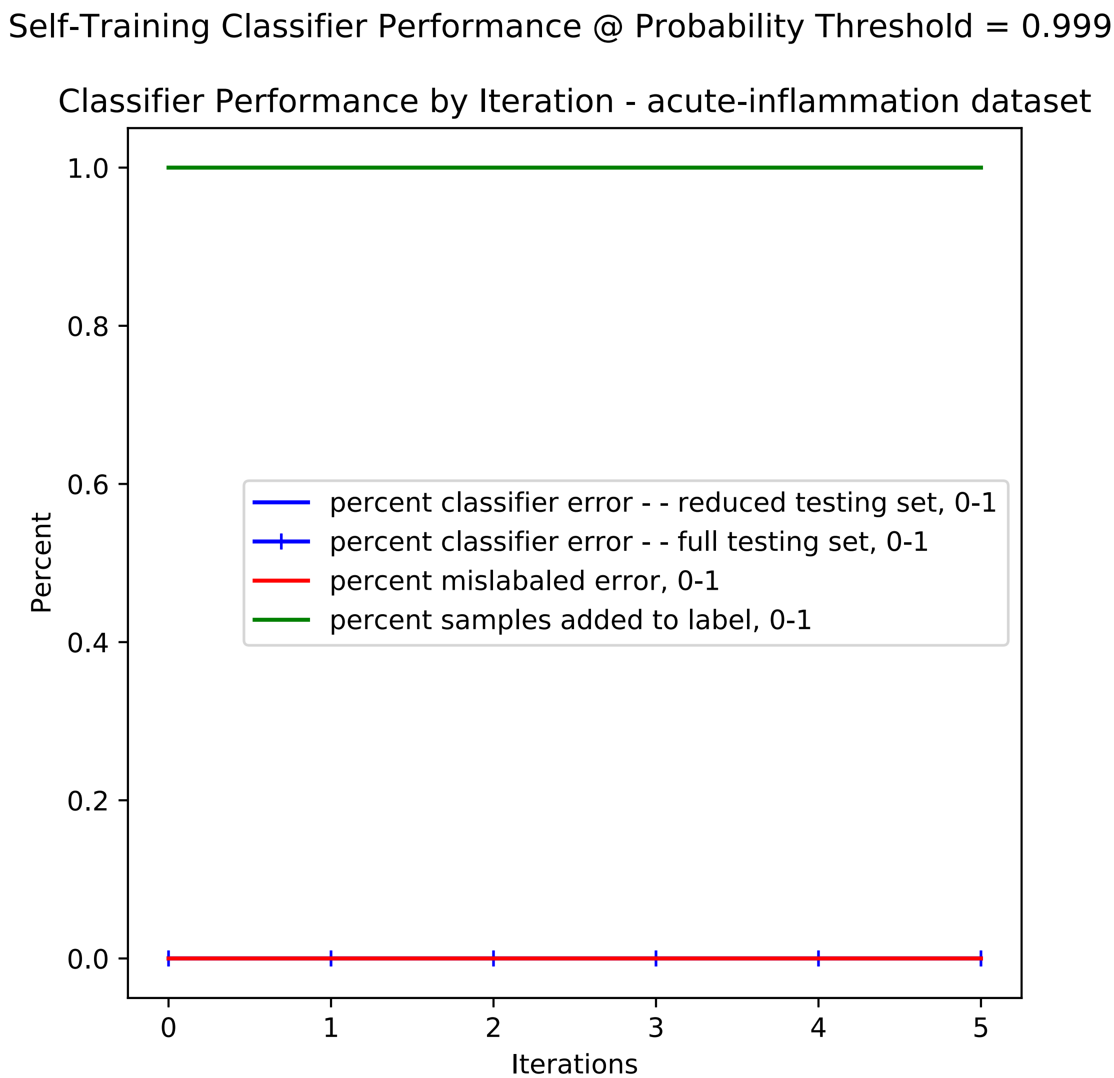



```
abalone
```

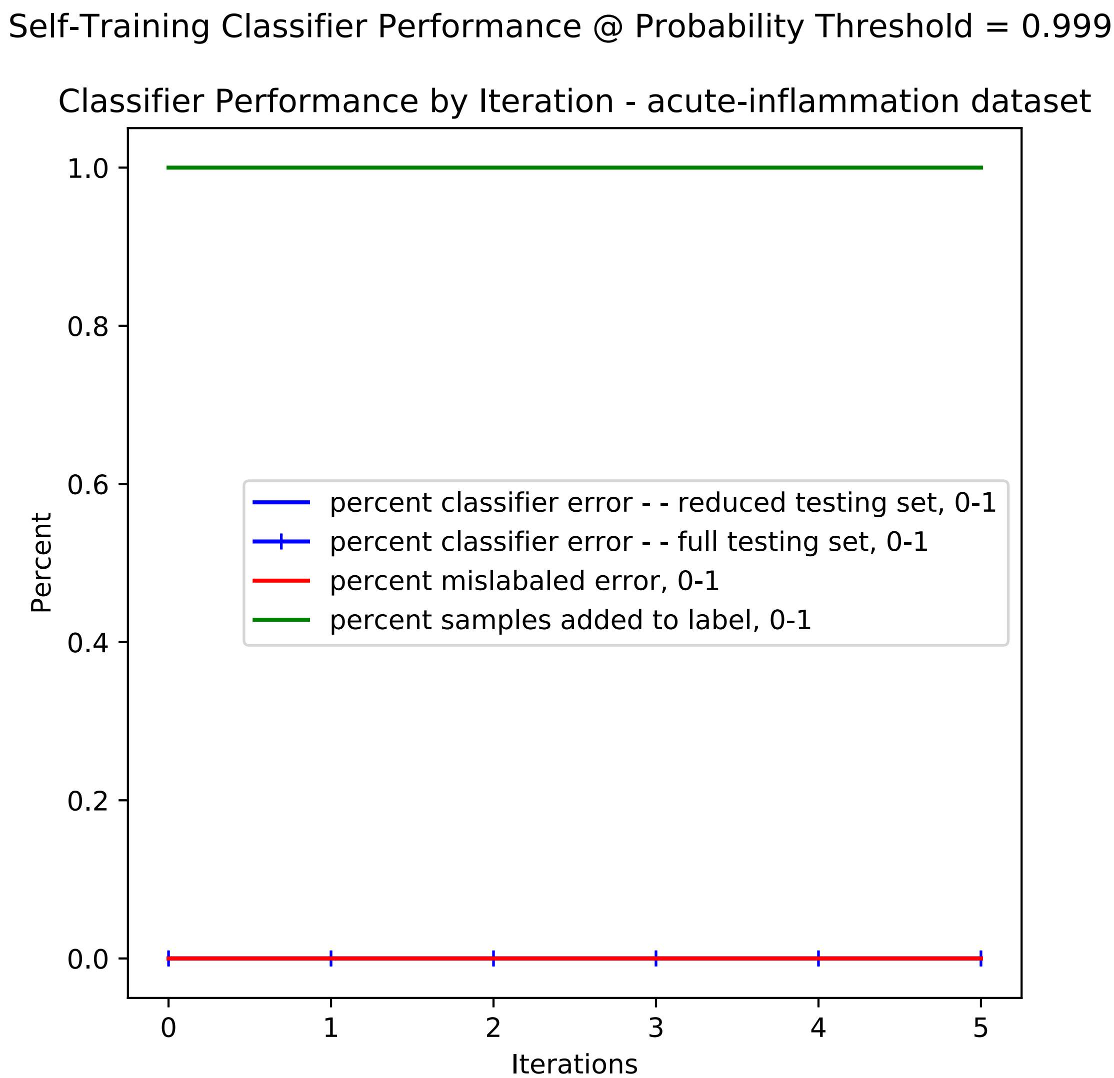
Self-Training Classifier Performance @ Probability Threshold = 0.999



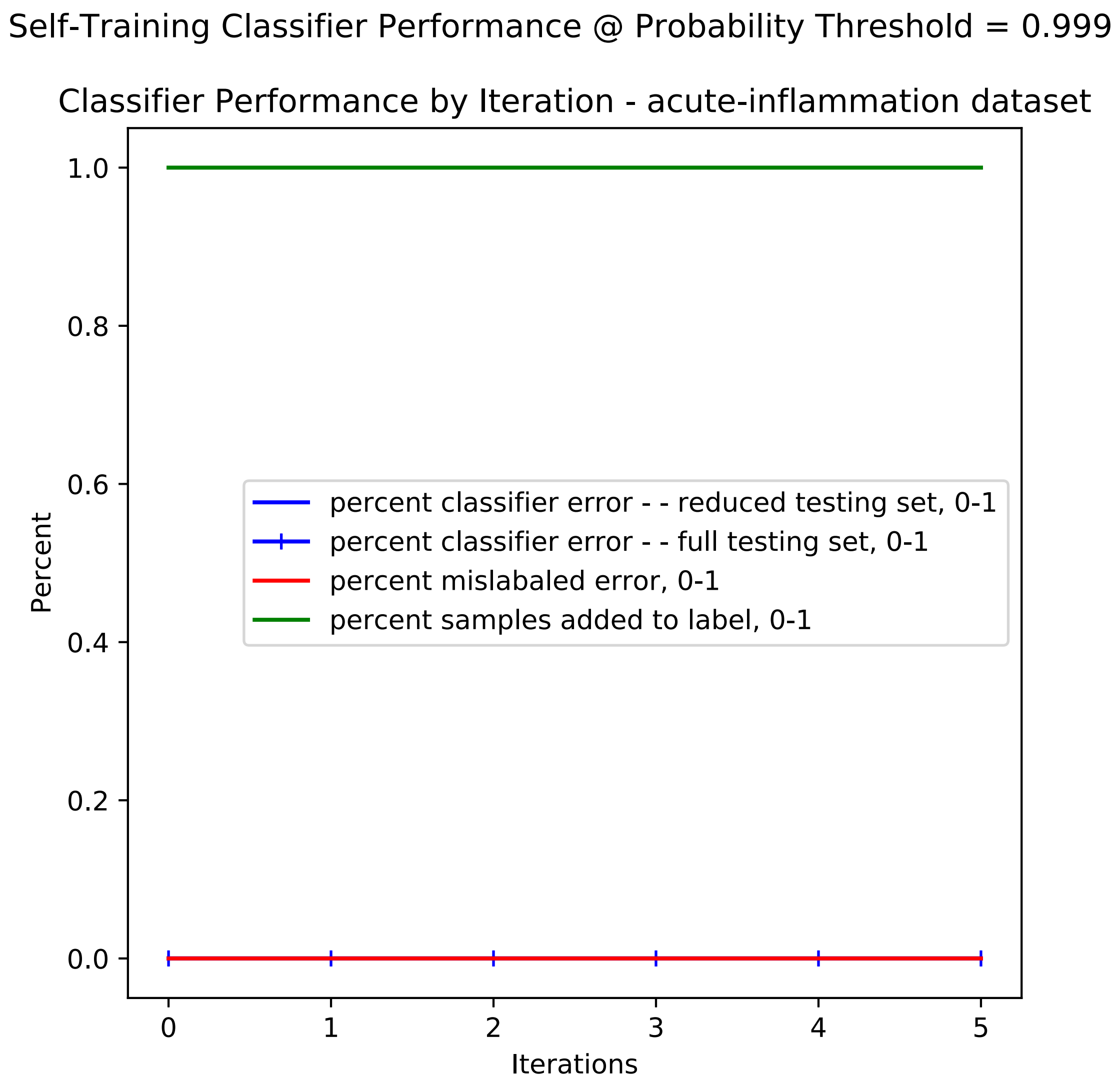

```
acute-inflammation
```



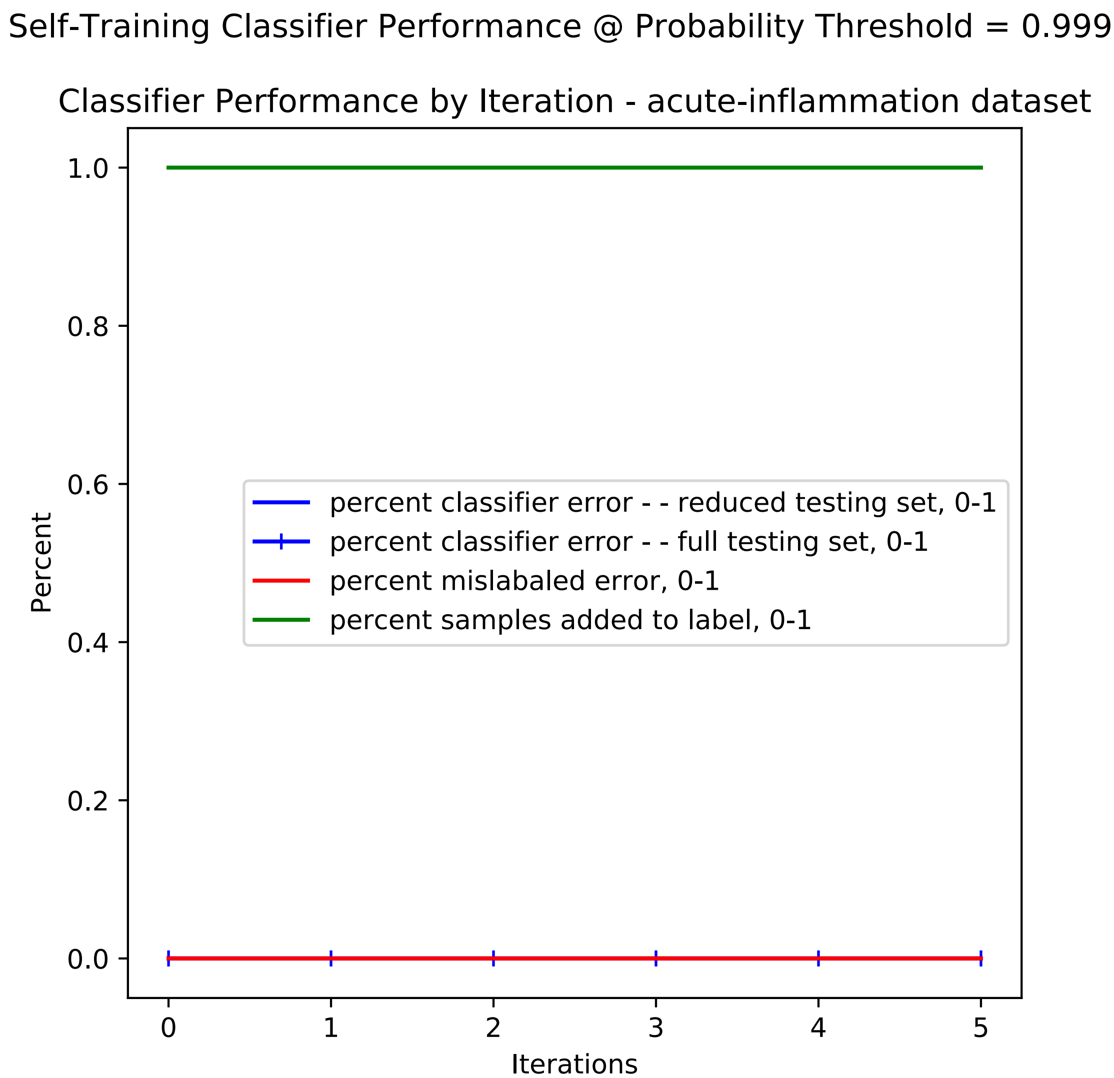

```
acute-inflammation
```



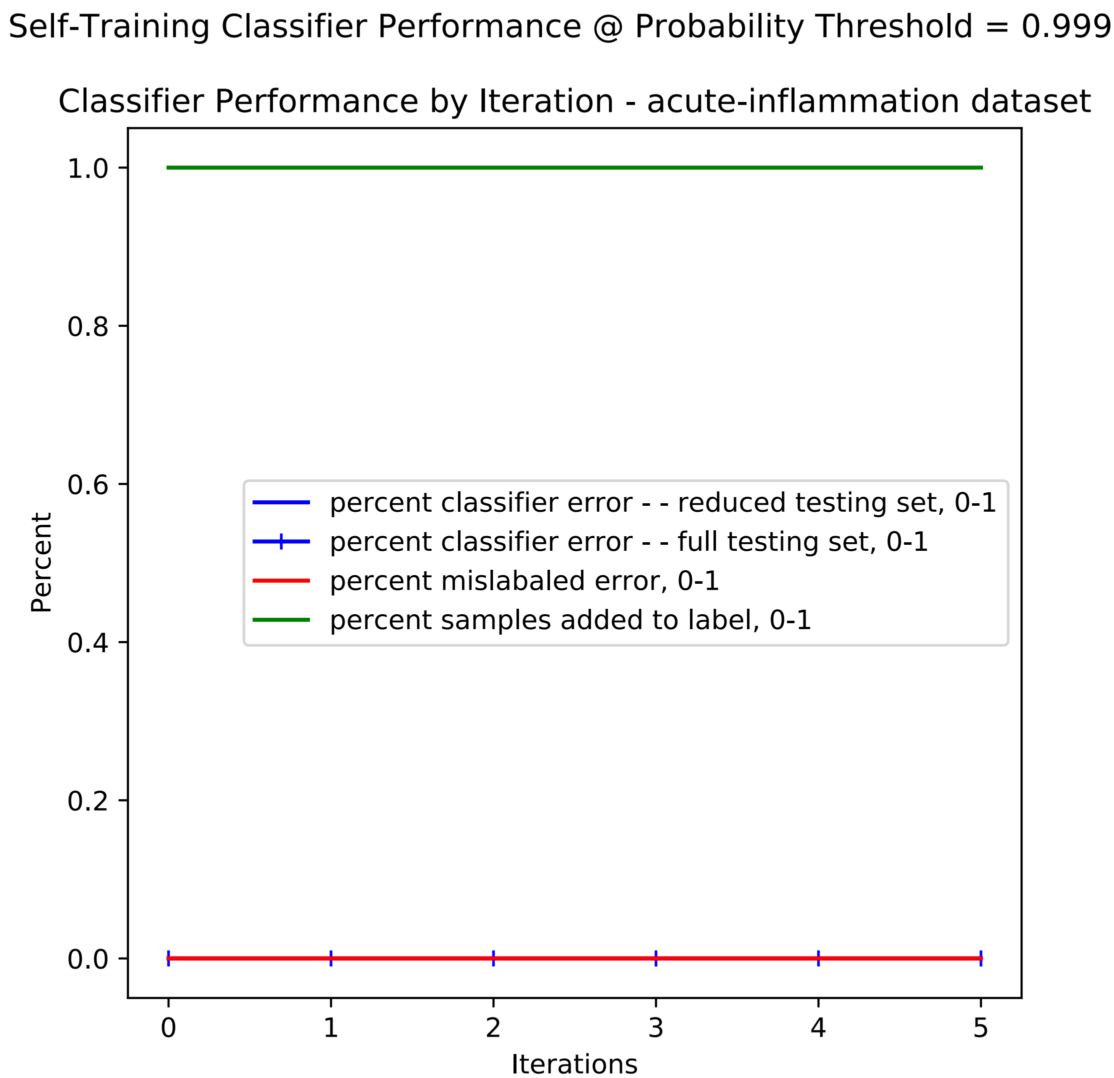

```
acute-inflammation
```



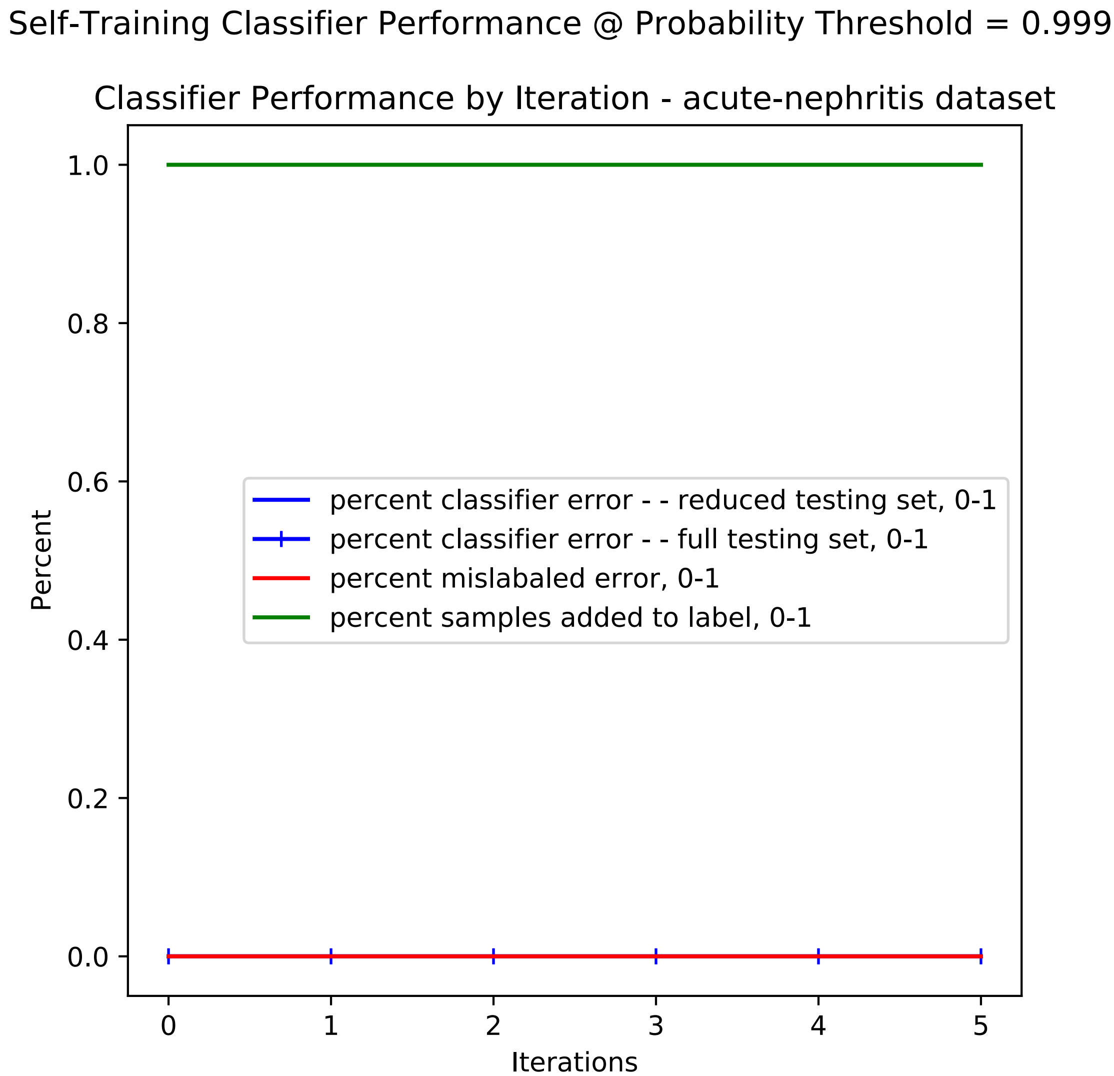

```
acute-inflammation
```




```
acute-inflammation
```

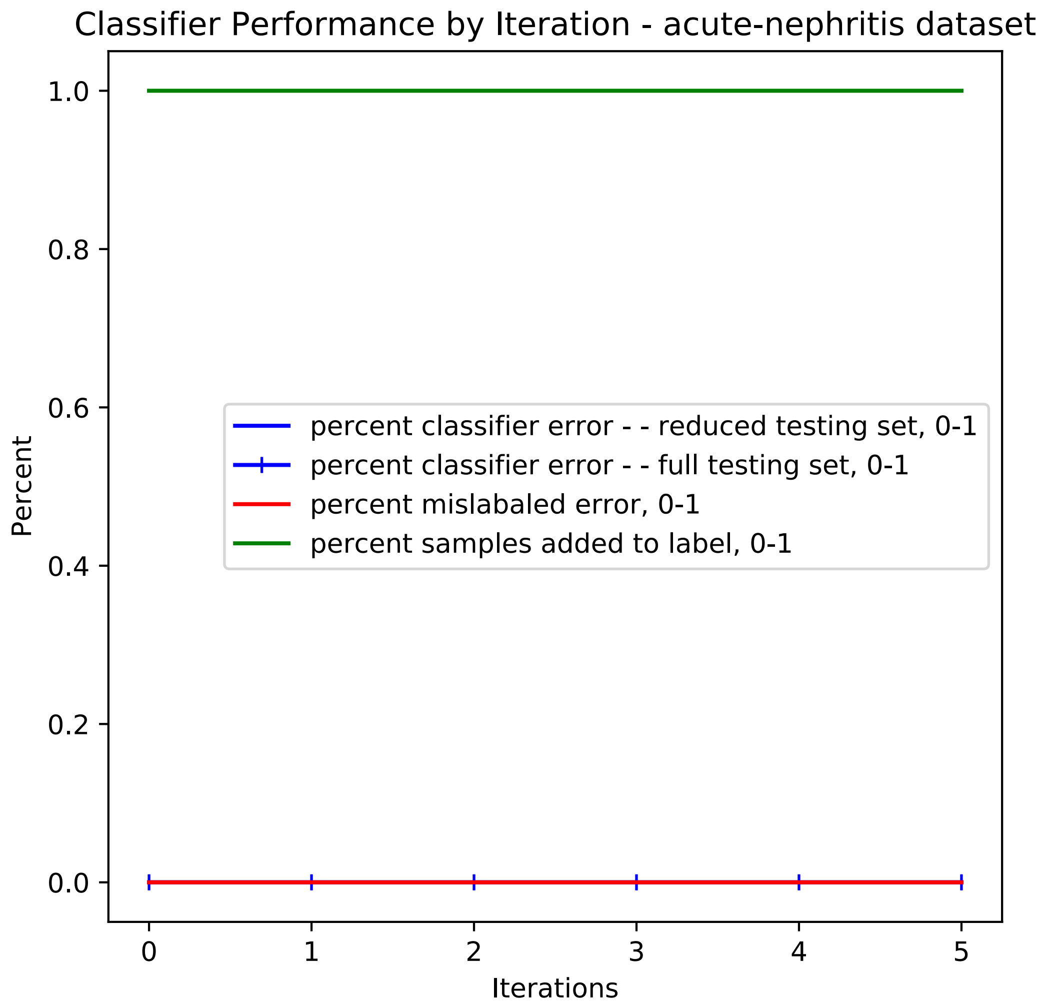



```
acute-nephritis
```

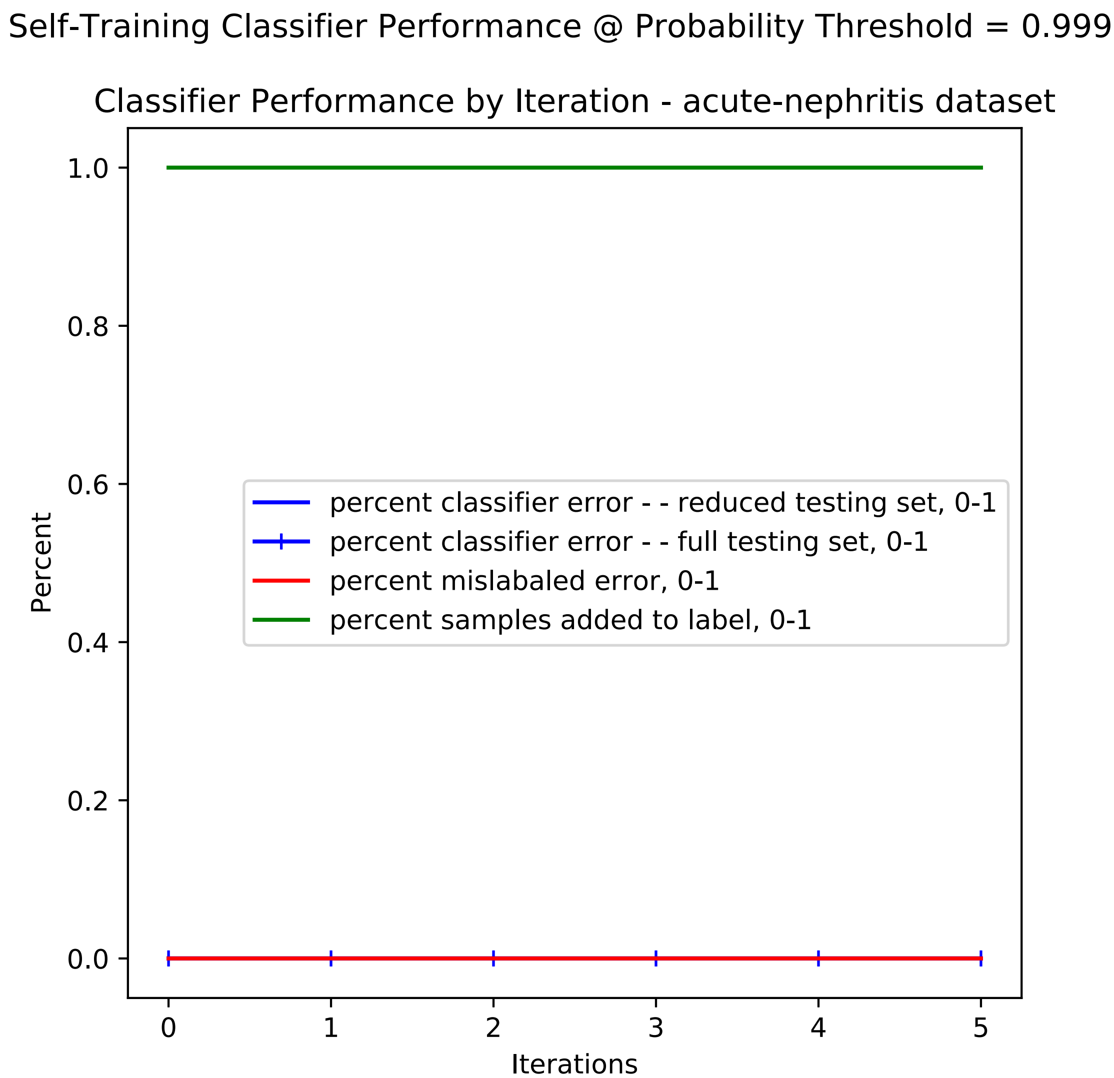



```
acute-nephritis
```

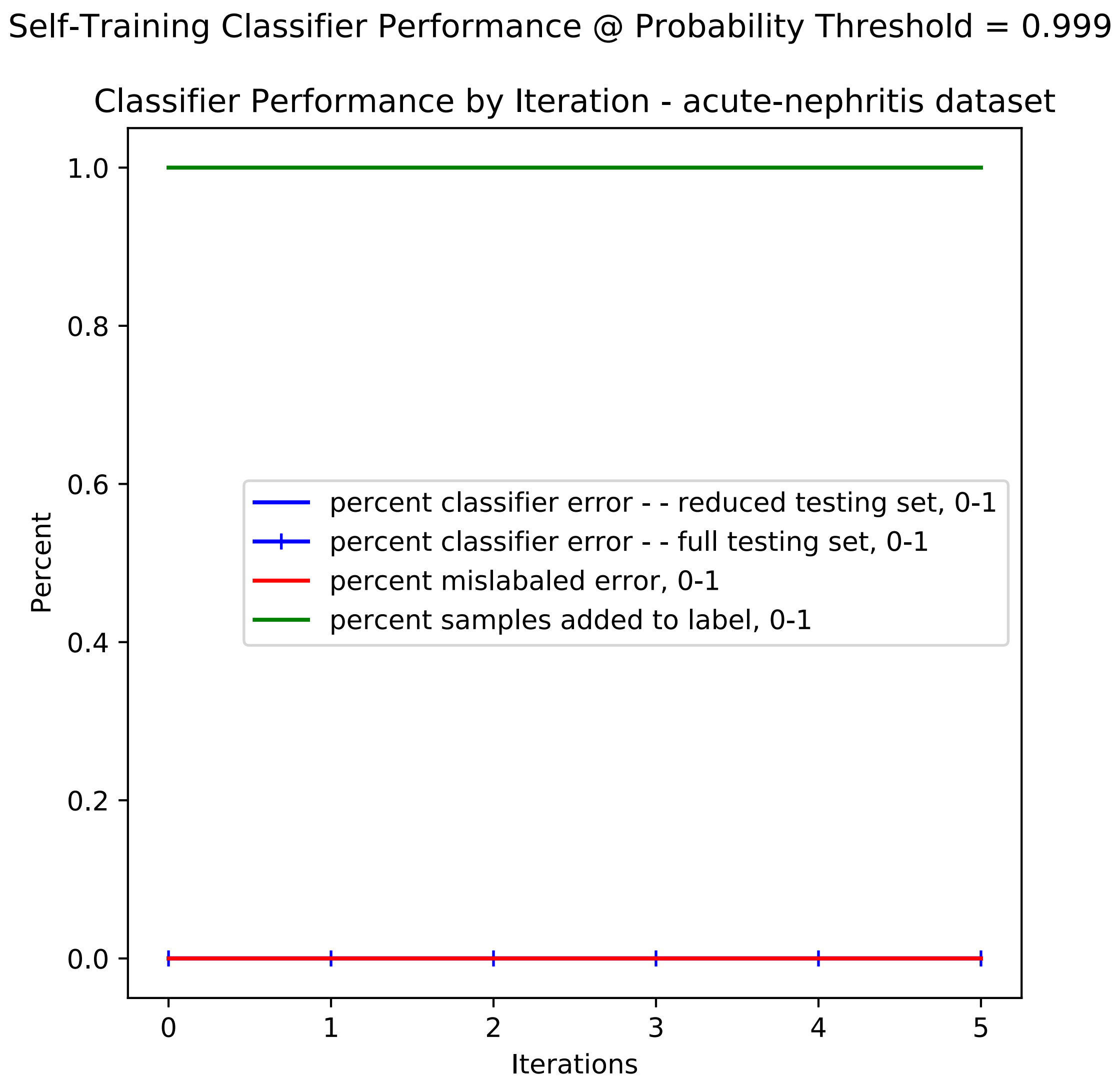
Self-Training Classifier Performance @ Probability Threshold = 0.999



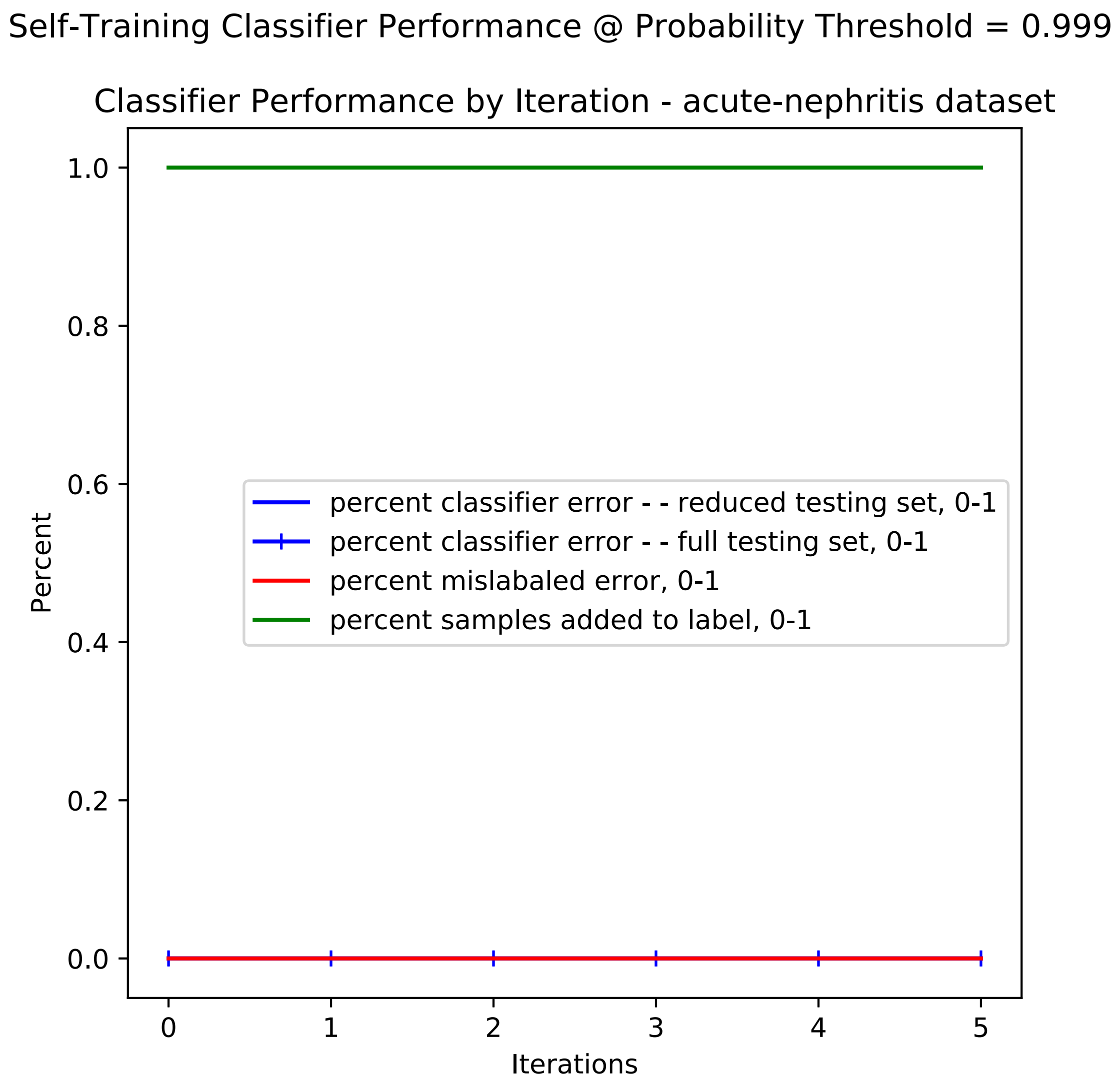

```
acute-nephritis
```



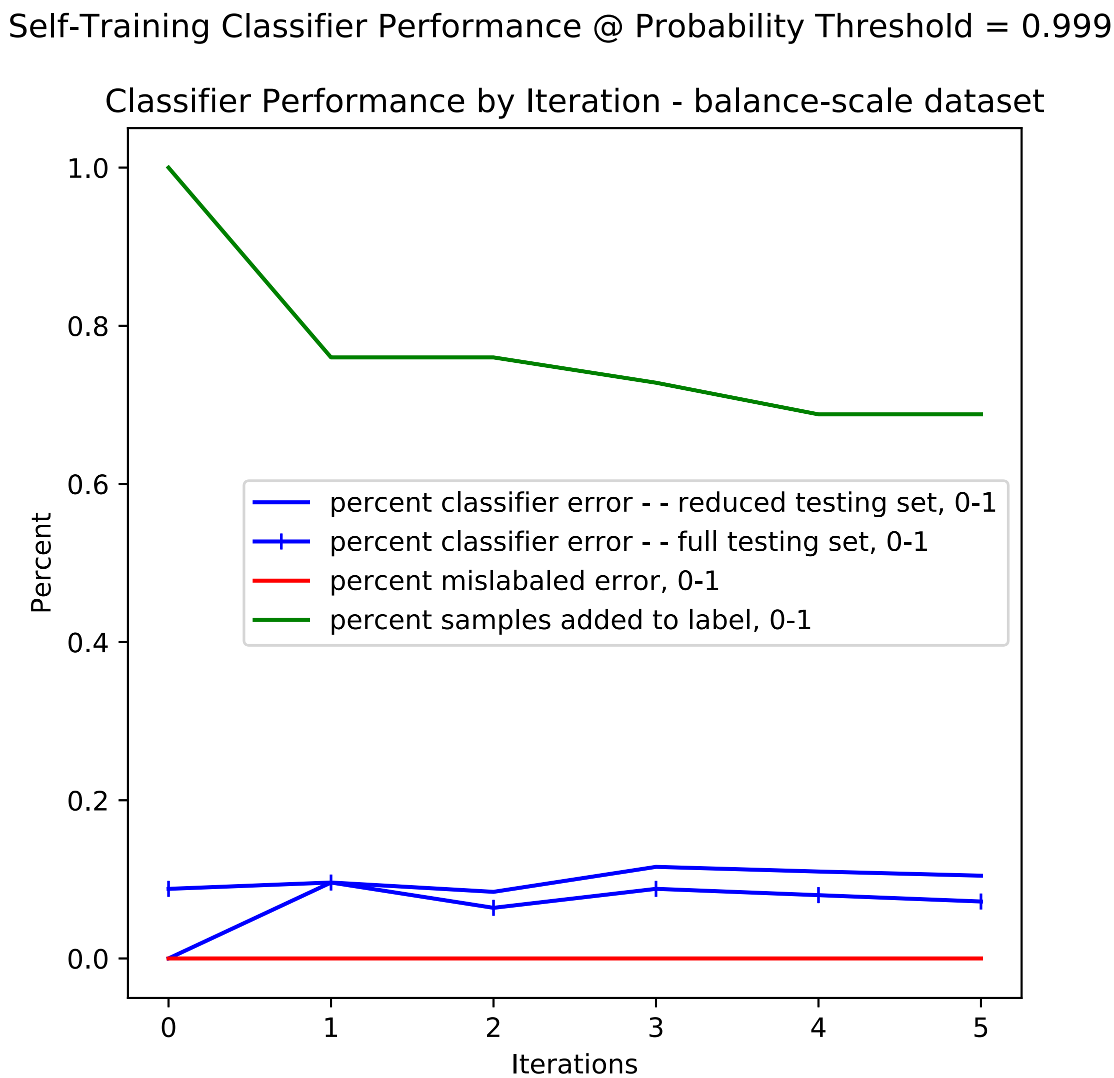

```
acute-nephritis
```



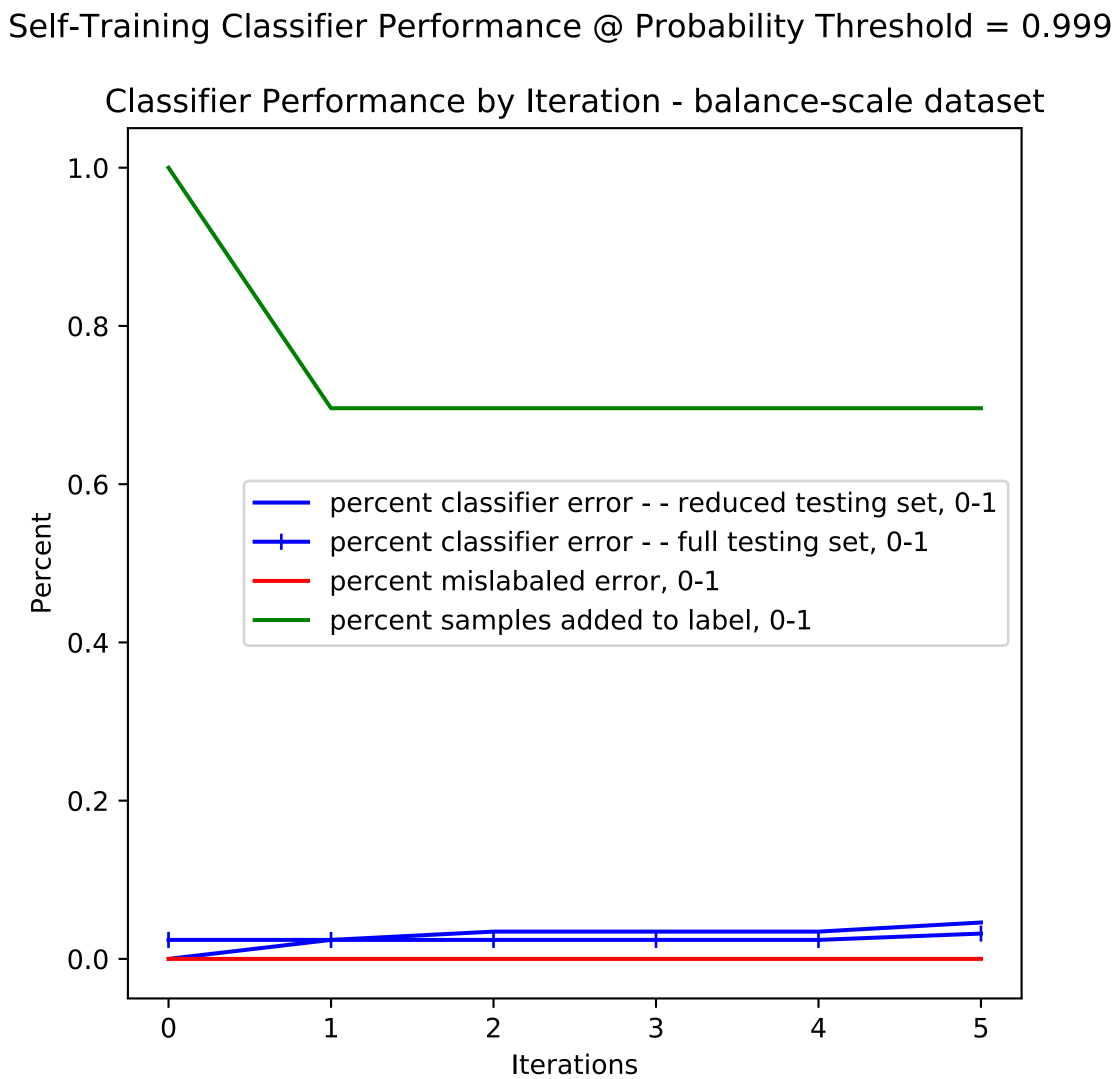

```
acute-nephritis
```



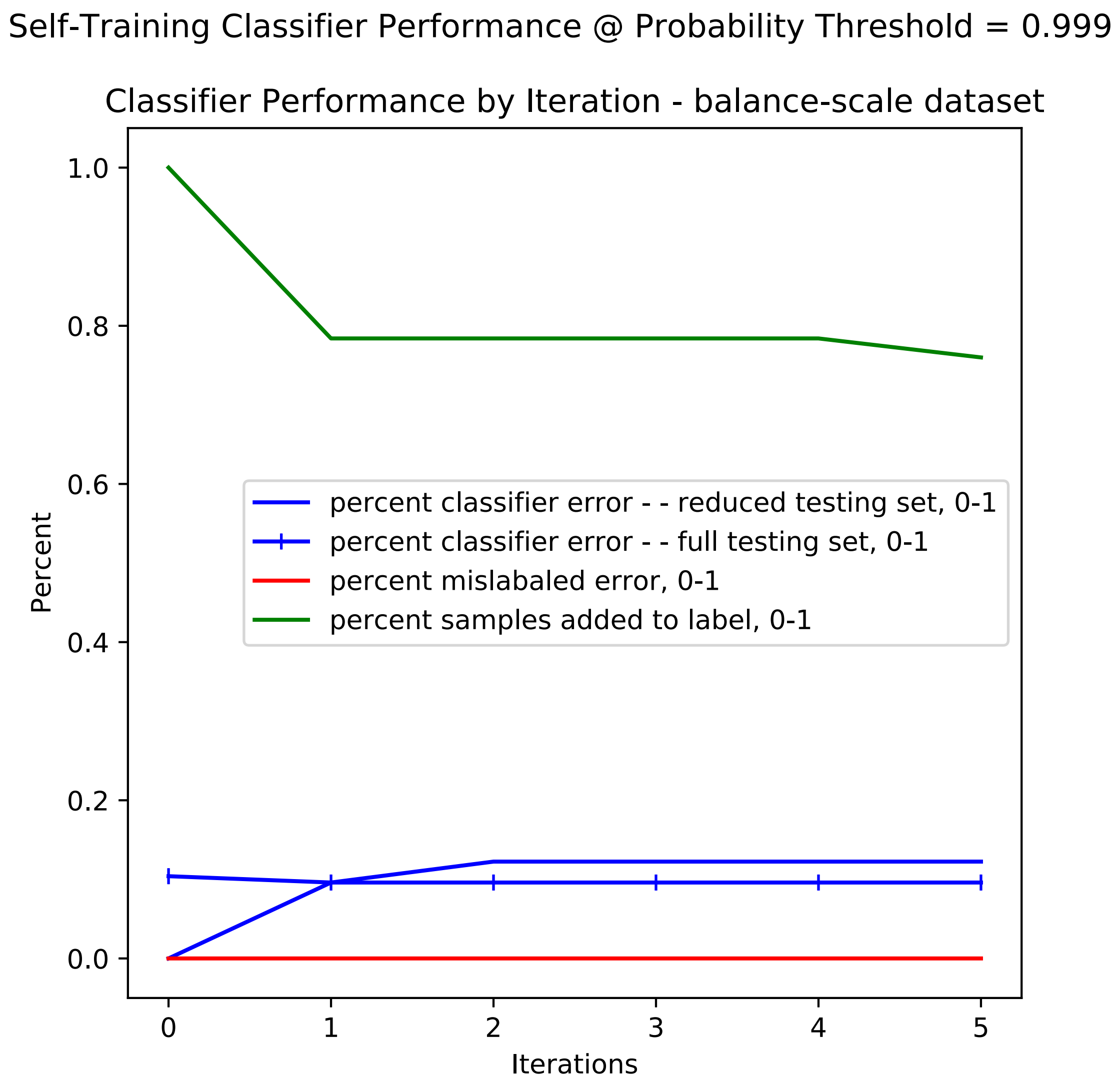

```
balance-scale
```




```
balance-scale
```

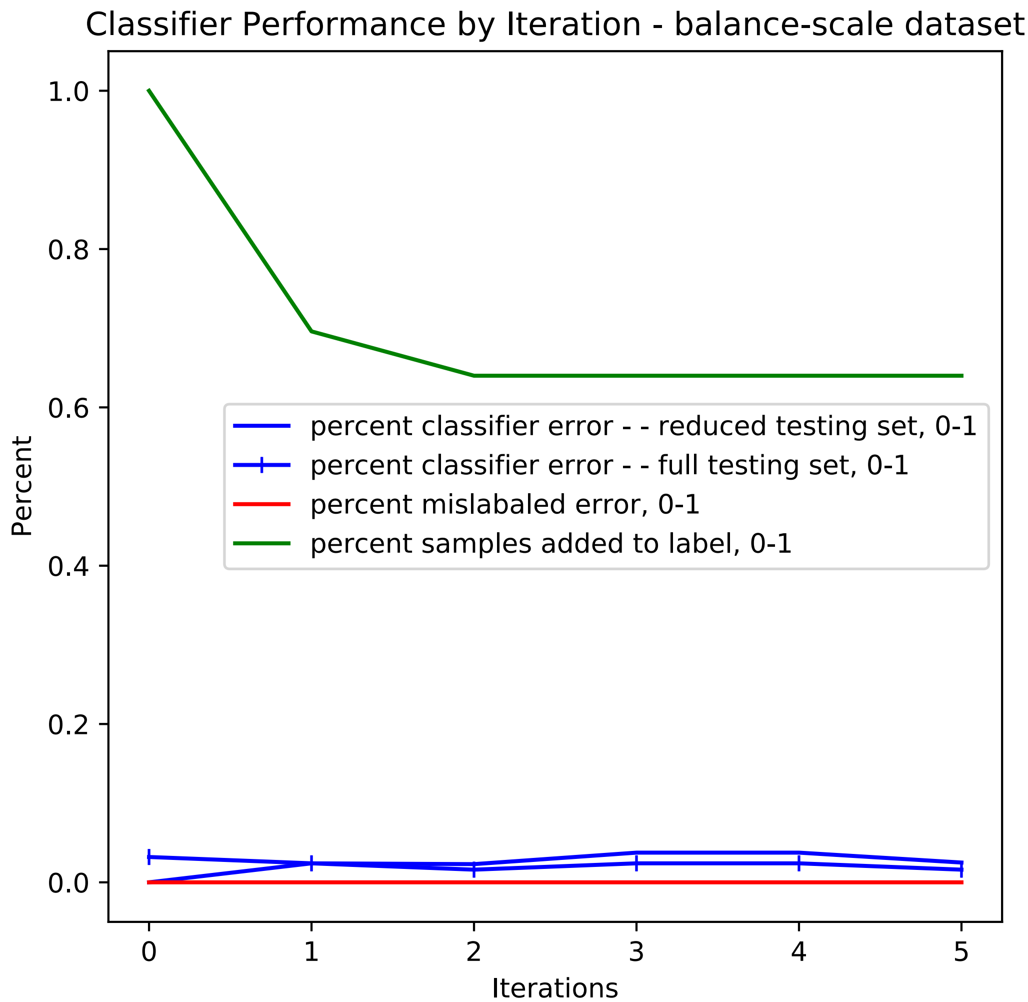



```
balance-scale
```

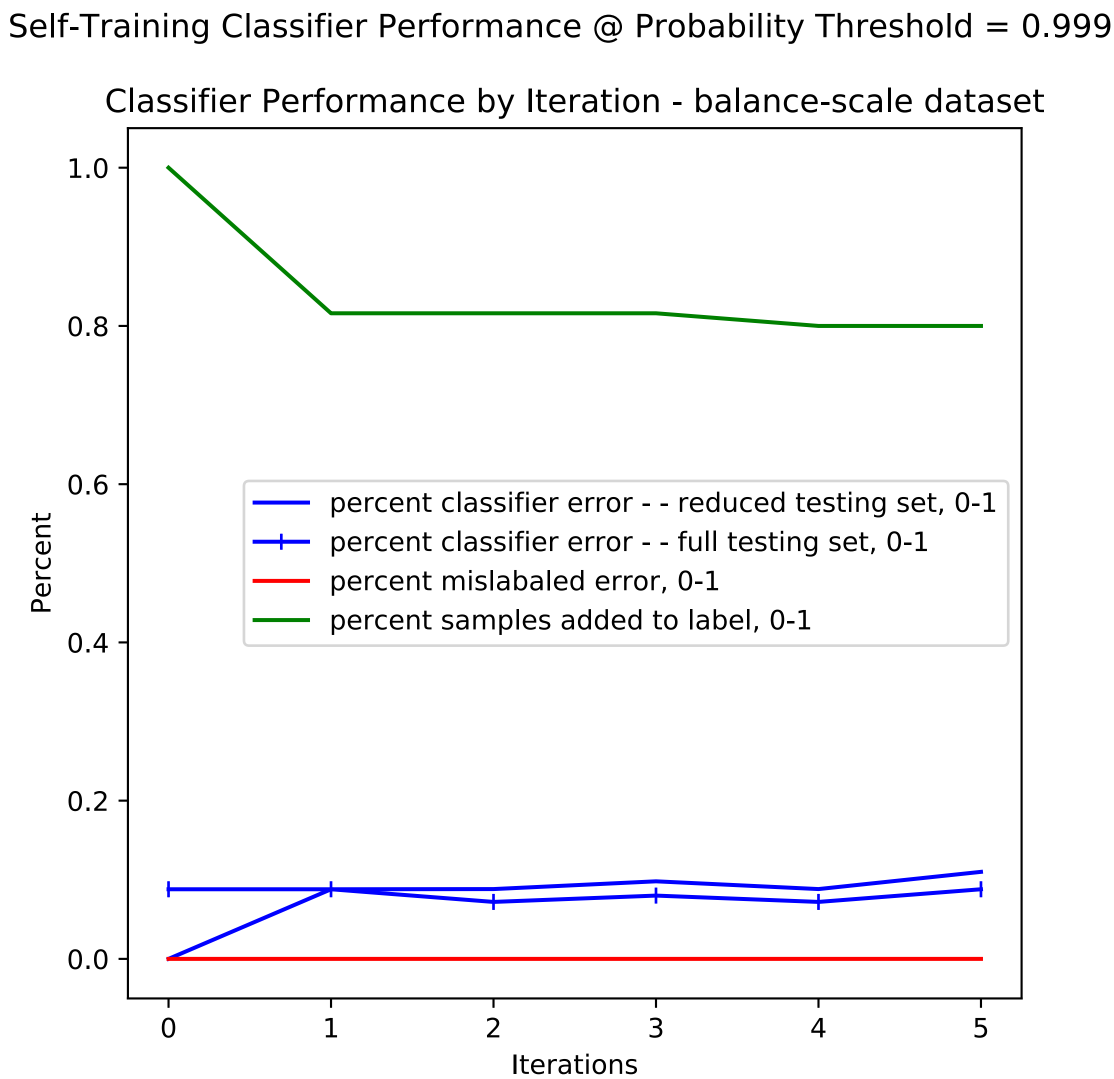



```
balance-scale
```

Self-Training Classifier Performance @ Probability Threshold = 0.999

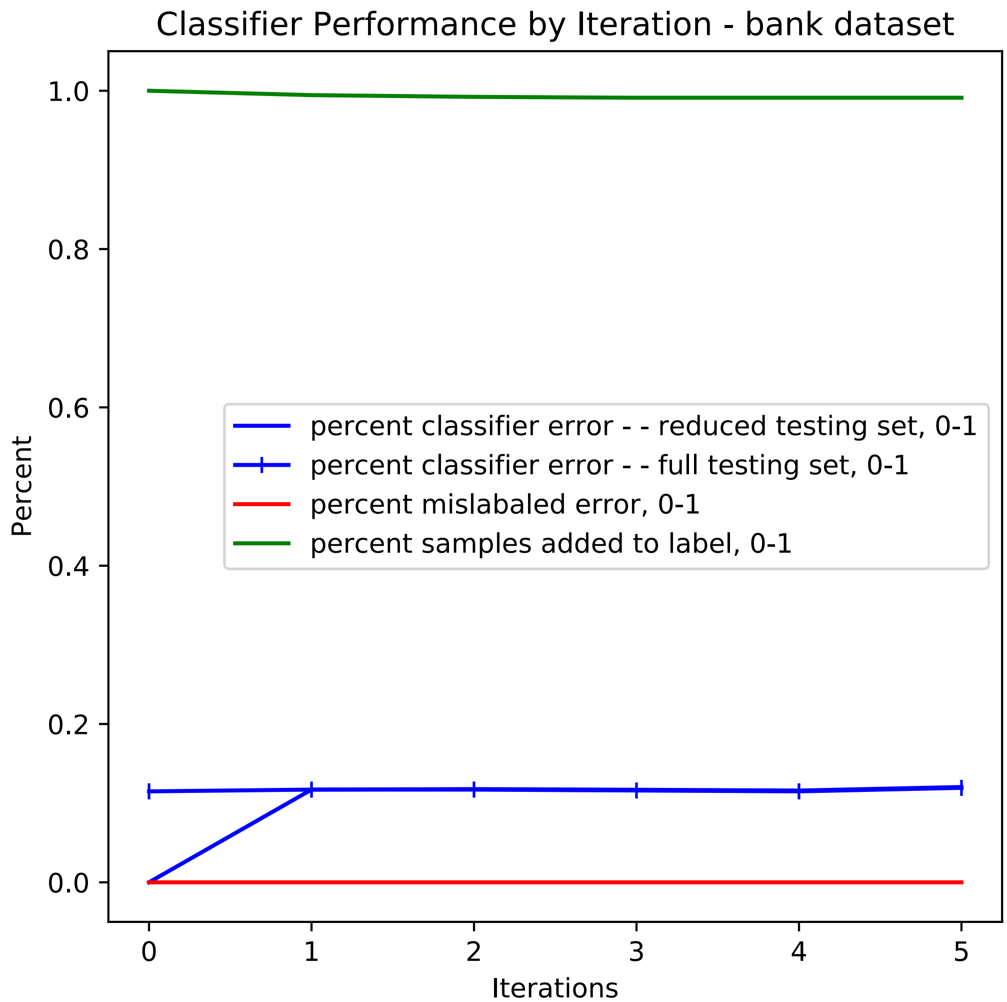



```
balance-scale
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - balance-scale dataset
1.0
0.8
0.6
0.4
0.2
0.0
Percent
0
1
2
3
4
5
Iterations
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1

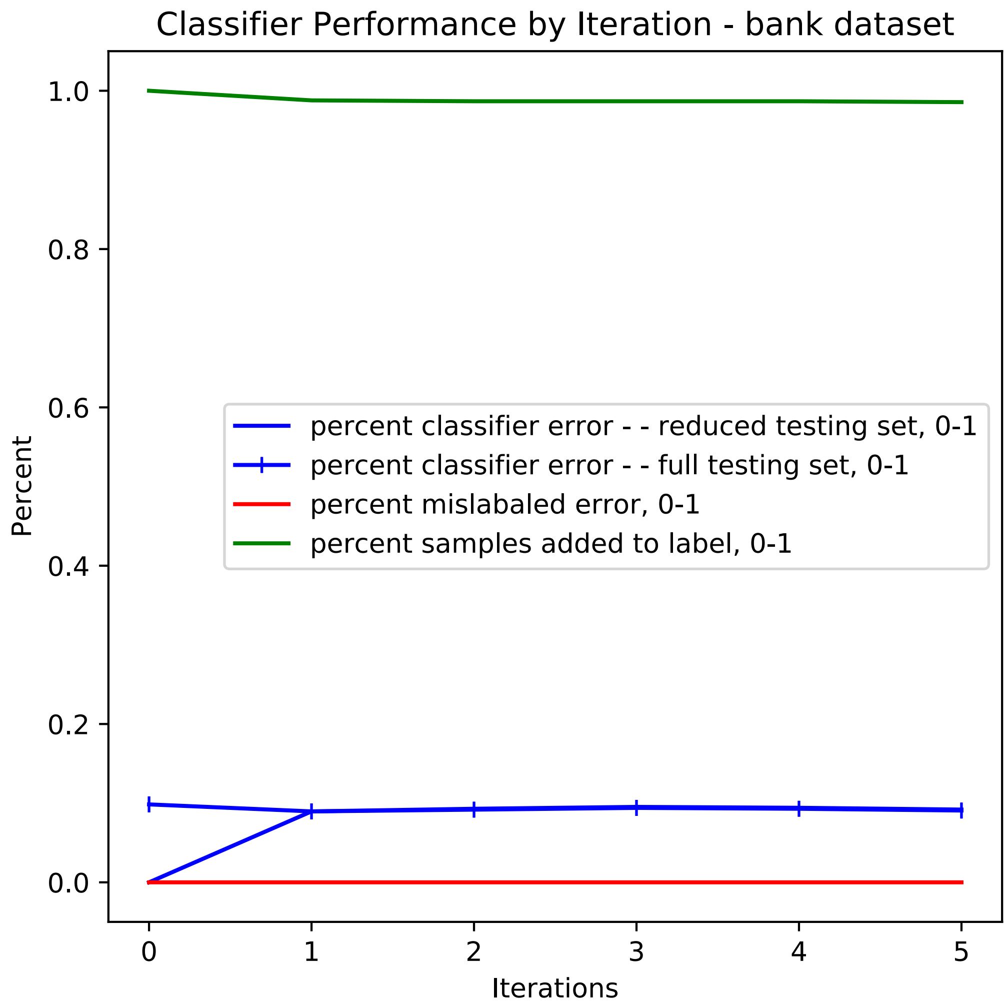
```
bank
```

Self-Training Classifier Performance @ Probability Threshold = 0.999

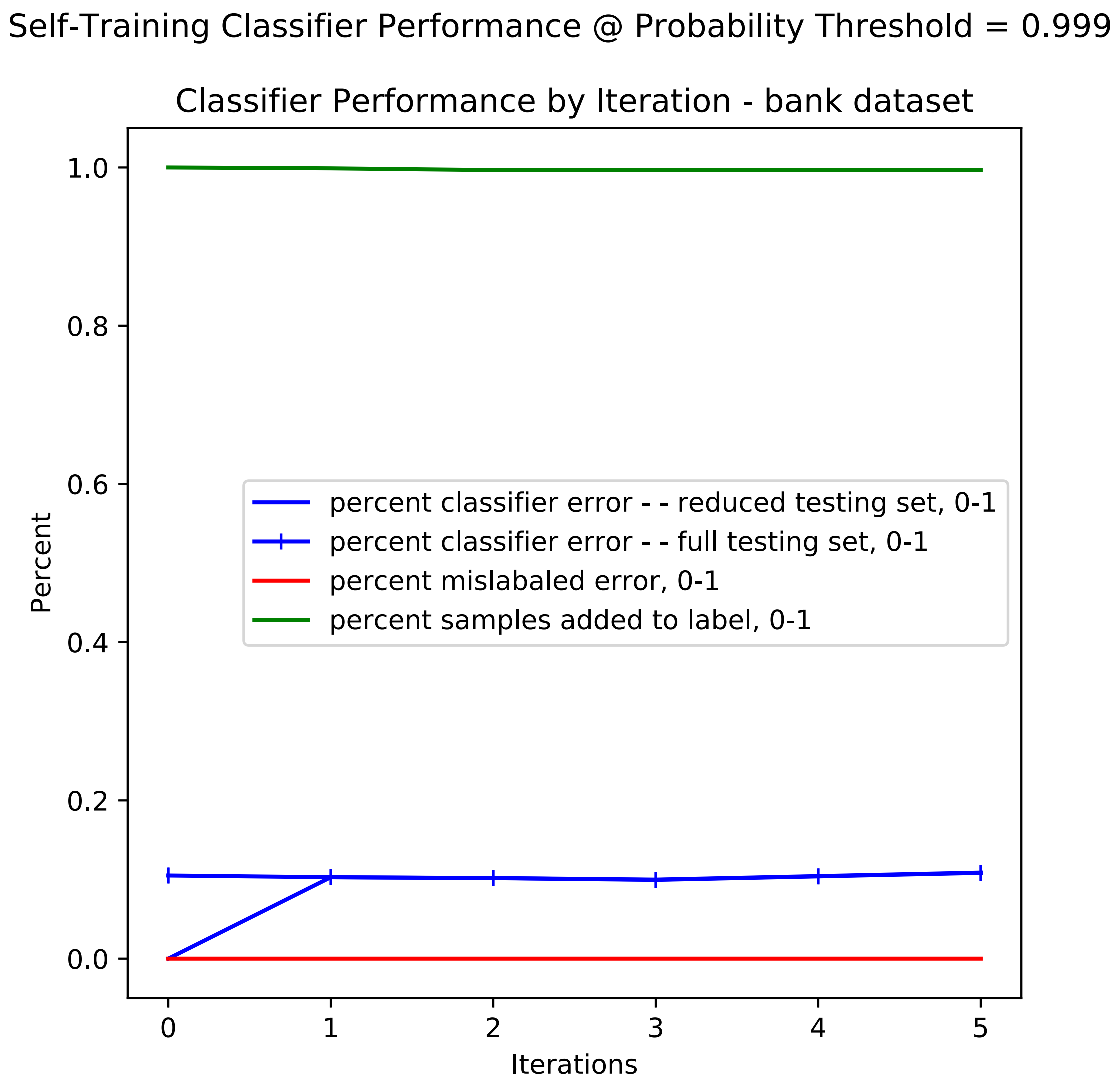



```
bank
```

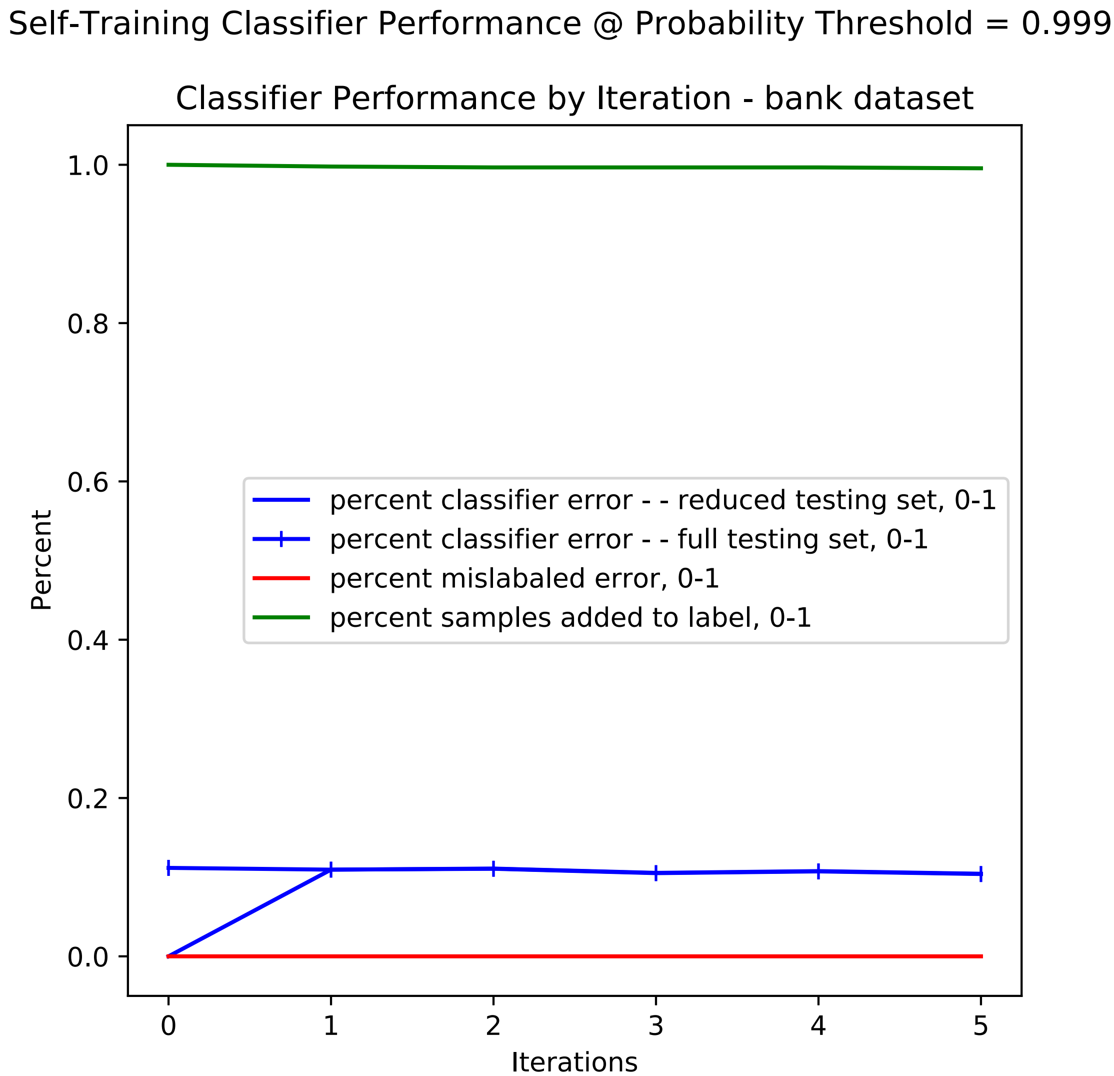
Self-Training Classifier Performance @ Probability Threshold = 0.999




```
bank
```

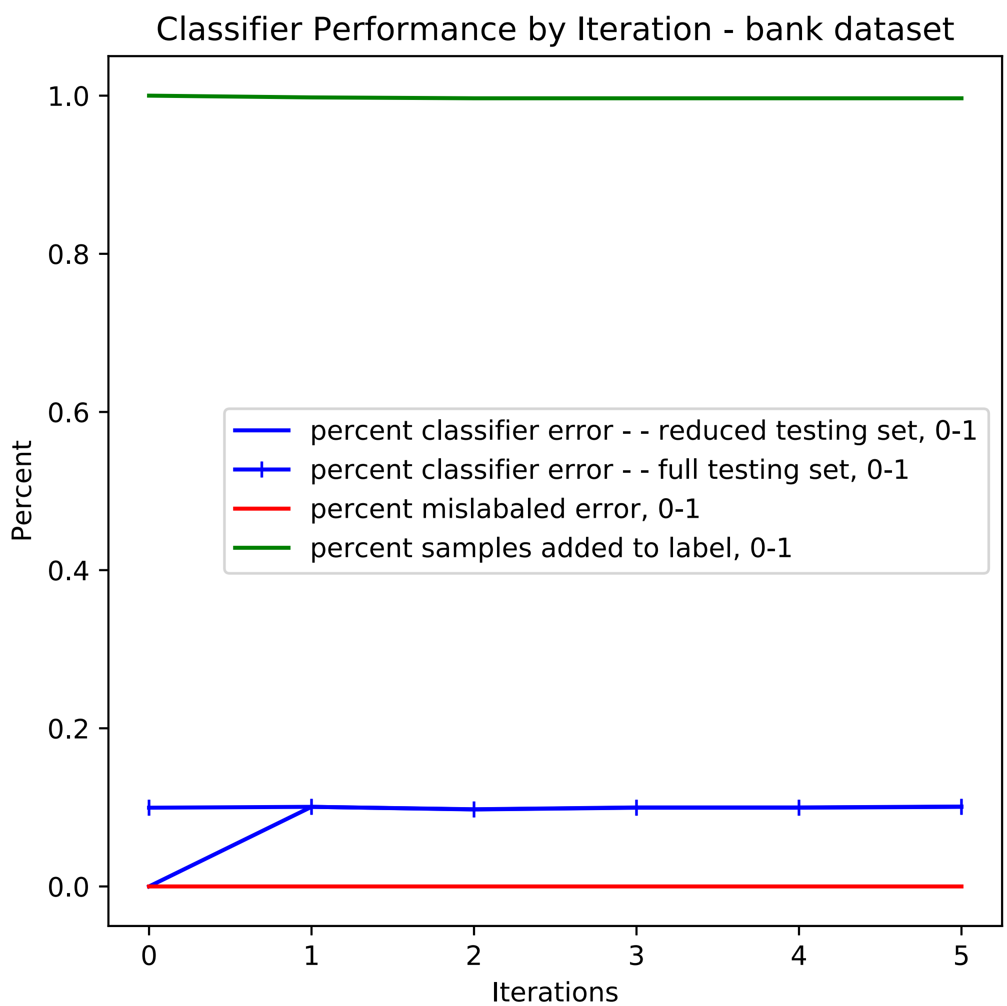

Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - bank dataset
1.0
0.8
0.6
0.4
0.2
0.0
Percent
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
0
1
2
3
4
5
Iterations

```
bank
```

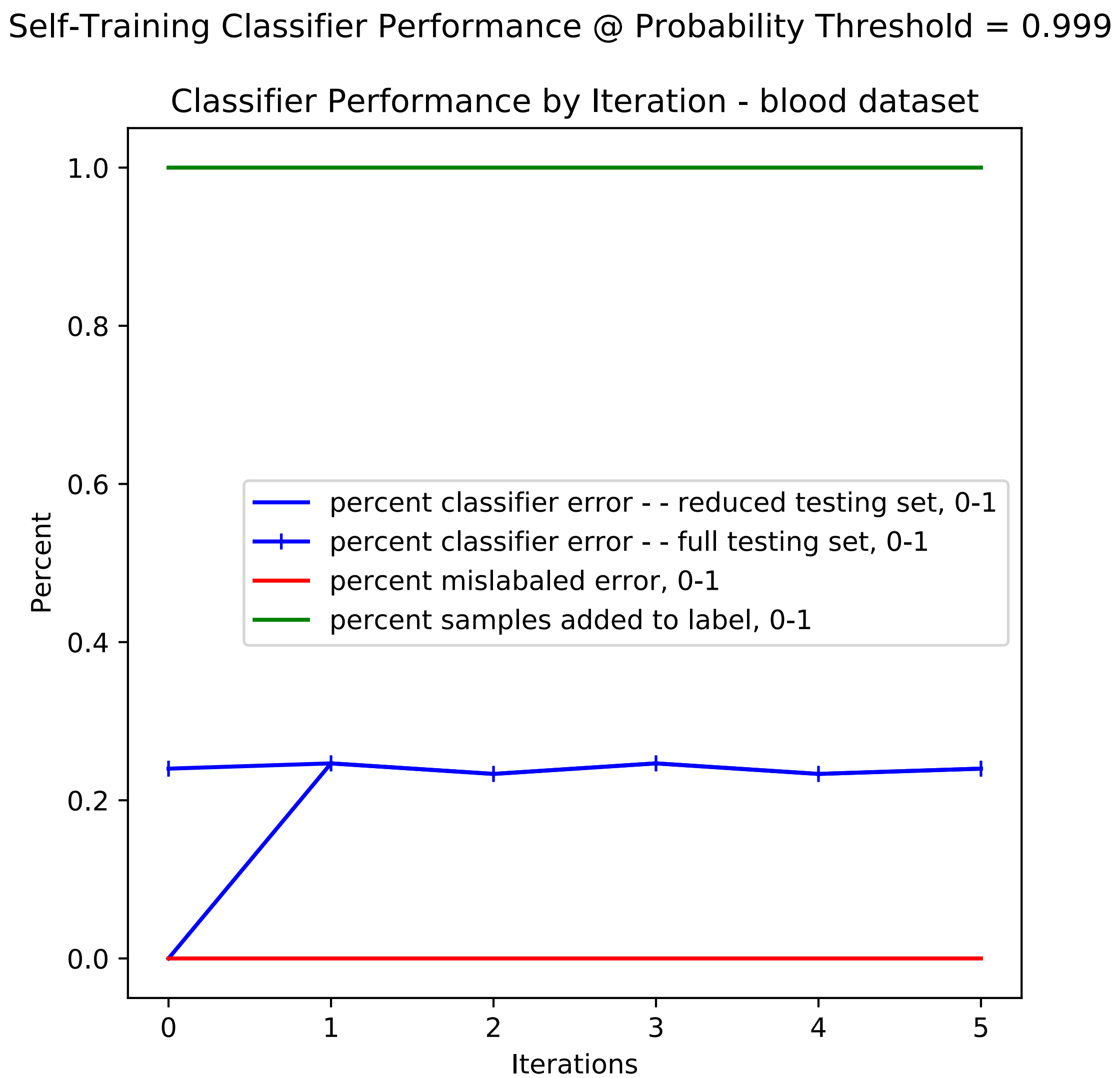

Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - bank dataset
1.0
0.8
0.6
0.4
0.2
0.0
Percent
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
0
1
2
3
4
5
Iterations

```
bank
```

Self-Training Classifier Performance @ Probability Threshold = 0.999

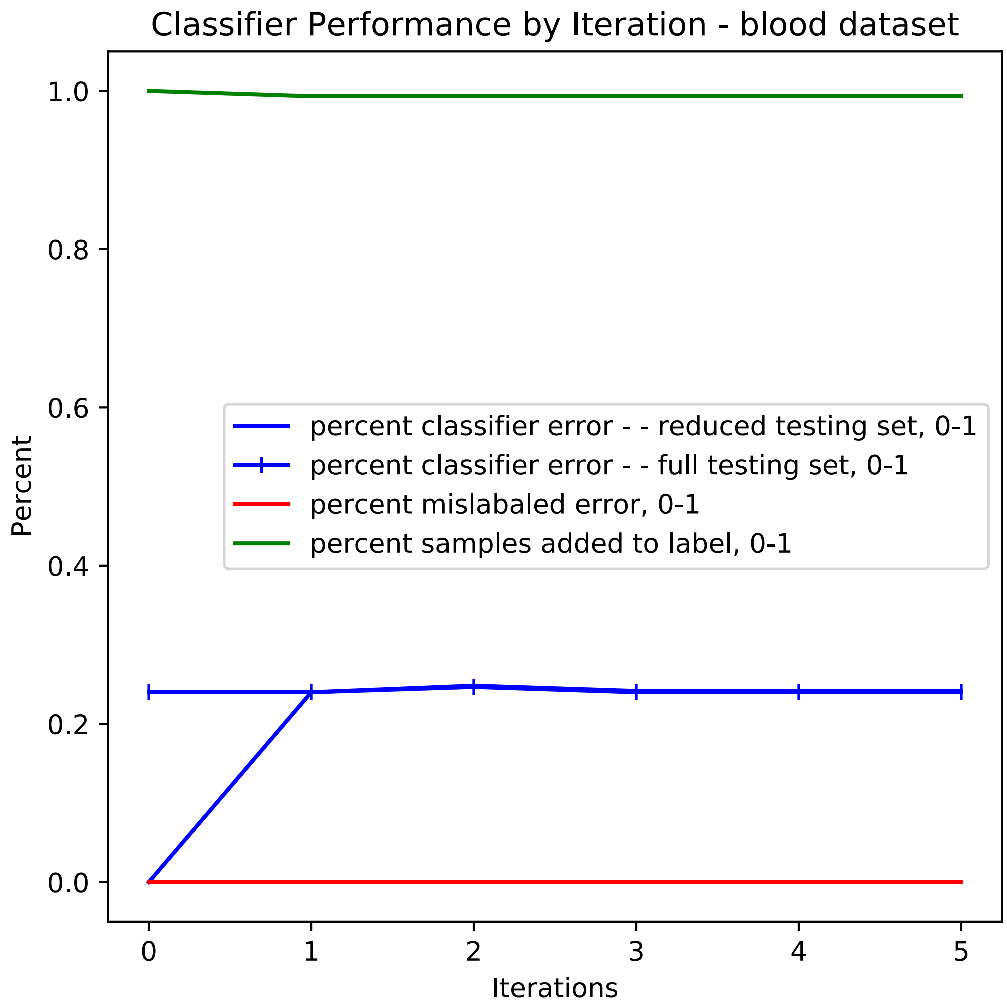



```
blood
```

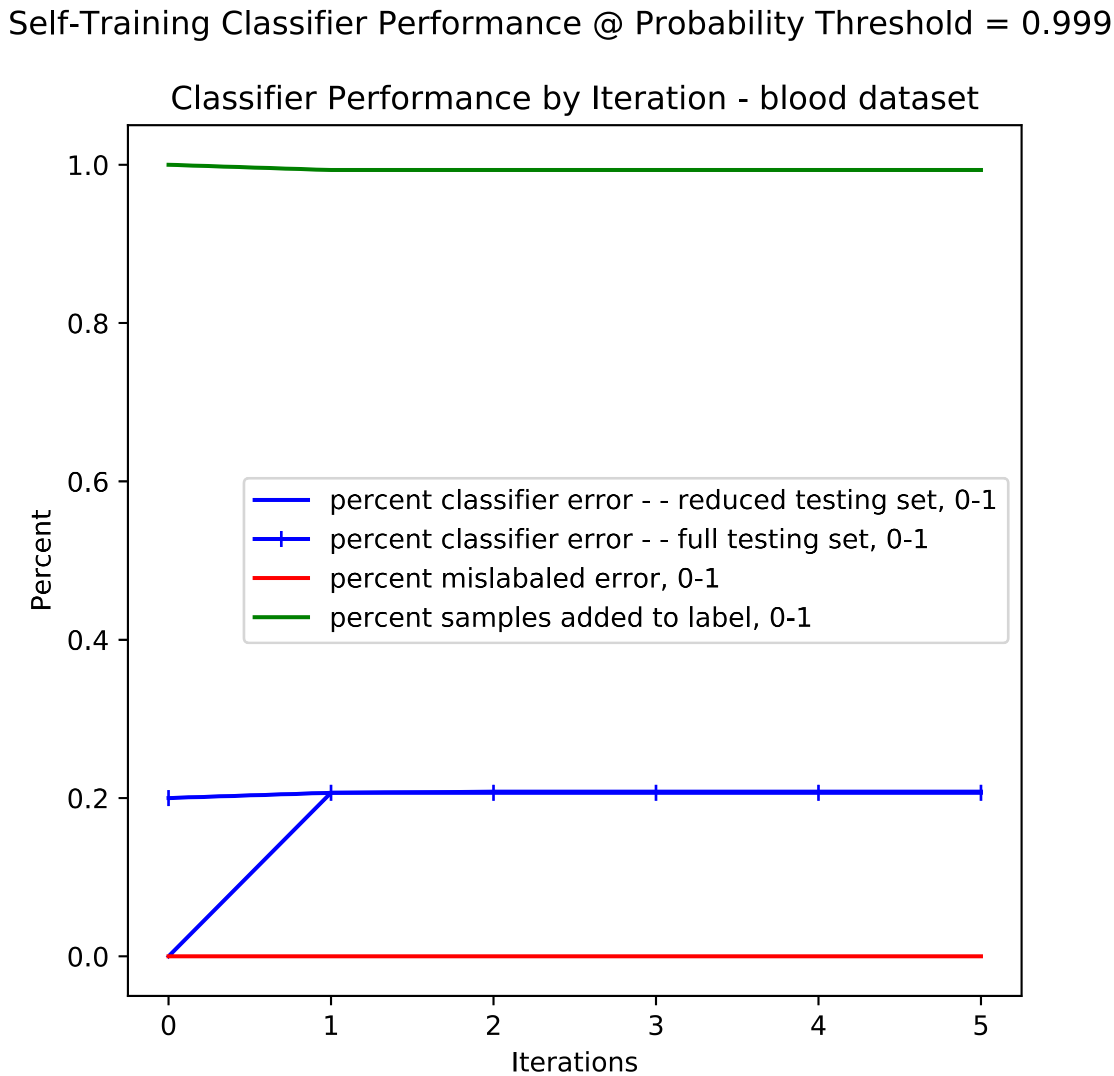

Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - blood dataset
1.0
0.8
0.6
0.4
0.2
0.0
Percent
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
0
1
2
3
4
5
Iterations

```
blood
```

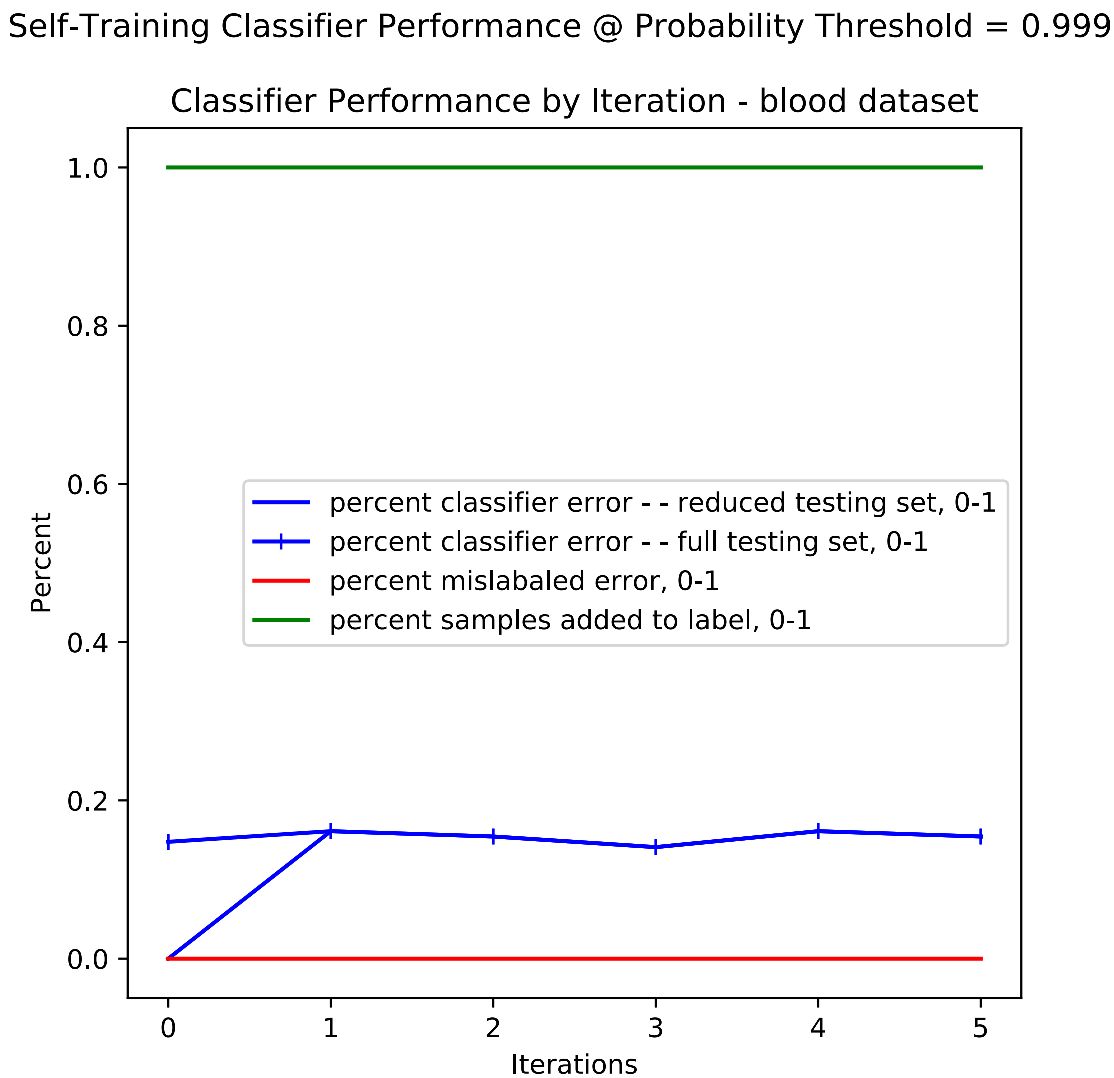
Self-Training Classifier Performance @ Probability Threshold = 0.999



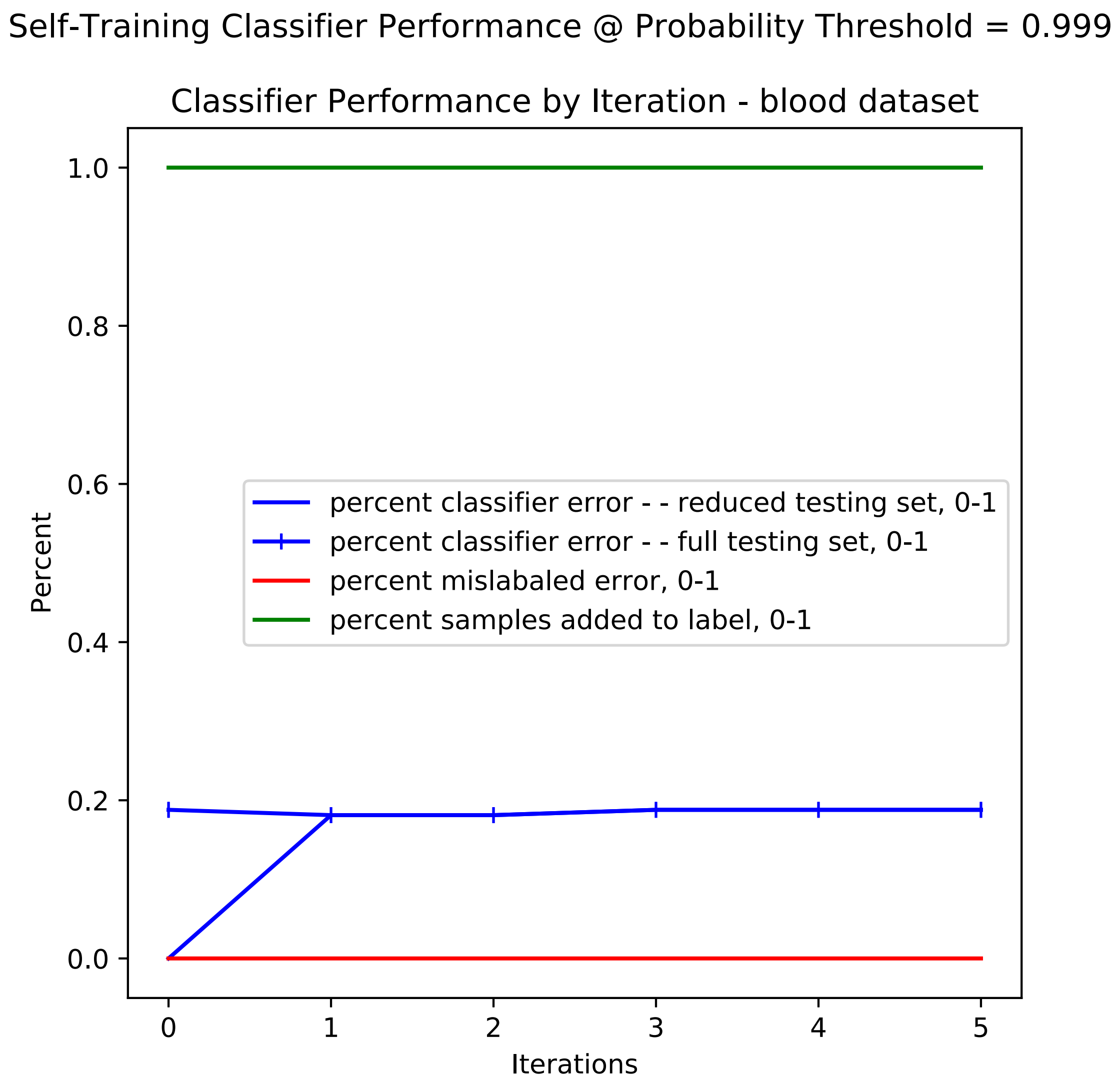

```
blood
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - blood dataset
1.0
0.8
0.6
0.4
0.2
0.0
Percent
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
0
1
2
3
4
5
Iterations

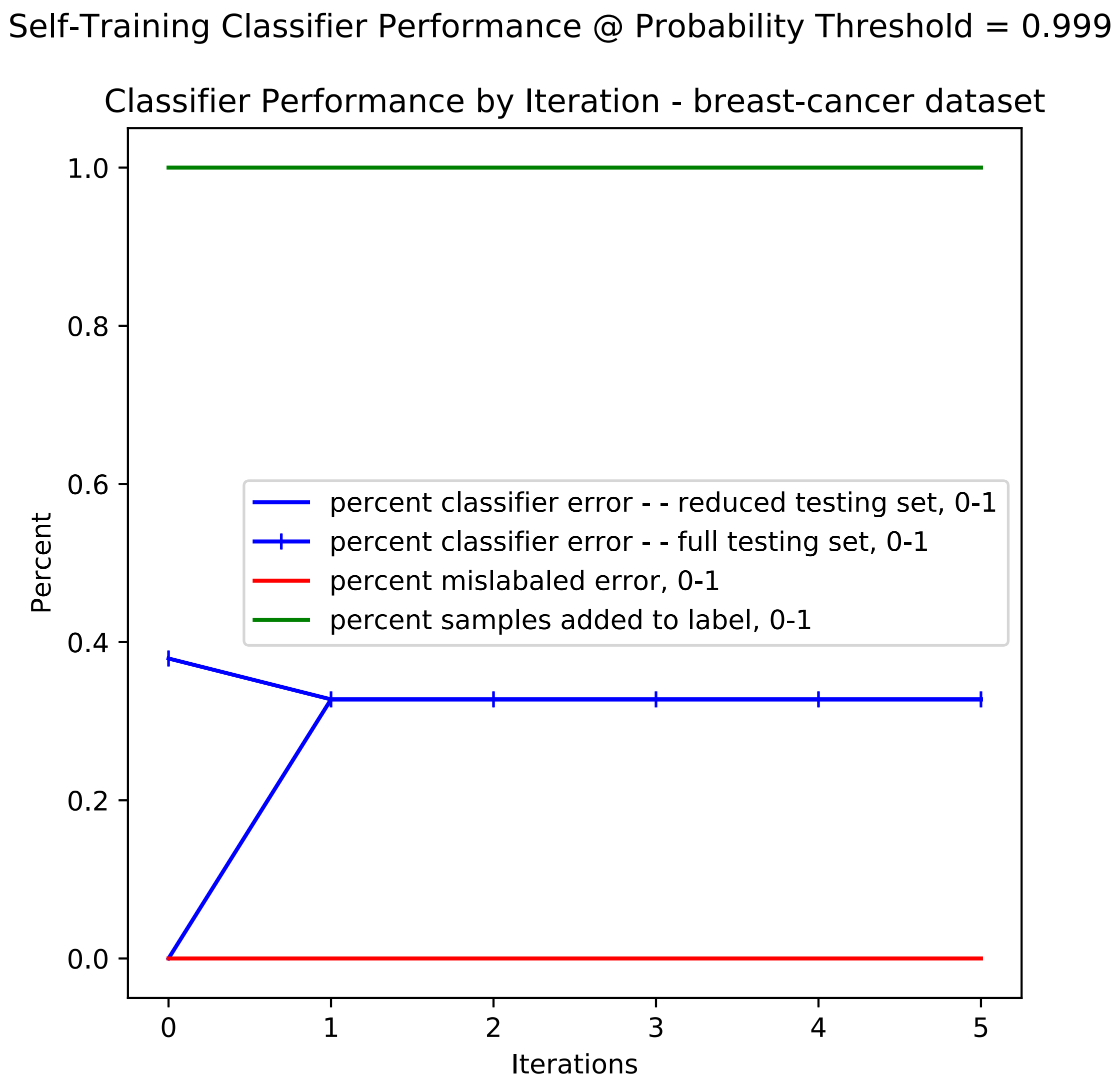
```
blood
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - blood dataset
1.0
0.8
0.6
0.4
0.2
0.0
Percent
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
0
1
2
3
4
5
Iterations

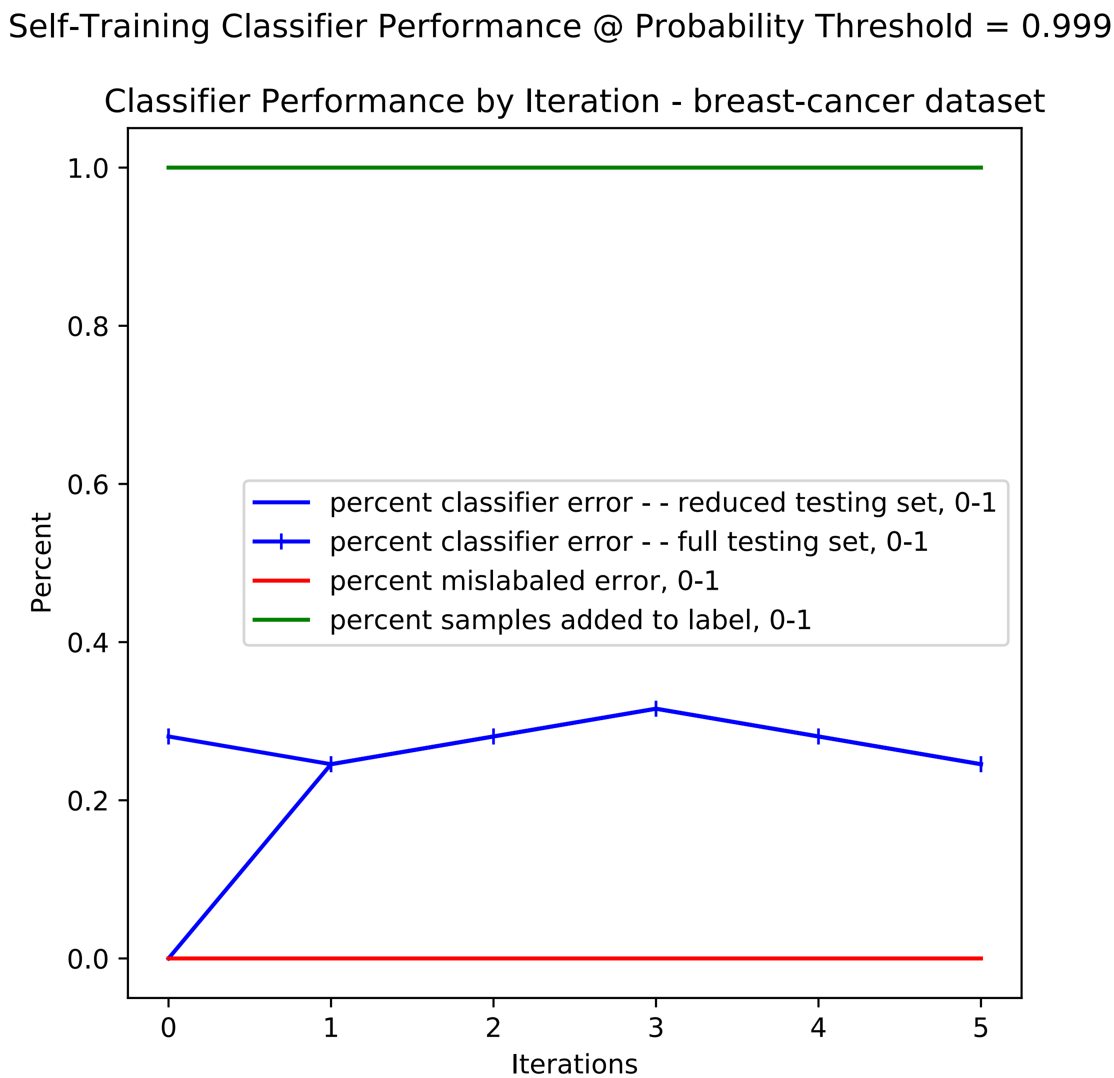
```
blood
```



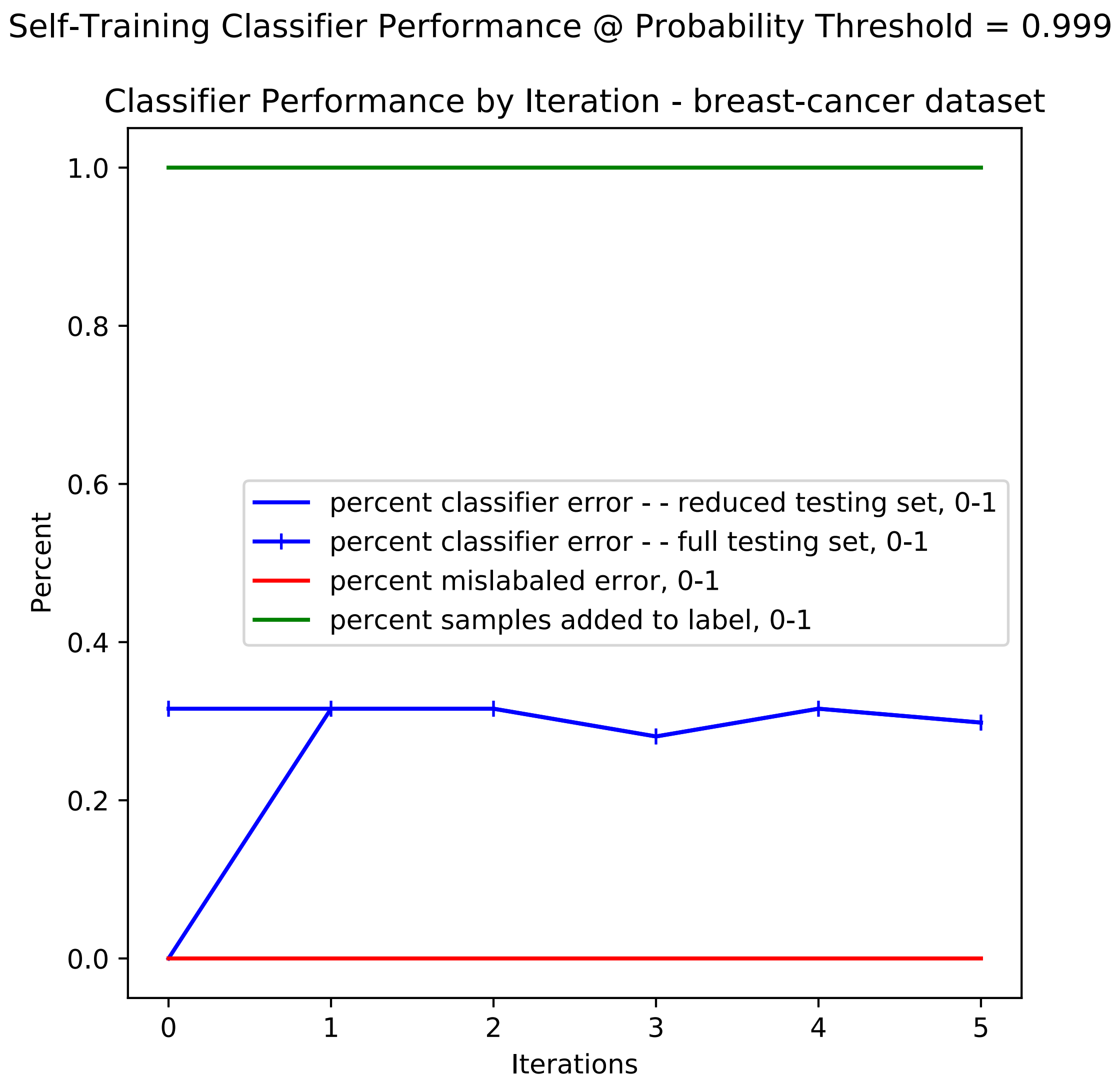

```
breast-cancer
```



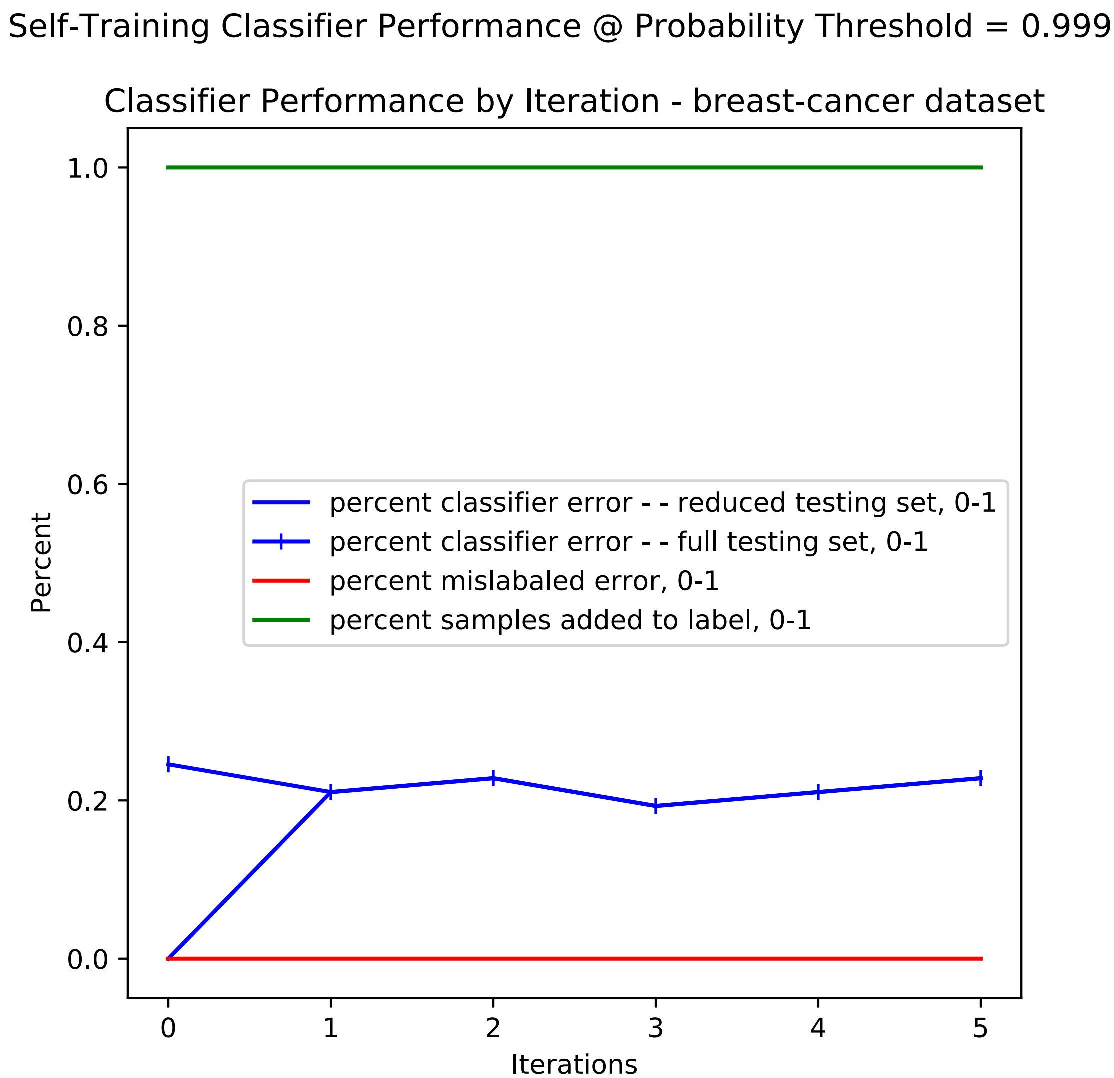

```
breast-cancer
```



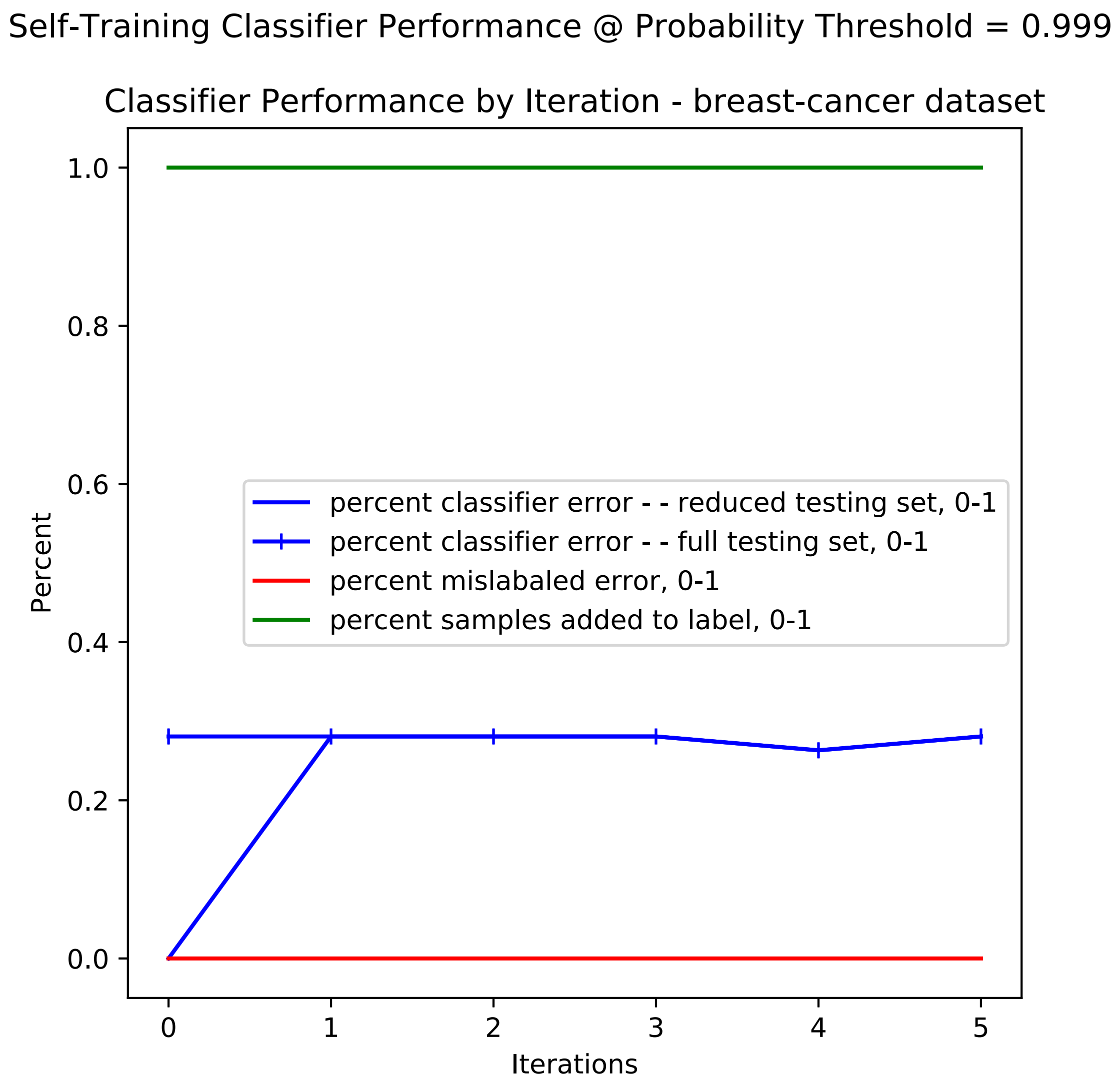

```
breast-cancer
```




```
breast-cancer
```

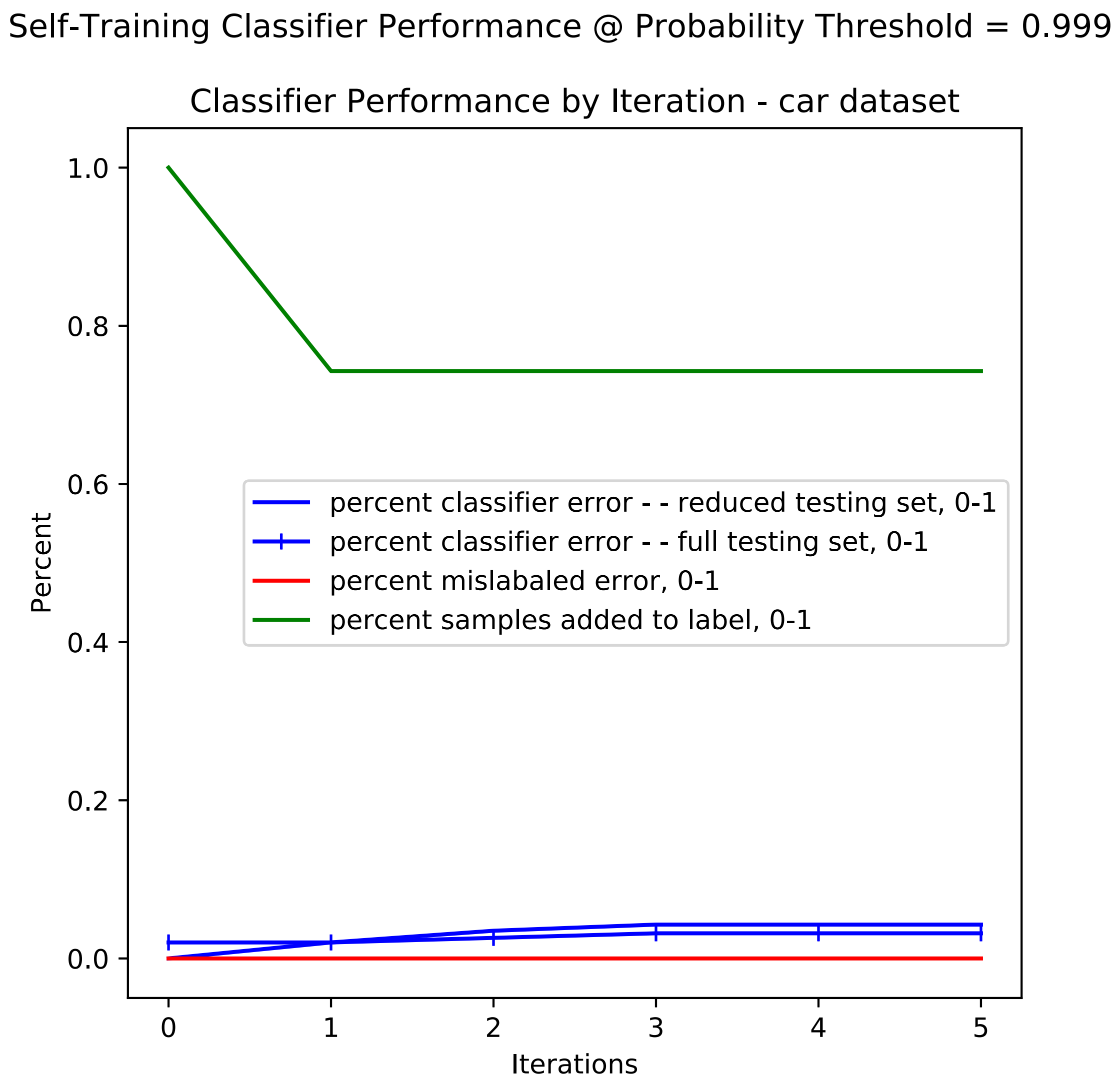



```
breast-cancer
```



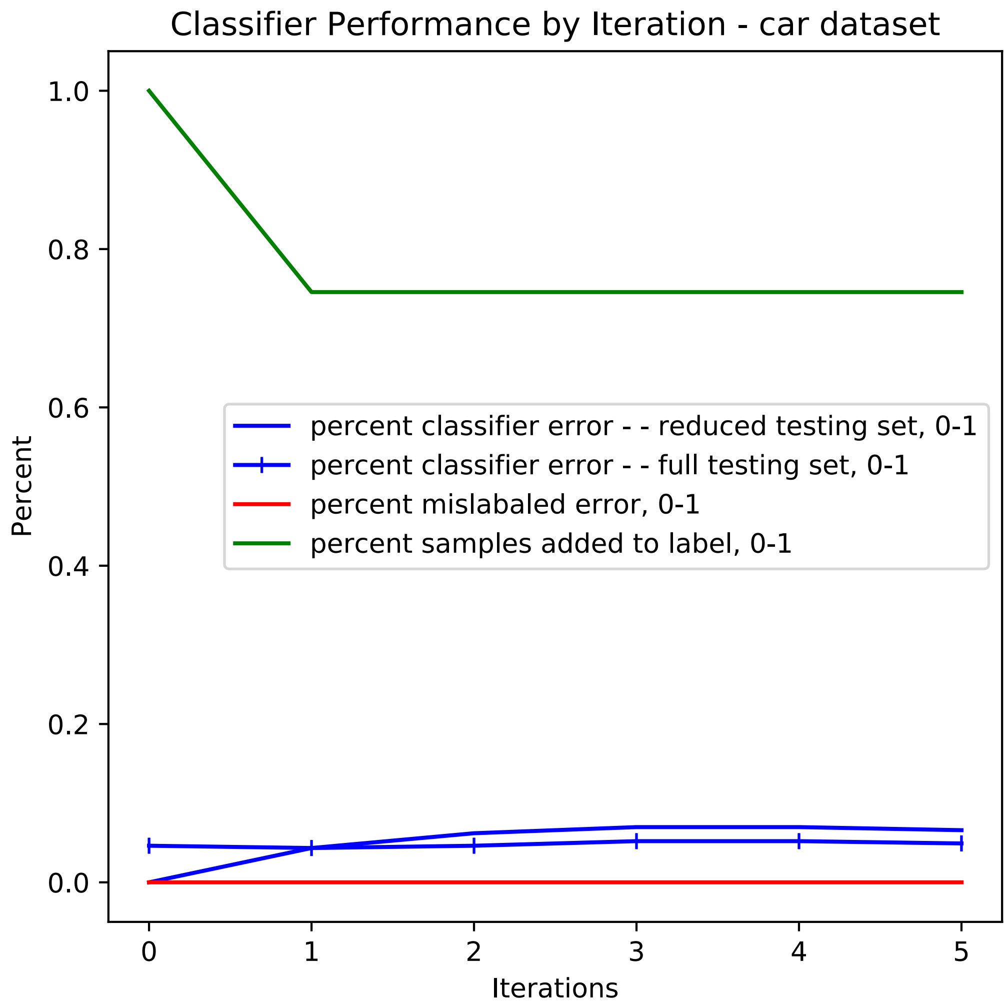
Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - breast-cancer dataset
Percent
Iterations
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5


```
car
```

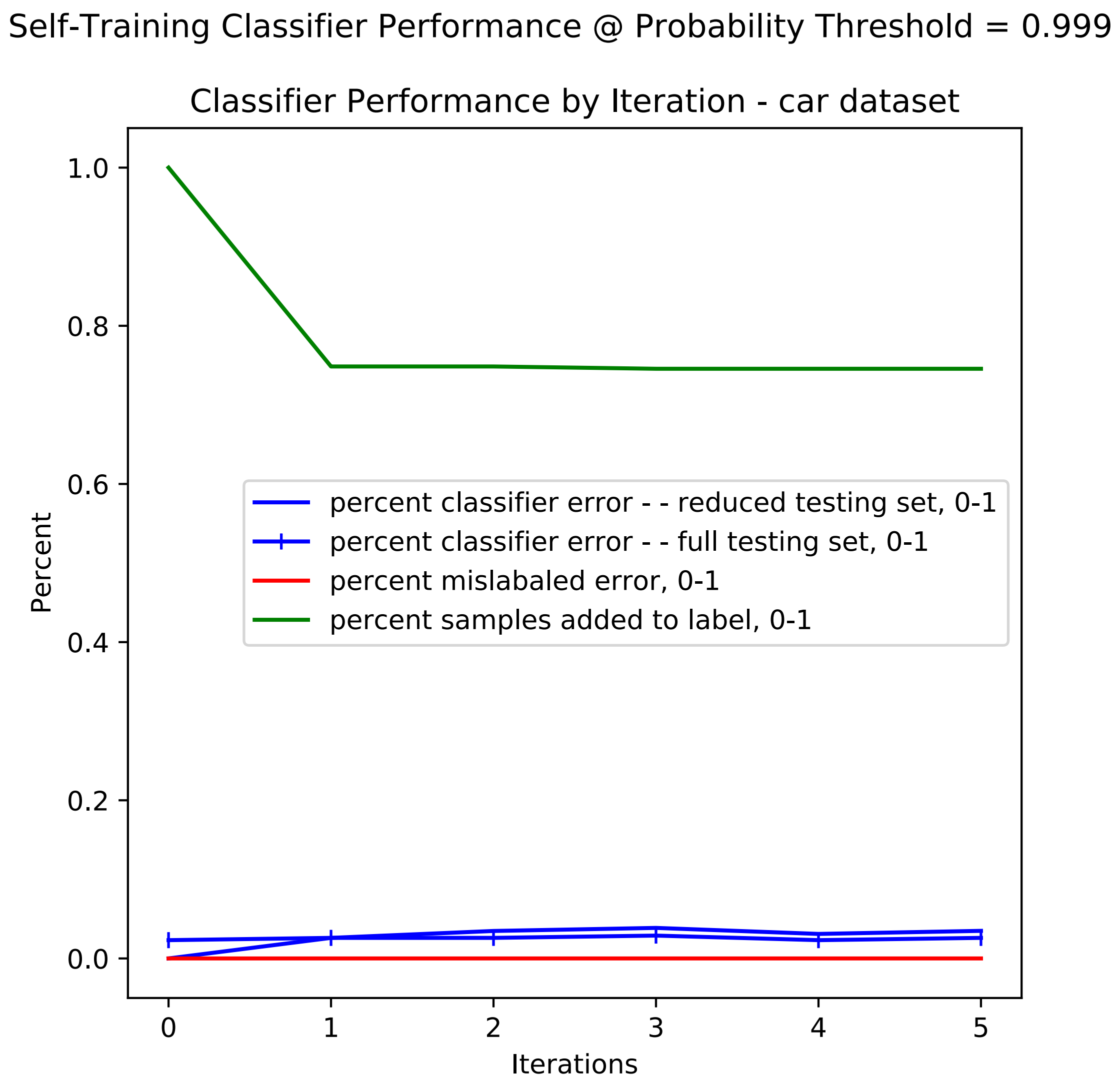

Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - car dataset
Percent
Iterations
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1

```
car
```

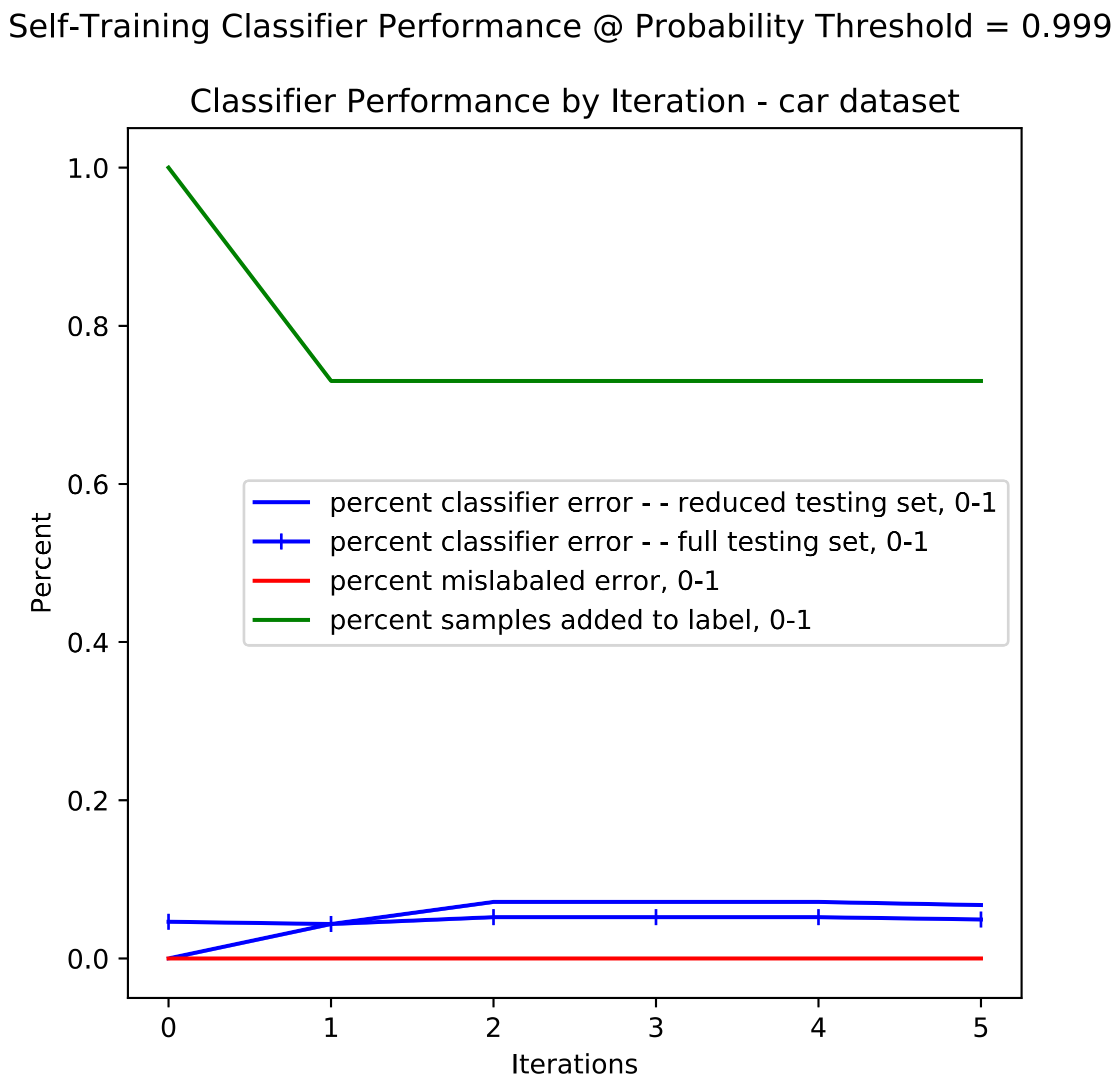
Self-Training Classifier Performance @ Probability Threshold = 0.999



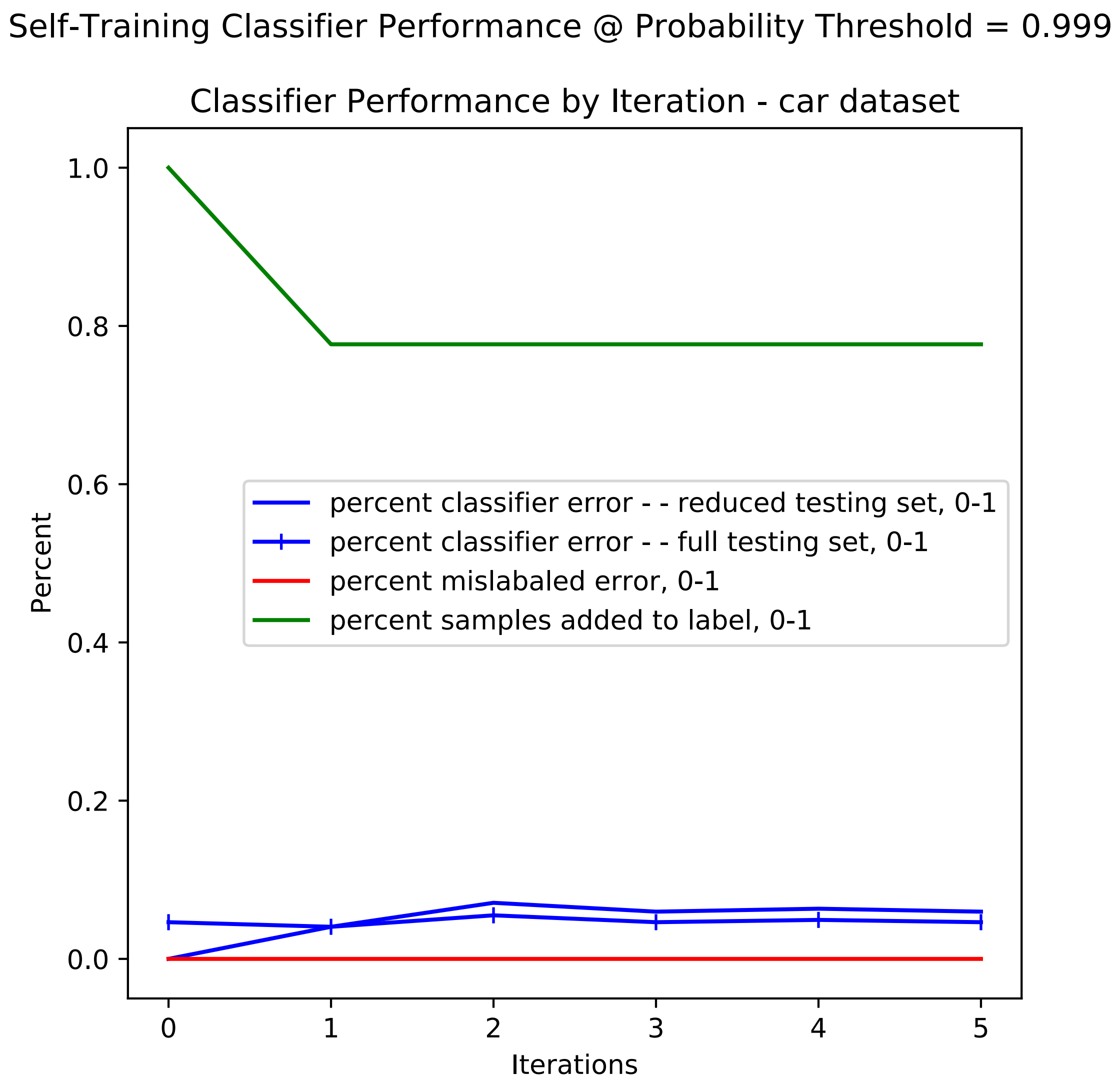

```
car
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - car dataset
Percent
Iterations
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1

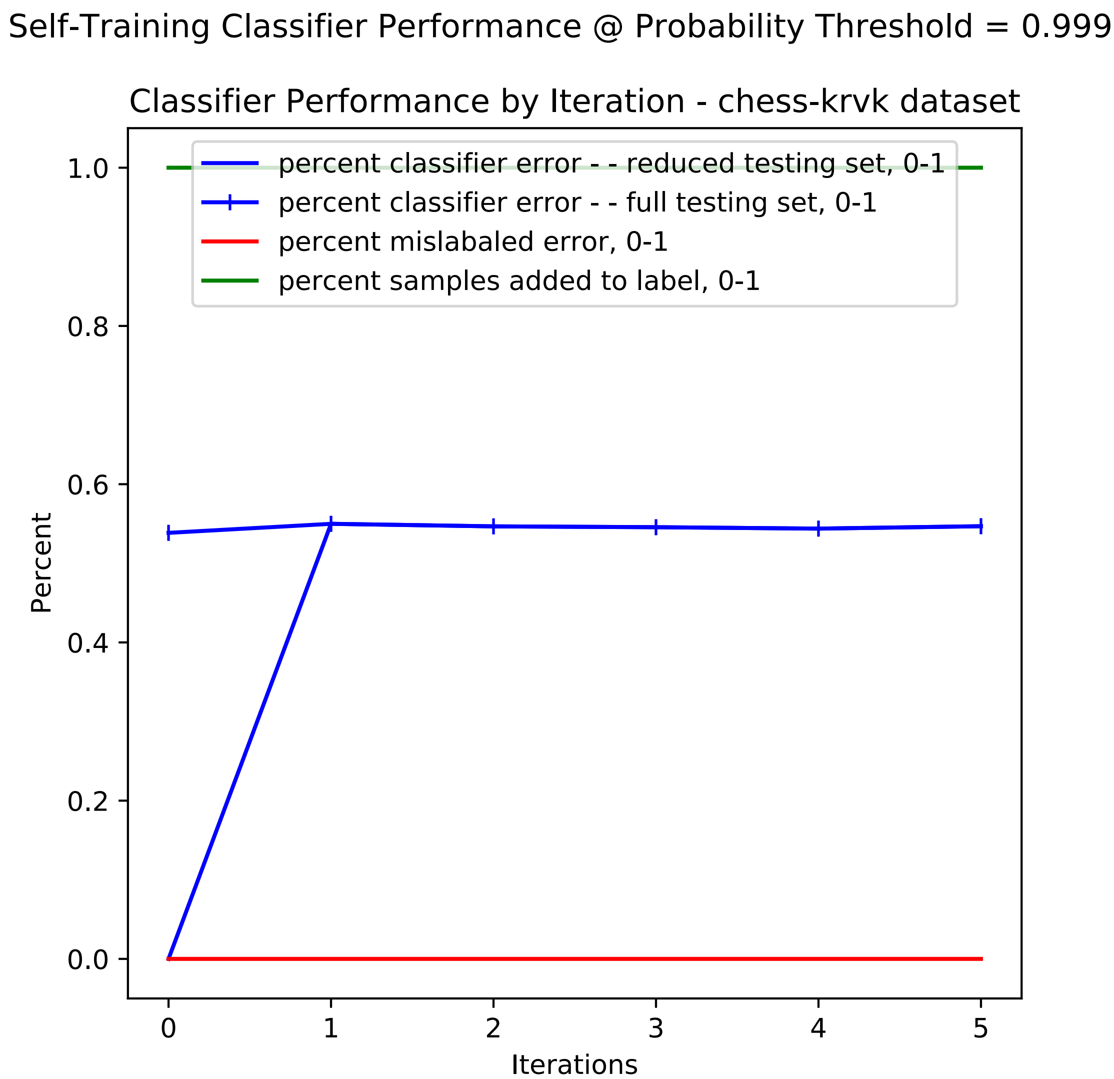
```
car
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - car dataset
Percent
Iterations
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1

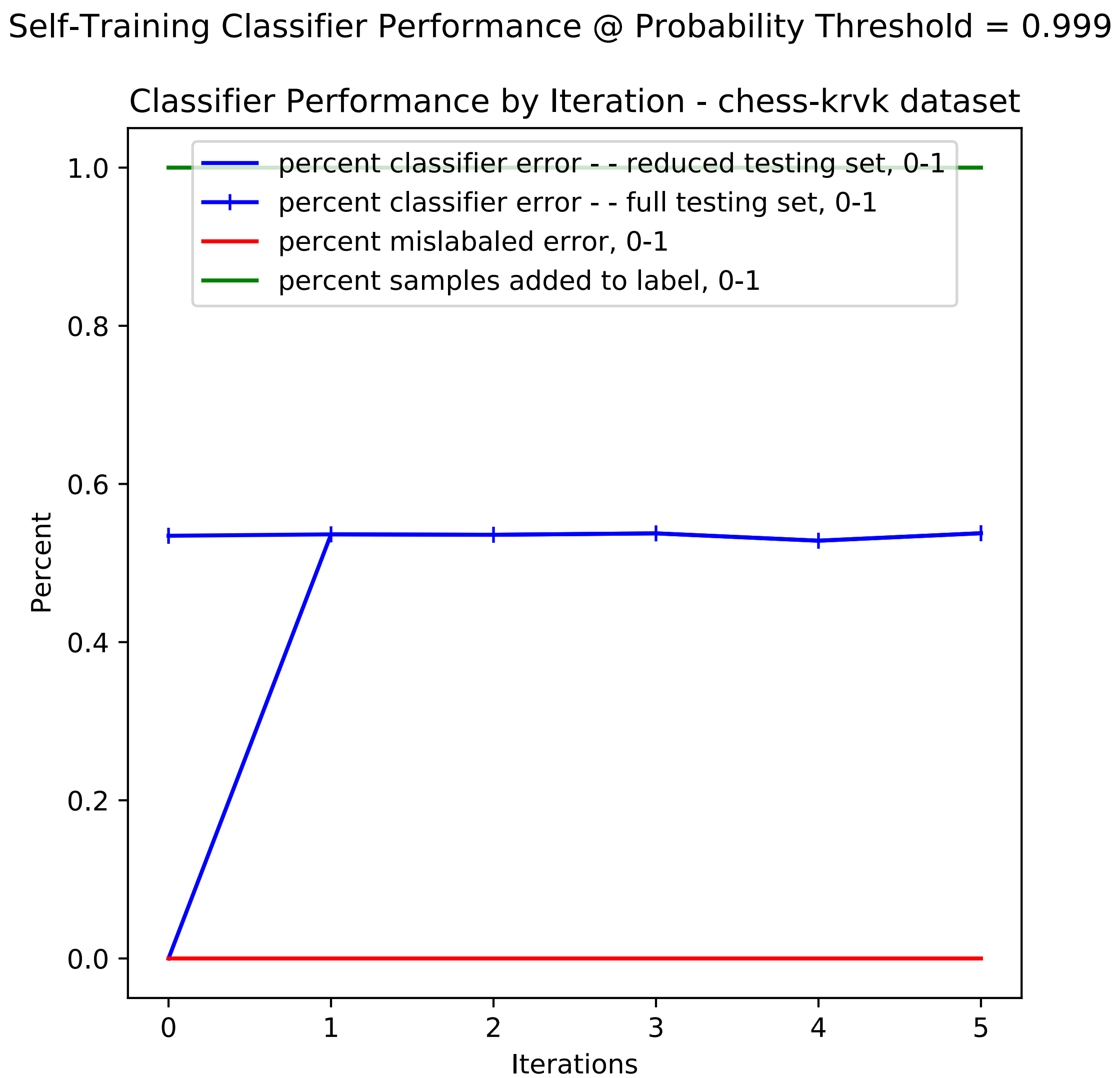
```
car
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - car dataset
Percent
Iterations
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5

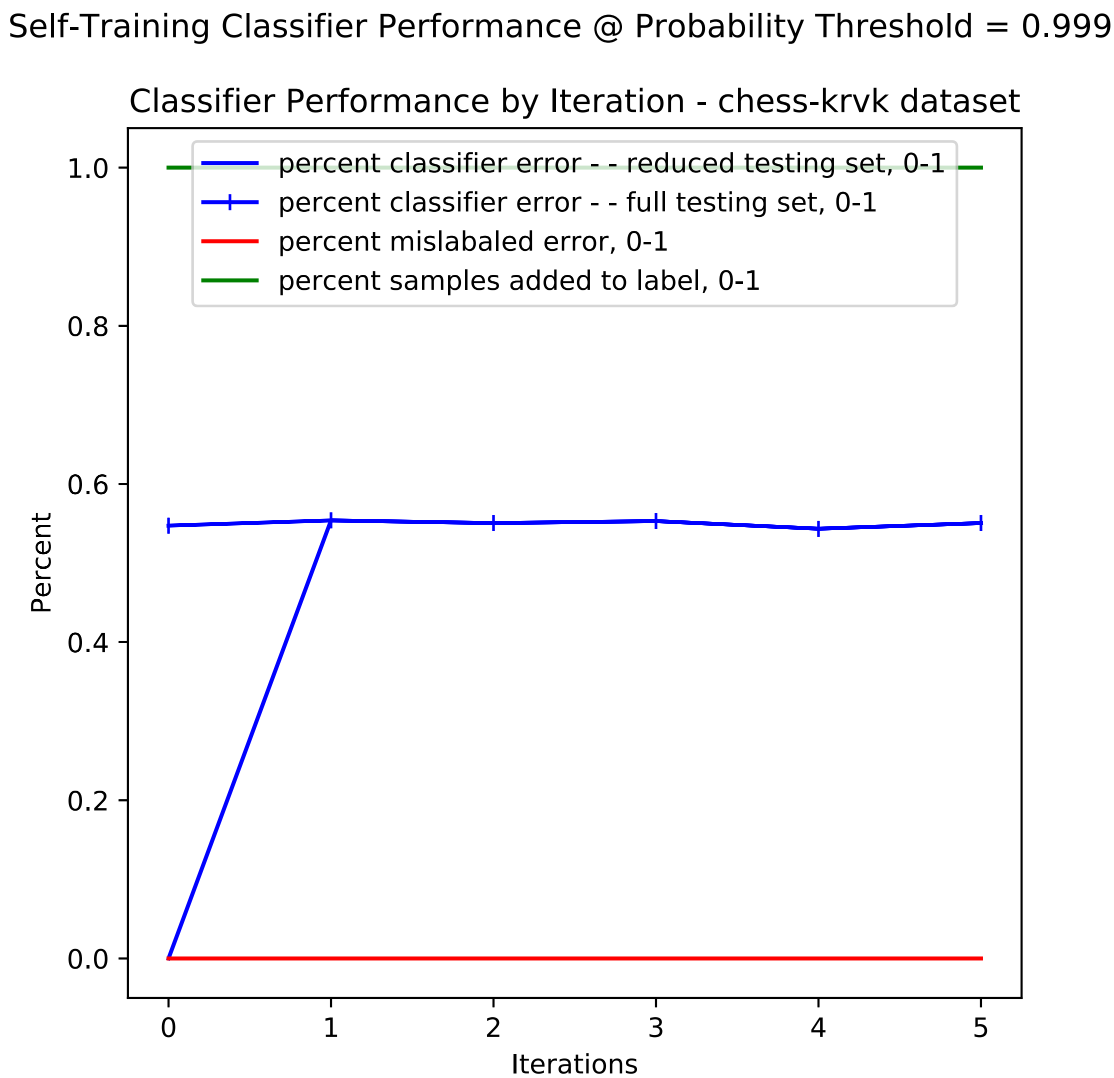
```
chess-krvk
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - chess-krvk dataset
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
1.0
0.8
0.6
0.4
0.2
0.0
Percent
0
1
2
3
4
5
Iterations

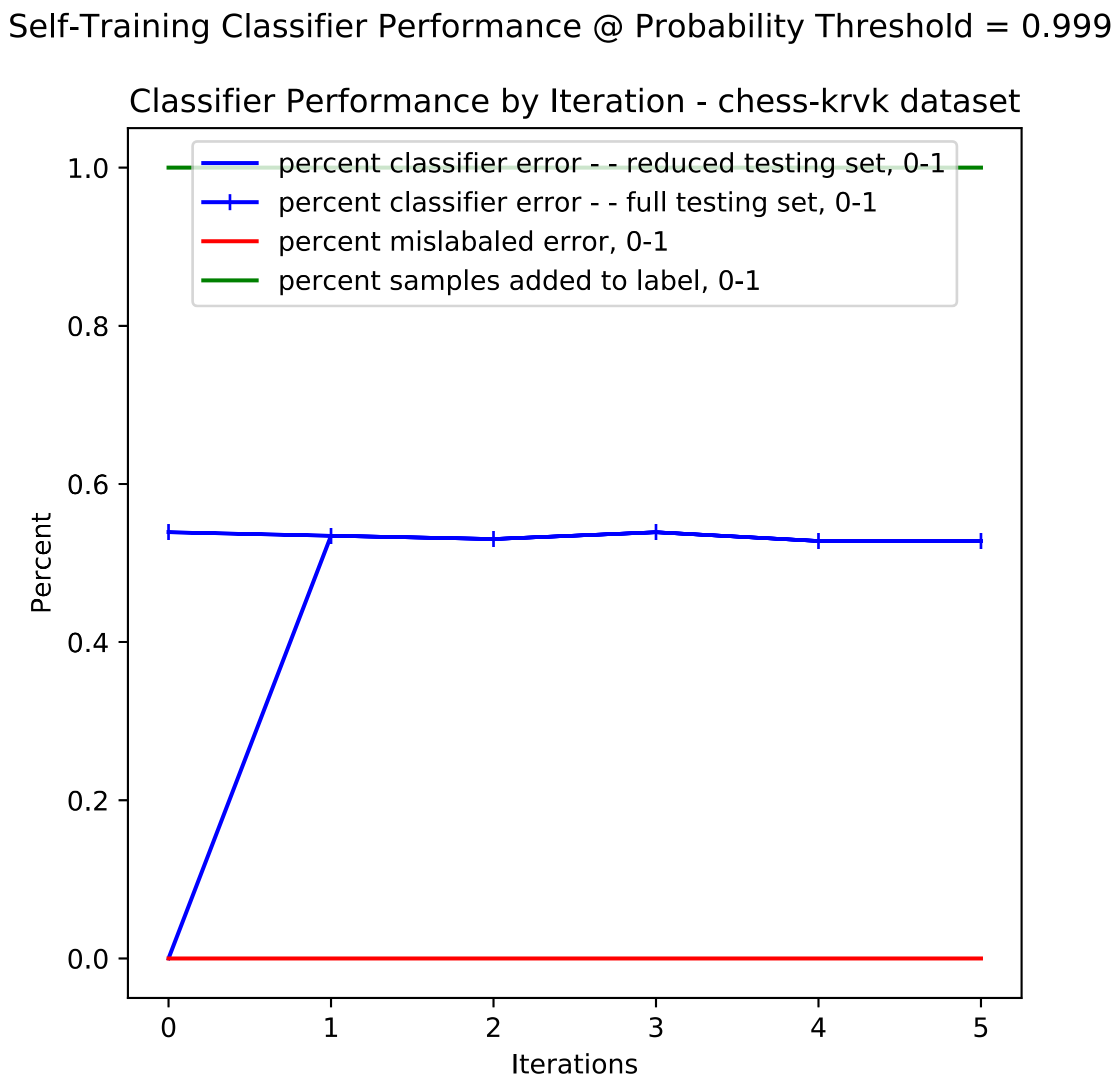
```
chess-krvk
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - chess-krvk dataset
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
Percent
Iterations
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5

```
chess-krvk
```

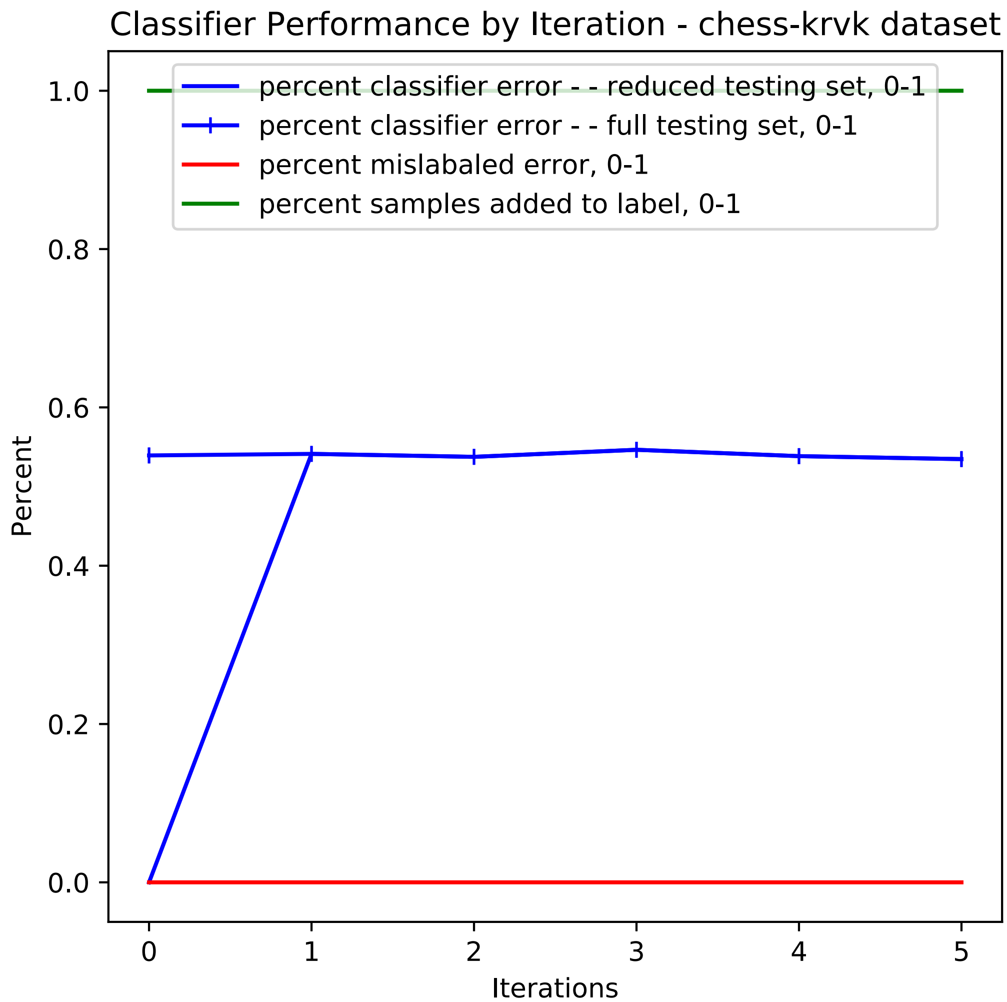

Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - chess-krvk dataset
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
1.0
0.8
0.6
0.4
0.2
0.0
Percent
0
1
2
3
4
5
Iterations

```
chess-krvk
```


Self-Training Classifier Performance @ Probability Threshold = 0.999
Classifier Performance by Iteration - chess-krvk dataset
percent classifier error - - reduced testing set, 0-1
percent classifier error - - full testing set, 0-1
percent mislabaled error, 0-1
percent samples added to label, 0-1
1.0
0.8
0.6
0.4
0.2
0.0
Percent
0
1
2
3
4
5
Iterations

```
chess-krvk
```

Self-Training Classifier Performance @ Probability Threshold = 0.999




```
In [ ]:
```