COMPSCI 371D Homework 6

Problem 0 (3 points)

Jihyeon Je (jj271), Tim Ho (th265), Rahul Prakash (rp221)

Preamble: The MNIST Dataset

In [1]:
```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

In [2]:
```
def standardize(data):
    scaler = StandardScaler().fit(data)
    data = scaler.transform(data).astype(np.float32)
    return data, scaler.mean_, scaler.scale_
```

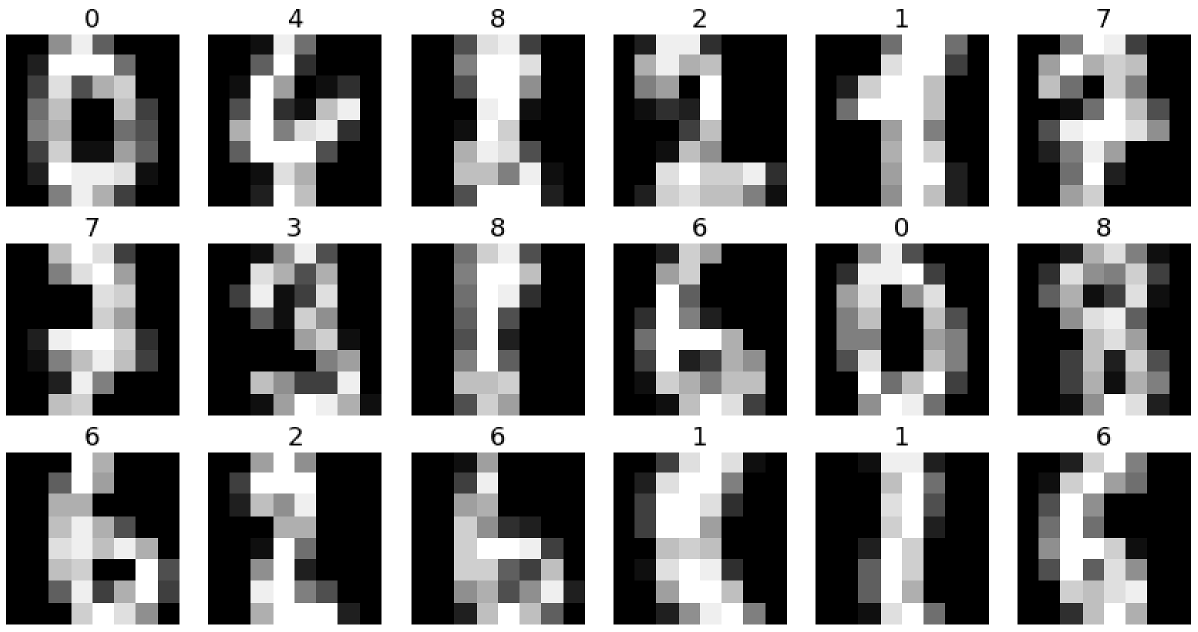
In [3]:
```
def load_mnist():
    digits = datasets.load_digits()
    n, shape = digits.images.shape[0], digits.images.shape[1:]
    xs = digits.images.reshape((n, -1)).astype(np.float32)
    max_pixel = np.max(xs)
    xs, mean, std = standardize(xs)
    ys = digits.target.astype(np.uint8)
    return xs, ys, mean, std, max_pixel, shape
```

In [4]:
```
def data_split(xs, ys, test_fraction=0.5):
    x_train, x_test, y_train, y_test = train_test_split(
        xs, ys, test_size=test_fraction, shuffle=True)
    train = {'x': x_train, 'y': y_train}
    test = {'x': x_test, 'y': y_test}
    data = {'train': train, 'test': test}
    return data
```

In [5]:
```
def x_to_image(x):
    x = np.round(x * pixel_std + pixel_mean)
    x = np.clip(x * 255. / pixel_max, 0., 255.).astype(np.uint8)
    return np.reshape(x, image_shape)
```

In [6]:
```
def show_random_images(xs, ys, rows=3, columns=6):
    rng = np.random.default_rng()
    indices = rng.integers(low=0, high=len(ys), size=rows * columns)
    plt.figure(figsize=(2 * columns, 2.1 * rows), tight_layout=True)
    for plot, index in enumerate(indices):
        image = x_to_image(xs[index])
        plt.subplot(rows, columns, plot + 1)
        plt.imshow(image, cmap='gray')
        plt.axis('off')
        plt.title(ys[index], fontsize=18)
    plt.show()
```

In [7]:
```
data_points, digit_labels, pixel_mean, pixel_std, pixel_max, image_shape
= load_mnist()
show_random_images(data_points, digit_labels)
digit_dataset = data_split(data_points, digit_labels)
```



Preamble: Three Classifiers

In [8]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

tolerance = 1.e-8
LogReg = LogisticRegression(
    C=1, solver='lbfgs', tol=tolerance,
    max_iter=100000, random_state=0)
LinearSvm = SVC(kernel='linear', tol=tolerance)
RbfSvm = SVC(kernel='rbf', tol=tolerance)
```

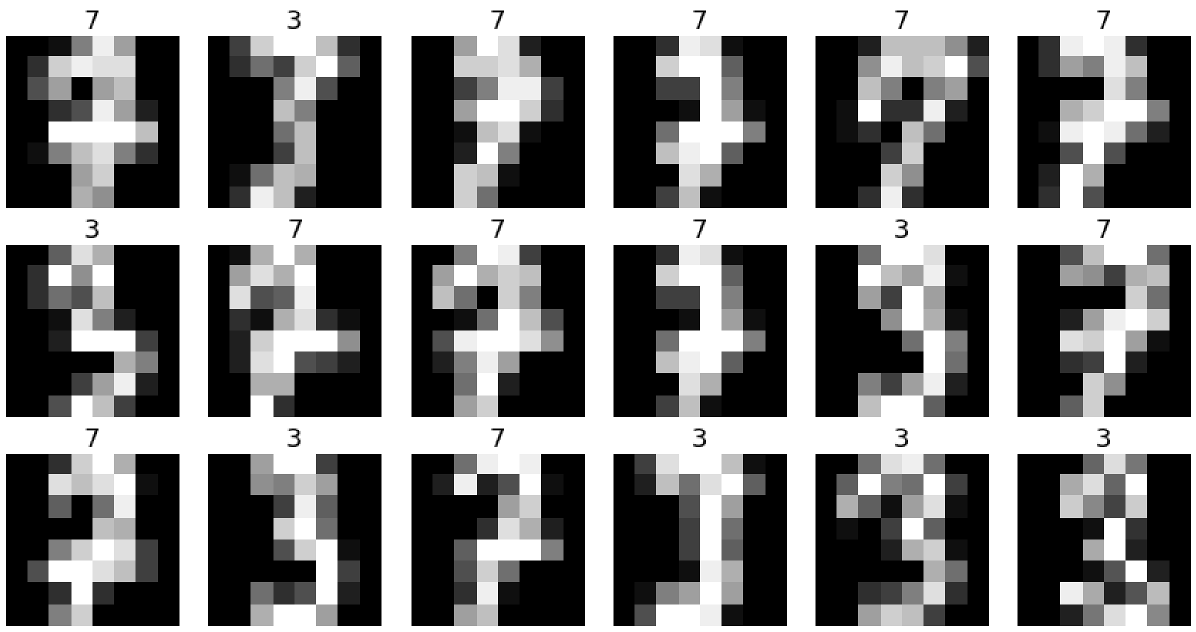
Part 1: Binary Classifiers

Problem 1.1

In [28]:
```
def evaluate(h, data):
    def error_rate(predictor, samples):
        x, y = samples['x'], samples['y']
        return (1 - predictor.score(x, y)) * 100

    e_train = error_rate(h, data['train'])
    e_test = error_rate(h, data['test'])
    return e_train, e_test
```

In [39]:
```
import numpy as np
def evaluate_classifiers(xs, ys):
    LogReg_test = []
    LogReg_train = []
    LinearSvm_test = []
    LinearSvm_train = []
    RbfSvm_test = []
    RbfSvm_train = []
    for i in range(0,10):
        split = data_split(xs, ys, test_fraction=0.5)
        LogReg_mod = LogReg.fit(split['train']['x'],split['train']['y'])
        LogReg_e = evaluate(LogReg_mod, split)
        LinearSvm_mod = LinearSvm.fit(split['train']['x'],split['train']
['y'])
        LinearSvm_e = evaluate(LinearSvm_mod, split)
        RbfSvm_mod = RbfSvm.fit(split['train']['x'],split['train']['y'])
        RbfSvm_e = evaluate(RbfSvm_mod, split)
        LogReg_test.append(LogReg_e[1])
        LogReg_train.append(LogReg_e[0])
        LinearSvm_test.append(LinearSvm_e[1])
        LinearSvm_train.append(LinearSvm_e[0])
        RbfSvm_test.append(RbfSvm_e[1])
        RbfSvm_train.append(RbfSvm_e[0])



    print(f'Error statistics for the LogReg classifier (percent): \n Tra
ining: min {min(LogReg_train):.3f}, max {max(LogReg_train):.3f}, mean {n
p.mean(LogReg_train):.3f}, std {np.std(LogReg_train):.3f}\n Testing: min
{min(LogReg_test):.3f}, max {max(LogReg_test):.3f}, mean {np.mean(LogReg
_test):.3f}, std {np.std(LogReg_test):.3f}\nError statistics for the Lin
earSvm classifier (percent): \n Training: min {min(LogReg_train):.3f}, m
ax {max(LogReg_train):.3f}, mean {np.mean(LogReg_train):.3f}, std {np.st
d(LinearSvm_train):.3f}\n Testing: min {min(LogReg_test):.3f}, max {max
(LinearSvm_test):.3f}, mean {np.mean(LinearSvm_test):.3f}, std {np.std(L
inearSvm_test):.3f}\nError statistics for the RbfSvm classifier (percen
t): \n Training: min {min(RbfSvm_train):.3f}, max {max(RbfSvm_train):.3
f}, mean {np.mean(RbfSvm_train):.3f}, std {np.std(RbfSvm_train):.3f}\n T
esting: min {min(RbfSvm_test):.3f}, max {max(RbfSvm_test):.3f}, mean {n
p.mean(RbfSvm_test):.3f}, std {np.std(RbfSvm_test):.3f}\n')
```

In [65]:
```
pair_data_points = data_points[np.where((digit_labels == 3) | (digit_lab
els==7))]
pair_labels = digit_labels[np.where((digit_labels == 3) | (digit_labels=
=7))]
show_random_images(pair_data_points, pair_labels)
evaluate_classifiers(pair_data_points, pair_labels)
```



```
Error statistics for the LogReg classifier (percent):
 Training: min 0.000, max 0.000, mean 0.000, std 0.000
 Testing: min 0.000, max 1.105, mean 0.387, std 0.354
Error statistics for the LinearSvm classifier (percent):
 Training: min 0.000, max 0.000, mean 0.000, std 0.000
 Testing: min 0.000, max 1.105, mean 0.166, std 0.354
Error statistics for the RbfSvm classifier (percent):
 Training: min 0.000, max 0.000, mean 0.000, std 0.000
 Testing: min 0.000, max 1.657, mean 0.939, std 0.432
```

Problem 1.2 (Exam Style)

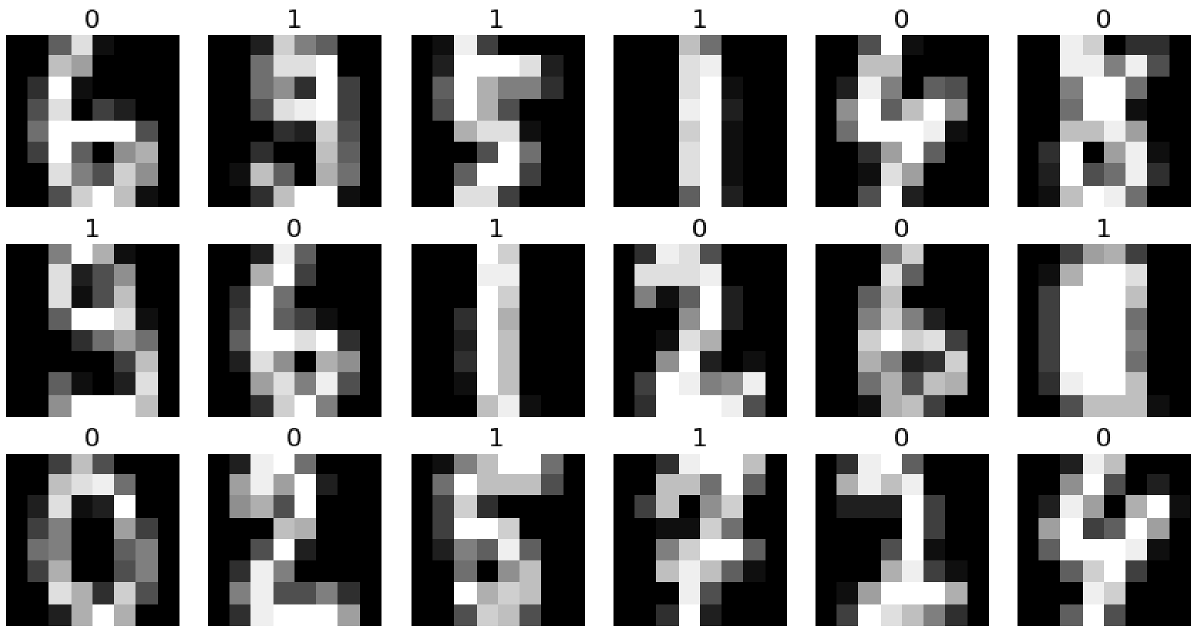
Some of the training sets in the ten splits are linearly separable. As the minimum percent error is zero for LogReg and LinearSVM models, we can conclude that some of the training sets in the ten splits must be linearly separable.

Problem 1.3 (Exam Style)

No it does not show conclusively that the RBF SVM generalize better or worse than the linear SVM. Running evaluate_classifier multiple times, we found that the means of the percent error in the testing set are relatively similar and the standard deviation show that the distributions of test error overlap.

Problem 1.4

In [50]:
```
parity_labels = np.where(digit_labels%2==0, 0,1)
show_random_images(data_points, parity_labels)
evaluate_classifiers(data_points, parity_labels)
```



```
Error statistics for the LogReg classifier (percent):
 Training: min 4.566, max 7.127, mean 6.058, std 0.789
 Testing: min 7.675, max 10.901, mean 9.021, std 0.904
Error statistics for the LinearSvm classifier (percent):
 Training: min 4.566, max 7.127, mean 6.058, std 0.696
 Testing: min 7.675, max 10.234, mean 9.210, std 0.562
Error statistics for the RbfSvm classifier (percent):
 Training: min 0.000, max 0.891, mean 0.490, std 0.278
 Testing: min 1.112, max 2.781, mean 2.058, std 0.446
```

Problem 1.5 (Exam Style)

The RBF SVM model performed the best because the mean percent error in both test and training set were the smallest. We hypothesize that the RBF SVM performs better as it performs better on non linearly separable data. Since the RBF SVM uses a guassian rather than linear kernel, it is able to fit a hypersurface rather than just a hyperplane to separate the data. After changing data labels to odd/even, the data is no longer linearly seprable, so the logistic regression classifier and linear SVM have poor performance.

Problem 1.6 (Exam Style)

RBF SVM overfit a little in the data set, because the testing mean percent error is higher than the training mean percent error. The model fits the training set too well, but it doesn't perform (generalize) as well on the testing set.

Part 2: Margins

In [70]:
```
def hyperplane(h):
    return h.intercept_, h.coef_[0]
```

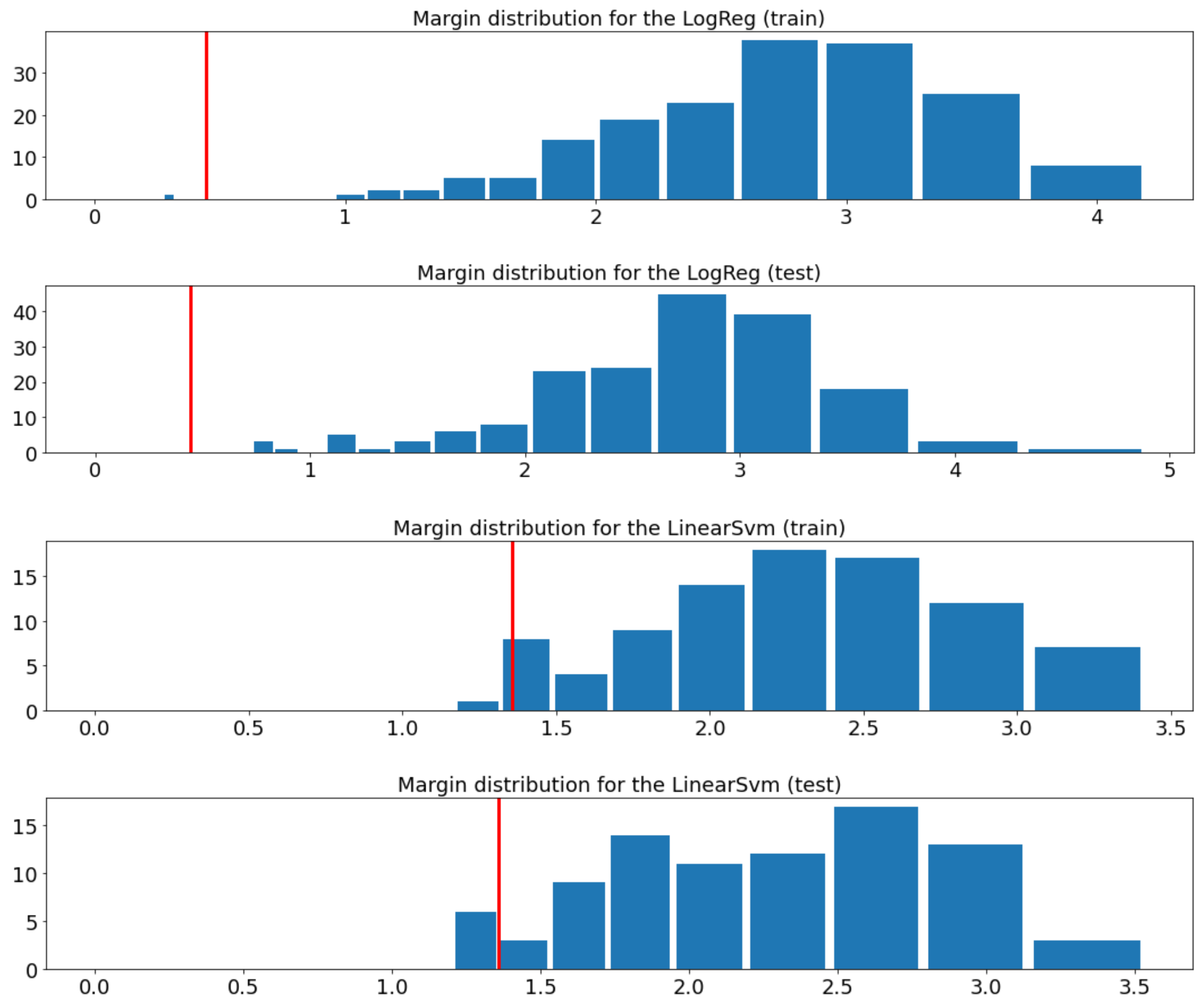
Problem 2.1

In [201]:
```
def margins(data,h):
    b, w = hyperplane(h)
    ref_margin  = 1/np.linalg.norm(w)
    n = w*ref_margin
    c = b*ref_margin
    data['y'] = np.where(data['y']==3, -1, 1)
    size = len(data['x'])
    margins = []
    for i in range(0,size):
        m = data['y'][i]*(np.matmul(n.T,data['x'][i])+c)
        margins.append(m[0])
    return margins, ref_margin
```

In [202]:
```
def show_margins(m, ref_margin, title, font_size=18):
    plt.figure(figsize=(15, 3), tight_layout=True)
    edges = np.logspace(-2, np.log10(max(m)), 50)
    plt.hist(m, bins=edges, rwidth=0.9)
    plt.axvline(ref_margin, color='r', lw=3.)
    title = 'Margin distribution for the {}'.format(title)
    plt.title(title, fontsize=font_size)
    plt.xticks(fontsize=font_size)
    plt.yticks(fontsize=font_size)
    plt.show()
```

In [203]:
```
def show_all_classifier_margins(dataset):
    lr_train = margins(dataset['train'], LogReg)
    lr_test = margins(dataset['test'], LogReg)
    ls_train = margins(dataset['train'], LinearSvm)
    ls_test = margins(dataset['test'], LinearSvm)
    show_margins(lr_train[0],lr_train[1], 'LogReg (train)')
    show_margins(lr_test[0],lr_test[1], 'LogReg (test)')
    show_margins(ls_train[0],ls_train[1], 'LinearSvm (train)')
    show_margins(ls_test[0],ls_test[1], 'LinearSvm (test)')
```

In [218]:
```
pair_dataset = data_split(pair_data_points, pair_labels)
show_all_classifier_margins(pair_dataset)
```



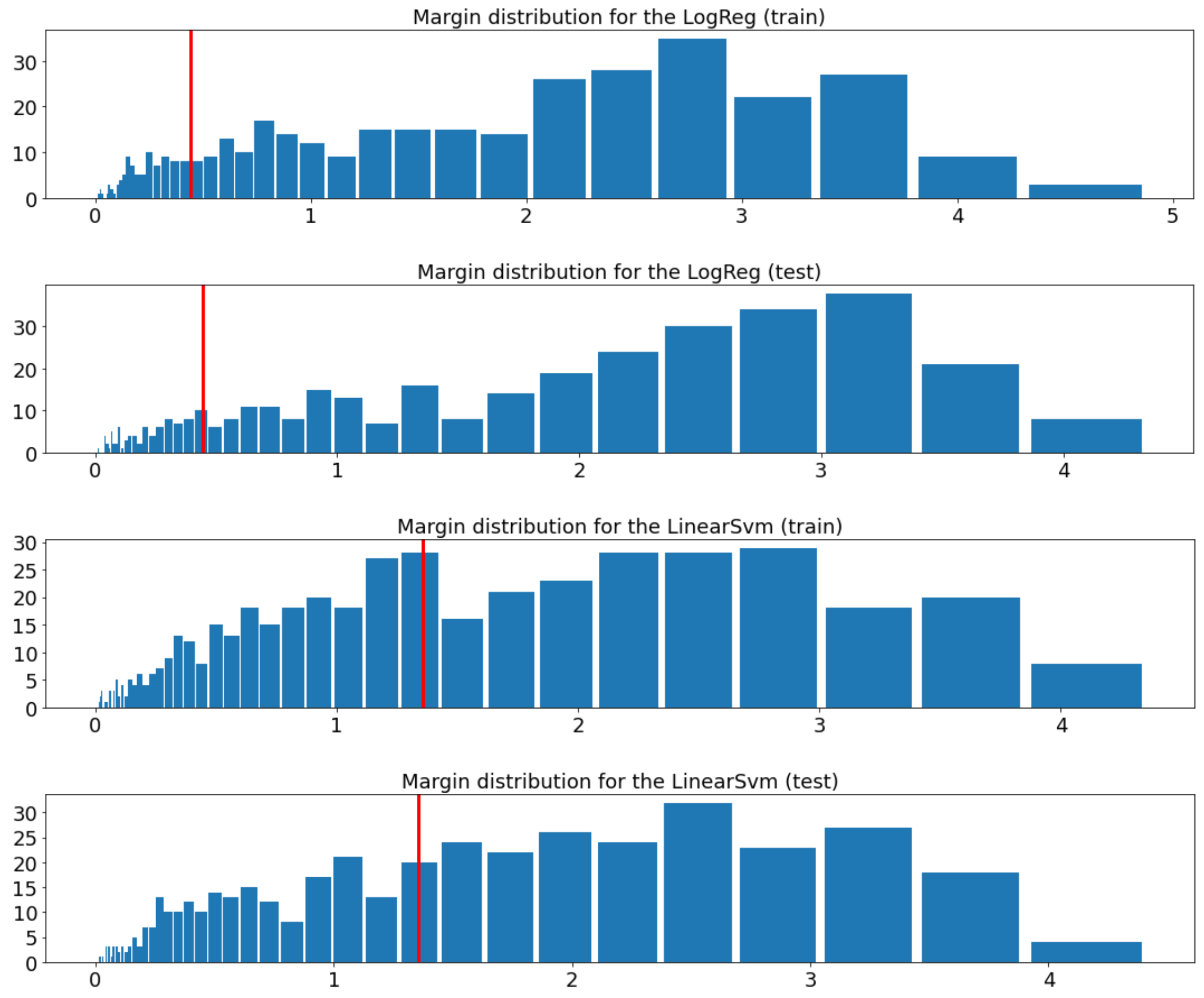
Problem 2.2 (Exam Style)

The distribution of data margins zin the training set is consistently different between the two classifiers. In the LinearSVM, the mean margin is smaller and the range is also reduced compared to LogReg. Furthermore, there are ~5-10 points within the reference margin in LinearSVM. However, in the LogReg classifier, there are consistently no datapoints within the reference margin. The reference margin itself is larger in the LinearSVM case. The pattern seems quite strong overall, since this difference in distributions is very consistent.

The pattern can be explained by the fact that LinearSVM maximizes the margin between datapoints from different classes near the decision boundary. LogReg also finds a hyperplane to separate the data but does not optimize for the distance of different class datapoints from this boundary.

Problem 2.3

In [234]:
```
parity_dataset = data_split(data_points, parity_labels)
show_all_classifier_margins(parity_dataset)
```



Problem 2.4 (Exam Style)

Yes, there are a consistently large number of data point that lies at and within the reference margin. Linear SVM maximizes the margin between the datapoints of different classes, and it also penalize for points that lies within the reference margin. Therefore, it will expand the margin to balance the margin size and penalty of points lying within the reference margin. Thus, we expect to see a large number of points clustered just at the reference margin. This reference margin also tend to be at a higher value than in LogReg.

Part 3: Multiclass Classifiers

Problem 3.1 (Partially Exam Style)

In [251]:
```
evaluate_classifiers(data_points, digit_labels)
```

```
Error statistics for the LogReg classifier (percent):
 Training: min 0.000, max 0.111, mean 0.067, std 0.055
 Testing: min 2.892, max 4.227, mean 3.648, std 0.427
Error statistics for the LinearSvm classifier (percent):
 Training: min 0.000, max 0.111, mean 0.067, std 0.000
 Testing: min 2.892, max 3.893, mean 2.481, std 0.669
Error statistics for the RbfSvm classifier (percent):
 Training: min 0.223, max 0.557, mean 0.345, std 0.116
 Testing: min 1.335, max 3.115, mean 2.425, std 0.526
```

Some of the data splits of the training set are linearly seperable, because we achieve 0 percent error in LogReg and Linear SVM on some of the splits. This implies that each pair of labels must've beem linearly seperable some of the time.

RBF SVM does the best (slightly better) most of the time. This conclusion is not reliable. It does not win by a large gap as the mean of the linear SVM percent error is similar to that of the Rbf SVM, at times even exactly identical. In addition, the standard deviation show that the distributions of test error overlap. However, we see that the max and min percent error of the RBF SVM is consistently smaller than those of Linear SVM.