# Homework 4 Solutions

Due May 01, 2020 by 11:59pm

**Instructions**: Upload your answers to the questions below to Canvas. Submit the answers to the questions in a PDF file and your code in a (single) separate file, including for the data competition exercise. Be sure to comment your code to indicate which lines of your code correspond to which question part. There are 3 study assignments and 2 exercises in this homework.

## Reading Assignments

- Review Lecture 4.
- Review Computer Lab. 4 in *canvas.uw.edu/courses/1371621/pages/course-materials* .
- Read and study Sec. 14.5-14.6 in *The Element of Statistical Learning* .

## 1 Exercise 1

In this exercise, you will implement in **Python** a first version of your own linear support vector machine with the huberized hinge loss.

The linear support vector machine with the huberized or smoothed hinge loss writes as

$$\min_{\beta\in\mathbb{R}^d} F(\beta) := \lambda\|\beta\|_2^2 + \frac{\rho}{n}\sum_{i=1;\, y_i=+1}^{n} \ell_{hh}(y_i, x_i^T\beta) + \frac{1-\rho}{n}\sum_{i=1;\, y_i=-1}^{n} \ell_{hh}(y_i, x_i^T\beta)\ ,$$

where $h = 0.5$ and

$$\ell_{\text{hh}}(y,t) := \begin{cases} 0 & \text{if} \quad yt > 1+h \\ \frac{(1+h-yt)^2}{4h} & \text{if} \quad |1-yt| \le h \\ 1-yt & \text{if} \quad yt < 1-h \end{cases} \tag{1}$$

You know now the fast gradient algorithm, so no need to recall it here.

- Consider the Vowel dataset from *The Elements of Statistical Learning*. Standardize the data, if you have not done so already.

```
import matplotlib.pyplot as plt
import numpy as np
```

```
import pandas as pd
import scipy.linalg
import scipy.stats
import sklearn.preprocessing
import sklearn.svm

# Part 1: Load the dataset. The standardization will be done later.
train_set = pd.read_table(
    'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/vowel.train',
    sep=',', header=0)
test_set = pd.read_table(
    'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/vowel.test',
    sep=',', header=0)

x_train = np.asarray(train_set)[:, 2:]
y_train = np.asarray(train_set)[:, 1]

x_test = np.asarray(test_set)[:, 2:]
y_test = np.asarray(test_set)[:, 1]

# Keep track of the number of samples and dimension of each sample
n_train = len(y_train)
n_test = len(y_test)
d = np.size(x_train, 1)
```

- Pick a positive class "+1" with examples from a single class and a negative class "-1" with examples from two other classes than the positive class.

```
# Part 2: Choose positive and negative classes
# 1 will be positive. 2, 3, will be negative
x_train = x_train[np.where(y_train <= 3)[0]]
y_train = y_train[np.where(y_train <= 3)[0]]
x_test = x_test[np.where(y_test <= 3)[0]]
y_test = y_test[np.where(y_test <= 3)[0]]
y_train = (y_train == 1)*2-1
y_test = (y_test == 1)*2-1
```

- Define a training set, a validation set, and a testing set, with roughly equal proportions of examples from the positive class and the negative class resp. within each set.

```
# Part 3: Define a training set, a validation set, and a testing set
ntrain = int(len(x_train)*0.8)
x_valid = x_train[ntrain:]
y_valid = y_train[ntrain:]
x_train = x_train[:ntrain]
```

```
y_train = y_train[:ntrain]
# Standardize
xbar = np.mean(x_train, axis=0)
xstd = np.std(x_train, axis=0)
x_train = (x_train - xbar)/xstd[np.newaxis, :]
x_valid = (x_valid - xbar)/xstd[np.newaxis, :]
x_test = (x_test - xbar)/xstd[np.newaxis, :]
```

- Derive the formula of the gradient of the objective function. You may proceed case by case and then define a general formula.
  First observe that, if the gradient exists, then defining $t_i = x_i^T \beta$, it is given by

$$\frac{dF}{d\beta} = 2\lambda\beta + \frac{\rho}{n} \sum_{i=1;\, y_i=+1}^{n} \frac{\partial \ell_{hh}}{\partial t_i} \frac{\partial t_i}{\partial \beta} + \frac{1-\rho}{n} \sum_{i=1;\, y_i=-1}^{n} \frac{\partial \ell_{hh}}{\partial t_i} \frac{\partial t_i}{\partial \beta} \, .$$

  As you know from the lectures, $\frac{\partial t_i}{\partial \beta} = x_i$. Therefore, it just remains to compute $\frac{\partial \ell_{hh}}{\partial t_i}$ (if it exists).

  First we need to check that $\ell_{hh}$ is continuous. At $yt = 1 + h$ the middle component is 0. Moreover, at $yt = 1 - h$ the middle component is

$$\frac{(2 - 2yt)^2}{4h} = \frac{(1 - yt)^2}{1 - yt} = 1 - yt \, .$$

  Therefore, since the left and right limits of $\ell_{hh}$ match at $yt = 1 + h$ and $yt = 1 - h$ the function is continuous.

  Next we will compute the derivative of each part and check that the left and right limits of the derivatives exist and match at $yt = 1 + h$ and $yt = 1 - h$. We find that (again, assuming the derivative exists)

$$\frac{\partial \ell_{hh}}{\partial t} = \begin{cases} 0 & \text{if} \quad yt > 1 + h \\ \frac{-2y(1+h-yt)}{4h} & \text{if} \quad |1 - yt| \leq h \\ -y & \text{if} \quad yt < 1 - h \end{cases}$$

$$= \begin{cases} 0 & \text{if} \quad yt > 1 + h \\ \frac{-y(1+h-yt)}{2h} & \text{if} \quad |1 - yt| \leq h \\ -y & \text{if} \quad yt < 1 - h \end{cases}$$

  Since the derivative of the center case at $yt = 1 + h$ is 0 and the derivative of the center case at $yt = 1 - h$ is

$$\frac{-y(1 + h - (1 - h))}{2h} = -y$$

  we conclude that the function $\ell_{hh}$ is differentiable.

Putting together the above computations and defining

$$\mathbb{1}(z < h) = \begin{cases} 1, & z < h \\ 0, & z \geq h \end{cases},$$

the gradient is given by

$$\begin{aligned}
\nabla F(\beta) = & \frac{\rho}{n} \sum_{i:y_i=+1} \left\{ \frac{-y_i(1+h-y_i x_i^T \beta)}{2h} \mathbb{1}(|1-y_i x_i^T \beta| < h) - y_i \mathbb{1}(y_i x_i^T \beta < 1-h) \right\} x_i \\
& + \frac{1-\rho}{n} \sum_{i:y_i=-1} \left\{ \frac{-y_i(1+h-y_i x_i^T \beta)}{2h} \mathbb{1}(|1-y_i x_i^T \beta| < h) - y_i \mathbb{1}(y_i x_i^T \beta < 1-h) \right\} x_i \\
& + 2\lambda\beta \\
= & \frac{\rho}{n} \sum_{i:y_i=+1} \left\{ \frac{-(1+h-x_i^T \beta)}{2h} \mathbb{1}(|1-x_i^T \beta| < h) - \mathbb{1}(x_i^T \beta < 1-h) \right\} x_i \\
& + \frac{1-\rho}{n} \sum_{i:y_i=-1} \left\{ \frac{1+h+x_i^T \beta}{2h} \mathbb{1}(|1+x_i^T \beta| < h) + \mathbb{1}(-x_i^T \beta < 1-h) \right\} x_i \\
& + 2\lambda\beta \, .
\end{aligned}$$

- Write a function *mylinearsvm* that implements the fast gradient algorithm to train the linear support vector machine with the huberized hinge loss. The function takes as input the training data, the regularization parameter $\lambda$, the imbalance parameter $\rho$, and the optimization accuracy $\varepsilon$.

```
# Part 5: Implementation
def objective(beta, lambduh, rho, x=x_train, y=y_train, h=0.5):
    n = len(y_train)
    yt = y*x.dot(beta)
    ell = (1+h-yt)**2/(4*h)*(np.abs(1-yt) <= h) + (1-yt)*(yt < (1+h))
    return rho/n*np.sum(ell[y == 1]) + (1-rho)/n*np.sum(ell[y == -1]) \
            + lambduh*np.dot(beta, beta)


def computegrad(beta, lambduh, rho, x=x_train, y=y_train, h=0.5):
    n = len(y_train)
    yt = y*x.dot(beta)
    ell_prime = -(1+h-yt)/(2*h)*y*(np.abs(1-yt) <= h) - y*(yt < (1-h))
    return rho/n*np.sum(ell_prime[y == 1, np.newaxis]*x[y == 1], axis=0) \
      + (1-rho)/n*np.sum(ell_prime[y != 1, np.newaxis]*x[y != 1], axis=0) \
      + 2*lambduh*beta


def bt_line_search(beta, lambduh, rho, eta=1, alpha=0.5, betaparam=0.8,
```

```
                    maxiter=1000, x=x_train, y=y_train):
    grad_beta = computegrad(beta, lambduh, rho, x=x, y=y)
    norm_grad_beta = np.linalg.norm(grad_beta)
    found_eta = 0
    iter_num = 0
    while found_eta == 0 and iter_num < maxiter:
        if objective(beta - eta * grad_beta, lambduh, rho, x=x, y=y) < \
            objective(beta, lambduh, rho, x=x, y=y) \
                - alpha * eta * norm_grad_beta ** 2:
            found_eta = 1
        elif iter_num == maxiter - 1:
            print('Warning: Max number of iterations of' \
                +  'backtracking line search reached')
            break
        else:
            eta *= betaparam
            iter_num += 1
    return eta


def mylinearsvm(beta_init, theta_init, lambduh, rho, eta_init, maxiter,
    x=x_train, y=y_train, eps=1e-6):
    beta = beta_init
    theta = theta_init
    grad_theta = computegrad(theta, lambduh, rho, x=x, y=y)
    beta_vals = beta
    theta_vals = theta
    iter_num = 0
    while iter_num < maxiter and np.linalg.norm(grad_theta) > eps:
        eta = bt_line_search(theta, lambduh, rho, eta=eta_init, x=x, y=y)
        beta_new = theta - eta*grad_theta
        theta = beta_new + iter_num/(iter_num+3)*(beta_new-beta)
        # Store all of the places we step to
        beta_vals = np.vstack((beta_vals, beta_new))
        theta_vals = np.vstack((theta_vals, theta))
        grad_theta = computegrad(theta, lambduh, rho, x=x, y=y)
        beta = beta_new
        iter_num += 1

    return beta_vals


def misclassification_error(beta, x=x_test, y=y_test):
    y_pred = (x.dot(beta) > 0)*2-1
    return np.mean(y_pred != y)

def sensitivity_specificity(beta, x=x_test, y=y_test):
```

```
    y_pred = (x.dot(beta) > 0)*2-1
    sensitivity = sum((y_pred == 1) & (y == 1))/sum(y == 1)
    specificity = sum((y_pred == -1) & (y == -1))/sum(y == -1)
    return sensitivity, specificity

def objective_plot(betas_fg, lambduh, rho, x=x_train, y=y_train):
    num_points = np.size(betas_fg, 0)
    objs_fg = np.zeros(num_points)
    for i in range(0, num_points):
        objs_fg[i] = objective(betas_fg[i, :], lambduh, rho, x=x, y=y)
    fig, ax = plt.subplots()
    ax.plot(range(1, num_points + 1), objs_fg)
    plt.xlabel('Iteration')
    plt.ylabel('Objective value')
    plt.title('Objective value vs. iteration when lambda='
        + str(lambduh) + ', rho=' + str(rho))
    plt.show()
```

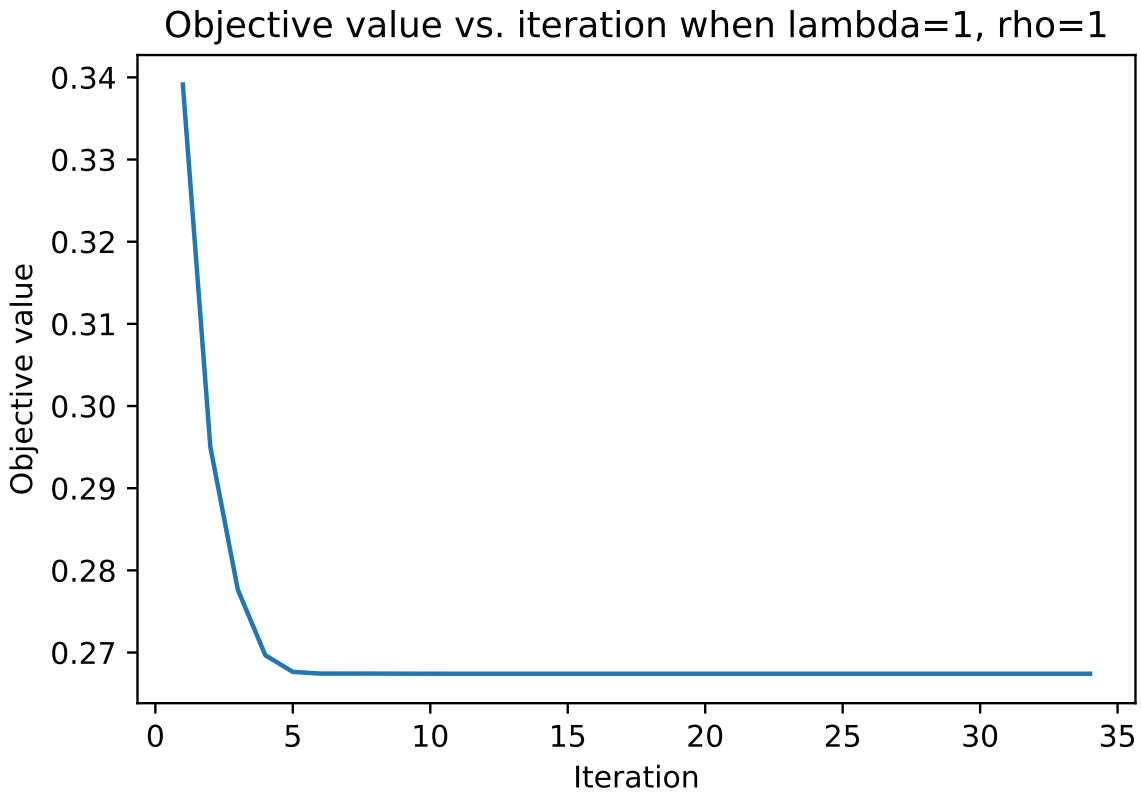
- Train your linear support vector machine with the huberized hinge loss on the the Vowel dataset for $\lambda = 1$ and $\rho = 1$. Report the misclassification error for these hyper-parameter values. Report the sensitivity and the specificity for these hyper-parameter values.[1]

```
# Part 6: Run for lambda=rho=1
lambduh = 1
rho = 1
beta_init = np.zeros(d)
theta_init = np.zeros(d)
max_eig = scipy.linalg.eigh(1 / len(y_train) * \
    x_train.T.dot(x_train), eigvals=(d - 1, d - 1),
    eigvals_only=True)[0]
eta_init = 1 / (max_eig + lambduh)
maxiter = 100

beta_vals = mylinearsvm(beta_init, theta_init, lambduh,
                        rho, eta_init, maxiter)
objective_plot(beta_vals, lambduh, rho)

for (name, x, y) in zip(('training', 'validation', 'test'),
                        (x_train, x_valid, x_test),
                        (y_train, y_valid, y_test)):
    error = misclassification_error(beta_vals[-1], x=x, y=y)
    sensitivity, specificity = sensitivity_specificity(beta_vals[-1],
                                                       x=x, y=y)
```

[1] You may set the optimization accuracy to a value of your choice based on your experience from previous homework assignments.

```
    print('Misclassification error on the ' + name +
          ' set: %0.2f'% error)
    print('Sensitivity and specificity on the ' + name +
          ' set: %0.2f, %0.2f'%(sensitivity, specificity))
```

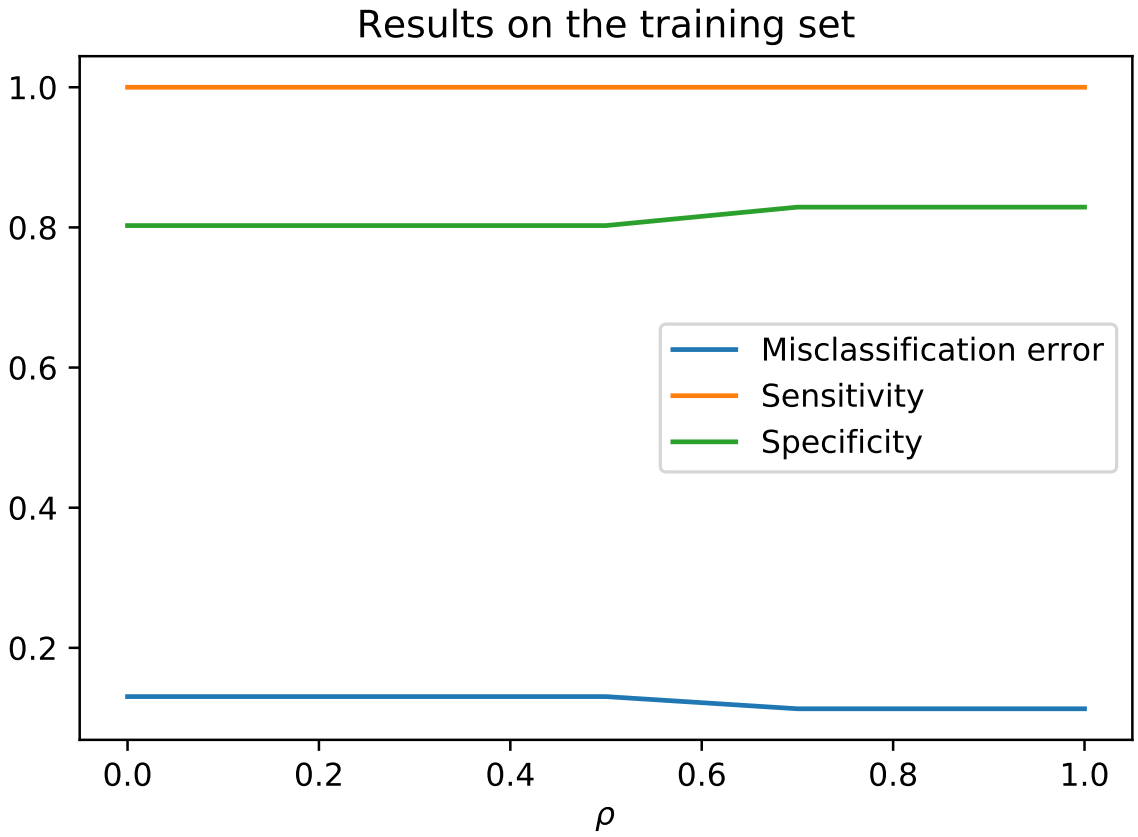
```
Misclassification error on the training set: 0.11
Sensitivity and specificity on the training set: 1.00, 0.83
Misclassification error on the validation set: 0.34
Sensitivity and specificity on the validation set: 0.33, 0.80
Misclassification error on the test set: 0.20
Sensitivity and specificity on the test set: 0.76, 0.82
```

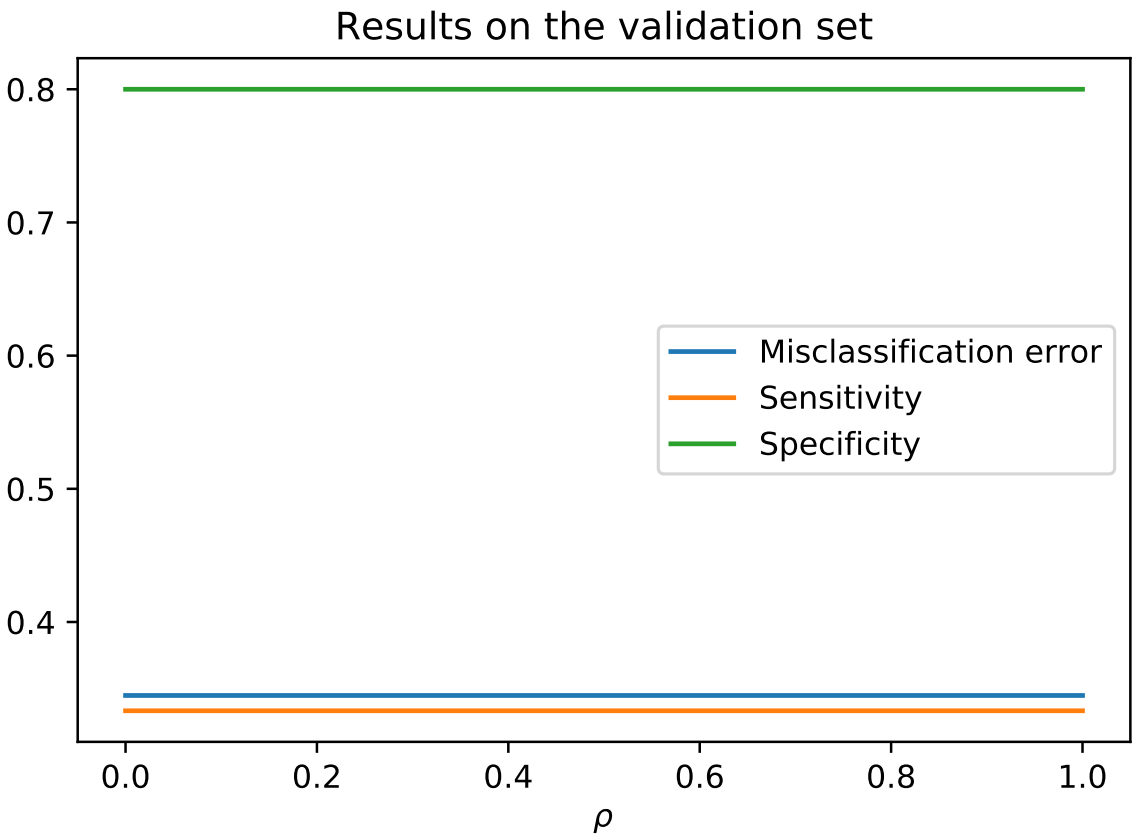


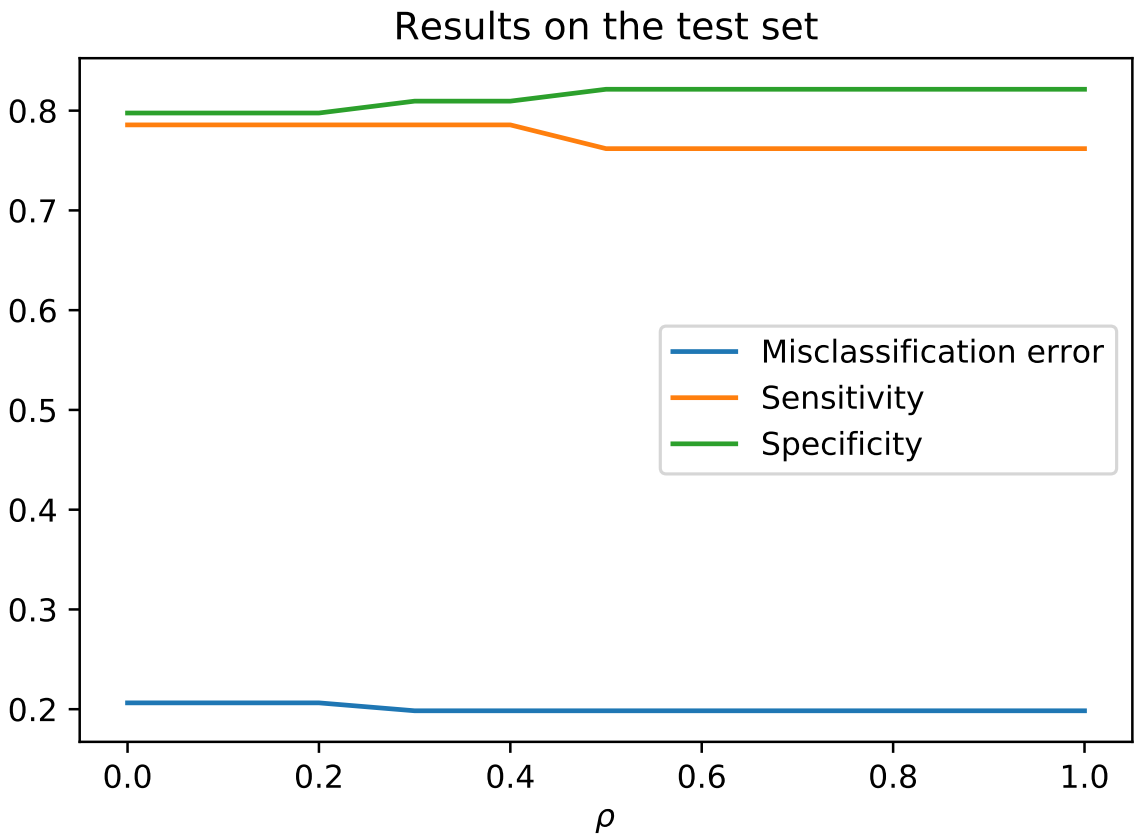

- Train your linear support vector machine with the huberized hinge loss on the the Vowel dataset for $\lambda = 1$ and $\rho = 0.1, 0.2, \ldots, 0.9, 1.0$. Plot the misclassification error vs. the imbalance parameter $\rho$. Plot on the same figure yet with different colors the sensitivity and the specificity vs. the imbalance parameter $\rho$.

```
# Part 7: Vary rho and plot the results
def plot_metric_results(results_dict, rhos, name):
    plt.clf()
    for metric in ['Misclassification error', 'Sensitivity',
                                'Specificity']:
        plt.plot(rhos, results_dict[name][metric], label=metric)
```

```
    plt.xlabel(r'$\rho$')
    plt.legend()
    plt.title('Results on the ' + name + ' set')
    plt.show()


rhos = np.arange(0, 1.1, 0.1)
results_dict = {}
for rho in rhos:
    beta_vals = mylinearsvm(beta_init, theta_init, lambduh,
                            rho, eta_init, maxiter)
    for (name, x, y) in zip(('training', 'validation', 'test'),
                                                (x_train, x_valid, x_test),
                                                (y_train, y_valid, y_test)):
        error = misclassification_error(beta_vals[-1], x=x, y=y)
        sensitivity, specificity = sensitivity_specificity(beta_vals[-1],

        if name not in results_dict:
            results_dict[name] = {}
            results_dict[name]['Misclassification error'] = [error]
            results_dict[name]['Sensitivity'] = [sensitivity]
            results_dict[name]['Specificity'] = [specificity]
        else:
            results_dict[name]['Misclassification error'].append(error)
            results_dict[name]['Sensitivity'].append(sensitivity)
            results_dict[name]['Specificity'].append(specificity)

for name in ['training', 'validation', 'test']:
    plot_metric_results(results_dict, rhos, name)
```


Results on the training set
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0
ρ
Misclassification error
Sensitivity
Specificity


Results on the validation set
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0
ρ
Misclassification error
Sensitivity
Specificity




- Select the best value of the pair $\lambda$ and $\rho$ on a grid of values of your choice in terms of area under the sensitivity/(1-specificity) curve obtained from the validation set. Compute the sensitivity and the specificity on the testing set for the best values found for the pair $\lambda$ and $\rho$.

```
# Part 8: Select the best lambda and rho from a grid
def auc_thresholds(xbeta):
    sorted_xbeta = np.sort(xbeta)
    thresholds = np.concatenate(([sorted_xbeta[0]-1],
                                 (sorted_xbeta[1:]+ sorted_xbeta[:-1])/2,
                                 [sorted_xbeta[-1]+1]))
    return thresholds

def trapezoid_rule(x, y):
    auc = np.sum((y[:-1] + y[1:]) / 2 * (x[1:] - x[:-1]))
    return auc

def compute_auc(beta, x, y):
    xbeta = x.dot(beta)
    thresholds = auc_thresholds(xbeta)
    y_pred = (xbeta[:, np.newaxis] > thresholds) * 2 - 1
    sensitivities = np.sum((y_pred == 1) & (y[:, np.newaxis] == 1),
```

```
                            axis=0) / sum(y == 1)
    specificities = np.sum((y_pred == -1) & (y[:, np.newaxis] == -1),
                            axis=0) / sum(y == -1)
    auc = trapezoid_rule(1 - specificities[::-1], sensitivities[::-1])
    return auc


lambdas = [2**i for i in range(-20, 11)]
best_auc = 0
results_dict = {}

for lam in lambdas:
    eta_init = 1 / (max_eig + lambduh)
    for rho in rhos:
        beta_vals = mylinearsvm(beta_init, theta_init, lam,
                                rho, eta_init, maxiter)

        for (name, x, y) in zip(('training', 'validation', 'test'),
                                (x_train, x_valid, x_test),
                                (y_train, y_valid, y_test)):
            error = misclassification_error(beta_vals[-1], x=x, y=y)
            sensitivity, specificity = sensitivity_specificity(
                                                beta_vals[-1], x=x, y=y)

            auc = compute_auc(beta_vals[-1], x, y)
            if name == 'validation' and auc > best_auc:
                best_auc = auc
                best_lam = lam
                best_rho = rho

            if name not in results_dict or rho not in results_dict[name]:
                if name not in results_dict:
                    results_dict[name] = {}
                results_dict[name][rho] = {}
                results_dict[name][rho]['Misclassification error'] = {}
                results_dict[name][rho]['Sensitivity'] = {}
                results_dict[name][rho]['Specificity'] = {}
                results_dict[name][rho]['AUC'] = {}

            results_dict[name][rho]['Misclassification error'][lam] = error
            results_dict[name][rho]['Sensitivity'][lam] = sensitivity
            results_dict[name][rho]['Specificity'][lam] = specificity
            results_dict[name][rho]['AUC'][lam] = auc
print('Best lambda and rho:', best_lam, best_rho)
print('Sensitivity and Specificity on the test set '
      'for these parameter values: %0.2f, %0.2f'%(
      results_dict[name][best_rho]['Sensitivity'][best_lam],
```

```
      results_dict[name][best_rho]['Specificity'][best_lam]))
```

```
Best lambda and rho: 0.0009765625 1.0
Sensitivity and Specificity on the test set for these parameter
values: 0.71, 0.82
```

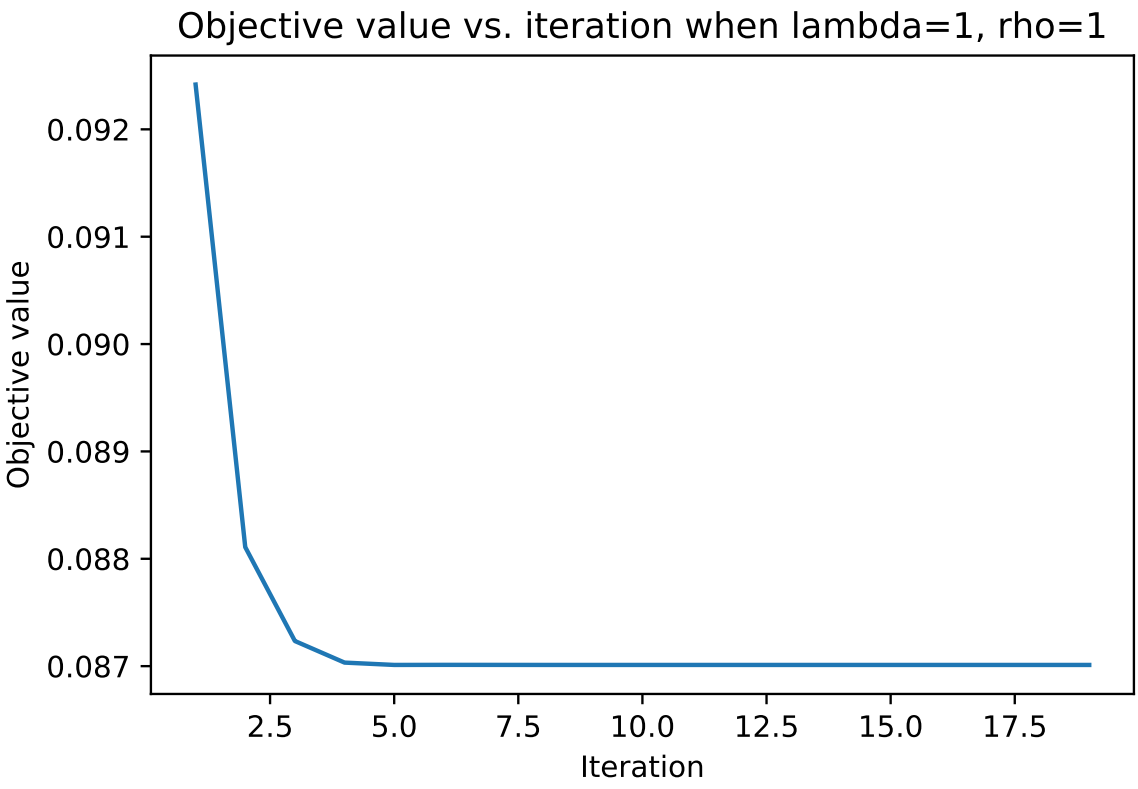
- Optional. Repeat all the above with obvious changes with a negative class "-1" with examples from all classes except positive class.

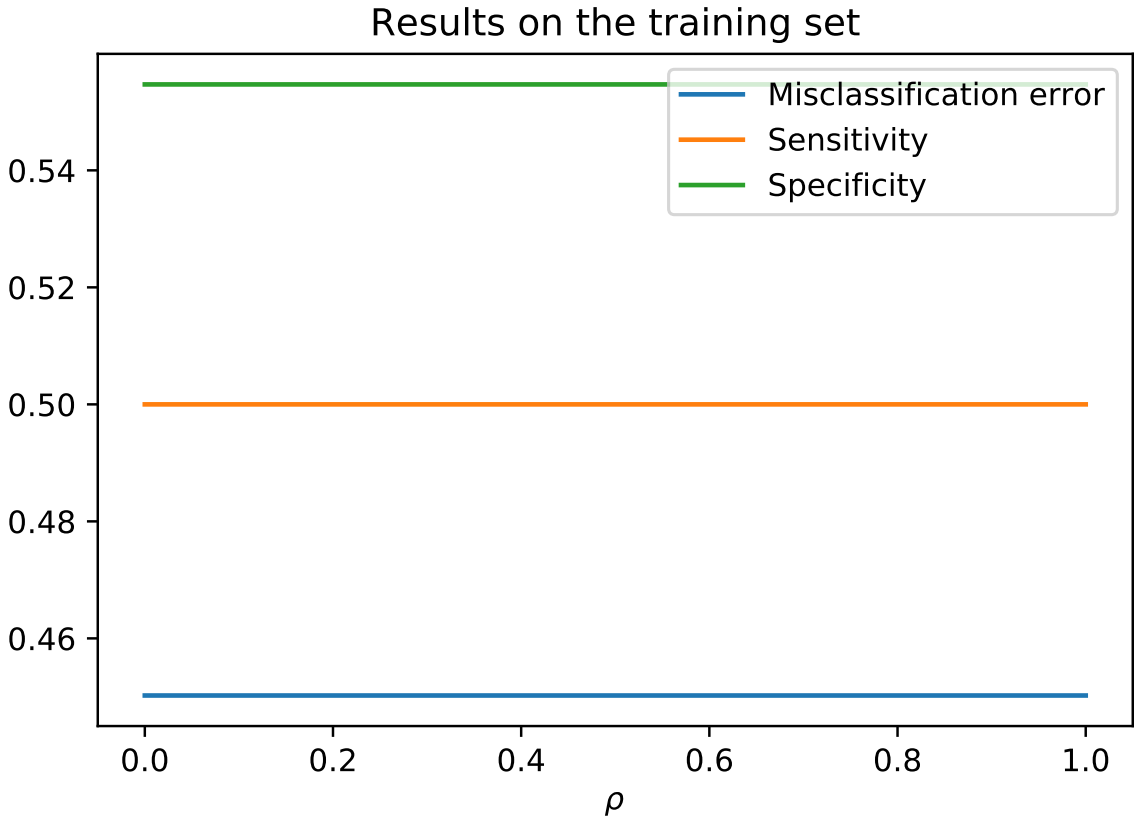
```
# Part 9: Repeat with all examples except class 1 as class -1
x_train = np.asarray(train_set)[:, 2:]
y_train = np.asarray(train_set)[:, 1]

x_test = np.asarray(test_set)[:, 2:]
y_test = np.asarray(test_set)[:, 1]

# Keep track of the number of samples and dimension of each sample
n_train = len(y_train)
n_test = len(y_test)
d = np.size(x_train, 1)

y_train = (y_train == 11)*2-1
y_test = (y_test == 11)*2-1

ntrain = int(len(x_train)*0.8)
x_valid = x_train[ntrain:]
y_valid = y_train[ntrain:]
x_train = x_train[:ntrain]
y_train = y_train[:ntrain]

xbar = np.mean(x_train, axis=0)
xstd = np.std(x_train, axis=0)
x_train = (x_train - xbar)/xstd[np.newaxis, :]
x_valid = (x_valid - xbar)/xstd[np.newaxis, :]
x_test = (x_test - xbar)/xstd[np.newaxis, :]

lambduh = 1
rho = 1
beta_init = np.zeros(d)
theta_init = np.zeros(d)
max_eig = scipy.linalg.eigh(1 / len(y_train) * \
    x_train.T.dot(x_train), eigvals=(d - 1, d - 1),
    eigvals_only=True)[0]
eta_init = 1 / (max_eig + lambduh)
maxiter = 100

beta_vals = mylinearsvm(beta_init, theta_init, lambduh,
```

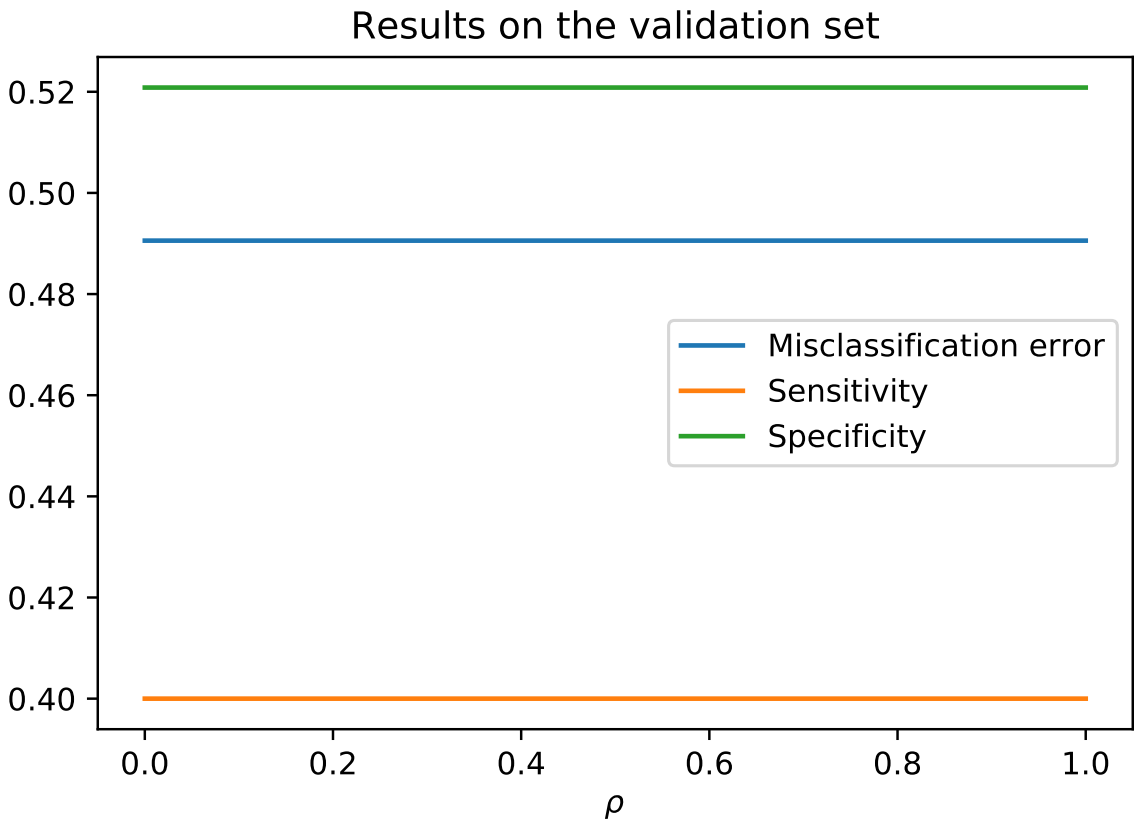
```
                        rho, eta_init, maxiter)
objective_plot(beta_vals, lambduh, rho)

for (name, x, y) in zip(('training', 'validation', 'test'),
                        (x_train, x_valid, x_test),
                        (y_train, y_valid, y_test)):
    error = misclassification_error(beta_vals[-1], x=x, y=y)
    sensitivity, specificity = sensitivity_specificity(beta_vals[-1],
                                                       x=x, y=y)
    print('Misclassification error on the ' + name +
          ' set: %0.2f'% error)
    print('Sensitivity and specificity on the ' + name +
          ' set: %0.2f, %0.2f'%(sensitivity, specificity))

rhos = np.arange(0, 1.1, 0.1)
results_dict = {}
for rho in rhos:
    beta_vals = mylinearsvm(beta_init, theta_init, lambduh,
                        rho, eta_init, maxiter)
    for (name, x, y) in zip(('training', 'validation', 'test'),
                            (x_train, x_valid, x_test),
                            (y_train, y_valid, y_test)):
        error = misclassification_error(beta_vals[-1], x=x, y=y)
        sensitivity, specificity = sensitivity_specificity(beta_vals[-1],
                                                           x=x, y=y)
        if name not in results_dict:
            results_dict[name] = {}
            results_dict[name]['Misclassification error'] = [error]
            results_dict[name]['Sensitivity'] = [sensitivity]
            results_dict[name]['Specificity'] = [specificity]
        else:
            results_dict[name]['Misclassification error'].append(error)
            results_dict[name]['Sensitivity'].append(sensitivity)
            results_dict[name]['Specificity'].append(specificity)

for name in ['training', 'validation', 'test']:
    plot_metric_results(results_dict, rhos, name)

lambdas = [2**i for i in range(-20, 11)]
best_auc = 0
results_dict = {}

for lam in lambdas:
    eta_init = 1 / (max_eig + lambduh)
    for rho in rhos:
        beta_vals = mylinearsvm(beta_init, theta_init, lam,
                            rho, eta_init, maxiter)
```

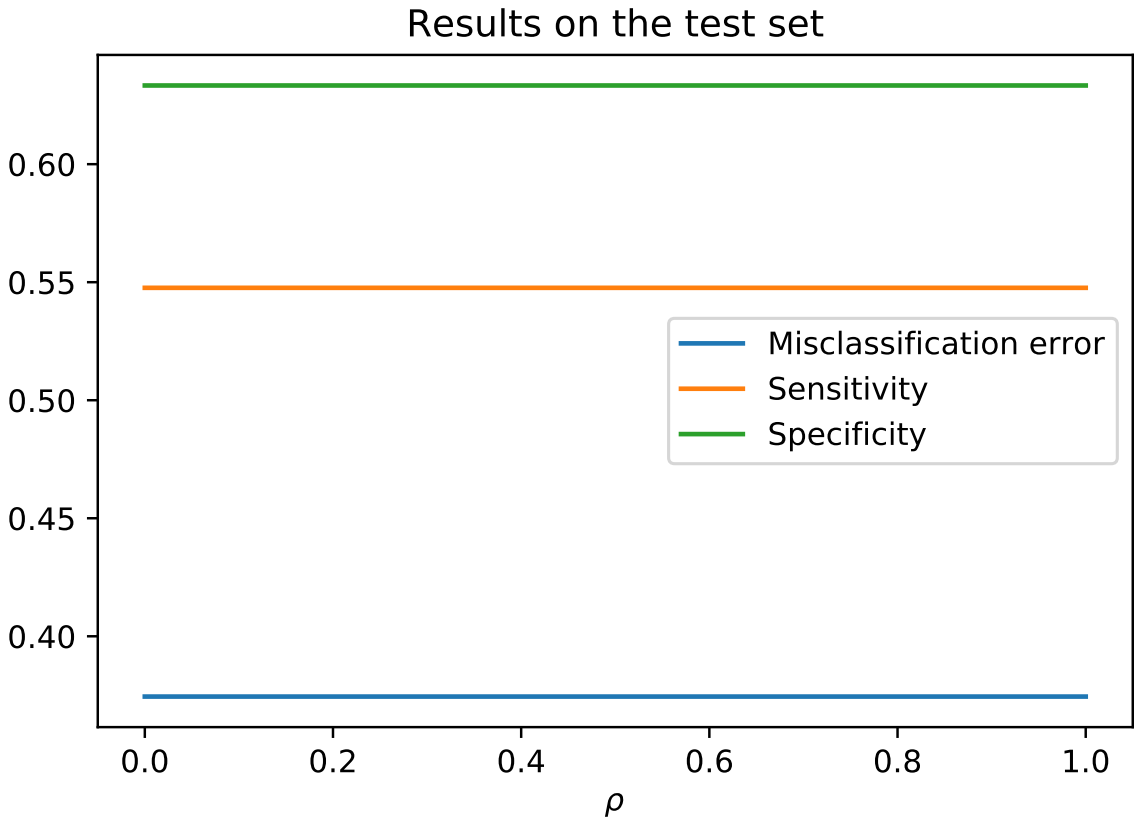
```
        for (name, x, y) in zip(('training', 'validation', 'test'),
                                (x_train, x_valid, x_test),
                                (y_train, y_valid, y_test)):
            error = misclassification_error(beta_vals[-1], x=x, y=y)
            sensitivity, specificity = sensitivity_specificity(
                                        beta_vals[-1], x=x, y=y)

            auc = compute_auc(beta_vals[-1], x, y)
            if name == 'validation' and auc > best_auc:
                best_auc = auc
                best_lam = lam
                best_rho = rho

            if name not in results_dict or rho not in results_dict[name]:
                if name not in results_dict:
                    results_dict[name] = {}
                results_dict[name][rho] = {}
                results_dict[name][rho]['Misclassification error'] = {}
                results_dict[name][rho]['Sensitivity'] = {}
                results_dict[name][rho]['Specificity'] = {}
                results_dict[name][rho]['AUC'] = {}

            results_dict[name][rho]['Misclassification error'][lam] = error
            results_dict[name][rho]['Sensitivity'][lam] = sensitivity
            results_dict[name][rho]['Specificity'][lam] = specificity
            results_dict[name][rho]['AUC'][lam] = auc

print('Best lambda and rho:', best_lam, best_rho)
print('Sensitivity and Specificity on the test set '
      'for these parameter values: %0.2f, %0.2f'%(
      results_dict[name][best_rho]['Sensitivity'][best_lam],
      results_dict[name][best_rho]['Specificity'][best_lam]))
```

```
Misclassification error on the training set: 0.45
Sensitivity and specificity on the training set: 0.50, 0.55
Misclassification error on the validation set: 0.49
Sensitivity and specificity on the validation set: 0.40, 0.52
Misclassification error on the test set: 0.37
Sensitivity and specificity on the test set: 0.55, 0.63
Best lambda and rho: 3.0517578125e-05 0.0
Sensitivity and Specificity on the test set for these parameter
values: 0.31, 0.60
```


Objective value vs. iteration when lambda=1, rho=1
Objective value
0.092
0.091
0.090
0.089
0.088
0.087
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Iteration


Results on the training set
Misclassification error
Sensitivity
Specificity
0.54
0.52
0.50
0.48
0.46
0.0
0.2
0.4
0.6
0.8
1.0
ρ


Results on the validation set
0.52
0.50
0.48
0.46
0.44
0.42
0.40
0.0
0.2
0.4
0.6
0.8
1.0
ρ
Misclassification error
Sensitivity
Specificity


Results on the test set
0.60
0.55
0.50
0.45
0.40
0.0
0.2
0.4
0.6
0.8
1.0
ρ
Misclassification error
Sensitivity
Specificity

## Exercise 3

In this exercise, you are facing several challenging situations and you are asked to suggest effective strategies to tackle them, based on your knowledge in statistical machine learning and data science in general. Please answer the questions for each situation in a few lines. Please feel free to write equations or draw figures if it helps.

**Linear regression insomnia.** Alex is Bobbie's manager. Alex is a workaholic and Bobbie suffers from insomnia. Bobbie wants to design a deep network to predict the covid-19 pandemic evolution. At 3am, Bobbie receives a text from Alex asking her to log in her computer for a zoom meeting. Alex wants Bobbie to design, implement, run, and validate a linear regression model on a dataset. Bobbie has 2hrs to deliver the model and the predictions to Alex. Tyler adamantly forbids Frankie to use any off-the-shelf package because of IP concerns. Bobbie calls **you** through zoom and asks for your advice regarding the best optimization algorithm to use in this case, since you took DATA 558 at UW. Which optimization algorithm do you advise your friend Bobbie to use? Explain why.
Fast gradient because the linear regression objective is convex and smooth.

**The next big thing.** Joey is a colleague of yours. Joey has been working with Tyler on a binary classification task for a particular real-world problem. Joey and Tyler have collected a dataset to work on this classification task. Joey sends you a zoom meeting request. While his kid is jumping on his lap, he claims he has just revolutionized machine learning. You ask him to tell you more. Joey says his method manages to predict perfectly on the test set. A striking feature of Joey's revolutionary approach is to use the test set (examples and labels) at training time. Joey says he did some research and apparently no one has ever applied this strategy in research papers or books. What do you think about Joey's revolutionary approach? How much would you invest in Joey's new company?
One should not train on the test set!