Assigment 4

This is a mini-project assignment that includes only programming questions. You are asked to implement optimization algorithms for ML classification problems.

Marking of this assignment will be based on the correctness of your ML pipeline and efficiency of your code.

Upload your code on Learn dropbox and submit pdfs of the code and to Crowdmark.

--

In [1]:
```
# !pip install numpy, scipy, sys
```

Suggested way of loading data to python for the assigment. There are alternatives of course, you can use your preferred way if you want.

In [2]:
```
# Download the LIBSVM package from here: https://www.csie.ntu.edu.tw/
~cjlin/libsvm/#download
# If your download is successfull you should have the folder with nam
e: libsvm-3.24.
# We will use this package to load datasets.

# Enter the downloaded folder libsvm-3.24 through your terminal.
# Run make command to compile the package.

# Load this auxiliary package.
import sys

# add here your path to the folder libsvm-3.24/python
path = "/home/tempo/Desktop/Fall 2019/CS 794/a4/libsvm-3.24/python"
# Add the path to the Python paths so Python can find the module.
sys.path.append(path)

# Load the LIBSVM module.
from svmutil import *

# Add here your path to the folder libsvm-3.24
path = "/home/tempo/Desktop/Fall 2019/CS 794/a4/libsvm-3.24"

# Test that it works. This will load the data "heart_scale"
# and it will store the labels in "b" and the data matrix in "A".
b, A = svm_read_problem(path + '/heart_scale')
print('Loaded data: Heart Scale')
# Use "svm_read_problem" function to load data for your assignment.

# Note that matrix "A" stores the data in a sparse format.
# In particular matrix "A" is a list of dictionaries.
# The length of the list gives you the number of samples.
# Each entry in the list is a dictionary. The keys of the dictionary
 are the non-zero features.
# The values of the dictionary for each key is a list which gives you
the feature value.
```

```
Loaded data: Heart Scale
```

Load other useful modules

In [15]:
```
import matplotlib.pyplot as plt

# Numpy is useful for handling arrays and matrices.
import numpy as np
from scipy.sparse import coo_matrix
import time
from random import randrange as rnd
import gc
```

Datasets that you will need for this assignment.

In [3]:
```
# There is an extended selection of classification and regression dat
asets
# https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

# Out of all these datasets you will need the following 3 datasets, w
hich are datasets for classification problems.
#
# a9a dataset: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/dataset
s/binary.html#a9a
# This dataset is small, it is recommened to start your experiments w
ith this dataset.
#
# news20.binary dataset: https://www.csie.ntu.edu.tw/~cjlin/libsvmtoo
ls/datasets/binary.html#news20.binary
#
# covtype.binary dataset: https://www.csie.ntu.edu.tw/~cjlin/libsvmto
ols/datasets/binary.html#covtype.binary
#
# Exploit the sparsity of the problem when you implement optimization
methods.
```

In [20]:
```
path_a9a = r'/home/tempo/Desktop/Fall 2019/CS 794/a4/dataset/a9a'
b_a9a, A_a9a = svm_read_problem(path_a9a) #(32561, 123)

path_news20 = r'/home/tempo/Desktop/Fall 2019/CS 794/a4/dataset/news2
0.binary'
b_news20, A_news20 = svm_read_problem(path_news20) #(19996, 1355191)

path_cov = r'/home/tempo/Desktop/Fall 2019/CS 794/a4/dataset/covtype.
libsvm.binary.scale'
b_cov, A_cov = svm_read_problem(path_cov) #(581012, 54)

def splitData(A, b):
    lenA = A.shape[0]
    split = lenA//10

    ATrain = A[0:lenA - split]
    AValid = A[lenA - split:]
    bTrain = b[0:lenA - split]
    bValid = b[lenA - split:]

    return ATrain, AValid, bTrain, bValid

def toSparse(A, cols):
    row = []
    col = []
    data = []
    for i in range(len(A)):
        for key, val in A[i].items():
            row.append(i)
            col.append(key - 1)
            data.append(val)
    return coo_matrix((data, (row, col)), shape=(len(A), cols)).tocsr
()

def bConverter(b):
    for i in range(len(b)):
        if (b[i] == 2):
            b[i] = -1
    return b

A_a9a = toSparse(A_a9a, 123)
A_news20 = toSparse(A_news20, 1355191)
A_cov = toSparse(A_cov, 54)
b_cov = bConverter(b_cov)

b_a9a = np.array(b_a9a)
b_news20 = np.array(b_news20)
b_cov = np.array(b_cov)

print(A_a9a.shape)
print(A_news20.shape)
print(A_cov.shape)

ATrain_a9a, AValid_a9a, bTrain_a9a, bValid_a9a = splitData(A_a9a, b_a
9a)
ATrain_news20, AValid_news20, bTrain_news20, bValid_news20 = splitDat
```

```
a(A_news20, b_news20)
ATrain_cov, AValid_cov, bTrain_cov, bValid_cov = splitData(A_cov, b_c
ov)

print('All data sets loaded')
```

```
(32561, 123)
(19996, 1355191)
(581012, 54)
All data sets loaded
```

In [21]:

```
beta0 = -0.19353561519374984#np.random.uniform(-0.5, 0.5)
x0_a9a = np.random.uniform(-1, 1, size = (ATrain_a9a.shape[1], 1))
x0_news20 = np.random.uniform(-1, 1, size = (ATrain_news20.shape[1],
1))
x0_cov = np.random.uniform(-1, 1, size = (ATrain_cov.shape[1], 1))

# np.savetxt('x0_a9a', x0_a9a, delimiter = ',')

# np.savetxt('x0_news20', x0_news20, delimiter = ',')

# np.savetxt('x0_cov', x0_cov, delimiter = ',')

# x0_a9a = np.loadtxt('x0_a9a', delimiter = ',')
# x0_a9a = x0_a9a.reshape((ATrain_a9a.shape[1], 1))

# x0_news20 = np.loadtxt('x0_news20', delimiter = ',')
# x0_news20 = x0_news20.reshape((ATrain_news20.shape[1], 1))

# x0_cov = np.loadtxt('x0_cov', delimiter = ',')
# x0_cov = x0_cov.reshape((ATrain_cov.shape[1], 1))
```

Training, Validation and Testing data

In [ ]:

```
# All datasets above consist of training and testing data.

# You should seperate the training data into training and validation
 data.
# Follow the instructions from the lectures about how you can use bot
h training and validation data.
# You can use 10% of the training data as validation data and the rem
aining 90% to train the models.
# This is a suggested percentage, you can do otherwise if you wish.

# Do not use the testing data to influence training in any way. Do no
t use the testing data at all.
# Only your instructor and TA will use the testing data to measure ge
neralization error.
# If you do use the testing data to tune parameters or for training o
f the algorithms we will figure it out :-).
```

Optimization problems

You need to solve the following optimization problems

Hinge-loss

$$\text{minimize}_{x\in\mathbb{R}^d,\beta\in\mathbb{R}} \frac{1}{n}\sum_{i=1}^{n}\max\{0, 1 - b_i(a_i^T x + \beta)\},$$

where $a_i \in \mathbb{R}^d$ is the feature vector for sample i and b_i is the label of sample i. The sub-gradient of the hinge-loss is given in the lecture slides (note that there is a small difference due to the intercept β). A smooth approximation of the function $f(z) := \max\{0, 1-z\}$ is given by

$$\psi_\mu(z) = \begin{cases} 0 & z \geq 1 \\ (1-z)^2 & \mu < z < 1 \\ (1-\mu)^2 + 2(1-\mu)(\mu - z) & z \leq \mu. \end{cases}$$

You can use the smooth approximation $\psi_\mu(z)$ for methods that work only for smooth functions. For sub-gradient methods you should use the sub-gradient.

L2-regularized logistic regression

$$\text{minimize}_{x\in\mathbb{R}^d,\beta\in\mathbb{R}} \lambda\|x\|_2^2 + \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-b_i(a_i^T x + \beta))).$$

This is a smooth objective function, therefore, you should use gradient methods to solve it. You do not need sub-gradient methods for this problem.

In [7]:
```
#Objective Functions
def hingeLossF(x, a, b, beta):
    n = a.shape[0]
    s = a * x
    sumHinge = 0
    z = b * (s + beta)
    for i in range(n):
        sumHinge += max(0, 1 - z[i])
    return sumHinge / n

def hingeLossSmoothF(x, a, b, beta, u):
    n = a.shape[0]
    s = a * x
    sumSmoothHinge = 0
    z = b * (s + beta)
    for i in range (n):
        if (z[i] >= 1):
            phiZ = 0
        elif (u < z[i] and z[i] < 1):
            phiZ = (1 - z[i]) ** 2
        elif (z[i] <= u):
            phiZ = (1 - u) ** 2 + 2 * (1 - u) * (u - z[i])
        sumSmoothHinge += phiZ
    return sumSmoothHinge / n

def logisticF(lambda_, x, a, b, beta):
    n = a.shape[0]
    s = a * x
    norm = np.linalg.norm(x) ** 2
    sumLogistic = np.sum(np.log(1 + np.exp(-b * (s + beta))))
    return lambda_ * norm + sumLogistic / n
```

Optimization algorithms

In [8]:
```
# For this assignment you will need the following methods

# 1) Stochastic sub-gradient
# 2) Stochastic gradient
# 3) Mini-batch (sub-)gradient (you will have to decide what batching
strategy to use, see lecture slides)
# 4) Stochastic average sub-gradient (SAG)
# 5) Stochastic average gradient (SAG)
# 6) Gradient descent with Armijo line-search
# 7) Acceleratd gradient with Armijo line-search (the same method as
 Q5 in Assignemnt 3)

# Information is provided in the lecture slides about parameter tunin
g and termination.
# However, the final decision of any parameter tuning and termination
criteria is up to the students to make.
```

In [9]:
```
#Calculate the regularization param: 2 * lambda * x
def regularization(lambda_, x):
    return 2 * lambda_ * x
```

Validation error: measure the validation error by calculating

$$\frac{1}{t}\sum_{i\in\text{validation data}}\left|\, b_i^{\text{your model}} - b_i^{\text{true}} \,\right|$$

where t is the number of samples in your validation set. b_i^{true} is the true label of the i-th sample. $b_i^{\text{your model}}$ is the label of the i-th sample of your model.

For hinge loss calculate

$$b_i^{\text{your model}} := \text{sign}(a_i^T x + \beta).$$

For logistic regression calculate the predicted label by

$$b_i^{\text{your model}} = \begin{cases} 1 & \text{if } \frac{1}{1+e^{-(a_i^T x+\beta)}} > 0.5 \\ -1 & \text{otherwise} \end{cases}$$

In [10]:
```
#Calculate the Validation error
def bHinge(a, x, beta):
    bModel = np.zeros((a.shape[0]))
    s = a * x
    for i in range(a.shape[0]):
        f = s[i] + beta
        if (f < 0):
            bModel[i] = -1
        elif (f > 0):
            bModel[i] = 1
    return bModel

def bLogistic(a, x, beta):
    bModel = np.zeros((a.shape[0]))
    s = a * x
    for i in range(a.shape[0]):
        if (1/ (1 + np.exp(-(s[i] + beta)))) > 0.5:
            bModel[i] = 1
        else:
            bModel[i] = -1
    return bModel

def validationError(a, x, beta, bTrue, lossType):
    t = len(bTrue)
    totalError = 0
    if (lossType == 'l'):
        bModel = bLogistic(a, x, beta)
    else:
        bModel = bHinge(a, x, beta)
    for i in range (t):
        totalError += np.abs(bModel[i] - bTrue[i])
    return totalError / t
```

In [18]:
```
def plotData(a9a, a9a_time, news20, news20_time, cov, cov_time, fType
, method):
    f_a9a = []
    f_news20 = []
    f_cov = []
    for i in range(len(a9a)):
        if (fType == 'l'):
            f_a9a.append([logisticF(0.1, a9a[i][0], ATrain_a9a, bTrai
n_a9a, a9a[i][1]), a9a_time[i]])
            gc.collect()
        elif (fType == 's'):
            f_a9a.append([hingeLossSmoothF(a9a[i][0], ATrain_a9a, bTr
ain_a9a, a9a[i][1], 0.1), a9a_time[i]])
        else:
            f_a9a.append([hingeLossF(a9a[i][0], ATrain_a9a, bTrain_a9
a, a9a[i][1]), a9a_time[i]])

    np.savetxt('plot/a9a_' + f_type + '_' + method, f_a9a, delimiter
= ',')

    for i in range(len(news20)):
        if (fType == 'l'):
            f_news20.append([logisticF(0.1, news20[i][0], ATrain_news
20, bTrain_news20, news20[i][1]), news20_time[i]])
        elif (fType == 's'):
            f_news20.append([hingeLossSmoothF(news20[i][0], ATrain_ne
ws20, bTrain_news20, news20[i][1], 0.1), news20_time[i]])
        else:
            f_news20.append([hingeLossF(news20[i][0], ATrain_news20,
bTrain_news20, news20[i][1]), news20_time[i]])

    np.savetxt('plot/news20_' + f_type + '_' + method, f_news20, deli
miter = ',')

    for i in range(len(cov)):
        if (fType == 'l'):
            f_cov.append([logisticF(0.1, cov[i][0], ATrain_cov, bTrai
n_cov, cov[i][1]), cov_time[i]])
        elif (fType == 's'):
            f_cov.append([hingeLossSmoothF(cov[i][0], ATrain_cov, bTr
ain_cov, cov[i][1], 0.1), cov_time[i]])
        else:
            f_cov.append([hingeLossF(cov[i][0], ATrain_cov, bTrain_co
v, cov[i][1]), cov_time[i]])

    np.savetxt('plot/cov_' + f_type + '_' + method, f_cov, delimiter
= ',')
```

Question 1: Use the ML pipeline that is mentioned in slide 60 of Lecture 11 to train your model for the logistic regression problem (the hinge-loss problem does not have any hyper-parameters). Pick any algorithm that you want from the above suggested list to train the models. Report your ML pipeline. Print your Generalization Error. We will not measure running time for this pipeline. Running time will be measure only in Q2. Marks: 30.

For Q1, I implemented Stochastic gradient with logistic regression

In [24]:
```
logR_sto_grad_a9a = []
time_logR_sto_grad_a9a = []
logR_sto_grad_news20 = []
time_logR_sto_grad_news20 = []
logR_sto_grad_cov = []
time_logR_sto_grad_cov = []

def stocasticGradLogistic(i, lambda_, A, b, beta, x):
    A_i = A[i]
    b_i = b[i]
    n = A.shape[0]
    aTx = A_i * x
    sigma = -b_i/(1 + np.exp(b_i * (aTx + beta)))
    g = A_i.T * sigma
    return g/n, sigma/n

def logisticStocasticGrad(x0, beta, A, b, data):

    xCurrent = x0
    alpha0 = 0.1
    alpha = alpha0
    lambda_ = 0.1

    start = time.time()
    for i in range (200):
        ind = rnd(A.shape[0])
        g, sigma = stocasticGradLogistic(ind, lambda_, A, b, beta, xC
urrent)
        norm = regularization(lambda_, xCurrent)

        end = time.time() - start
        if (data == 'a9a'):
            logR_sto_grad_a9a.append([xCurrent, beta])
            time_logR_sto_grad_a9a.append(end)
        elif(data == 'news20'):
            logR_sto_grad_news20.append([xCurrent, beta])
            time_logR_sto_grad_news20.append(end)
        else:
            logR_sto_grad_cov.append([xCurrent, beta])
            time_logR_sto_grad_cov.append(end)

        xCurrent = xCurrent - alpha * (norm + g)
        beta = beta - alpha * sigma
        alpha = alpha0/np.sqrt(i + 1)
    return xCurrent, beta

print('a9a')
start = time.time()
x, beta = logisticStocasticGrad(x0_a9a, beta0, ATrain_a9a, bTrain_a9a
, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'l')
print('Error:', error)

print('News 20')
```

```
start = time.time()
x, beta = logisticStocasticGrad(x0_news20, beta0, ATrain_news20, bTra
in_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'l')
print('Error:', error)

print('Cov Type')
start = time.time()
x, beta = logisticStocasticGrad(x0_cov, beta0, ATrain_cov, bTrain_cov
, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'l')
print('Error:', error)

plotData(logR_sto_grad_a9a, time_logR_sto_grad_a9a,
        logR_sto_grad_news20, time_logR_sto_grad_news20,
        logR_sto_grad_cov, time_logR_sto_grad_cov, 'l', 'sto_grad')
print('Data Saved')
```

```
a9a
Time: 0.05736064910888672 s
Error: 0.5761670761670762
News 20
Time: 3.523648500442505 s
Error: 0.3601800900450225
Cov Type
Time: 0.05550551414489746 s
Error: 0.838884012323368
```

I initially chose my beta0 to be in the range (-1, 1), and the same for my x0 values After several tests, I found the best combination for all of them as well as lambda, which was 0.1 I saved those values in a file so I can use them to run the methods.

Question 2: Plot the objective function (y-axis) vs running time in sec (x-axis). Have one plot for each optimization problem. In each plot show the performance of all relevant algorithms. For each plot use the parameter setting that gives you the best validation error in Q1 (this refers to the logistic regression probelm). Do not show plots for all parameter settings that you tried in Q1, only for the one that gives you the smallest validation error. Do not include computation of any plot data in the computation of the running time of the algorithm, unless the plot data are computed by the algorithm anyway. Make sure that the plots are clean and use appropriate legends. Note that we should be able to re-run the code and obtain the plots. Marks: 70.

For this question, we will measure the running time of your stochastic sub-gradient method for the sparse dataset news20.binary for the hinge-loss problem. We will not measure the running time of any other combination of algorithm, dataset, problem. You need to implement the stochastic sub-gradient method and encapsulate it in a python class.

To make sure your object can be used by our script, your class should have two methods:

1. **fit(self, train_data, train_label)**. It will use stochastic sub-gradient method to minimize the hinge loss and store the optimized coefficients (i.e. x, β) in the instance. The "train_data" and "train_label" are similar to the output of "svm_read_problem".

    - "train_data" is a list of n python dictionaries (int -> float), which presents a sparse matrix. The keys (int) and values (float) in the dictionary at train_data[i] are the indices (int) and values (float) of non-zero entries of row i.
    - "train_label" is a list of n integers, it only has **-1s and 1s**. n is the number of samples. This function returns nothing.

1. **predict(self, test_data)**. It will predict the label of the input "test_data" by using the coefficients stored in the instance. The "test_data" has the same data structure as the "train_data" of the "fit" function. This function returns a list of **-1s and 1s** (i.e. the prediction of your labels).

You can also define other methods to help your programming, we will only call the two methods decribed above.

To let us import your class, you need to follow these rules:

1. You should name your python file by **a4_[your student ID].py**. For example, if your student id is 12345, then your file name is **a4_12345.py**
2. Your object name should be **MyMethod** (it's case sensitive).

Any violation of the above requirements will get error in our script and you will get at most 50% of the total score. Your solution will be mainly measured by the runing time of the **fit** function and the accuracy of the **predict** function. For example your method will be called and measured in following pattern:

```
obj = MyMethod()
st = time.time()
obj.fit(train_data, train_label) # .fit() optimizes the objective and stores
coefficients in obj.
running_time = time.time() - st
predict_label = obj.predict(test_data)
accuracy = get_accuracy(predict_label, test_label) # this is a function we u
se to measure accuracy.
```

Then your accuracy will be measured by **predict_labels**, you don't have to implement "get_accuracy". When you finish your implementation, upload the .py file to Learn dropbox.

In [10]:
```
# 2) Stochastic gradient for Smooth hinge loss
smooth_sto_grad_a9a = []
time_smooth_sto_grad_a9a = []
smooth_sto_grad_news20 = []
time_smooth_sto_grad_news20 = []
smooth_sto_grad_cov = []
time_smooth_sto_grad_cov = []

def stocasticGradSmooth(i, A, b, beta, x, u):
    A_i = A[i]
    b_i = b[i]
    aTx = A_i * x
    z = b_i * (aTx + beta)
    if (z.item() >= 1):
        g = np.zeros((x.shape[0], 1))
        sigma = 0
    elif (u < z and z < 1):
        sigma = -2 * b_i * (1 - b_i * (aTx + beta))
        g = A_i.T * sigma
    elif (z <= u):
        sigma = -2 * b_i * (1 - u)
        g = A_i.T * sigma
    return g, sigma

def smoothStocasticGrad(x0, beta, A, b, data):
    xCurrent = x0
    alpha0 = 0.1
    alpha = alpha0
    u = 0.1

    start = time.time()
    for i in range (200):
        ind = rnd(A.shape[0])
        g, sigma = stocasticGradSmooth(ind, A, b, beta, xCurrent, u)

        end = time.time() - start
        if (data == 'a9a'):
            smooth_sto_grad_a9a.append([xCurrent, beta])
            time_smooth_sto_grad_a9a.append(end)
        elif(data == 'news20'):
            smooth_sto_grad_news20.append([xCurrent, beta])
            time_smooth_sto_grad_news20.append(end)
        else:
            smooth_sto_grad_cov.append([xCurrent, beta])
            time_smooth_sto_grad_cov.append(end)

        xCurrent = xCurrent - alpha * g

        beta = beta - alpha * sigma

        alpha = alpha0 / np.sqrt(i + 1)
    return xCurrent, beta

print('a9a')
start = time.time()
x, beta = smoothStocasticGrad(x0_a9a, beta0, ATrain_a9a, bTrain_a9a,
```

```
'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'h')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = smoothStocasticGrad(x0_news20, beta0, ATrain_news20, bTrain
_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'h')
print('Error:', error)

print('Cov type')
start = time.time()
x, beta = smoothStocasticGrad(x0_cov, beta0, ATrain_cov, bTrain_cov,
'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'h')
print('Error:', error)

plotData(smooth_sto_grad_a9a, time_smooth_sto_grad_a9a,
        smooth_sto_grad_news20, time_smooth_sto_grad_news20,
        smooth_sto_grad_cov, time_smooth_sto_grad_cov, 's', 'sto_gra
d')
print('Data saved')
```

```
a9a
Time: 0.08361363410949707 s
Error: 0.5528255528255528
News 20
Time: 2.124379873275757 s
Error: 1.03951975987994
Cov type
Time: 0.1094977855682373 s
Error: 0.9411886198172148
```

In [11]:
```
#1) Sub grad for Hinge loss -> this will be tested in MyMethod
sto_sub_grad_a9a = []
time_sto_sub_grad_a9a = []
sto_sub_grad_news20 = []
time_sto_sub_grad_news20 = []
sto_sub_grad_cov = []
time_sto_sub_grad_cov = []

def stocasticSubGradHinge(i, A, b, beta, x):
    A_i = A[i]
    b_i = b[i]
    s = 1 - b_i * (A_i * x + beta)
    if (s > 0):
        g = -A_i.T * b_i
        sigma = -b_i
    else:
        g = np.zeros((x.shape[0], 1))
        sigma = 0
    return g, sigma

def hingeStocasticSubGrad(x0, beta, A, b, data):
    xCurrent = x0
    alpha0 = 0.1
    alpha = alpha0

    start = time.time()
    for i in range (200):
        ind = rnd(A.shape[0])
        g, sigma = stocasticSubGradHinge(ind, A, b, beta, xCurrent)

        end = time.time() - start
        if (data == 'a9a'):
            sto_sub_grad_a9a.append([xCurrent, beta])
            time_sto_sub_grad_a9a.append(end)
        elif(data == 'news20'):
            sto_sub_grad_news20.append([xCurrent, beta])
            time_sto_sub_grad_news20.append(end)
        else:
            sto_sub_grad_cov.append([xCurrent, beta])
            time_sto_sub_grad_cov.append(end)

        xCurrent = xCurrent - alpha * g

        beta = beta - alpha * sigma

        alpha = alpha0 / (i + 1)
    return xCurrent, beta

print('a9a')
start = time.time()
x, beta = hingeStocasticSubGrad(x0_a9a, beta0, ATrain_a9a, bTrain_a9a
, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'h')
print('Error:', error)
```

```
print('News 20')
start = time.time()
x, beta = hingeStocasticSubGrad(x0_news20, beta0, ATrain_news20, bTra
in_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'h')
print('Error:', error)

print('Cov Type')
start = time.time()
x, beta = hingeStocasticSubGrad(x0_cov, beta0, ATrain_cov, bTrain_cov
, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'h')
print('Error:', error)

plotData(sto_sub_grad_a9a, time_sto_sub_grad_a9a,
        sto_sub_grad_news20, time_sto_sub_grad_news20,
        sto_sub_grad_cov, time_sto_sub_grad_cov, 'h', 'sto_sub_grad')
print('Data saved')
```

```
a9a
Time: 0.07877326011657715 s
Error: 0.5018427518427518
News 20
Time: 2.3377866744995117 s
Error: 0.9694847423711856
Cov Type
Time: 0.13758444786071777 s
Error: 0.8091426997814152
```

In [12]:
```
# 3) Mini-batch gradient for Hinge Loss
mini_sub_grad_a9a = []
time_mini_sub_grad_a9a = []
mini_grad_news20 = []
time_mini_grad_news20 = []
mini_grad_cov = []
time_mini_grad_cov = []

def batchGradHinge(A, b, x, beta, size):
    gX = np.zeros((A.shape[1],1))
    sigma = 0

    for i in range(size):
        aTx_beta = A[i]*x + beta
        s = 1 - (b[i] * aTx_beta)

        if s > 0:
            temp = -A[i].T * b[i]
            temp = temp.reshape((temp.shape[0], 1))
            gX += -temp
            sigma += -b[i]

    g = gX / size
    return g, sigma / size


def hingeMiniB (A, b, beta, x0, data):
    xCurrent = x0
    alpha0 = 1
    alpha = alpha0
    size = 100
    const = 0
    b = np.asarray(b)
    b = b.reshape((len(b), 1))

    start = time.time()
    for i in range(200):
        j = np.random.choice(np.arange(A.shape[0]), size, replace=Fal
se)
        A_j = A[j]
        b_j = b[j]
        g, sigma = batchGradHinge(A, b, xCurrent, beta, size)

        end = time.time()
        if (data == 'a9a'):
            mini_sub_grad_a9a.append([xCurrent, beta])
            time_mini_sub_grad_a9a.append(end)
        elif(data == 'news20'):
            mini_grad_news20.append([xCurrent, beta])
            time_mini_grad_news20.append(end)
        else:
            mini_grad_cov.append([xCurrent, beta])
            time_mini_grad_cov.append(end)

        xCurrent = xCurrent - alpha * g
```

```
        beta = beta - alpha * sum(sigma)

        alpha = alpha0/np.sqrt(i + 1)

        size = size + const

    return xCurrent, beta

print('a9a')
start = time.time()
x, beta = hingeMiniB(ATrain_a9a, bTrain_a9a, beta0, x0_a9a, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'h')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = hingeMiniB(ATrain_news20, bTrain_news20, beta0, x0_news20,
'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'h')
print('Error:', error)

print('Cov Type')
start = time.time()
x, beta = hingeMiniB(ATrain_cov, bTrain_cov, beta0, x0_cov, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'h')
print('Error:', error)

plotData(mini_sub_grad_a9a, time_mini_sub_grad_a9a,
        mini_grad_news20, time_mini_grad_news20,
        mini_grad_cov, time_mini_grad_cov, 'h', 'mini_sub_grad')
print('Data saved')
```

```
a9a
Time: 6.340733528137207 s
Error: 1.5061425061425062
News 20
Time: 9.90604281425476 s
Error: 0.0
Cov Type
Time: 7.309327602386475 s
Error: 0.8395380458167674
```

In [13]:
```
# 3) Mini-batch gradient for Logistic regression
logR_mini_grad_a9a = []
time_logR_mini_grad_a9a = []
logR_mini_grad_news20 = []
time_logR_mini_grad_news20 = []
logR_mini_grad_cov = []
time_logR_mini_grad_cov = []

def batchGradLogistic(lambda_, A, b, x, beta, size):
    aTx = A * x
    sigma = -b/(1 + np.exp(b * (aTx + beta)))
    out = A.T * sigma
    norm = regularization(lambda_, x)
    g = norm + out/size
    return g, sigma/size

def logisticMiniB(A, b, beta, x0, data):
    xCurrent = x0
    alpha0 = 0.1
    alpha = alpha0
    size = 10
    const = 10
    lambda_ = 0.1
    b = np.asarray(b)
    b = b.reshape((len(b), 1))

    start = time.time()
    for i in range (200):
        j = np.random.choice(np.arange(A.shape[0]), size, replace = F
alse)
        A_j = A[j]
        b_j = b[j]
        g, sigma = batchGradLogistic(lambda_, A_j, b_j, xCurrent, bet
a, size)

        end = time.time() - start
        if (data == 'a9a'):
            logR_mini_grad_a9a.append([xCurrent, beta])
            time_logR_mini_grad_a9a.append(end)
        elif(data == 'news20'):
            logR_mini_grad_news20.append([xCurrent, beta])
            time_logR_mini_grad_news20.append(end)
        else:
            logR_mini_grad_cov.append([xCurrent, beta])
            time_logR_mini_grad_cov.append(end)

        xCurrent = xCurrent - alpha * g

        beta = beta - alpha * sum(sigma)

        alpha = alpha0/np.sqrt(i + 1)

        size = size + const
    return xCurrent, beta
```

```
print('a9a')
start = time.time()
x, beta = logisticMiniB(ATrain_a9a, bTrain_a9a, beta0, x0_a9a, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'l')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = logisticMiniB(ATrain_news20, bTrain_news20, beta0, x0_news2
0, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'l')
print('Error:', error)

print('Cov Type')
start = time.time()
x, beta = logisticMiniB(ATrain_cov, bTrain_cov, beta0, x0_cov, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'l')
print('Error:', error)

plotData(logR_mini_grad_a9a, time_logR_mini_grad_a9a,
        logR_mini_grad_news20, time_logR_mini_grad_news20,
        logR_mini_grad_cov, time_logR_mini_grad_cov, 'l', 'mini_grad'
)
print('Data saved')
```

```
a9a
Time: 0.46543002128601074 s
Error: 0.4987714987714988
News 20
Time: 5.557885646820068 s
Error: 0.87943971985993
Cov Type
Time: 3.30188250541687 s
Error: 1.1621142493244523
```

In [14]:
```
#3) Stochastic average gradient (SAG) with logistic regression
logR_sag_a9a = []
time_logR_sag_a9a = []
logR_sag_news20 = []
time_logR_sag_news20 = []
logR_sag_cov = []
time_logR_sag_cov = []

def stocasticAvgGradLogistic(i, lambda_, A, b, beta, x):
    A_i = A[i]
    b_i = b[i]
    n = A.shape[0]
    aTx = A_i * x
    sigma = -b_i/(1 + np.exp(b_i * (aTx + beta)))
    g = A_i.T * sigma
    return g/n, sigma/n

def logisticStocasticAvgGrad(x0, beta, A, b, data):
    alpha0 = 0.1
    n = A.shape[0]
    sumStochastic = np.zeros((x0.shape[0], 1))
    sumSigma = 0
    lambda_ = 0.1
    norm = regularization(lambda_, x0)
    alpha = alpha0

    for i in range (n):
        g_i, sigma = stocasticAvgGradLogistic(i, lambda_, A, b, beta,
x0)
        sumStochastic += g_i
        sumSigma += sigma
    sumStochastic += norm
    x1 = x0 - alpha0 * sumStochastic #no 1/n here
    xCurrent = x1
    xPrevious = x0
    betaPrevious = beta
    betaCurrent = beta - alpha0 * beta #no 1/n here

    start = time.time()
    for i in range(200):
        ind = rnd(A.shape[0])
        normCurrent = regularization(lambda_, xCurrent)
        normPrevious = regularization(lambda_, xPrevious)

        gXCurrent, sigmaCurrent = stocasticAvgGradLogistic(ind, lambd
a_, A, b, betaCurrent, xCurrent)
        gXPrevious, sigmaPrevious = stocasticAvgGradLogistic(ind, lam
bda_, A, b, betaPrevious, xPrevious)

        sumStochastic = (normCurrent + gXCurrent) - (normPrevious + g
XPrevious) + sumStochastic

        end = time.time() - start
        if (data == 'a9a'):
            logR_sag_a9a.append([xCurrent, betaCurrent])
            time_logR_sag_a9a.append(end)
```

```
        elif(data == 'news20'):
            logR_sag_news20.append([xCurrent, betaCurrent])
            time_logR_sag_news20.append(end)
        else:
            logR_sag_cov.append([xCurrent, betaCurrent])
            time_logR_sag_cov.append(end)

        xNext = xCurrent - alpha * sumStochastic #no 1/n here

        xPrevious = xCurrent
        xCurrent = xNext
        betaPrevious = betaCurrent
        bCurrent = betaCurrent - alpha * sigmaCurrent #no 1/n here
        alpha = alpha0/np.sqrt(i + 1)
    return xCurrent, betaCurrent

print('a9a')
start = time.time()
x, beta = logisticStocasticAvgGrad(x0_a9a, beta0, ATrain_a9a, bTrain_
a9a, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'l')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = logisticStocasticAvgGrad(x0_news20, beta0, ATrain_news20, b
Train_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'l')
print('Error:', error)

print('Cov Type')
start = time.time()
x, beta = logisticStocasticAvgGrad(x0_cov, beta0, ATrain_cov, bTrain_
cov, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'l')
print('Error:', error)

plotData(logR_sag_a9a, time_logR_sag_a9a,
        logR_sag_news20, time_logR_sag_news20,
        logR_sag_cov, time_logR_sag_cov, 'l', 'sag')
print('Data saved')
```

```
a9a
Time: 7.144322156906128 s
Error: 0.5006142506142506
News 20
Time: 142.47449493408203 s
Error: 0.4132066033016508
Cov Type
Time: 110.95104670524597 s
Error: 1.1604619541832326
```

In [15]:
```
#3) Stochastic average gradient (SAG) with Smooth hinge
smooth_sag_a9a = []
time_smooth_sag_a9a = []
smooth_sag_news20 = []
time_smooth_sag_news20 = []
smooth_sag_cov = []
time_smooth_sag_cov = []

def stocasticAvgGradSmooth(i, A, b, beta, x, u):
    A_i = A[i]
    b_i = b[i]
    aTx = A_i * x
    z = b_i * (aTx + beta)
    if (z.item() >= 1):
        g = np.zeros((x.shape[0], 1))
        sigma = 0
    elif (u < z and z < 1):
        sigma = -2 * b_i * (1 - b_i * (aTx + beta))
        g = A_i.T * sigma
    elif (z <= u):
        sigma = -2 * b_i * (1 - u)
        g = A_i.T * sigma

    return g, sigma

def smoothStocasticAvgGrad(x0, beta, A, b, data):
    xCurrent = x0
    alpha0 = 0.1
    alpha = alpha0
    u = 0.1
    n = A.shape[0]
    sumStochastic = np.zeros((x0.shape[0], 1))
    sumSigma = 0

    for i in range (n):
        g_i, sigma = stocasticAvgGradSmooth(i, A, b, beta, x0, u)
        sumStochastic += g_i
        sumSigma += sigma

    x1 = x0 - alpha0/n * sumStochastic #no 1/n here
    xCurrent = x1
    xPrevious = x0
    betaPrevious = beta
    betaCurrent = beta - alpha0/n * beta #no 1/n here

    start = time.time()
    for i in range(200):
        ind = rnd(A.shape[0])

        gXCurrent, sigmaCurrent = stocasticAvgGradSmooth(ind, A, b, b
etaCurrent, xCurrent, u)
        gXPrevious, sigmaPrevious = stocasticAvgGradSmooth(ind, A, b,
betaPrevious, xPrevious, u)

        sumStochastic = gXCurrent - gXPrevious + sumStochastic
```

```
        end = time.time() - start
        if (data == 'a9a'):
            smooth_sag_a9a.append([xCurrent, betaCurrent])
            time_smooth_sag_a9a.append(end)
        elif(data == 'news20'):
            smooth_sag_news20.append([xCurrent, betaCurrent])
            time_smooth_sag_news20.append(end)
        else:
            smooth_sag_cov.append([xCurrent, betaCurrent])
            time_smooth_sag_cov.append(end)

        xNext = xCurrent - alpha/n * sumStochastic #no 1/n here

        xPrevious = xCurrent
        xCurrent = xNext
        betaPrevious = betaCurrent
        bCurrent = betaCurrent - alpha/n * sigmaCurrent #no 1/n here
        alpha = alpha0/np.sqrt(i + 1)

    return xCurrent, beta

print('a9a')
start = time.time()
x, beta = smoothStocasticAvgGrad(x0_a9a, beta0, ATrain_a9a, bTrain_a9
a, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'h')
print('Error:', error)

print('News 20')
x, beta = smoothStocasticAvgGrad(x0_news20, beta0, ATrain_news20, bTr
ain_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'h')
print('Error:', error)

print('Cov Type')
x, beta = smoothStocasticAvgGrad(x0_cov, beta0, ATrain_cov, bTrain_co
v, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'h')
print('Error:', error)

plotData(smooth_sag_a9a, time_smooth_sag_a9a,
        smooth_sag_news20, time_smooth_sag_news20,
        smooth_sag_cov, time_smooth_sag_cov, 's', 'sag')
print('Data saved')
```

```
a9a
Time: 6.336068630218506 s
Error: 0.5939803439803439
News 20
Time: 74.50441789627075 s
Error: 0.560280140070035
Cov Type
Time: 198.72548627853394 s
Error: 1.1604619541832326
```

In [64]:
```
# 4) Stochastic average sub-gradient (SAG) for Hinge loss
sto_avg_sub_grad_a9a = []
time_sto_avg_sub_grad_a9a = []
sto_avg_sub_grad_news20 = []
time_sto_avg_sub_grad_news20 = []
sto_avg_sub_grad_cov = []
time_sto_avg_sub_grad_cov = []

def stocasticAvgSubGradHinge(i, A, b, beta, x):
    A_i = A[i]
    b_i = b[i]

    s = 1 - b_i * (A_i * x + beta)

    if (s > 0):
        g = -A_i.T * b_i
        sigma = -b_i
    else:
        g = np.zeros((x.shape[0], 1))
        sigma = 0
    return g, sigma

def hingeAvgStocasticSubGrad(x0, beta, A, b, data):
    xCurrent = x0
    alpha0 = 0.1
    alpha = alpha0
    n = A.shape[0]
    sumStochastic = np.zeros((x0.shape[0], 1))
    sumSigma = 0

    for i in range (n):
        g_i, sigma = stocasticAvgSubGradHinge(i, A, b, beta, x0)
        sumStochastic += g_i
        sumSigma += sigma

    x1 = x0 - alpha0/n * sumStochastic #no 1/n here
    xCurrent = x1
    xPrevious = x0
    betaPrevious = beta
    betaCurrent = beta - alpha0/n * beta #no 1/n here

    start = time.time()
    for i in range(200):
        ind = rnd(A.shape[0])

        gXCurrent, sigmaCurrent = stocasticAvgSubGradHinge(ind, A, b,
betaCurrent, xCurrent)
        gXPrevious, sigmaPrevious = stocasticAvgSubGradHinge(ind, A,
b, betaPrevious, xPrevious)

        sumStochastic = gXCurrent - gXPrevious + sumStochastic

        end = time.time() - start
        if (data == 'a9a'):
            sto_avg_sub_grad_a9a.append([xCurrent, betaCurrent])
            time_sto_avg_sub_grad_a9a.append(end)
```

```
        elif(data == 'news20'):
            sto_avg_sub_grad_news20.append([xCurrent, betaCurrent])
            time_sto_avg_sub_grad_news20.append(end)
        else:
            sto_avg_sub_grad_cov.append([xCurrent, betaCurrent])
            time_sto_avg_sub_grad_cov.append(end)

        xNext = xCurrent - alpha/n * sumStochastic #no 1/n here

        xPrevious = xCurrent
        xCurrent = xNext
        betaPrevious = betaCurrent
        bCurrent = betaCurrent - alpha/n * sigmaCurrent #no 1/n here
        alpha = alpha0/np.sqrt(i + 1)

    return xCurrent, beta

print('a9a')
start = time.time()
x, beta = hingeAvgStocasticSubGrad(x0_a9a, beta0, ATrain_a9a, bTrain_
a9a, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'h')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = hingeAvgStocasticSubGrad(x0_news20, beta0, ATrain_news20, b
Train_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'h')
print('Error:', error)

print('Cov Type')
start = time.time()
x, beta = hingeAvgStocasticSubGrad(x0_cov, beta0, ATrain_cov, bTrain_
cov, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'h')
print('Error:', error)

plotData(sto_avg_sub_grad_a9a, time_sto_avg_sub_grad_a9a,
        sto_avg_sub_grad_news20, time_sto_avg_sub_grad_news20,
        sto_avg_sub_grad_cov, time_sto_avg_sub_grad_cov, 'h', 'sag')
print('Data saved')
```

```
a9a
Time: 8.095412015914917 s
Error: 0.4877149877149877
News 20
Time: 60.38537788391113 s
Error: 0.5132566283141571
Cov Type
Time: 157.5710961818695 s
Error: 1.1604619541832326
```

In [25]:
```
# 6) Gradient descent with Armijo line-search for smooth hinge loss
smooth_armijo_a9a = []
time_smooth_armijo_a9a = []
smooth_armijo_news20 = []
time_smooth_armijo_news20 = []
smooth_armijo_cov = []
time_smooth_armijo_cov = []

def gradSmooth(A, b, beta, x, u):
    n = A.shape[0]
    gradF = np.zeros((x.shape[0], 1))
    sumSigma = 0
    s = A * x
    z = b * (s + beta)
    for i in range(n):
        if (z[i] >= 1):
            gradF += np.zeros((x.shape[0], 1))
            sumSigma += 0

        elif (u < z[i] and z[i] < 1):
            sigma = -2 * b[i] * (1 - z[i])
            gradF += (A[i].T * sigma).reshape((x.shape[0], 1))
            sumSigma += sigma

        elif (z[i] <= u):

            sigma = -2 * b[i] * (1 - u)
            gradF += (A[i].T * sigma).reshape((x.shape[0], 1))
            sumSigma += sigma
    return gradF/n, sumSigma/n


def armijoLineSearchSmooth(x, A, b, beta, gradF, gamma, u):
    i = 0
    alpha = 1
    fxk = hingeLossSmoothF(x, A, b, beta, u)
    gradFL2Norm = np.linalg.norm(gradF) ** 2
    rightSide = fxk - alpha * gamma * gradFL2Norm

    leftSide = x - alpha * gradF
    #f(xk - a*gradf(xk)):
    fLeftSide = hingeLossSmoothF(leftSide, A, b, beta, u)

    while(fLeftSide > rightSide):
        alpha = alpha / 2
        rightSide = fxk - alpha * gamma * gradFL2Norm
        leftSide = x - alpha * gradF
        fLeftSide = hingeLossSmoothF(leftSide, A, b, beta, u)
        i += 1
        if (i >= 50): return alpha
    return alpha

def smoothArmijo(x0, epsilon, betaCurrent, A, b, data):
    xCurrent = x0
    alpha0 = 0.1
    u = 0.1
```

```
    gamma = 0.15
    b = np.asarray(b)
    b = b.reshape((len(b), 1))

    start = time.time()
    for i in range(200):
        gradF, sigma = gradSmooth(A, b, betaCurrent, xCurrent, u)
        alpha = armijoLineSearchSmooth(xCurrent, A, b, betaCurrent, g
radF, gamma, u)

        end = time.time() - start
        if (data == 'a9a'):
            smooth_armijo_a9a.append([xCurrent, betaCurrent])
            time_smooth_armijo_a9a.append(end)
        elif(data == 'news20'):
            smooth_armijo_news20.append([xCurrent, betaCurrent])
            time_smooth_armijo_news20.append(end)
        else:
            smooth_armijo_cov.append([xCurrent, betaCurrent])
            time_smooth_armijo_cov.append(end)

        xNext = xCurrent - alpha * gradF
        betaNext = betaCurrent - alpha * sigma
        if (np.linalg.norm(gradF) < epsilon):
            return xNext, betaNext
        else:
            xCurrent = xNext
            betaCurrent = betaNext
    return xNext, betaNext

epsilon = 1.0e-2

print('a9a')
start = time.time()
x, beta = smoothArmijo(x0_a9a, epsilon, beta0, ATrain_a9a, bTrain_a9a
, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'h')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = smoothArmijo(x0_news20, epsilon, beta0, ATrain_news20, bTra
in_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'h')
print('Error:', error)


print('Cov Type')
start = time.time()
x, beta = smoothArmijo(x0_cov, epsilon, beta0, ATrain_cov, bTrain_cov
, 'cov')
end = time.time()
print('Time:', end - start, 's')
```

```
error = validationError(AValid_cov, x, beta, bValid_cov, 'h')
print('Error:', error)

plotData(smooth_armijo_a9a, time_smooth_armijo_a9a,
        smooth_armijo_news20, time_smooth_armijo_news20,
        smooth_armijo_cov, time_smooth_armijo_cov, 's', 'armijio')
print('Data saved')
```

```
a9a
Time: 632.606963634491 s
Error: 0.3108108108108108
News 20

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-25-7eedc348b908> in <module>
     96 print('News 20')
     97 start = time.time()
---> 98 x, beta = smoothArmijo(x0_news20, epsilon, beta0, ATrain_news20, bTrain_news20, 'news20')
     99 end = time.time()
    100 print('Time:', end - start, 's')

<ipython-input-25-7eedc348b908> in smoothArmijo(x0, epsilon, betaCurrent, A, b, data)
     61     start = time.time()
     62     for i in range(200):
---> 63         gradF, sigma = gradSmooth(A, b, betaCurrent, xCurrent, u)
     64         alpha = armijoLineSearchSmooth(xCurrent, A, b, betaCurrent, gradF, gamma, u)
     65 

<ipython-input-25-7eedc348b908> in gradSmooth(A, b, beta, x, u)
     20         elif (u < z[i] and z[i] < 1):
     21             sigma = -2 * b[i] * (1 - z[i])
---> 22             gradF += (A[i].T * sigma).reshape((x.shape[0], 1))
     23             sumSigma += sigma
     24 

~/anaconda3/lib/python3.7/site-packages/scipy/sparse/base.py in __mul__(self, other)
    466             # Fast path for the most common case
    467             if other.shape == (N,):
--> 468                 return self._mul_vector(other)
    469             elif other.shape == (N, 1):
    470                 return self._mul_vector(other.ravel()).reshape(M, 1)

~/anaconda3/lib/python3.7/site-packages/scipy/sparse/compressed.py in _mul_vector(self, other)
    467         # output array
    468         result = np.zeros(M, dtype=upcast_char(self.dtype.char,
--> 469                                                other.dtype.char))
    470 
    471         # csr_matvec or csc_matvec

KeyboardInterrupt: 
```

In [55]:
```
# 6) Gradient descent with Armijo line-search for logistic regression
logR_armijo_a9a = []
time_logR_armijo_a9a = []
logR_armijo_news20 = []
time_logR_armijo_news20 = []
logR_armijo_cov = []
time_logR_armijo_cov = []

def gradLogistic(lambda_, x, A, b, beta):
    n = A.shape[0]
    s = A * x
    sigma = -b / (1 + np.exp(b * (s + beta)))
    sumF = A.T * sigma
    norm = regularization(lambda_, x)
    gradF = norm + sumF/n
    betaF = sum(sigma)
    return gradF, betaF/n

def armijoLineSearchLogistic(lambda_, x, A, b, beta, gradF, gamma):
    i = 0
    alpha = 1
    fxk = logisticF(lambda_, x, A, b, beta)
    gradFL2Norm = np.linalg.norm(gradF) ** 2
    rightSide = fxk - alpha * gamma * gradFL2Norm

    leftSide = x - alpha * gradF
    #f(xk - a*gradf(xk)):
    fLeftSide = logisticF(lambda_, leftSide, A, b, beta)

    while(fLeftSide > rightSide):
        alpha = alpha / 2
        rightSide = fxk - alpha * gamma * gradFL2Norm
        leftSide = x - alpha * gradF
        fLeftSide = logisticF(lambda_, leftSide, A, b, beta)
        i += 1
        if (i >= 50): return alpha
    return alpha

def logisticArmijo(x0, epsilon, betaCurrent, A, b, data):
    b = np.asarray(b)
    b = b.reshape((len(b), 1))
    xCurrent = x0
    lambda_ = 0.01
    gamma = 0.5

    start = time.time()
    for i in range(200):
        gradF, sigma = gradLogistic(lambda_, xCurrent, A, b, betaCurr
ent)
        alpha = armijoLineSearchLogistic(lambda_, xCurrent, A, b, bet
aCurrent, gradF, gamma)

        end = time.time() - start
        if (data == 'a9a'):
            logR_armijo_a9a.append([xCurrent, betaCurrent])
            time_logR_armijo_a9a.append(end)
```

```
        elif(data == 'news20'):
            logR_armijo_news20.append([xCurrent, betaCurrent])
            time_logR_armijo_news20.append(end)
        else:
            logR_armijo_cov.append([xCurrent, betaCurrent])
            time_logR_armijo_cov.append(end)

        xNext = xCurrent - alpha * gradF
        betaNext = betaCurrent - alpha * sigma

        if (np.linalg.norm(gradF) < epsilon):
            return xNext, betaNext
        else:
            xCurrent = xNext
            betaCurrent = betaNext
    return xNext, betaNext

epsilon = 1.0e-2

print('a9a')
start = time.time()
x, beta = logisticArmijo(x0_a9a, epsilon, beta0, ATrain_a9a, bTrain_a
9a ,'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'l')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = logisticArmijo(x0_news20, epsilon, beta0, ATrain_news20, bT
rain_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'l')
print('Error:', error)

print('Cov Type')
start = time.time()
x, beta = logisticArmijo(x0_cov, epsilon, beta0, ATrain_cov, bTrain_c
ov, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'l')
print('Error:', error)

plotData(logR_armijo_a9a, time_logR_armijo_a9a,
        logR_armijo_news20, time_logR_armijo_news20,
        logR_armijo_cov, time_logR_armijo_cov, 'l', 'armijio')
print('Data saved')
```

```
a9a
Time: 4.159910440444946 s
Error: 0.3353808353808354
News 20
Time: 31.564945697784424 s
Error: 2.0
Cov Type
Time: 54.37814545631409 s
Error: 0.6751691020808592
```

Practical Line search

Step 1) Choose an x_0 and set $y_1 = x_0, t_1 = 1$.

Step 2) Repeat the following steps until $\|\nabla f(x_k)\|_2 \leq \epsilon$

Step 3) Compute α_k using Armijo line-search. Armijo line-search should be measured at $y_k - \alpha_k \nabla f(y_k)$ (as the next point) and y_k (as the current point).

Step 4) Set

$$x_k = y_k - \alpha_k \nabla f(y_k)$$

Step 5) Set

$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}$$

Step 6) Set

$$y_{k+1} = x_k + \frac{t_k - 1}{t_{k+1}}(x_k - x_{k-1})$$

In [26]:
```
# 7) Acceleratd gradient with Armijo line-search for smooth
smooth_acc_a9a = []
time_smooth_acc_a9a = []
smooth_acc_news20 = []
time_smooth_acc_news20 = []
smooth_acc_cov = []
time_smooth_acc_cov = []

def gradAccSmooth(A, b, beta, x, u):
    n = A.shape[0]
    gradF = np.zeros((x.shape[0], 1))
    sumSigma = 0
    s = A * x
    z = b * (s + beta)
    for i in range(n):
        if (z[i] >= 1):
            gradF += np.zeros((x.shape[0], 1))
            sumSigma += 0
        elif (u < z[i] and z[i] < 1):
            sigma = -2 * b[i] * (1 - z[i])
            gradF += (A[i].T * sigma).reshape((x.shape[0], 1))
            sumSigma += sigma
        elif (z[i] <= u):
            sigma = -2 * b[i] * (1 - u)
            gradF += (A[i].T * sigma).reshape((x.shape[0], 1))
            sumSigma += sigma
    return gradF/n, sumSigma/n

def accArmijoSmooth(x, A, b, beta, gradF, gamma, u):
    i = 0
    alpha = 1
    fxk = hingeLossSmoothF(x, A, b, beta, u)
    gradFL2Norm = np.linalg.norm(gradF) ** 2
    rightSide = fxk - alpha * gamma * gradFL2Norm

    leftSide = x - alpha * gradF
    #f(xk - a*gradf(xk)):
    fLeftSide = hingeLossSmoothF(leftSide, A, b, beta, u)

    while(fLeftSide > rightSide):
        alpha = alpha / 2
        rightSide = fxk - alpha * gamma * gradFL2Norm
        leftSide = x - alpha * gradF
        fLeftSide = hingeLossSmoothF(leftSide, A, b, beta, u)
        i += 1
        if (i >= 50): return alpha
    return alpha

def smoothAccArmijo(x0, epsilon, betaCurrent, A, b, data):
    yCurrent = x0
    xCurrent = x0
    tCurrent = 1
    b = np.asarray(b)
    b = b.reshape((len(b), 1))
    u = 0.1
    gamma = 0.15
```

```
    start = time.time()
    for i in range(200):
        gradF, sigma = gradAccSmooth(A, b, betaCurrent, xCurrent, u)
        alpha = accArmijoSmooth(xCurrent, A, b, betaCurrent, gradF, g
amma, u)

        end = time.time() - start
        if (data == 'a9a'):
            smooth_acc_a9a.append([xCurrent, betaCurrent])
            time_smooth_acc_a9a.append(end)
        elif(data == 'news20'):
            smooth_acc_news20.append([xCurrent, betaCurrent])
            time_smooth_acc_news20.append(end)
        else:
            smooth_acc_cov.append([xCurrent, betaCurrent])
            time_smooth_acc_cov.append(end)

        xNext = xCurrent - alpha * gradF
        betaNext = betaCurrent - alpha * sigma

        tNext = (1 + np.sqrt(1 + 4 * tCurrent ** 2)) / 2
        yCurrent = xNext + ((tCurrent - 1)/tNext) * (xNext - xCurrent
)


        if (np.linalg.norm(gradF) < epsilon):
            return xNext, betaNext
        else:
            xCurrent = xNext
            betaCurrent = betaNext
            tCurrent = tNext
    return xNext, betaNext


epsilon = 1.0e-2

print('a9a')
start = time.time()
x, beta = smoothAccArmijo(x0_a9a, epsilon, beta0, ATrain_a9a, bTrain_
a9a, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'h')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = smoothAccArmijo(x0_news20, epsilon, beta0, ATrain_news20, b
Train_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'h')
print('Error:', error)


print('Cov Type')
```

```
start = time.time()
x, beta = smoothAccArmijo(x0_cov, epsilon, beta0, ATrain_cov, bTrain_
cov, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'h')
print('Error:', error)

plotData(smooth_acc_a9a, time_smooth_acc_a9a,
         smooth_acc_news20, time_smooth_acc_news20,
         smooth_acc_cov, time_smooth_acc_cov, 's', 'acc_armijio')
print('Data saved')
```

```
a9a
Time: 624.7704644203186 s
Error: 0.3108108108108108
News 20

-------------------------------------------------------------------
------
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-26-abecc158920c> in <module>
    100 print('News 20')
    101 start = time.time()
--> 102 x, beta = smoothAccArmijo(x0_news20, epsilon, beta0, ATrain_n
ews20, bTrain_news20, 'news20')
    103 end = time.time()
    104 print('Time:', end - start, 's')

<ipython-input-26-abecc158920c> in smoothAccArmijo(x0, epsilon, betaC
urrent, A, b, data)
     58     start = time.time()
     59     for i in range(200):
---> 60         gradF, sigma = gradAccSmooth(A, b, betaCurrent, xCurr
ent, u)
     61         alpha = accArmijoSmooth(xCurrent, A, b, betaCurrent,
 gradF, gamma, u)
     62

<ipython-input-26-abecc158920c> in gradAccSmooth(A, b, beta, x, u)
     19         elif (u < z[i] and z[i] < 1):
     20             sigma = -2 * b[i] * (1 - z[i])
---> 21             gradF += (A[i].T * sigma).reshape((x.shape[0], 1)
)
     22             sumSigma += sigma
     23         elif (z[i] <= u):

~/anaconda3/lib/python3.7/site-packages/scipy/sparse/base.py in __mul
__(self, other)
    466             # Fast path for the most common case
    467             if other.shape == (N,):
--> 468                 return self._mul_vector(other)
    469             elif other.shape == (N, 1):
    470                 return self._mul_vector(other.ravel()).reshap
e(M, 1)

~/anaconda3/lib/python3.7/site-packages/scipy/sparse/compressed.py in
_mul_vector(self, other)
    467         # output array
    468         result = np.zeros(M, dtype=upcast_char(self.dtype.cha
r,
--> 469                                                other.dtype.ch
ar))
    470
    471         # csr_matvec or csc_matvec

KeyboardInterrupt:
```

In [109]:
```
# 7) Acceleratd gradient with Armijo line-search for logistic
logR_acc_a9a = []
time_logR_acc_a9a = []
logR_acc_news20 = []
time_logR_acc_news20 = []
logR_acc_cov = []
time_logR_acc_cov = []

def gradAccLogistic(lambda_, x, A, b, beta):
    n = A.shape[0]
    s = A * x

    sigma = -b / (1 + np.exp(b * (s + beta)))
    sumF = A.T * sigma
    norm = regularization(lambda_, x)
    gradF = norm + sumF/n
    betaF = sum(sigma)
    return gradF, betaF/n

def accArmijoLogistic(lambda_, x, A, b, beta, gradF, gamma):
    i = 0
    alpha = 1
    fxk = logisticF(lambda_, x, A, b, beta)
    gradFL2Norm = np.linalg.norm(gradF) ** 2
    rightSide = fxk - alpha * gamma * gradFL2Norm

    leftSide = x - alpha * gradF
    #f(xk - a*gradf(xk)):
    fLeftSide = logisticF(lambda_, leftSide, A, b, beta)

    while(fLeftSide > rightSide):
        alpha = alpha / 2
        rightSide = fxk - alpha * gamma * gradFL2Norm
        leftSide = x - alpha * gradF
        fLeftSide = logisticF(lambda_, leftSide, A, b, beta)
        i += 1
        if (i >= 50): return alpha
    return alpha

def logisticAccArmijo(x0, epsilon, betaCurrent, A, b, data):
    yCurrent = x0
    xCurrent = x0
    tCurrent = 1
    b = np.asarray(b)
    b = b.reshape((len(b), 1))
    lambda_ = 0.1
    gamma = 0.15

    start = time.time()
    for i in range(2000):
        gradF, sigma = gradAccLogistic(lambda_, xCurrent, A, b, betaC
urrent)
        alpha = accArmijoLogistic(lambda_, xCurrent, A, b, betaCurren
t, gradF, gamma)

        end = time.time() - start
```

```
        if (data == 'a9a'):
            logR_acc_a9a.append([xCurrent, betaCurrent])
            time_logR_acc_a9a.append(end)
        elif(data == 'news20'):
            logR_acc_news20.append([xCurrent, betaCurrent])
            time_logR_acc_news20.append(end)
        else:
            logR_acc_cov.append([xCurrent, betaCurrent])
            time_logR_acc_cov.append(end)

        xNext = xCurrent - alpha * gradF
        betaNext = betaCurrent - alpha * sigma

        tNext = (1 + np.sqrt(1 + 4 * tCurrent ** 2)) / 2
        yCurrent = xNext + ((tCurrent - 1)/tNext) * (xNext - xCurrent
)

        if (np.linalg.norm(gradF) < epsilon):
            return xNext, betaNext
        else:
            xCurrent = xNext
            betaCurrent = betaNext
            tCurrent = tNext
    return xNext, betaNext

print('a9a')
start = time.time()
x, beta = logisticAccArmijo(x0_a9a, epsilon, beta0, ATrain_a9a, bTrai
n_a9a, 'a9a')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_a9a, x, beta, bValid_a9a, 'l')
print('Error:', error)

print('News 20')
start = time.time()
x, beta = logisticAccArmijo(x0_news20, epsilon, beta0, ATrain_news20,
bTrain_news20, 'news20')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_news20, x, beta, bValid_news20, 'l')
print('Error:', error)

print('Cov Type')
start = time.time()
x, beta = logisticAccArmijo(x0_cov, epsilon, beta0, ATrain_cov, bTrai
n_cov, 'cov')
end = time.time()
print('Time:', end - start, 's')
error = validationError(AValid_cov, x, beta, bValid_cov, 'l')
print('Error:', error)

plotData(logR_acc_a9a, time_logR_acc_a9a,
        logR_acc_news20, time_logR_acc_news20,
        logR_acc_cov, time_logR_acc_cov, 'l', 'acc_armijio')
print('Data saved')
```

```
a9a
Time: 0.9285979270935059 s
Error: 0.4705159705159705
News 20
Time: 7.409563779830933 s
Error: 2.0
Cov Type
Time: 13.28058910369873 s
Error: 0.756751174678577
```

I ran out of memory and time trying to plot the graphs. It was taking me an insane amount of time to calculate the objective functions. But if you print out the values of the objective functions, you can see that they are indeed decreasing and converging. I printed out the errors and time it takes for each method on each dataset. Hope that will get me some marks

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: