In [9]:
```
#Part 1
'''
#1 Train a logistic regression model using Gradient Descent.
Show , via a graph, how the loss function changes as you perform iteration
Report the total time required to train the model.
Use only 2/3 of the data to train the model, test it with the rest of the
```

Out[9]:
```
'\n#1 Train a logistic regression model using Gradient Descent. \nShow
, via a graph, how the loss function changes as you perform iterations
of GD. \nReport the total time required to train the model. \nUse only
2/3 of the data to train the model, test it with the rest of the data.\
n'
```

In [128]:
```
# %install_ext https://raw.github.com/cpcloud/ipython-autotime/master/autc
# %load_ext autotime

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model, model_selection, metrics #SGD, cross_val
from glob import glob
```

Out[128]:
```
['accuracy_results_100runs.png',
 'accuracy_results_200runs.png',
 'accuracy_results_50runs.png',
 'CS688HW3.pdf',
 'data.txt',
 'HW3_Problem_1.ipynb',
 'sgb_accuracy_results_200runsv2.png',
 'sgb_accuracy_results_250runsv2.png']
```

In [173]:
```
# read in data and add in missing values
df = pd.read_csv('data.txt')
# setting up y df
df_y = pd.DataFrame(df['diagnosis'])
df_y[df_y=='B'] = 0
df_y[df_y=='M'] = 1

# getting rid of unnecessary columns for data
df_x = df.drop(labels=['diagnosis','Unnamed: 32','id'], axis=1)

# cleaning up data
df_x.replace(to_replace=0,value=df_x.mean()).head()
df_x.replace(to_replace=np.NaN, value=df_x.mean())

# normalizing data
col_names = list(df_x.columns.values)
for col in col_names:
    df_x[col]= (df[col]-df[col].mean())/(df[col].max() - df[col].min())
```

Out[173]:

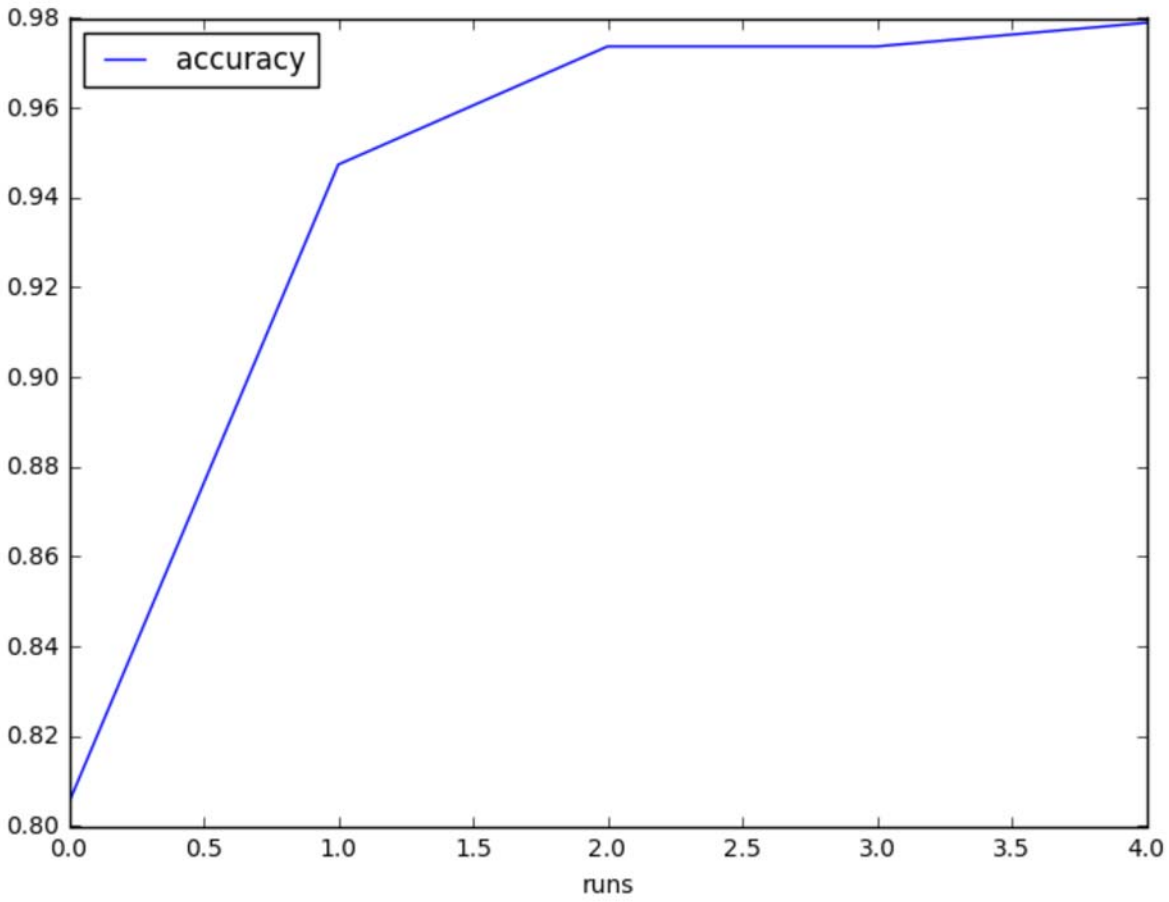
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compac
0	0.182815	-0.301307	0.213053	0.146813	0.198968	0.53143
1	0.304923	-0.051392	0.282848	0.284671	-0.104905	-0.07883
2	0.263274	0.066295	0.262808	0.232497	0.119524	0.17041
3	-0.128132	0.036874	-0.099434	-0.114014	0.416536	0.55076
4	0.291671	-0.167388	0.298051	0.272369	0.035567	0.08729

5 rows × 30 columns

In [353]:
```
# loss - the type of loss function to use, using logistic
# n_iter - number of times stochastic descent is iterated through
# n_jobs - number of cores to use, -2 is all but one
# alpha- scaler of gradient descent
# random_state - None randomizes parameters

def run_batchGD(x, y, partition_size, n_runs, n_replications, reg):
    n_GD_replications = range(0,n_replications)
    replication_accuracy = []
    replication_log_loss = []
    random = None
    descent_params = 0
    run_accuracy = []
    run_list = []
    replication_list = []
    for i in n_GD_replications:
#         print('this is i: {}'.format(i))
        n = 0

        run_log_loss = []
        while n < n_runs:
#             print('this is n: {}'.format(n))
            if( n == 0 ):
                train_x, test_x, train_y, test_y = model_selection.train_t
                x, y, test_size=partition_size, random_state=0)

                sgd = linear_model.SGDClassifier(loss = 'log', penalty=reg
                                                 n_jobs=-2, alpha=.000001,

            sgd.partial_fit(train_x, train_y, classes = np.unique(y)) #cla

            accuracy = sgd.score(test_x,test_y)
            run_accuracy.append(accuracy)
            run_list.append(n)
            replication_list.append(i)
            n += 1

        # saving average accuracy of malignant predictions
        replication_accuracy.append(np.mean(run_accuracy))
```

In [384]:
```
%%time
# converting to numpy arrays for scikit-learn
x = np.asarray(df_x)
y = np.ravel(df_y).tolist()

# run batch gradient descent
test_size = 1/3
mean_runs = 5
num_reps = 1
l1_regularization = 'l1'
batchGD_accuracy, batch_runs, replication_runs = run_batchGD(x, y, test_si
                                                             l1_regulariza
print(batchGD_accuracy)
```

```
[0.80526315789473679, 0.94736842105263153, 0.97368421052631582, 0.97368
421052631582, 0.97894736842105268]
[0.80526315789473679, 0.94736842105263153, 0.97368421052631582, 0.97368
421052631582, 0.97894736842105268]
[0, 1, 2, 3, 4]
```




```
Wall time: 151 ms
```

In [ ]:
```
# plot prediction accuracy
print(batchGD_accuracy)
print(batch_runs)
results_df = pd.DataFrame({'replication': replication_runs, 'runs': batch_
# sorted_df = results_df.sort_values(['runs']))
fig, ax = plt.subplots(figsize=(8,6))
for i, group in results_df.groupby('replication'):
    group.plot(x = 'runs', y= 'accuracy', ax=ax)
```

In [62]:
```
# is0 = train_y==0
# is1 = train_y==1
# is_1or0 = is0 | is1
# is_1or0.head()
```

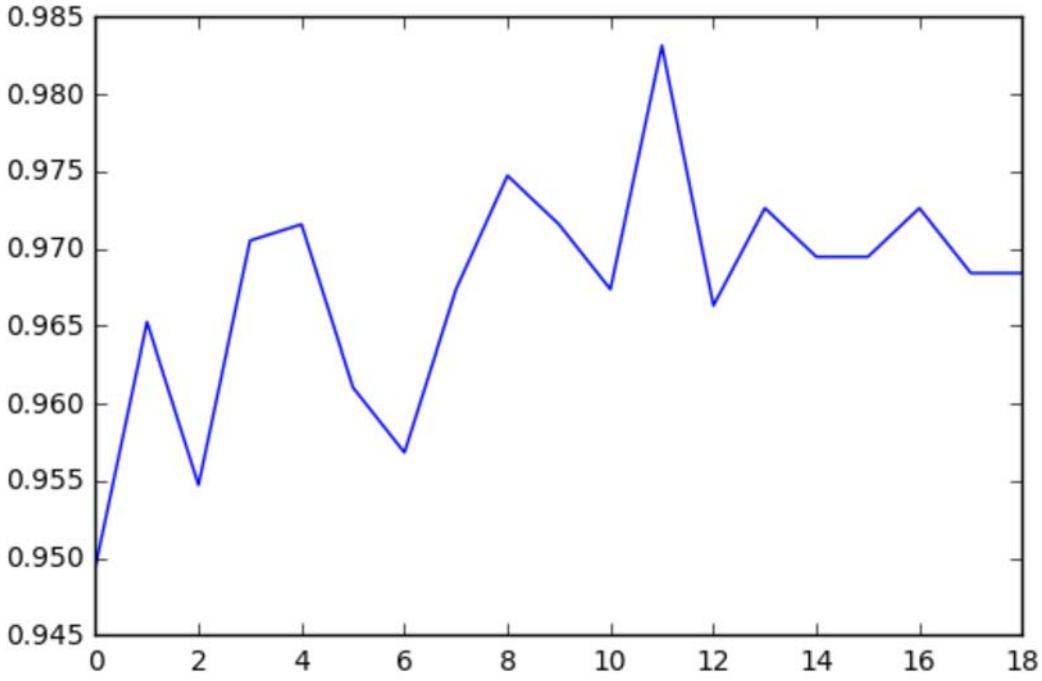
In [ ]:
```
'''
Repeat i) using Stochastic Gradient Descent.
Again, show the change of the loss function as you perform iterations of S
Report the total time needed.
```

In [387]:
```
# loss - the type of loss function to use, using logistic
# n_iter - number of times stochastic descent is iterated through
# n_jobs - number of cores to use, -2 is all but one
# learning_rate - scaler of gradient descent, 'optimal' is the default, ch
# random_state - None randomizes parameters

def run_sgd(x, y, partition_size, n_runs, n_iters, reg):
    n_SGD_iterations = range(1,n_iters)
    iteration_accuracy = []
    iteration_log_loss = []
    random = None
    for i in n_SGD_iterations:
            run_error = []
            run_log_loss = []
            n = 0
            while n < n_runs:
                train_x, test_x, train_y, test_y = model_selection.train_t
                    x, y, test_size=partition_size, random_state=random)
                sgd = linear_model.SGDClassifier(loss = 'log', penalty=reg
                                                 n_jobs=-2, learning_rate=
                sgd.fit(train_x, train_y)
                run_error.append(sgd.score(test_x,test_y))
                run_log_loss.append(metrics.log_loss(test_y, sgd.predict(t
                n += 1

            # saving average accuracy of malignant predictions
            iteration_accuracy.append(np.mean(run_error))
            iteration_log_loss.append(np.mean(run_log_loss))
```

In [409]:
```
%%time
# converting to numpy arrays for scikit-learn
x = np.asarray(df_x)
y = np.ravel(df_y).tolist()

# run stochastic gradient descent
test_size = 1/3
num_runs = 5
num_iters = 20
l1_regularization = 'l1'
accuracy_results, log_loss = run_sgd(x, y, test_size, num_runs, num_iters,
```

```
[0.94947368421052636, 0.96526315789473682, 0.95473684210526311, 0.97052
631578947357, 0.9715789473684211, 0.96105263157894727, 0.95684210526315
783, 0.96736842105263166, 0.97473684210526312, 0.9715789473684211, 0.96
736842105263166, 0.98315789473684201, 0.96631578947368424, 0.9726315789
473684, 0.96947368421052649, 0.96947368421052627, 0.9726315789473684, 0
.96842105263157896, 0.96842105263157896]
Wall time: 197 ms
```

In [410]:
```
# plot prediction accuracy
plt.plot(list(range(num_iters-1)), accuracy_results)
```



In [ ]:
```
'''
Repeat i) and ii) using L2-Regularization.
Use only 50% of the data to train the model.
Use approximately 1/3 of the data to validate the regularization constant
Use the rest of the data to test the resulting model.
```

In [446]:
```
%%time
# converting to numpy arrays for scikit-learn
x = np.asarray(df_x)
y = np.ravel(df_y).tolist()

# run stochastic gradient descent
test_size = 1/3
num_runs = 50
num_iters = 45
l2_regularization = 'l2'
sgd_results, log_loss = run_sgd(x, y, test_size, num_runs, num_iters,l2_re
```

```
Wall time: 4.72 s
```

In [455]:
```
# run batch gradient descent
test_size = 1/3
num_runs = 9
num_reps = 2
batchGD_accuracy, batch_runs, replication_runs = run_batchGD(x, y, test_si
                                                             l2_regulariza
```

```
[0.92105263157894735, 0.93157894736842106, 0.94736842105263153, 0.95263
157894736838, 0.84736842105263155, 0.9631578947368421, 0.97368421052631
582, 0.93157894736842106, 0.97368421052631582, 0.88421052631578945, 0.9
631578947368421, 0.95263157894736838, 0.95263157894736838, 0.9315789473
6842106, 0.91578947368421049, 0.97368421052631582, 0.95263157894736838,
 0.93157894736842106]
```

In [456]:
```
# plot prediction accuracy
mean_bgd_result = [np.mean(batchGD_accuracy) for i in range(len(sgd_result
plt.plot(list(range(num_iters-1)), sgd_results)
plt.plot(list(range(num_iters-1)), mean_bgd_result)
```