Expectation-Maximization vs Gradient Ascent

The following experiment tries to differentiate between two parameter learning algorithms for incomplete data. It uses Student Bayesian Network used in the class assignments for generating the data. Then another program removes the random data entries given the probability of missing entries. Data in such file is divided into train and test set with 80/20 distribution. Finally, both algorithms learn parameters for given data. With those parameters, I calculate the likelihood for the test data.

Data Generation:
I simply used forward sampling ***bn.bn.sample()*** over student bn to generate complete data using ***exp_invoker.generate_data***. It generates ***studentbn<numberOfEntries>_data.csv*** with complete data. Then for incomplete data, I used ***data_remover.py***, which uses generated complete data via ***full_data_file_name*** variable to create incomplete data out of it. It has a variable ***missing_entry_prob*** to set missing entries probability. The data remover generates file with name ***studentbn<numberOfEntries>_<missingEntryProbability>.csv***.

Running the algorithms:
Both the algorithms are implemented in the same function ***bn.bnstructure.learn_params()***.
The function takes following arguments:

- **data_file_name:** The data file to use for training the parameters.
- **var_order:** order of variables in either data file or data_array(passed below).
- **learning_rate=0.0001:** learning rate to be used for gradient descent.
- **alg=None:** algorithm to be used to learn parameters either ‘em’ or ‘gd’
- **data_array=None:** a NumPy array of data. Setting this will make the function overlook the previously mentioned data file.
- **starting_point=None:** assignments of the factors to be used at the beginning of the algorithm. Initialization of parameters.

The algorithm is invoked from ***exp_invoker.py, if __name__ == '__main__'***
The file ***exp_invoker.py*** takes two command-line arguments the size of the data generated and the probability of missing entry.
The program learns the parameters 3 times to check with multiple random initializations.

Experiment:

- I generated complete data files of sizes 100 and 1000(At 1000 rows themselves both algorithms started giving identical parameters and likelihood for data the trend continued for 10000 rows as well so I selected 100 and 1000 for experimentation).
- With these files I then generated MCAR incomplete data with missing entries probability of 0.1, 0.3, 0.5, 0.7 and 0.9, which is 10%, 30%, 50%, 70% and 90% missing data.
- Both the algorithms were tested on these files and their progression of loglikelihood during training is collected.

Results:
For data with 1000 entries:

Missing probability	**Gradient Ascent iterations avg.**	**Expectation-Maximization iterations avg.**	**LLikelihood diff 100*(gd-em)/gd avg.**	**LH diff test 100*(gd-em)/gd avg.**
0.1	158	6.66	0.125	-1.9
0.3	181	14.66	0.135	-0.56
0.5	263.33	21.66	0.164	-0.007
0.7	375.66	53	0.273	-0.31
0.9	1293	300.33	1.3	-0.86

As we can clearly see the the EM is converging almost dozens of times faster when running for 10% missing data. As the missing data increases the speed of EM decreases quicker than GD. Apart from speed both of them show very little difference in final training log-likelihoods. It seems the EM is performing a pinch better than GD in each case. But this is reversed for test data likelihoods, the GD has a pinch better performance than the EM.

Observations:

- This experiment boldly shows that EM converges fast. When checking likelihood progression it takes a lot bigger steps towards maxima.
- The speed of GD heavily depends on its learning rate setting it to a higher value and then decaying it gradually does give some improvement over speed.

The Likelihood difference on test data is a notable phenomenon. I tried this algorithm on smaller data size.(Please continue on next page).

For data with 100 entries:

Missing probability	**Gradient Ascent iterations avg.**	**Expectation-Maximization iterations avg.**	**LogLikelihood diff 100*(gd-em)/gd avg.**	**LLH diff test 100*(gd-em)/gd avg.**	**Best case llh diff(test) for loosing alg**
0.1	867	6.33	0.76	-10.86	10.15
0.3	1253.33	18.33	1.13	-25.64	25.27
0.5	1319.66	37.66	1.16	-79.21	64.53
0.7	2757.66	114.66	2.23	2.28	0.73
0.9	4300.33	503.66	7.57	11.78	4.91

The GD took far more iterations than for 1000 entries. EM took almost same number of iterations. The likelihood differences for the train has similarly low difference. The test differences are very high. GD is performing considerably better for 0.1, 0.3 and 0.5 and best cases where GD is lacking on average are not very bad.

Observations:

- The GD takes more iterations as the amount of data decreases. The EM has little effect of the iteration of data size.
- The GD although taking a lot of iterations is converging to a good point that works better on test data when compared with EM.

Conclusions:

- GD has one more parameter than EM which is the learning rate. It needs to calibrated based on the data there more parameters needed to speed up the GD. This makes GD to be difficult to calibrate and case specific.
- The EM on the other hand does not need any hyperparameters its steps are relatively bigger and in direction of local maxima that helps it converge very fast.
- The complexity of GD while making it slower allows us to find better solutions due to its smaller steps. The EM on the other hand might end up on suboptimal plateau when compared with GD.
- The differences of accuracy between these to are visible when we have a smaller amount of data that might not give best estimate of original distribution.
- With rich data both of them perform equally well but EM performs better on speed as well.
- The GD hence can be used when we have lack of data whereas the EM is good estimator with good amount of data