Online Learning Report

Eric Westman, Sun Chunyang Jane and R. Arun Srivatsan

November 13 2014

1 Introduction

For this lab, we implemented bayesian linear regression, online gradient descent, and kernelized SVM. For reference, Figure 1 shows the test set with ground truth labels.

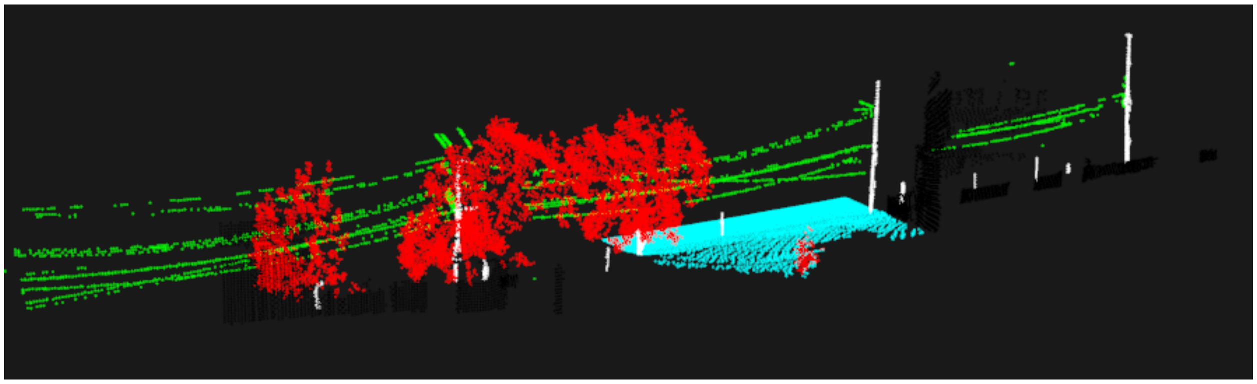


Figure 1: Test data with ground truth labels.

2 Online gradient descent

1. How well did it perform for online learning? Does it perform well on the held-out data?
Online gradient descent algorithm achieves 80% to 85% success with no more than 40% of the individual class misclassified.

2. Are there any classes that did not get classified well? Why do you think that is?
The wire class is not classified well, probably because it has the least data points to learn from.

3. How easy was the learner to implement?
The learner is of medium difficulty for implementation.

4. How long does the learner take (in terms of data points, dimensions, classes, etc...) for training and prediction
Gradient descent is very efficient. It is basically linear in the number of training examples. It is O(n) for training as well as for prediction. In the actual implementation, the complexity measured by time elapsed is shown in Table 1. For the data points, we halved the numbers. For the classes, we decreased from 4 classes to 3 and 2. For features, we halved the dimensions.

5. Show images/movies of the classified data.
Figure 2 shows the classification results for online gradient descent.

Table 1: Time complexity for online gradient descent

Variable	Action	Training (s)	Complexity	Testing (s)	Complexity
Data Points	Half the data points	13, 6.7, 3	O(n)	0.6, 0.3, 0.16	O(n)
Classes	4, 3, 2	6.7, 4.1, 2.7	O(log(n))	0.3, 0.3, 0.3	O(log(n))
Feature Dimensions	Half the dimensions	6.9, 6.8, 9.2	O(1)	0.32, 0.32, 0.44	O(1)

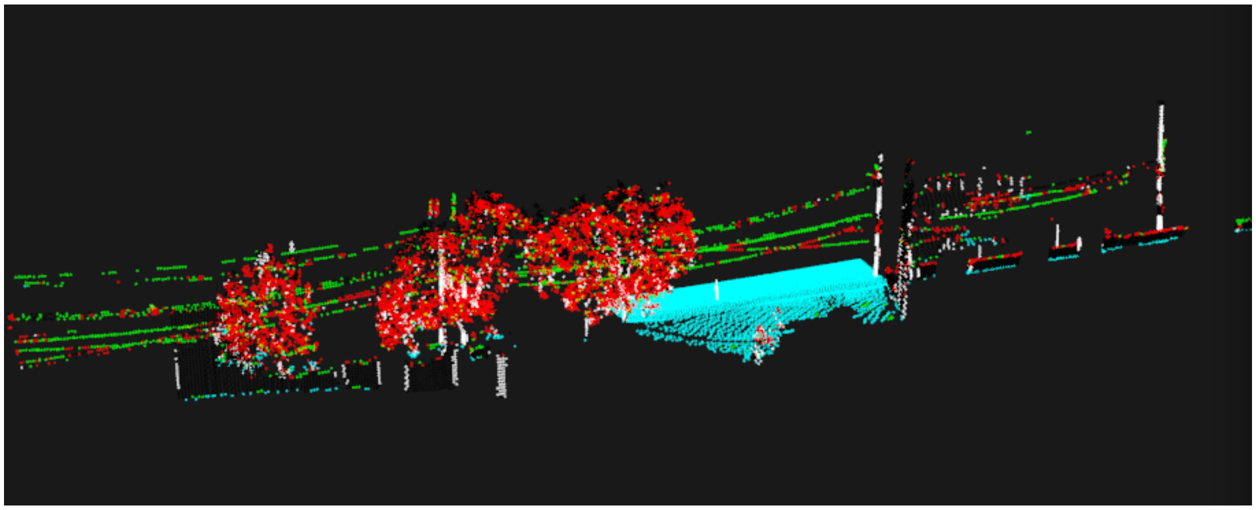


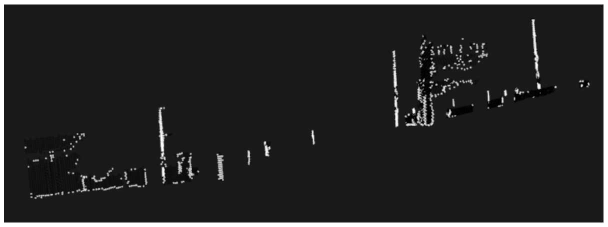
Figure 2: Online gradient descent classification results on the test data

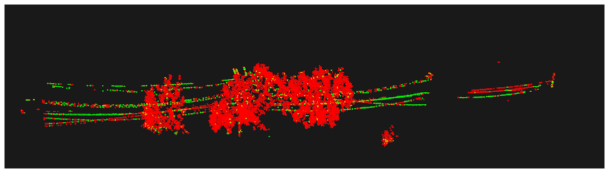
6. How did you choose (hyper)parameters (priors, kernel width, noise variance, prior variance, learning rate, etc. . .)?

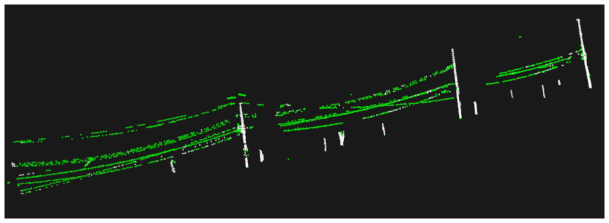
- The most important parameter to choose in OGD is the learning rate. From the theoretical discussions on regret, we chose $\alpha = \frac{1}{\sqrt{T}}$ where T is the total number of data points. The results were very poor when we used $\alpha = \frac{1}{T}$ or random small number such as $10^{-3}, 10^{-6}$ etc.
- Also the algorithm performed well when the indexing order of the training data was randomized. The training data seemed to have members of classes in batch. So randomizing it improved the prediction rate.
- Some of the classes had fewer members than the others. Hence we duplicated the data to have similar number of members in all the classes.
- Running the training algorithm multiple times on the data also helped in improving the prediction rate.

7. How robust is this algorithm to noise?

With 4 more noise corrupted versions of the features, the algorithm stayed at 80% success within 8 trials. With 4 more random noise features, the error fluctuated between 85% to 75% within 8 trials. Overall the algorithm is pretty robust.


(a) Facade vs pole


(b) Veg vs wire


(c) Pole vs wire

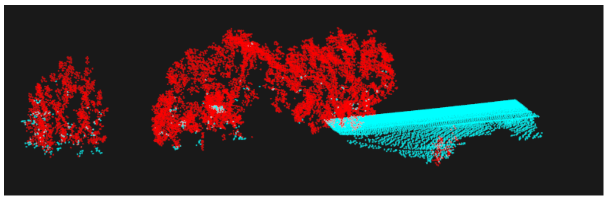

(d) Veg vs ground

Figure 3: Bayesian linear regression classification results on test data.

Variable	Action	Training (s)	Complexity	Testing (s)	Complexity
Data Points	Half the data points	0.23, 0.12, 0.09	O(n)	0.33, 0.17, 0.03	O(n)
Feature Dimensions	Half the dimensions	0.23, 0.22, 0.23	O(1)	0.33, 0.34, 0.33	O(1)

Table 2: Time complexity for bayesian linear regression.

3 Bayesian Linear Regression

1. How well did it perform for online learning? Does it perform well on the held-out data?
Depending on the classes that you select to classify, it can produce as little as 3% classification error, or as much as 60% classification error.

2. Are ther e any classes that did not get classified well? Why do you think that is?
The classes that were classified worst were generally veg, facade, and pole. In our implementation, we gave the classifier the same number of examples from each class, so this discrepancy between the classes is probably due to the feature set rather than the data itself.

3. How easy was the learner to implement?
This learner was fairly easy to implement because it was just a binary classifier (one class vs. the other).

4. How long does the learner take (in terms of data points, dimensions, classes, etc...) for training and prediction?
Bayesian Regression's computational complexity is about O(n) with the number of samples. In the actual implementation, the complexity measured by time elapsed is shown in Figure 2. For the data points, we halved the numbers. For features, we halved the dimensions.

5. Show images/movies of the classified data.
See Figure 3 for a visual representation of the classifier operating on four different pairs of classes.

6. How did you choose (hyper)parameters (priors, kernel width, noise variance, prior variance, learning rate, etc...)?
We chose zero mean, and a somewhat arbitrary prior variance. We initially used a high variance because there is very little certainty at the beginning of this algorithm. The hyperparameters did not seem to mat-

ter for this algorithm. Adjusting the variance of the data did not matter, as we held it constant for each point.

7. How robust is this algorithm to noise?

With 4 more noise corrupted versions of the features, the algorithm stayed at 70% success for one class and 92% success for the other within 4 trials. With 4 more random noise features, the success rate of the two classes fluctuated between 90% and 10% within 4 trials. Overall, it handles noise poorly.

4 Kernelized SVM

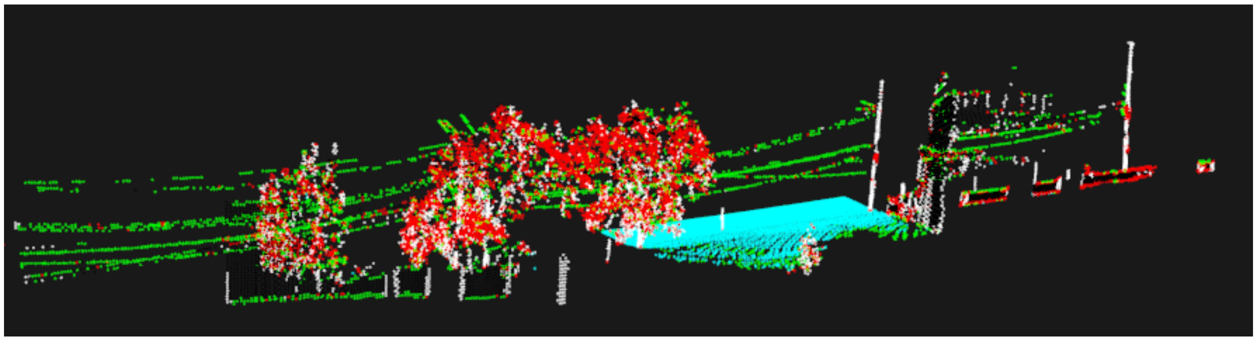


Figure 4: Kernelized SVM classification results on test data.

1. How well did it perform for online learning? Does it perform well on the held-out data?

This algorithm learns the features online somewhat well, and performs comparably on the held-out data. It results in around 80% correct classification, with the no individual class misclassified more than 40% incorrectly.

2. Are there any classes that did not get classified well? Why do you think that is?

The classes that were classified worst were generally veg, facade, and pole. In our implementation, we gave the classifier the same number of examples from each class, so this discrepancy between the classes is probably due to the feature set rather than the data itself.

3. How easy was the learner to implement?

This learner was the most difficult to implement than the other learners we implemented (BLR and online gradient descent). The complexity added by the kernels is more difficult to grasp conceptually, but also introduces the potential for more bugs, which consumed most of our development time for this algorithm.

4. How long does the learner take (in terms of data points, dimensions, classes, etc...) for training and prediction?

SVM with RBF's computational complexity is more than quadratic with the number of samples for training and O(n) for each data point in the prediction. It is not very well scaled with the number of samples and run the longest among three algorithms. In the actual implementation, the complexity measured by time elapsed is shown below. For the data points, we halved the numbers. For the classes, we decreased from 4 classes to 3 and 2. For features, we halved the dimension.

Variable	Action	Training (s)	Complexity	Testing (s)	Complexity
Data Points	Half the data points	12.3, 6.3, 3.1	O(n)	68, 34.3, 17.3	O(n)
Classes	4, 3, 2	3.7, 1.9, 0.6	O(n^2) to O($n^2 log(n)$)	27.7, 21, 14	slightly worse than O($\sqrt{n}$)
Feature Dimensions	Half the dimensions	6.1, 6.3, 9.1	O(1)	34.5, 34.3, 54	O(1)

5. Show images/movies of the classified data.
Figure 4 shows the classification results for the kernelized SVM.

6. How did you choose (hyper)parameters (priors, kernel width, noise variance, prior variance, learning rate, etc...)?
Like the other algorithms, our prior was zero, the kernels had unit variance, the features were standardized to zero mean and unit variance, the learning rate was $frac1\sqrt{T}$, and the weight of the squared norm in the lost function was $\lambda = 0.01$. The choice of kernel width was somewhat arbitrary, the learning rate was selected for no-regret, and the weight of the squared norm was chosen to be small to allow the function to conform to the nonlinearities of the feature set.

7. How robust is this algorithm to noise?
With 4 more noise corrupted versions of the features, the algorithm stayed at 80% success in 2 trials. With 4 more random noise features, the algorithm stayed at 70% success in 2 trials.

5 Conclusion

The online gradient descent and the kernelized SVM worked best in terms of prediction accuracy, with the online gradient descent having a lower training as well as predicting time. The kernels did help in improving the prediction accuracy albeit at the cost of additional run time due to computational overheads. Of these three algorithms in their current state, would use online gradient descent on a real robot as it is linear in complexity and gives a very good accuracy. However, given more time to improve the kernelized SVM, we would deploy the kernelized SVM algorithm because it seems to have higher potential for correct classification. Future work on these algorithms would include tuning parameters