Online Learning

Akshay Pramod Roy* and Dhaneshwaran Jotheeswaran*

School of Electrical and Computer Engineering

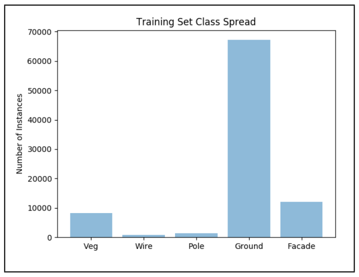
E-mail: apr8@gatech.edu; dhaneshwaran@gatech.edu

Abstract

Classifying objects using Ladar data is crucial for Robot Navigation, as it enables the robot to navigate smoothly. In this project we implemented 3 Online Learning algorithms to classify 5 objects namely, ground, vegetation, facade, pole, and wire.

Introduction

The dataset we are using is the 3D point-cloud data of Oakland in Pittsburg, collected from a Mobile Robot. It has 5 classes, and 9 features (plus a bias term). The dataset was split into a training set and a test set. As we can see, the dataset is highly skewed, both in the training set and the test set. The Ground class dominates the training set, so we might see other classes misclassified a fair amount, as we wouldn't have seen enough examples that is representative of these classes. The approach that we took to implement the Learning algorithms was the same. We trained on the larger dataset, by shuffling it and going through the entire dataset 10 times, to simulate an online-to-batch problem. Then, we chose the last weights, and tested it on the test set. For checking robustness, we added 5 features with uniform noise between [-1, 1] and ran the algorithms. In all Confusion Matrices, y-axis is actual label, and x-axis is predicted label. The plots for average regret over time are also shown. Note that this is not Regret, rather Average Regret. All average regrets go to zero in the limit. The regret plot would show the sub-linear growth of regret, but owing to space, we show average regret zoomed in.



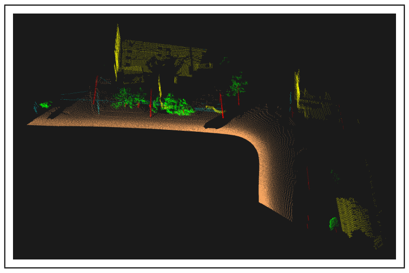


Figure 1: Training Set Spread, and Visualization

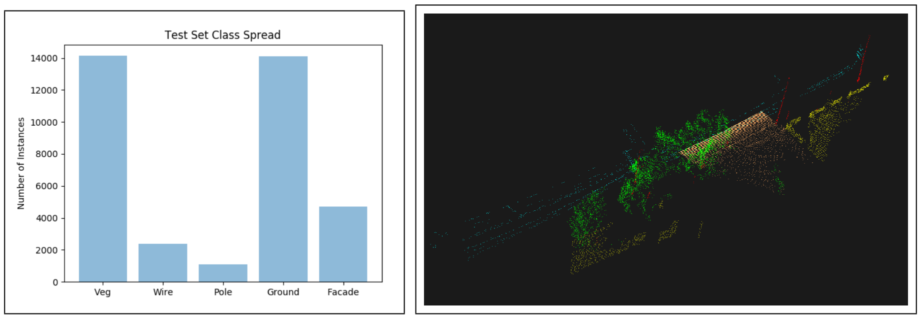


Figure 2: Testing Set Spread, and Visualization

Linear Regression

This is the simplest of all algorithms implemented. We try a fit a hyperplane that separates the individual classes. We use Gradient Descent on Squared Loss to optimize the weights for each feature. We performed Grid search over a few learning rates, and chose the best one for our test set. However, better results could be obtained using learning rate as a function of dataset size or by having a learning that would adapt itself based on how much the values have converged. And also, we went over the data 10 times, to make it more accurate(online to batch learning). As we can see, Veg is classified as Facade a lot of times. This might be because these 2 classes are not linearly separable. And, as expected Wire and Pole are misclassified a lot of times. This might be for the same reason mentioned above and also because there aren't enough instances of this class in the training set.

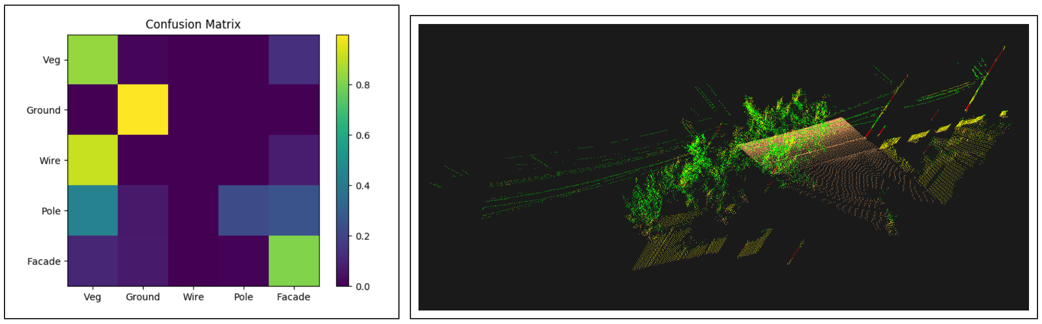
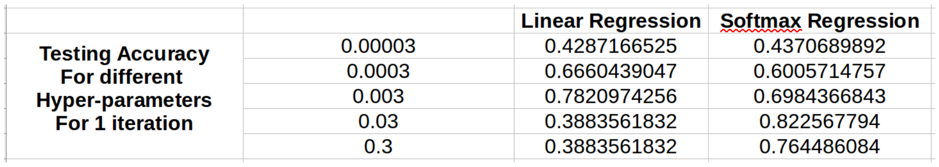


Figure 3: Linear Regression: Confusion Matrix, and Visualization



		Linear Regression	**Softmax Regression**
Testing Accuracy For different Hyper-parameters For 1 iteration	0.00003	0.4287166525	0.4370689892
	0.0003	0.6660439047	0.6005714757
	0.003	0.7820974256	0.6984366843
	0.03	0.3883561832	0.822567794
	0.3	0.3883561832	0.764486084

Figure 4: Hyperparameter Tweaking: Linear Regression and Softmax Regression

The plot for average regret over time is shown below. And also, on the noisy data, the results were almost the same. So, this is a robust algorithm.

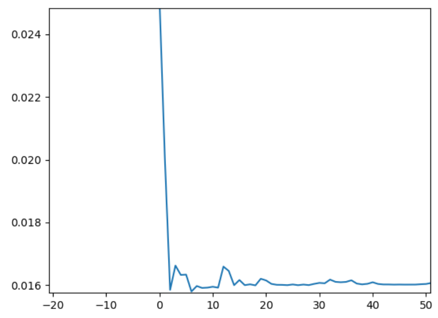


Figure 5: Linear Regression: Average Regret vs Iterations

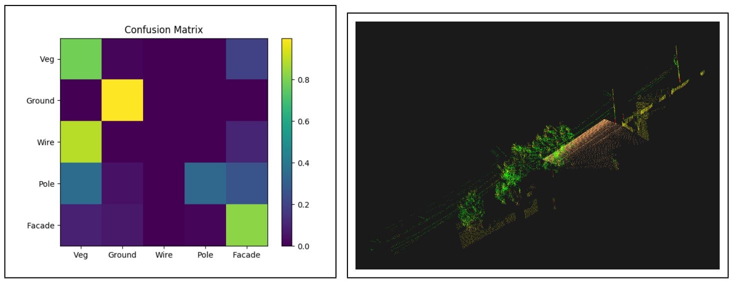


Figure 6: Linear Regression: Confusion Matrix, and Visualization for Noisy Data

Bayesian Linear Regression

This was also fairly easy to implement. Here, we try a fit a hyperplane that separates the two classes (We chose Ground and Veg), but instead of using Gradient Descent to optimize the weights for the features, we do that by using Bayes Rule. We tried different Co-variance Matrices and Mean Vectors initialization, and ended up choosing a scaled identity Matrix for co-variance, and a vector of ones, for the mean vector. We chose large variances, as we knew little about the data we were dealing with. But, we could have done more intelligent things here, like having a validation set, and seeing how our model performs on it to choose the best hyperparameters, or we could have done k-fold Cross-Validation. This classifier performs well on these 2 classes, which might be because these 2 classes are linearly separable and/or there are enough examples of these 2 classes in the training set.

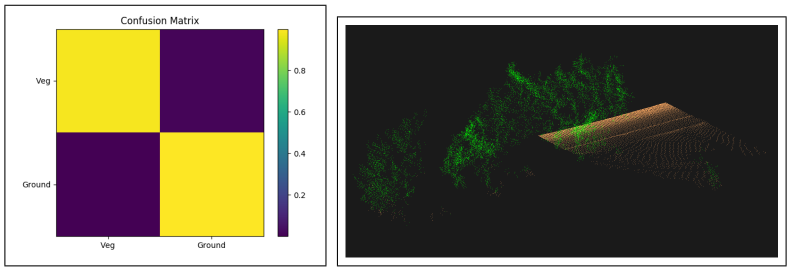


Figure 7: Bayesian Regression: Confusion Matrix, and Visualization

The plot for average regret over time is shown below. And also, on the noisy data, the results were almost the same. So, this is a robust algorithm.

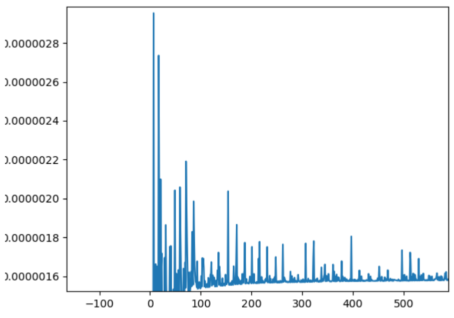
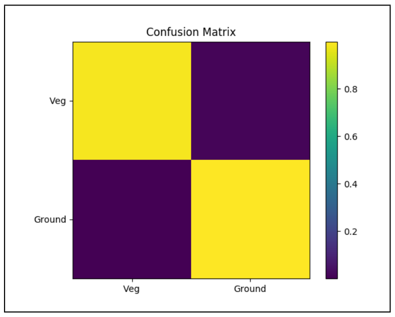


Figure 8: Bayesian Regression: Average Regret Vs Iterations



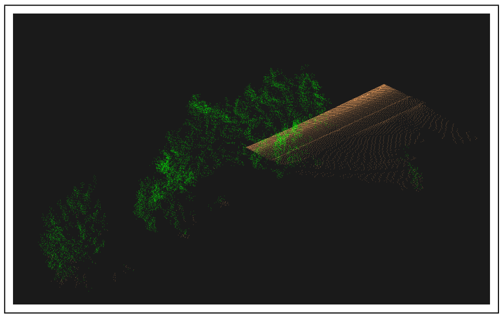


Figure 9: Bayesian Regression: Confusion Matrix, and Visualization for Noisy Data

Softmax Regression (Multi-Class Logistic Regression)

This algorithm was also fairly easy to implement. We find the probability that the datapoint came from each class using the softmax function, which is just exponentiation of the output from the linear combination of all the features, and normalizing them. This is a generalization of logistic regression for multi-class classification. We use Gradient Descent on Cross-Entropy Loss to optimize the weights for each feature. We performed Grid search over a few learning rates, and chose the best one for our test set. However, better results could be obtained using learning rate as a function of dataset size or by having a learning that would adapt itself based on how much the values have converged. And also, we went over the data 10 times, to make it more accurate. Similar problems which existed previously were also present in the Softmax Classifier, but not to the extent of the other 2 algorithms. But, wire got classified correctly here, even with the few data available, as can be seen from the confusion matrix.

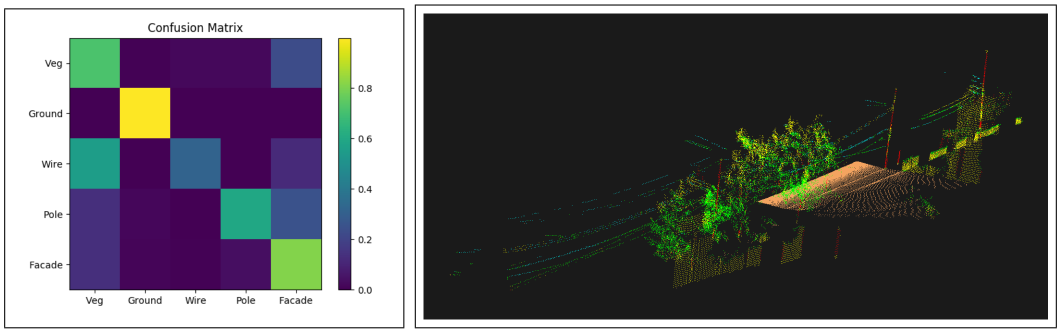


Figure 10: Softmax Regression: Confusion Matrix, and Visualization

The plot for average regret over time is shown below. And also, on the noisy data, the results were almost the same. So, this is a robust algorithm.

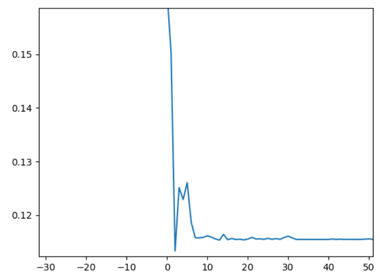


Figure 11: Softmax Regression: Average Regret Vs Iterations

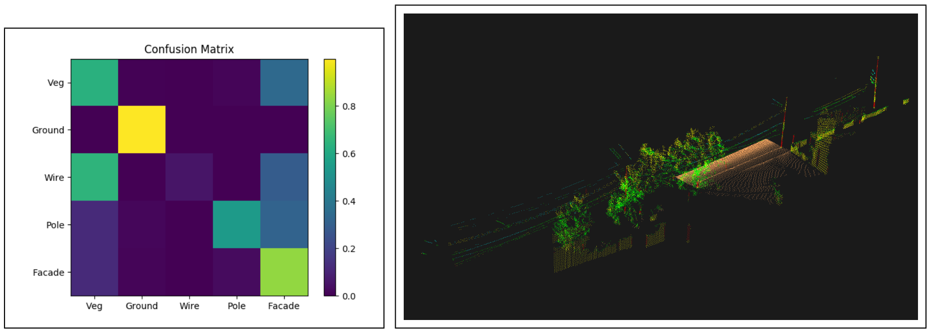
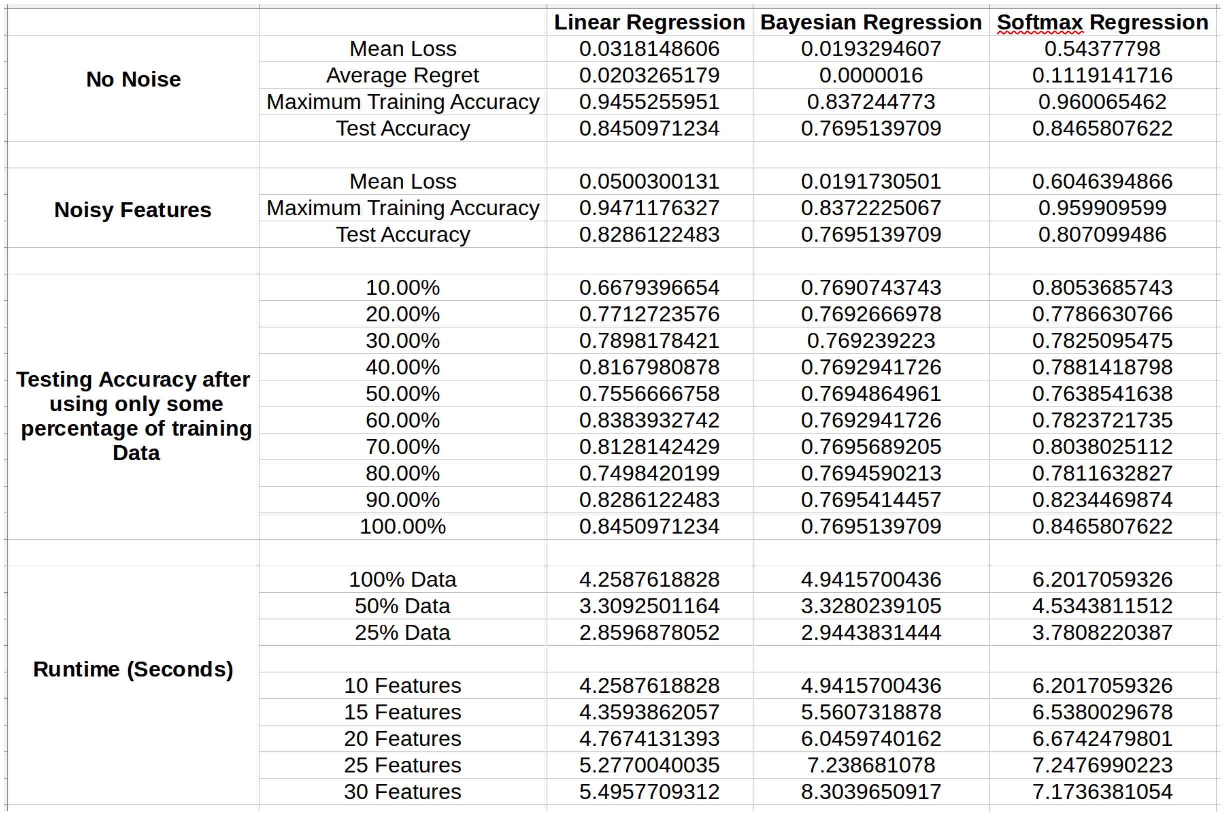


Figure 12: Softmax Regression: Confusion Matrix, and Visualization for Noisy Data

Conclusion

All the algorithms gave good accuracy on the test set, but if we had to rank them, Softmax Classifier would come first, Linear Regression a close second, and Bayes Regression a distant third. In terms of computational efficiency, Linear Regression is fastest, followed by Bayes Linear Regression, and then Softmax Regression. Considering all things, we would choose the Softmax Classifier among these 3 classifiers, because it gave the highest accuracy, and is robust. But above all, from the table below, we will choose Softmax for our Robot. Because, it requires the fewest data to reach high accuracy, and also it gave the highest accuracy for Wire. But, though it is not as robust as the Linear Regressor, we will make this trade-off because of its high accuracy and the sample efficiency. A detailed comparison is shown in the table.



		Linear Regression	**Bayesian Regression**	**Softmax Regression**
No Noise	Mean Loss	0.0318148606	0.0193294607	0.54377798
	Average Regret	0.0203265179	0.0000016	0.1119141716
	Maximum Training Accuracy	0.9455255951	0.837244773	0.960065462
	Test Accuracy	0.8450971234	0.7695139709	0.8465807622
Noisy Features	Mean Loss	0.0500300131	0.0191730501	0.6046394866
	Maximum Training Accuracy	0.9471176327	0.8372225067	0.959909599
	Test Accuracy	0.8286122483	0.7695139709	0.807099486
Testing Accuracy after using only some percentage of training Data	10.00%	0.6679396654	0.7690743743	0.8053685743
	20.00%	0.7712723576	0.7692666978	0.7786630766
	30.00%	0.7898178421	0.769239223	0.7825095475
	40.00%	0.8167980878	0.7692941726	0.7881418798
	50.00%	0.7556666758	0.7694864961	0.7638541638
	60.00%	0.8383932742	0.7692941726	0.7823721735
	70.00%	0.8128142429	0.7695689205	0.8038025112
	80.00%	0.7498420199	0.7694590213	0.7811632827
	90.00%	0.8286122483	0.7695414457	0.8234469874
	100.00%	0.8450971234	0.7695139709	0.8465807622
Runtime (Seconds)	100% Data	4.2587618828	4.9415700436	6.2017059326
	50% Data	3.3092501164	3.3280239105	4.5343811512
	25% Data	2.8596878052	2.9443831444	3.7808220387
	10 Features	4.2587618828	4.9415700436	6.2017059326
	15 Features	4.3593862057	5.5607318878	6.5380029678
	20 Features	4.7674131393	6.0459740162	6.6742479801
	25 Features	5.2770040035	7.238681078	7.2476990223
	30 Features	5.4957709312	8.3039650917	7.1736381054

Figure 13: Comparison of algorithms