ECE/CS/ME 532 Project Report

Victor Freire
Department of Mechanical Engineering
University of Wisconsin-Madison
Maddison, WI 53706
`freiremelgiz@wisc.edu`

Project GitHub Page:
`https://github.com/freiremelgiz/ECE532_FinalProject`

1 Introduction

I am a Mechanical Engineering MS - Research student working with Professor Xiangru Xu in research about control theory applied to quadcopters. I found a dataset containing network traffic data for devices labeled as either UAV or something else. Classifying network device type based on traffic data may not be directly applicable to control theory. However, a control algorithm could use labels provided by this classifier to make path-planning decisions in a cluttered network environment. In this document I describe the dataset and classification algorithms I used in detail. Please refer to my Project GitHub page for tables and plots showing the results of each classification algorithm.

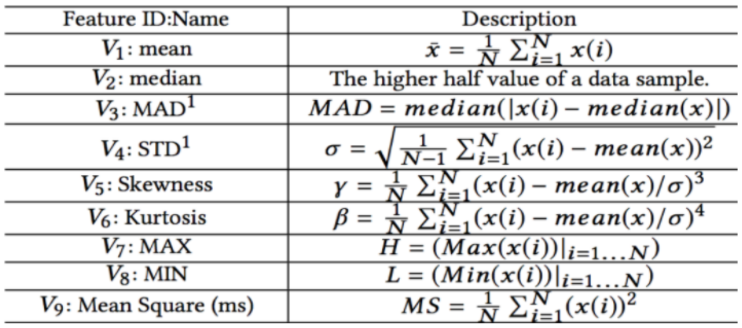
2 Project Dataset - Unmanned Aerial Vehicle (UAV) Intrusion Detection

The dataset I chose was posted at the *UCI Machine Learning Repository*. However, the data folder was empty. I searched for the author and found his personal *website*. In his space, there is more information about the dataset and it can be downloaded.

The author collected data for 3 quadcopter brands and 2 data flow directions for each brand (6 datasets). Each dataset also contains two matrices of "testing data" and "training data". Each matrix consists of 9 statistic measurements of network traffic **timing** and **size** for a total of 18 features. The data collection was performed in both **unidirectional** and **bidirectional** data flow. The bidirectional flow datasets include 3x more features because the data comes from uplink, downlink and a combined total. Figure 1 shows a mathematical description of each feature. The datasets are also labeled to train classifiers. Traffic features coming from a UAV are labeled with +1. Other features are labeled as 0. Table 1 summarizes the dimensions of each dataset.

dataset	m: datapoints [training-testing]	n: features
1	1751-17629	54
2	1569-15687	54
3	500-5000	54
4	1063-10600	18
5	1351-13513	18
6	7-66	18

Table 1: Dataset dimension summary.



Feature ID:Name	Description
V_1: mean	$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x(i)$
V_2: median	The higher half value of a data sample.
V_3: MAD[1]	$MAD = median(\|x(i) - median(x)\|)$
V_4: STD[1]	$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x(i) - mean(x))^2}$
V_5: Skewness	$\gamma = \frac{1}{N}\sum_{i=1}^{N}(x(i) - mean(x)/\sigma)^3$
V_6: Kurtosis	$\beta = \frac{1}{N}\sum_{i=1}^{N}(x(i) - mean(x)/\sigma)^4$
V_7: MAX	$H = (Max(x(i))\|_{i=1\ldots N})$
V_8: MIN	$L = (Min(x(i))\|_{i=1\ldots N})$
V_9: Mean Square (ms)	$MS = \frac{1}{N}\sum_{i=1}^{N}(x(i))^2$

Figure 1: Feature description based on raw data. The features are found in the dataset along with labels.

3 Classification Algorithms

A number of classification algorithms and techniques were used to sort each dataset into UAV (+1) or non-UAV (-1). The general approach was to train the classifier using the provided training data and then examine its performance on the test set. This process was done for each algorithm on each of the 6 datasets.

3.1 Least Squares

Training a set of weights $\mathbf{w}$ using Least Squares involves solving the convex optimization problem

$$\arg\min_{\mathbf{w}} ||X\mathbf{w} - \mathbf{y}||_2^2$$

This problem can be solved in closed form with

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$$

Computing the $(X^T X)^{-1}$ matrix can sometimes be computationally expensive. One way to speed-up the computation of the inverse is to use a low-rank approximation of the data matrix. I used Principal Component Analysis to understand which features are most important in each dataset and try to solve the Least Squares problem on a low-rank approximation of the training data matrix. The rank-r approximation was computed using truncated Singular Value Decomposition.

$$X_r = \sum_{i=1}^{r} \mathbf{u}_i \sigma_i \mathbf{v}_i^T$$

I examined the performance of the classifier weights $\mathbf{w}$ trained with the full-rank training data as well as on the low-rank approximation. An interesting exercise I didn't explore would be to benchmark the training time for full-rank and low-rank models. This should help understand the benefits of low-rank models.

3.2 Ridge Regression

The Ridge Regression algorithm (Also known as Tikhonov Regularization) is nothing more than Least Squares with an L2 norm regularization term on the weights $\mathbf{w}$.

$$\arg\min_{\mathbf{w}} ||X\mathbf{w} - \mathbf{y}||_2^2 + \lambda||\mathbf{w}||_2^2$$

This problem can also be solved in closed form with

$$\mathbf{w} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}$$

This algorithm addresses the tendency of Least Squares to overfitting the data. I chose an appropriate regularization parameter λ for each dataset using cross-validation. I first trained a number of classifiers $\mathbf{w}_\lambda$ with different values of λ. Then, I tested the performance of each $\mathbf{w}_\lambda$ on a holdout set of 10% of the testing data. I chose the largest λ that resulted in the fewest amount of misclassifications to be the best.

3.3 Hinge Loss

The Hinge Loss cost function addresses the limitation of the Mean Squared Error cost function with regard to data outliers. In other words, Hinge Loss is insensitive to datapoints which are far away from the decision boundary. Training a classifier with Hinge Loss involves solving the convex optimization problem

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} (1 - y_i \mathbf{x}_i^T \mathbf{w})_+$$

This problem has no closed-form solution. I used iterative Gradient Descent to find the minimum. The gradient of the Hinge Loss function is

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = -\sum_{i=1}^{N} \frac{1}{2} y_i (1 + sign(1 - y_i \mathbf{x}_i^T \mathbf{w}))\mathbf{x}_i$$

With this gradient, the optimal weights $\mathbf{w}$ can be found iterating

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \tau \nabla_{\mathbf{w}} f(\mathbf{w})$$

For a sufficiently small step size τ to avoid divergence. Given the large singular values associated with my training matrices, this step size τ was very small in my algorithm. This resulted in slow convergence rates for gradient descent. Please take a look at the `IterReg (Convex)` feature described on the Project GitHub page for the strategy I used to solve this problem.

3.4 Support Vector Machines

The Support Vector Machine classification algorithm uses the Hinge Loss cost function with a L2 norm regularization term. This regularization term ensures that the minimization converges to the max-margin decision boundary. This is the boundary at the maximum distance from the closest datapoints.

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} (1 - y_i \mathbf{x}_i^T \mathbf{w})_+ + \lambda ||\mathbf{w}||_2^2$$

I used the same values for λ that I found on the Ridge Regression algorithm. The gradient for the SVM problem is

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = -\sum_{i=1}^{N} \big(\frac{1}{2} y_i (1 + sign(1 - y_i \mathbf{x}_i^T \mathbf{w}))\mathbf{x}_i\big) + 2\lambda \mathbf{w}$$

And iterative gradient descent can also be used

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \tau \nabla_{\mathbf{w}} f(\mathbf{w})$$

3.5 Neural Network

Neural Networks are nonlinear functions capable of creating arbitrarily complex decision boundaries. As proven by the Universal Approximation Therorem, a neural network with one hidden layer and enough hidden nodes can be used to approximate any function. I use this theorem to train a neural network with one hidden layer and one output to approximate a complex decision boundary for each dataset. I trained a total of 6 different neural networks (one for each dataset). The objective function I minimized to train the neural network is the Squared Error Loss

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} \frac{1}{2}(\hat{y}_i - y_i)^2$$

where

$$\hat{y}_i = \sigma\left(\sum_{k=1}^{r} v_k \sigma\left(\sum_{j=1}^{n} x_{ij} w_k\right)\right)$$

For r hidden nodes, n features and $i = 1, 2, ..., N$ datapoints. I used the logistic function (sigmoid) as the activation function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

I used backpropagation of the training data via Stochastic Gradient Descent to train each neural network. This minimization problem is non-convex. This means there may be multiple local minimum where Gradient Descent could get stuck while iterating. Please take a look at the `IterReg (Non-Convex)` feature described on the Project GitHub page for the strategy I used address this issue.

4 Results

Please refer to the Project GitHub README page for tables and plots showing the classification results for each algorithm.

Overall, I found classifying the datasets which had the most training data to be easiest. Many algorithms correctly classified datapoints in these cases. For dataset 6, which had only 7 training datapoints and 18 features, Least Squares performed worse than a coin-toss. Hinge loss was marginally better. Regularization techniques showed little improvement. A trained neural network even with only 7 training datapoints, however, achieved a misclassification rate of only 10% in this scarce dataset. It is also worth noting that Hinge Loss and SVM achieved perfect classification of dataset 2.

5 Conclusion

Deciding which algorithm to use to classify a certain dataset is a complicated task. This project helped me understand the strengths and limitations of just a few classification algorithms. Often times, the metrics being balanced are computational complexity, ability to generalize, training set performance and more. Certain algorithms are developed to address specifically some of these metrics. The table below provides an overall summary on which algorithm was most effective at classifying each dataset.

Dataset	**Best Algorithm**	**Error**
1	Least Squares	0.20 %
2	Hinge Loss	0.00 %
3	Least Squares	1.78 %
4	Least Squares	3.25 %
5	Neural Network	4.68 %
6	Neural Network	10.61 %