How Much Did it Rain?

MATP-4820 Computational Optimization

Matthew Poegel, Thomas Wagner, Jacob Coutu

May 13, 2016

Abstract

Efforts were made to improve the accuracy of current methods to predict actual rainfall measured by rain gauges by means of machine learning on two different kinds of radar data, NEXRAD and MADIS. This problem was tackled with linear regression, support vector regression, and support vector machines that used robust optimization algorithms in MATLAB and NEOS to solve the underlying nonlinear programs. Difficulties emerged in the shear size and magnitude of the data, making it challenging to run complete analyses given the computational constraints. Simple linear regression was unable to fully model the data, as it never predicted rainfall outside the neighborhood of $\hat{y} = 0$. SVR with a fourth degree polynomial kernel was much more fit to solve the problem, but cross validation could not be run due to the computational limits. Thus it did not generalize well to the test data. The final model used an SVM to first classify the data into either "rain" or "no rain" classes and then ran linear regression on the "rain" class with some degree of success.

1 Introduction

Accurately predicting the weather is a very difficult task. Yet as we evolve and create more advanced and distributed instruments, society is improving upon its ability to forecast the weather. This investigation focuses on predicting rainfall—after the fact. Given constantly improving radar techniques, the goal is to refine these instruments to measure how much rain will fall in an area. This would replace physical rain gauges, which are often inaccurate or misused and are limited by location, whereas radar can more easily cover a large area.

2 Problem Overview

The goal of this analysis was to predict the actual amount of rainfall from the polarimetric radar measurements. Since predictions from radar measurements are only estimates and thereby not completely accurate indicators for the amount of rainfall, machine learning techniques were applied to create a better estimator. Given radar readings over a period of time, the goal was to estimate how much rain fell during that time period. This estimator could then used independently from rain gauges on the ground.

3 The Data

3.1 Description

The data was generated by Next-Generation Radar (NEXRAD) and Meteorological Assimilation Data Ingest System (MADIS) between April and August of 2014 over the corn-growing states in the Midwest. The data consists of polarimetric radar measurements and actual rainfall gauge measurements. The data was grouped by hour, with multiple and varying intervals at which measurements were taken. The actual rainfall gauge measurement was reflective of the reading at the end of the hour interval. Lakshmana et al. of the Artificial Intelligence Committee of the American Meteorological Society aggregated the data and made it available to the public through Kaggle [3] [1].

3.2 Munging

Before any analysis could begin, the data first had to be cleaned and manipulated into a usable form. Furthermore, since the training data alone was over one gigabyte, computational restrictions prevented the analysis of the data direct from the source. To combat this, the data set was purged of implausible data using a multithreaded program written in C. An implausible data point was defined as any data for which the corresponding rain gauge measurement was above the world record for the most rainfall in one hour. This record is 305mm from Holt, Missouri in 1947 and Kauai, Hawaii in 1956 [2].

However, this still left the training much too large, given the computational restrictions of the available resources. There was also still a lot of missing data. To fix both problems, data points missing any measurements at all were also discarded. This relies on the assumption that missing data was randomly distributed in the data. What remained was a $2,756,266 \times 24$ matrix.

Finally, two techniques were devised to overcome the challenge of dealing with a variable number data points per hour. The first, and more simple, technique was to collapse the readings down to the mean. This will be referred to as Mean Compression. While this loses a lot of information, it provided for easier computation and was a quicker solution. The matrix that this produced was 391254×24. The second technique was to interpolate a polynomial function for each variable for each hour and re-sample that polynomial at constant intervals. This will be referred to as Interpolation Expansion. This algorithm was slightly more intricate, using Newton's Divided Differences method to calculate the interpolating polynomial [6]. This technique had the advantage of maintaining the time dimension of the data but drastically increased the feature space. Using $n = 5$ sampling times, the matrix produced was 391254×103.

4 Nonlinear Programming Problem

In the scope of the data in this report the dimensionality of the data is as follows.

Mean Compression	**Interpolation Expansion**
$C \in \mathbb{R}, N = 391254, b \in \mathbb{R}$	$C \in \mathbb{R}, N = 391254, b \in \mathbb{R}$
$\mathbf{w} \in \mathbb{R}^{22}, \mathbf{z} \in \mathbb{R}^N, \mathbf{y} \in \mathbb{R}^N$	$\mathbf{w} \in \mathbb{R}^{102}, \mathbf{z} \in \mathbb{R}^N, \mathbf{y} \in \mathbb{R}^N$
$X \in \mathbb{R}^{Nx22}$	$X \in \mathbb{R}^{Nx22}$

Table 1: Dimensionality of the data.

4.1 Ridge Regression

Ridge regression expands the simpler implementation of linear regression by adding a regularization term to the objective function. The ridge regression problem is as follows.

$$\min_{\mathbf{w}} \quad (\mathbf{X}\mathbf{w} - \mathbf{y})^2 + \lambda \mathbf{w}^\intercal \mathbf{w} \tag{1}$$

Here λ is a parameter which is given to the algorithm in order to penalize the function for choosing large values for the componenets of $\mathbf{w}$. Problem 1 was solved using NEOS as described in Section 5.3.

4.2 Support Vector Regression

The support vector regression (SVR) method was used as a more robust model than simple linear regression. The SVR problem is as follows [7].

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}(\mathbf{z}_n - \mathbf{z}_n^*) \\ \text{s.t} \quad & \mathbf{y}_n - \mathbf{w}^\intercal \mathbf{x}_n - b \leq \epsilon + \mathbf{z}_n \\ & \mathbf{w}^\intercal \mathbf{x}_n + b - \mathbf{y}_n \leq \epsilon + \mathbf{z}_n \\ & \mathbf{z}, \mathbf{z}^* \geq 0 \end{aligned} \tag{2}$$

Instead of solving this problem, the dual of Problem 2 was solved [7].

$$\begin{aligned} \max_{\lambda} \quad & -\frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}(\lambda_i - \lambda_i^*)(\lambda_k - \lambda_k^*)\mathbf{x}_i^\intercal \mathbf{x}_k - \epsilon\sum_{i=1}^{N}(\lambda_i - \lambda_i^*) + \sum_{i=1}^{N}\mathbf{y}_i(\lambda_i - \lambda_i^*) \\ \text{s.t.} \quad & \sum_{i=0}^{N}(\lambda_i - \lambda_i^*) = 0 \\ & \lambda, \lambda^* \in [0, C] \end{aligned} \tag{3}$$

The dual SVR Problem 3 was solved using `quadprog` as described in Section 5.2.

4.3 Support Vector Machine

The support vector machine (SVM) algorithm was employed to created to a two-class classifier. Because the data was not expected to be separable, the soft-margin SVM was used, namely,

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\mathbf{z}_i \\ \text{s.t.} \quad & \mathbf{z}_i + \mathbf{y}_i(\mathbf{x}_i\mathbf{w} - b) - 1 \geq 0 \end{aligned} \tag{4}$$

Problem (4) is known as the primal soft-margin SVM. The algorithm that was written to create the classifier actually solves the dual of (4).

$$\begin{aligned} \min_{\lambda} \quad & \frac{1}{2}\sum_{i=1}^{m}\sum_{k=1}^{m}\lambda_i\lambda_k\mathbf{y}_i\mathbf{y}_k\mathbf{x}_i^\intercal\mathbf{x}_k - \sum_{i=1}^{m}\lambda_i \\ \text{s.t.} \quad & \sum_{i=0}^{m}\lambda_i\mathbf{y}_i = 0 \\ & \lambda \geq 0 \end{aligned} \tag{5}$$

The dual problem corresponds to finding a separator for the reduced convex hulls for each class. Program (5) was solved using `fmincon` as described in Section 5.1.

5 Optimization Algorithm

5.1 fmincon

As shown in Section 4.3, the dual of the primal SVM problem tries to find the minimum separator for the reduced convex hulls for each class. As a convex minimization problem, this problem could be solved using the Matlab command `fmincon`.[4]

5.2 quadprog

As shown by Equation 3 in Section 4.2, the dual of the primal SVR problem is a quadratic program. Thus, this program was best solved using the Matlab command `quadprog`.[4] The `quadprog` command was set to use the Interior Point Convex Method to solve the program.

5.3 NEOS

As shown in Section 4.1, the Ridge Regression problem tries to find the minimum value for an unconstrained convex minimization problem. As a convex minimization problem, it could be solved using the MINOS solver supplied by NEOS.[5]

6 Experiments

The data was separated into training and test sets and munged independently using identical procedures as described in Section 3.2. The training data consisted of the first twenty days of each month (April to August) and the testing data consisted of the remaining ten or eleven days of each month. The models were tuned on the training data to create E_{in} and evaluated on the testing data to yield E_{test}. Two different loss functions were compared for linear regression: root mean squared error and mean absolute error. MAE was considered because this was the loss function used in the Kaggle competition that was justified by Willmott and Matsuura as being more suitable for climate-related models over RMSE [8].

6.1 Linear Regression

The linear model was created for each version of the data. First, Ridge Regression was used on the Mean Compressed data. For this model, the training data used the first 10,000 points and was tested on 2,000 points. Because the analytical form for linear regression was used,

$$\mathbf{w} = (\mathbf{X}^\intercal\mathbf{X} - \lambda^*\bar{\mathbf{I}})^{-1}\mathbf{X}^\intercal\mathbf{y}$$

[9]
a large sample of the data could be used to train the model. Cross validation was used to select the regularization parameter, λ^*, using the formula

$$E_{\text{cv}} = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{\hat{y}_n - y_n}{1 - \mathsf{H}_{nn}(\lambda)}\right)^2 \tag{6}$$

where $\mathsf{H}(\lambda) = \mathbf{X}(\mathbf{X}^\intercal\mathbf{X} - \lambda\mathbf{I})^{-1}\mathbf{X}$ [9]. From the cross validation, λ^* was chosen such that the E_{cv} was at its minimum. Then the optimal model was created and E_{in} and E_{test} were calculated. These steps were repeated for the Interpolation Expanded data.

6.2 Support Vector Regression

The SVR model was created with the parameters $C = 10.0$ and $\epsilon = 1 \times 10^-5$. Only $N = 1000$ data points could be used to create the model because of computational restrictions. For the same reason, cross validation could not reasonably be used to choose the parameters C and ϵ without sacrificing too many data points. The kernel function that was used was the fourth degree polynomial kernel,

$$\mathsf{K}(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\intercal\mathbf{y} + 1)^d$$

where $d = 4$. The models were tested on 200 data points.

6.3 Combination Method

The final strategy for creating a model combined both classification and linear regression. The idea of this was model was to classify the data as either "rain" or "no rain". For class "no rain", the final $\hat{y}$ outputted was 0. For class "rain", an additional regression model was created using only the points in the "rain" class. This regression outputted the final $\hat{y}$ value for "rain" class only.

To accomplish this, the data was separated into two classes using $\epsilon = 0.025$. If $\mathbf{y}_n > \epsilon$, then $\mathbf{x}_n \in C_1$. If $\mathbf{y}_n \leq \epsilon$ then $\mathbf{x}_n \in C_{-1}$. The threshold, ϵ, was chosen after a careful review of figure 1. Then the SVM model was created using 10-fold cross validation to choose C^* with a fifth order inhomogenous kernel, as shown in section 6.2 where d = 5. Since the cross validation computation was so demanding, the input data had to be restricted to $n = 750$: $\mathbf{X} \in \mathbb{R}^{750\times 101}$, which is less than ideal. From there the regression model was created in the same manner described in Section 6.1 and the training and test errors were computed.

These classification models were judged using two different error metrics. The first being raw error, that is the total number of miss-classified points divided by the total number of points evaluated. The second being the percentage of false negatives, that being the number of "rain" points which the model classified as "not rain" points divided

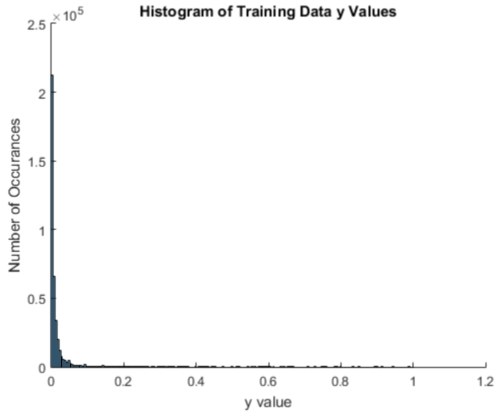


Figure 1: Histogram of the y values in the training data.

by the number of "rain" points. This second error measure was decided to be more valuable because a regression model would simply identify any false positives as points with little rainfall whereas false negatives would be lost from the model at this point.

This model, with the C* parameter set, was then evaluated on a set of testing data using both of these error metrics.

7 Results

7.1 Linear Regression

For the Mean Compressed data, the training MAE, E_{in}, was 0.0184. The testing MAE, E_{test} was 0.0180. The relevant plots depicting the performance of the linear regression model for these data can be seen in 2.

For the Interpolated Expanded data, the training MAE, E_{in}, was 0.0193. The testing MAE, E_{test} was 0.0208. The relevant plots depicting the performance of the linear regression model for these data can be seen in 3.

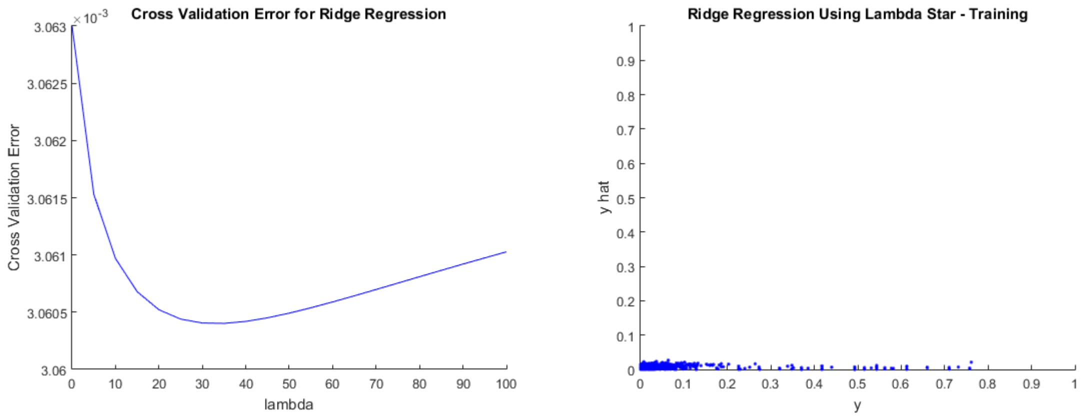
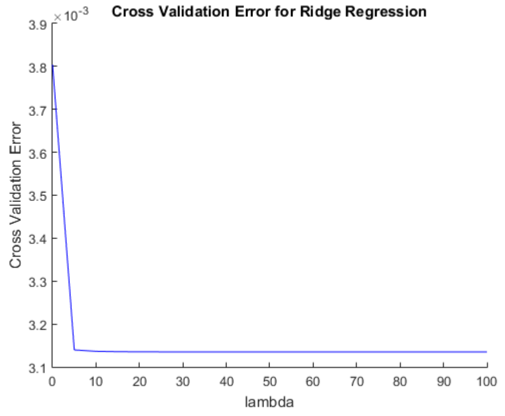


Figure 2: Left: Cross validation error plotted against the regularization parameter, λ, for the Mean Compressed training data. Right: Scatter plot of the predicted value, $\hat{y}$, against the the actual y value using the Ridge Regression model created using $\lambda^* = 35$ on the Mean Compressed training data.



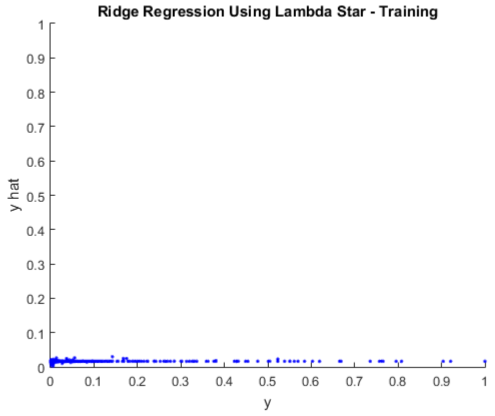


Figure 3: Left: Cross validation error plotted against the regularization parameter, λ, for the Interpolation Expanded training data. Right: Scatter plot of the predicted value, $\hat{y}$, against the the actual y value using the Ridge Regression model created using $\lambda^* = 60$ on the Interpolation Expanded training data.

7.2 Support Vector Regression

For the Mean Compressed data, the training error, E_{in} for the SVR with a fourth degree polynomial was 0.0103. The testing error, E_{test} was 0.0586. The relevant plots depicting the performance of the SVR model are seen in Figure 4.

For the Interpolation Expanded data, the training error, E_{in} for the SVR with a fourth degree polynomial was 0.0152. The testing error, E_{test} was 0.0150. The relevant plots depicting the performance of the SVR model are seen in Figure 5.

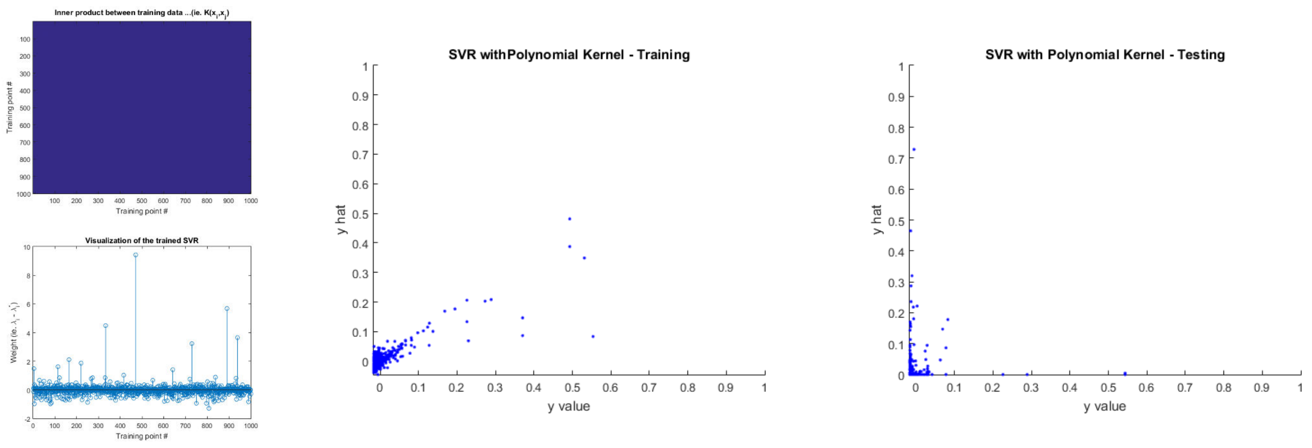


Figure 4: Summary of the SVR model created using the Mean Compressed data. Top Left: The Gram matrix, K, for the SVR model produced, where $K_{i,j} = \mathsf{K}(\mathbf{x}_i, \mathbf{x}_j)$. Bottom Left: The weights of each data point in the SVR model, $\lambda_i - \lambda_i^*$. The points whose weight is between 0 and C are considered the support vectors. Center: Scatter plot of the predicted value, $\hat{y}$, against the actual y value for the training data. Right: Scatter plot of the predicted value, $\hat{y}$, against the actual y value for the testing data.

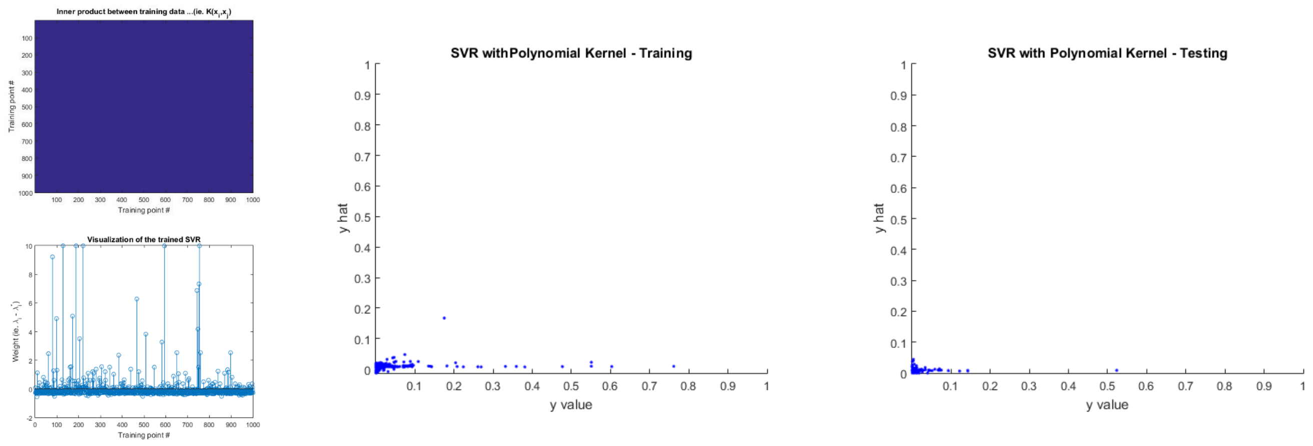


Figure 5: Summary of the SVR model created using the Interpolation Expanded data. Top Left: The Gram matrix, K, for the SVR model produced, where $K_{i,j} = \mathsf{K}(\mathbf{x}_i, \mathbf{x}_j)$. Bottom Left: The weights of each data point in the SVR model, $\lambda_i - \lambda_i^*$. The points whose weight is between 0 and C are considered the support vectors. Center: Scatter plot of the predicted value, $\hat{y}$, against the actual y value for the training data. Right: Scatter plot of the predicted value, $\hat{y}$, against the actual y value for the testing data.

7.3 Combination Method

As stated in Section 6.3, the C* parameter was chosen by comparing the Cross Validation Errors of models built using different candidate values. Each of these models were evaluated using both the Raw Error and False Negative Percentage metrics as shown in Figure 6. With these error metrics in mind it was decided that the optimal value for the C parameter is 0.55. Although this parameter leads to a rather large raw error, it allows for the frequency of False Negative predictions produced by the model to be minimized.

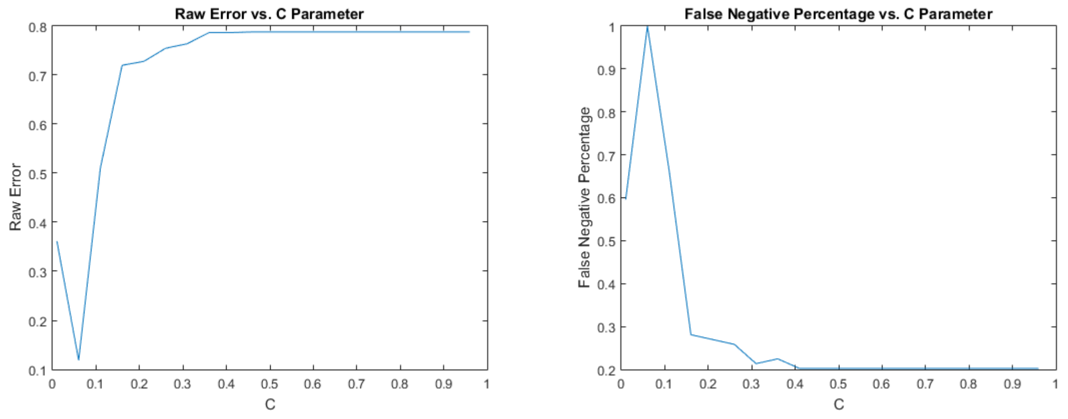


Figure 6: Summary of Cross Validation Error measures for SVM classification model. Left: Plot of Raw Error vs. C Parameter values from 0.01 to 0.96. Right: Plot of False Negative Percentage vs. C Parameter values from 0.01 to 0.96.

The SVM model for classification with this C parameter set produced training errors of 81% for the the raw metric and 16% for the False Negative metric. Similarly the testing errors produced were 79% for the raw metric and 17% for the False Negative metric.

8 Discussion

The Ridge Regression models, while computationally efficient, did not prove to be valuable in the analysis other than to point out that the data is evidently not linear in nature. Despite utilizing the most data out of all the models, the algorithm was unable to find a suitable model that came anywhere near fitting the data. It is also noted that expanding the feature space through interpolation did not increase the success of the algorithm. It appears that due to the distribution of the data, with the majority of the data near $y = 0$, the model was unable to generalize to points outside the neighborhood of $y = 0$. The algorithm focused on the points near $y = 0$ and thus never predicted rainfall more than 0.05.

With these insights in mind, support vector regression models were considered as a means to efficiently transform the data to a higher space where the linear model could succeed. This indeed was the case: SVR could easily fit the training data that it was given with little training error. However, the training data size had to be restricted to only 1,000 points and even so cross validation was not feasible, thus reasonable estimates for C, ϵ and the kernel parameter, d, had to be made. The final model using $d = 4$ for the fourth degree polynomial kernel was able to perform well with respect to both training and testing for the Mean Compressed data. Since cross validation was not possible, the model did not generalize to the test data very well. In comparison to simple linear regression, the model more accurately predicted rain levels both inside and outside the neighborhood of $y = 0$.

Interestingly, SVR did not perform as well when trained using the Interpolation Expanded data—it fell victim to the same gripe that plagued linear regression by rarely predicting rainfall outside the neighborhood of $\hat{y} = 0$. The plot in the bottom left of Figure 5 shows many more support vectors than the same plot for the Mean Compressed SVR model. This and the poor performance may indicate that the interpolation introduced a significant level of noise that obstructed the learning algorithm.

The distribution of the data created an interesting trade-off when trying to produce a successful SVM model for classification. When simply trying to minimize the raw error metric, the model produced would classify all points as "not rain" and produce a fairly successful albeit inflexible model. The ability to use SVR to catch any false positives in the model's predictions allowed for an error metric which would specifically look to avoid false negative classifications. The combination of these models was able to achieve a much higher degree of success than any of the other models tested.

9 Extensions

Several creative extensions were applied to the analysis in an attempt to better results. The first being the Interpolation Expansion of the data that used Newton's Divided Differences method to interpolate the data for each hour for re-sampling and the expansion of the feature space. This was done because it was thought that there would be too much loss of information by compressing the data down the mean for each hour. However, this method had mixed results because the expansion of the feature space proved to come at the expense of added noise and computation time for all the optimization algorithms.

The second extension involved classifying the data first and then applying a regression model. This was an attempt to compensate for the distribution of the data being heavily concentrated around $y = 0$, because it rains a little or not at all more often than it rains a lot. The details of this method are discussed in Sections 6.3 and 7.3.

10 Conclusion

The combination of SVM and SVR models were able to model the data to a relatively high degree of success. However, a few notes could be improved upon for further work in the area. First, these models could simply be optimized or tested on more powerful machines so that more data can be used to train the models. If cross validation could be performed for the SVR model, it would be very powerful. Additionally the identification and implementation of a Kernel function designed to support the Zipf distribution of this data set could enable both the SVM and SVR models to produce more accurate output.

References

[1] *How Much Did It Rain II.* 2015. URL: `https://www.kaggle.com/c/how-much-did-it-rain-ii`.

[2] Krause and Flood. *Weather and Climate Extremes.* Sept. 1997, p. 88.

[3] V. Lakshmanan et al. "The AMS-AI 2015-2016 Contest: Probabilistic estimate of hourly rainfall from radar". In: *13th Conference on Artificial Intelligence* (2015).

[4] *MATLAB*. 2015.

[5] *Neos Solvers*. 2015. URL: `https://neos-server.org/neos/solvers/nco:MINOS/AMPL.html`.

[6] Timothy Sauer. *Numerical Analysis*. Second. New York: Pearson Addison Wesley, 2006. Chap. Interpolation, pp. 138–87.

[7] Alex J. Smola and Bernhard Scholkopf. “A Tutorial on Support Vector Regression”. In: (Oct. 1998).

[8] C. Willmott and K. Matsuura. “Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance”. In: (2005), pp. 79–82.

[9] Abu-Mostafa Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from Data: A Short Course*. United States, 2012. URL: `AMLBook.com`.

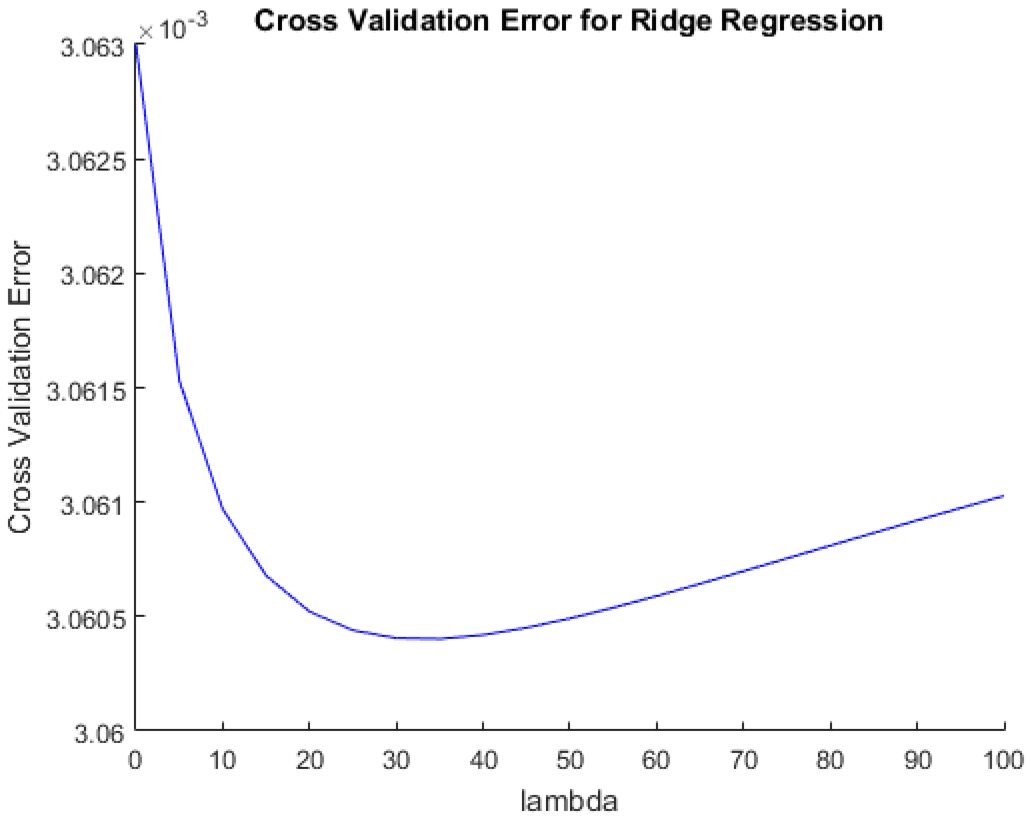
Appendix A: Sample Runs

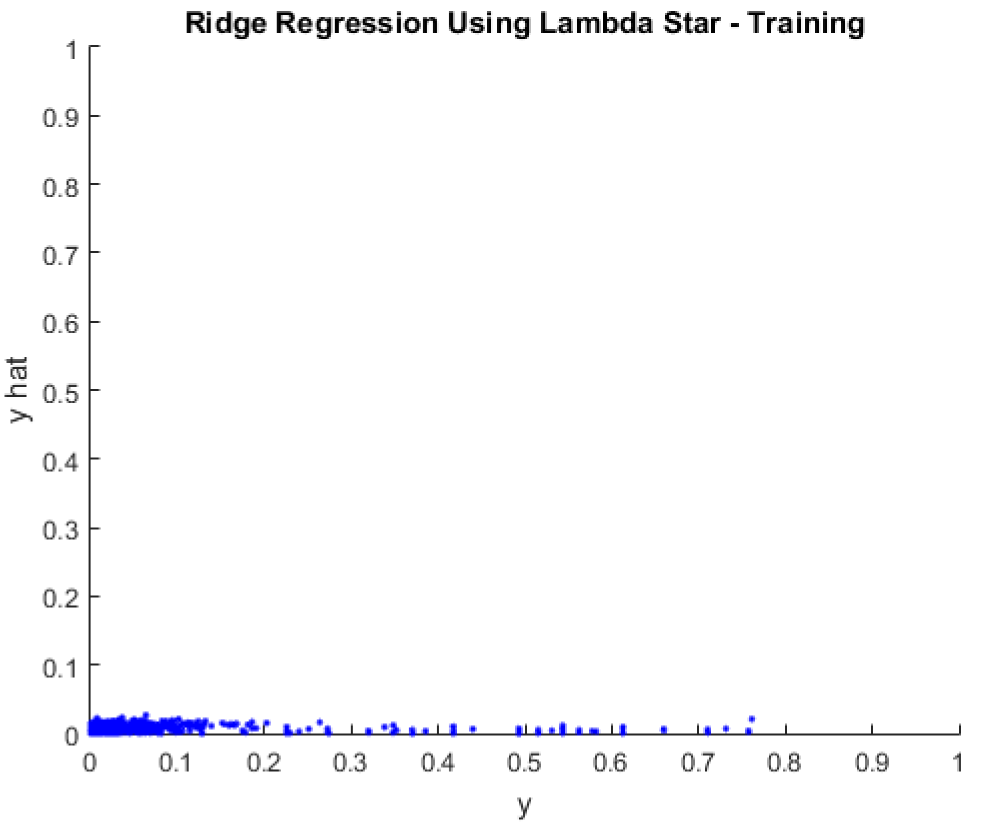
1. Ridge Regression
2. SVM Cross Validation
3. SVM Final Model
4. SVR Interpolation Expanded Model
5. SVR Mean Compressed Model

Table of Contents

Ridge Regression on the Mean Compressed Data

```
M = csvread('train_filtered_b.sorted.squashed.norm.sample.csv', 1, 0);
max_n = 10000;
ratio = 0.2;

rng(24);
[ Xtrain, Ytrain, Xtest, Ytest, ntrain, mtrain, ntest, mtest ] = ...
        getTrainAndTestSets(M, max_n, ratio);

w = RidgeRegression(Xtrain, Ytrain);


lambda_star =

    35
```


Cross Validation Error for Ridge Regression
×10^-3
3.063
3.0625
3.062
3.0615
3.061
3.0605
3.06
Cross Validation Error
0
10
20
30
40
50
60
70
80
90
100
lambda


Ridge Regression Using Lambda Star - Training
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y hat
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
y

Compute Ein

```
MAE = 0;
yhat = zeros(ntrain,1);
for i = 1:ntrain
    yh = Xtrain(i,:) * w;
    MAE = MAE + abs( yh - Ytrain(i) );
    yhat(i) = yh;
end
MAE = MAE / ntrain;

fprintf( 'Ein = %0.4f\n', MAE );
```

```
Ein = 0.0184
```

Compute Etest

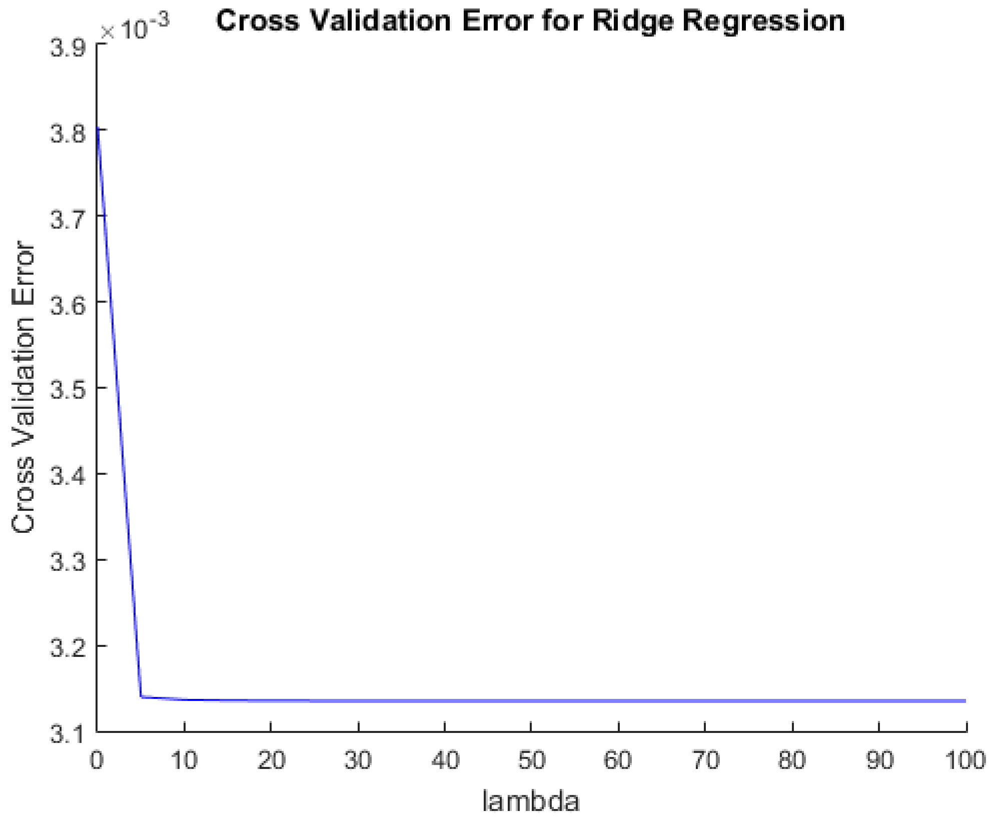
```
MAE = 0;
yhat = zeros(ntest,1);
for i = 1:ntest
    yh = Xtest(i,:) * w;
    MAE = MAE + abs( yh - Ytest(i) );
    yhat(i) = yh;
end
MAE = MAE / ntest;

fprintf( 'Etest = %0.4f\n', MAE );
```

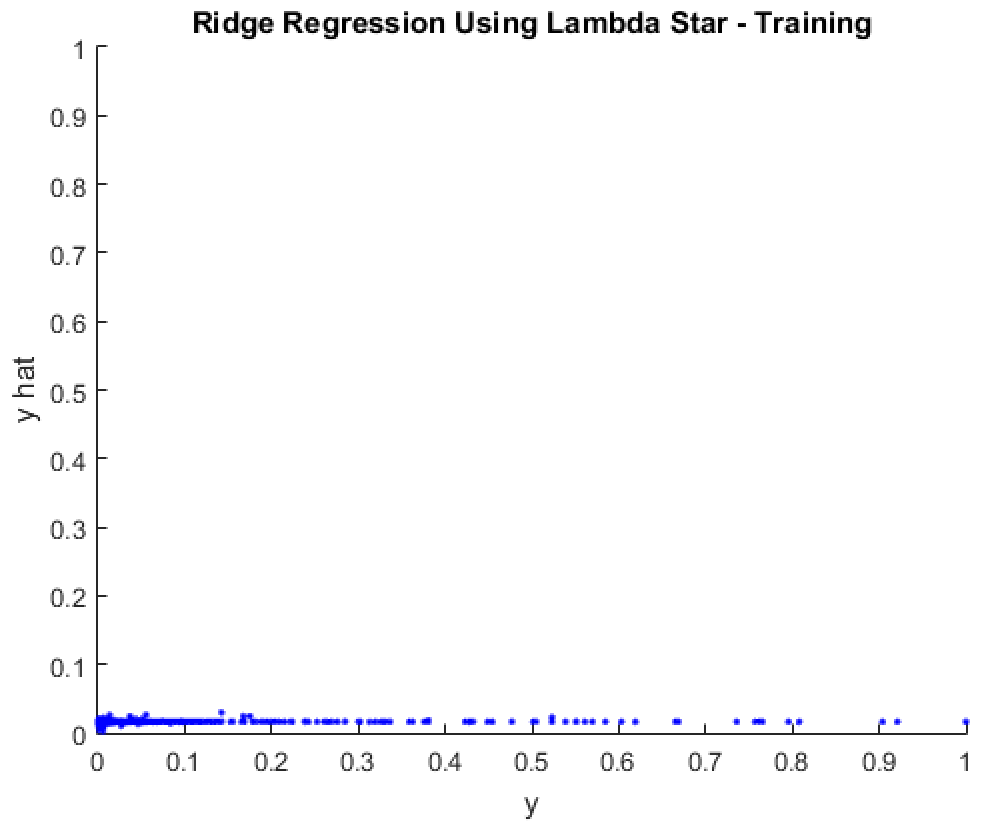
```
Etest = 0.0180
```

Ridge Regression on the Interpolation Expanded Data

```
M = csvread('train_filtered_b.sorted.interpolated.norm.sample.csv', 1,
 0);

rng(24);
[ Xtrain, Ytrain, Xtest, Ytest, ntrain, mtrain, ntest, mtest ] = ...
        getTrainAndTestSets(M, max_n, ratio);

w = RidgeRegression(Xtrain, Ytrain);
```

```
lambda_star =

    60
```


Cross Validation Error for Ridge Regression
×10^-3
3.9
3.8
3.7
3.6
3.5
3.4
3.3
3.2
3.1
Cross Validation Error
0
10
20
30
40
50
60
70
80
90
100
lambda


Ridge Regression Using Lambda Star - Training
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y hat
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
y

Compute Ein

```
MAE = 0;
yhat = zeros(ntrain,1);
for i = 1:ntrain
    yh = Xtrain(i,:) * w;
    MAE = MAE + abs( yh - Ytrain(i) );
    yhat(i) = yh;
end
MAE = MAE / ntrain;

fprintf( 'Ein = %0.4f\n', MAE );
```

```
Ein = 0.0193
```

Compute Etest

```
MAE = 0;
yhat = zeros(ntest,1);
for i = 1:ntest
    yh = Xtest(i,:) * w;
    MAE = MAE + abs( yh - Ytest(i) );
    yhat(i) = yh;
end
MAE = MAE / ntest;

fprintf( 'Etest = %0.4f\n', MAE );
```

```
Etest = 0.0208
```

Published with MATLAB® R2015a

Table of Contents

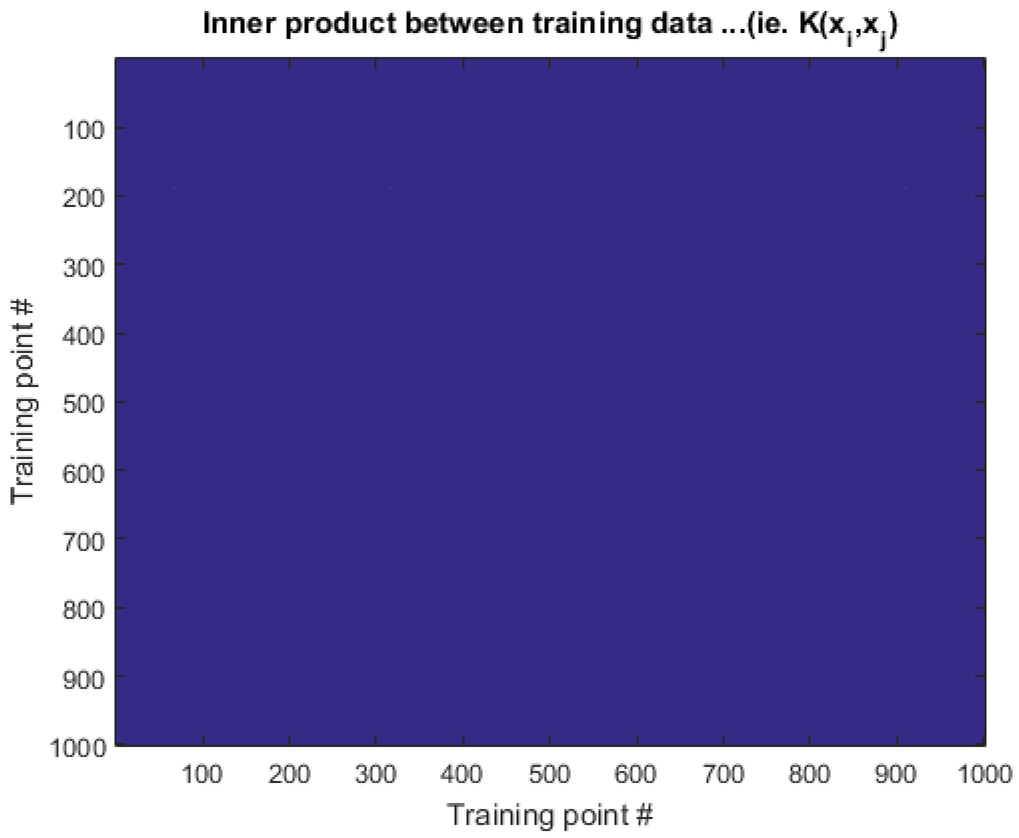
Ready the data

```
M = csvread('train_filtered_b.sorted.squashed.norm.sample.csv', 1, 0);
max_n = 1000;
ratio = 0.2;

rng(24);
[ Xtrain, Ytrain, Xtest, Ytest, ntrain, mtrain, ntest, mtest ] = ...
        getTrainAndTestSets(M, max_n, ratio);
```

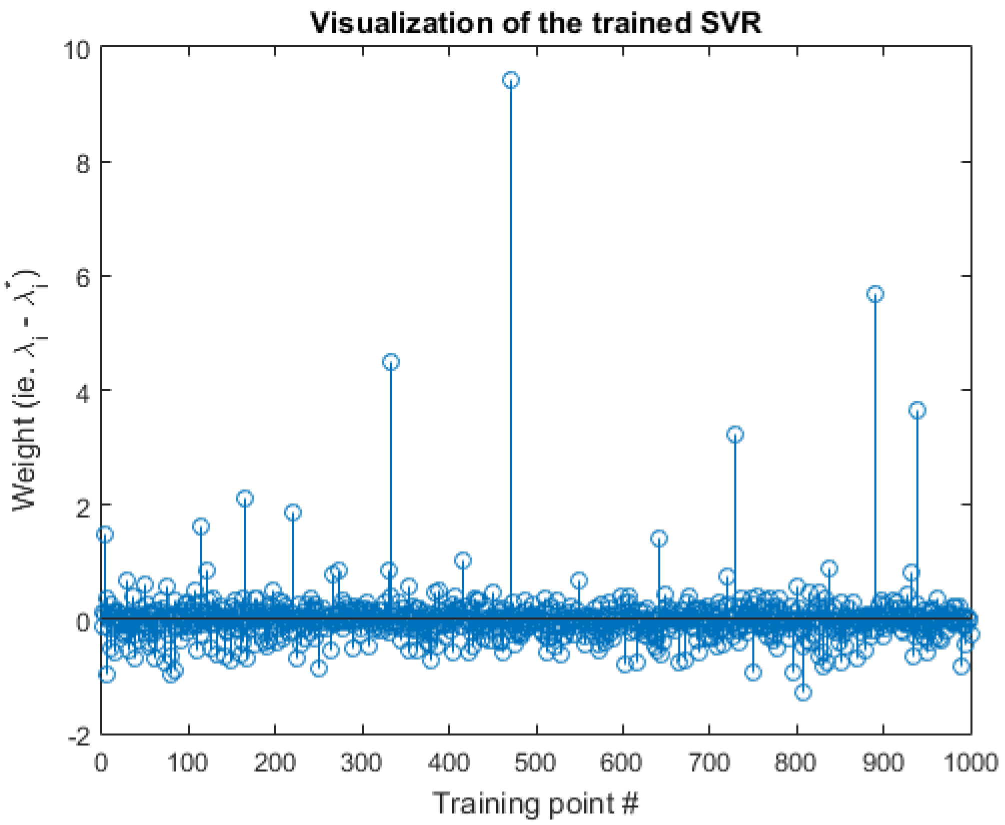
SVR

```
C = 10.0;
epsilon = 0.00001;

svr = svr_trainer( Xtrain, Ytrain, C, epsilon, 'Polynomial', 4 )
```

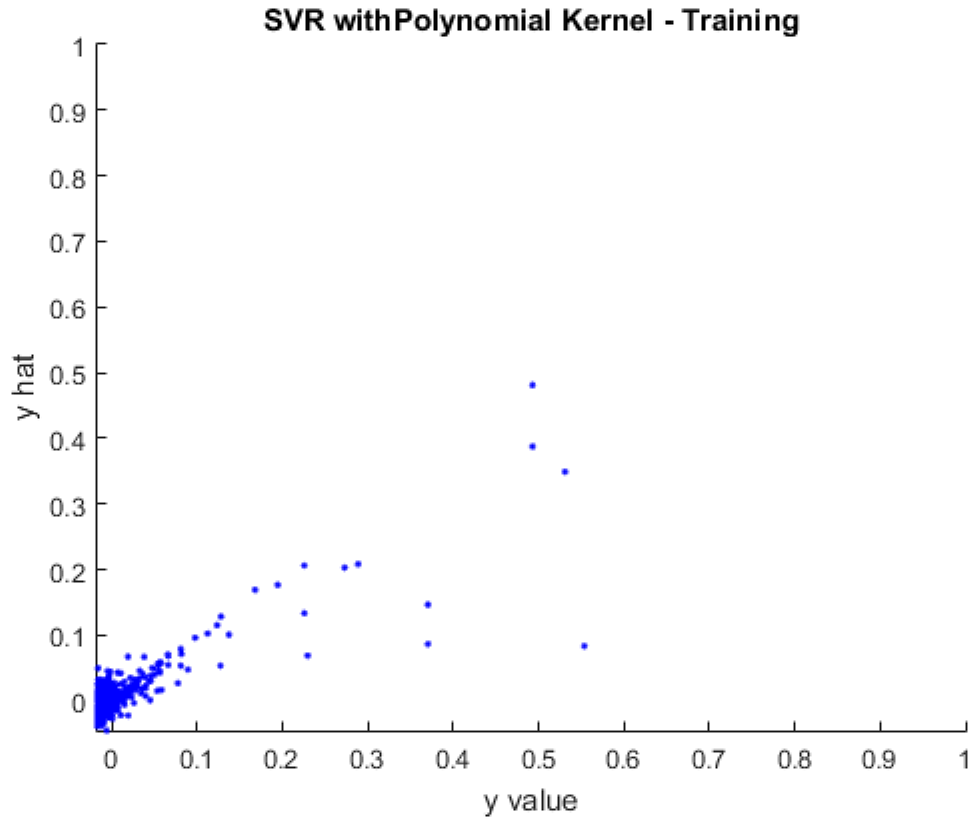
```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the default value of the function
 tolerance,
and constraints are satisfied to within the default value of the
 constraint tolerance.




svr =

         alpha: [1000x1 double]
             b: 0.0015
        kernel: @(x,y)(x.feature*y.feature'+1)^d
    train_data: [1000x22 double]
       predict: @(x)cellfun(@(u)svr_eval(u),num2cell(x,2))
```



Inner product between training data ...(ie. $K(x_i,x_j)$)
Training point #
Training point #



Visualization of the trained SVR
Weight (ie. $\lambda_i - \lambda_i^*$)
Training point #



Evaluate Ein Using MAE

```
MAE = 0;
yhat = zeros(ntrain,1);
for i = 1:ntrain
    yh = svr.predict( Xtrain(i,:) );
    MAE = MAE + abs( yh - Ytrain(i) );
    yhat(i) = yh;
end
MAE = MAE / ntrain;

fprintf( 'Ein = %0.4f\n', MAE );
```

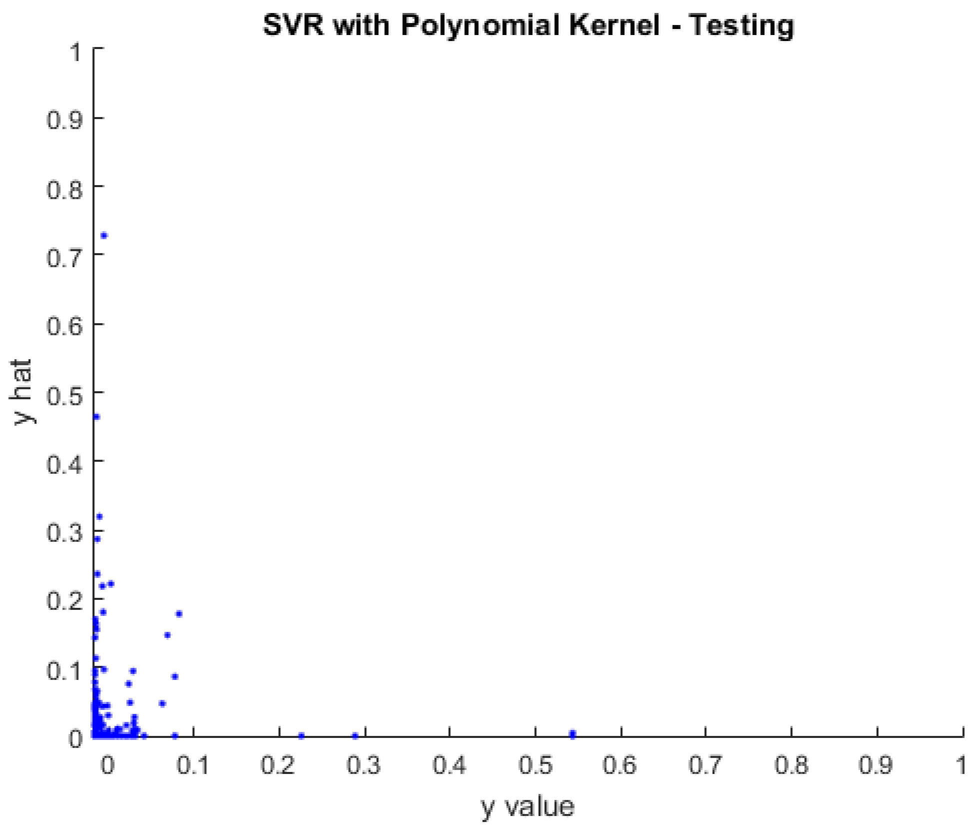
```
Ein = 0.0103
```

Compute Etest

```
MAE = 0;
yhat = zeros(ntest,1);
for i = 1:ntest
    yh = svr.predict( Xtest(i,:) );
    yh = max(yh, 0);
    MAE = MAE + abs( yh - Ytest(i) );
    yhat(i) = yh;
end
```

```
MAE = MAE / ntest;

fprintf( 'Etest = %0.4f\n', MAE );

figure;
hold on;
plot( Ytest, yhat, 'b.' );
title( 'SVR with Polynomial Kernel - Testing' );
xlabel( 'y value' );
ylabel( 'y hat' );
axis([min(Ytest) 1 min(yhat) 1]);
hold off;
```

```
Etest = 0.0586
```



Published with MATLAB® R2015a

Table of Contents

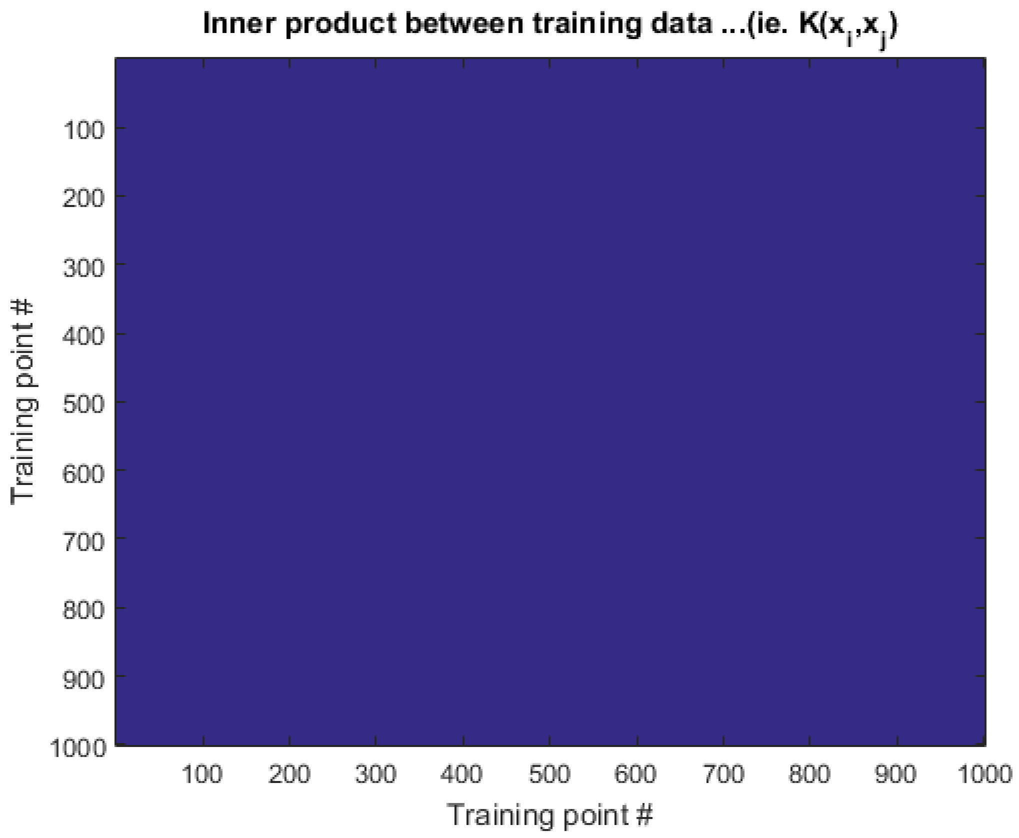
Ready the data

```
M = csvread('train_filtered_b.sorted.interpolated.norm.sample.csv', 1,
 0);
max_n = 1000;
ratio = 0.2;

rng(24);
[ Xtrain, Ytrain, Xtest, Ytest, ntrain, mtrain, ntest, mtest ] = ...
        getTrainAndTestSets(M, max_n, ratio);
```

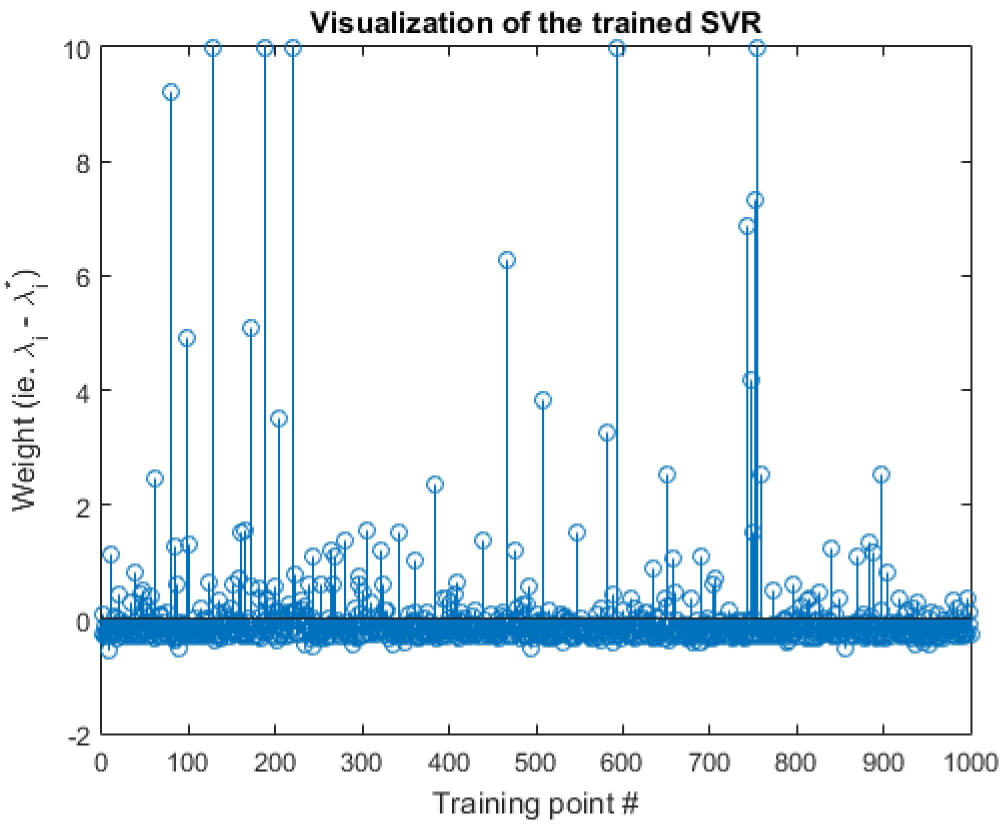
SVR

```
C = 10.0;
epsilon = 0.00001;

svr = svr_trainer( Xtrain, Ytrain, C, epsilon, 'Polynomial', 4 )
```

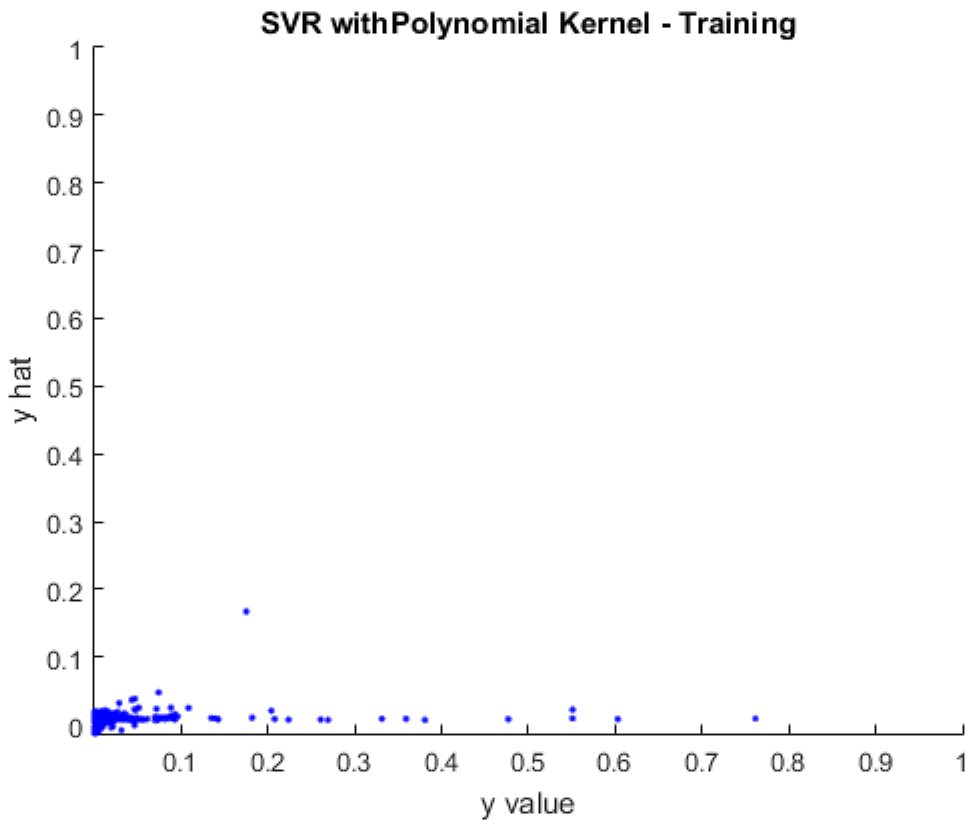
```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the default value of the function
 tolerance,
and constraints are satisfied to within the default value of the
 constraint tolerance.




svr =

         alpha: [1000x1 double]
             b: -0.0155
        kernel: @(x,y)(x.feature*y.feature'+1)^d
    train_data: [1000x101 double]
       predict: @(x)cellfun(@(u)svr_eval(u),num2cell(x,2))
```



Inner product between training data ...(ie. $K(x_i,x_j)$)
Training point #
Training point #
100
200
300
400
500
600
700
800
900
1000



Visualization of the trained SVR
Weight (ie. $\lambda_i - \lambda_i^*$)
Training point #
-2
0
2
4
6
8
10
0
100
200
300
400
500
600
700
800
900
1000



Evaluate Ein Using MAE

```
MAE = 0;
yhat = zeros(ntrain,1);
for i = 1:ntrain
    yh = svr.predict( Xtrain(i,:) );
    MAE = MAE + abs( yh - Ytrain(i) );
    yhat(i) = yh;
end
MAE = MAE / ntrain;

fprintf( 'Ein = %0.4f\n', MAE );
```

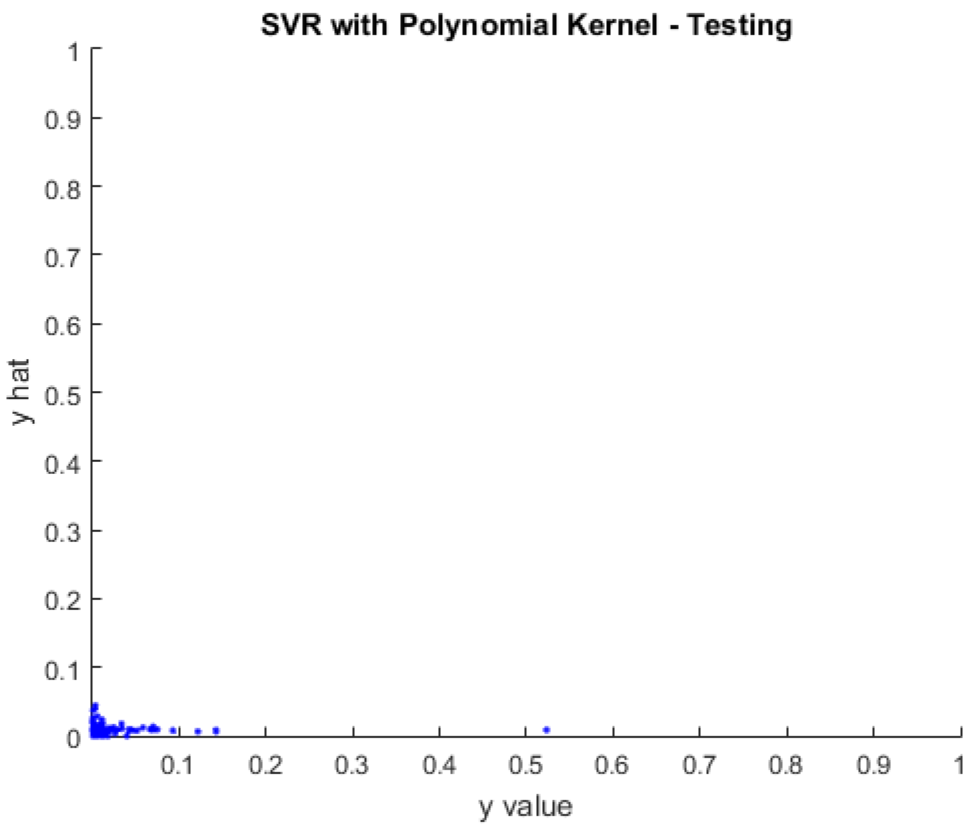
```
Ein = 0.0152
```

Compute Etest

```
MAE = 0;
yhat = zeros(ntest,1);
for i = 1:ntest
    yh = svr.predict( Xtest(i,:) );
    yh = max(yh, 0);
    MAE = MAE + abs( yh - Ytest(i) );
    yhat(i) = yh;
end
```

```
MAE = MAE / ntest;

fprintf( 'Etest = %0.4f\n', MAE );

figure;
hold on;
plot( Ytest, yhat, 'b.' );
title( 'SVR with Polynomial Kernel - Testing' );
xlabel( 'y value' );
ylabel( 'y hat' );
axis([min(Ytest) 1 min(yhat) 1]);
hold off;
```

```
Etest = 0.0150
```



Published with MATLAB® R2015a

Table of Contents

Read in Data

```
clear;
format long;
M = csvread('train_filtered_b.sorted.interpolated.norm.sample.csv', 0,
 0);
id = M(:,1);
Y = M(:,end)';
X = M(:,2:end-1)';
```

Take sample of training data that our model can handle

```
model_size = 750;
[m,n] = size(X);
r = randperm( n );
X = X( :, r );
Y = Y( r );
X = X( :, 1:model_size );
Y = Y( :, 1:model_size );
```

Classify Y values as raining when Y > epsilon, otherwise not raining

Store classification in Yclass vector

```
[a,b] = size(Y);
epsilon = 0.025;
Yclass = ones(a,b);
for i = 1:b
    if Y(i) > epsilon
        Yclass(i) = 1;
    else
        Yclass(i) = -1;
    end
end
```

Test values for SVM parameter using 5th order inhomogenous kernel

```
[d,N] = size(X);
K = @(x,xprime) (1 + x' * xprime)^5;


index = 0;
for C=.01:.05:1
    index = index+1;
    predictions = [];
    for cv=1:10
         mult = N/10;
         CVPoints = X(:,(cv-1)*(mult)+1: cv*mult);
         Ycv = Yclass((cv-1)*(mult)+1: cv*mult);
         Xtrain = X; Xtrain(:,(cv-1)*(mult)+1: cv*mult) = [];
         Ytrain = Yclass; Ytrain(:,(cv-1)*(mult)+1: cv*mult) = [];

         for i=1:N-mult
            for j=1:N-mult
                G(i,j) = Ytrain(i) * Ytrain(j) *
 K(Xtrain(:,i),Xtrain(:,j));
            end
         end

        H = double(G);
        f = -1 * ones(N-mult,1);
        A = zeros(1,N-mult);
        b = 0;
        Aeq = double(Ytrain);
        beq = 0;
        lb = zeros(N-mult,1);
        ub = repmat(C,N-mult,1);
        x0 = ones(N-mult,1);
        input_function = @(x) 0.5*x'*H*x + f'*x;
        alpha = fmincon(input_function, x0,A,b,Aeq,beq,lb,ub);

        for i=1:N-mult
            if(alpha(i) > C/100  && alpha(i) < (C*99/100))
                s = i;
            end
        end
        bstar = Ytrain(s);
        for i=1:N-mult
            if(alpha(i) > 0.01)
                bstar = bstar - alpha(i)*Ytrain(i) *
 K(Xtrain(:,i),Xtrain(:,s));
            end
        end

        for i=1:mult
            val = 0.0;
            for j=1:N-mult
                val = val +
 alpha(j)*Ytrain(j)*K(CVPoints(:,i),Xtrain(:,j));
            end
            predictions((cv-1)*(mult)+i) = sign(val + bstar);
        end

    end
    Ecv = sum(predictions~=Yclass);
    FalseNegative = sum( predictions(Yclass == 1) == -1 );
    fprintf( 'Using parameter C = %d\n', C );
    fprintf( '\tCross validation error = %d\n', Ecv/N );
    fprintf('\tFalse Negative percentage = %d\n',FalseNegative/
sum(Yclass==1));
 end
```

```
Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 3000 (the default value).


Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 3000 (the default value).


Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 3000 (the default value).


Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 3000 (the default value).


Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 3000 (the default value).


Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 3000 (the default value).


Solver stopped prematurely.
```

Table of Contents

Read in Data

```
clear;
format long;
M = csvread('train_filtered_b.sorted.interpolated.norm.sample.csv', 0,
 0);
id = M(:,1);
Y = M(:,end)';
X = M(:,2:end-1)';
```

Take sample of training data that our model can handle

```
model_size = 750;
[m,n] = size(X);
r = randperm( n );
X = X( :, r );
Y = Y( r );
TestX = X( :, model_size+1:end );
TestY = Y( :, model_size+1:end );
X = X( :, 1:model_size );
Y = Y( :, 1:model_size );
```

Classify Y values as raining when Y > epsilon, otherwise not raining

Store classification in Yclass vector

```
[a,b] = size(Y);
epsilon = 0.025;
Yclass = ones(a,b);
for i = 1:b
    if Y(i) > epsilon
        Yclass(i) = 1;
    else
        Yclass(i) = -1;
    end
end
```

Build SVM with 5th order inhomogenous Kernel

```
[d,N] = size(X);
K = @(x,xprime) (1 + x' * xprime)^5;

  for i=1:N
     for j=1:N
         G(i,j) = Yclass(i) * Yclass(j) * K(X(:,i),X(:,j));
     end
  end

C = .55;

H = double(G);
f = -1 * ones(N,1);
A = zeros(1,N);
b = 0;
Aeq = double(Yclass);
beq = 0;
lb = zeros(N,1);
ub = repmat(C,N,1);
x0 = ones(N,1);

input_function = @(x) 0.5*x'*H*x + f'*x;
alpha = fmincon(input_function, x0,A,b,Aeq,beq,lb,ub);


s = 1;
for i=1:N
    if(alpha(i) > C/100  && alpha(i) < (C*99/100))
        s = i;
    end
end
bstar = Yclass(s);
for i=1:N
    if(alpha(i) > C/100)
        bstar = bstar - alpha(i)*Yclass(i) * K(X(:,i),X(:,s));
    end
end



for i=1:N
    val = 0.0;
    for j=1:N
        if(alpha(j) > 0.01)
            val = val + alpha(j)*Yclass(j)*K(X(:,i),X(:,j));
        end
    end
    g(i) = sign(val + bstar);
end
```

```
Solver stopped prematurely.

fmincon stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 3000 (the default value).
```

Describe training error

```
fprintf('Raw training error = %d\n', sum(Yclass ~= g)/model_size);
fprintf('False Negative Percentage = %d\n',...
    sum(g(Yclass==1)==-1)/sum(Yclass==1));
```

```
Raw training error = 8.093333e-01
False Negative Percentage = 1.571429e-01
```

Classify TestY values as raining when Y > epsilon, otherwise not raining

Store classification in YTestclass vector

```
[a,b] = size(TestY);
epsilon = 0.025;
YTestclass = ones(a,b);
for i = 1:b
    if TestY(i) > epsilon
        YTestclass(i) = 1;
    else
        YTestclass(i) = -1;
    end
end
```

Describe Testing Error

```
[Features, XtestN] = size(TestX);
[Features, XtrainN] = size(X);
predictions = [];

for i = 1:XtestN
    val = 0.0;
    for j = 1:XtrainN
        val = val + alpha(j)*YTestclass(j)*K(TestX(:,i),X(:,j));
    end
    predictions(i) = sign(val + bstar);
end

fprintf('Raw training error = %d\n', sum(YTestclass ~= predictions)/
XtestN);
fprintf('False Negative Percentage = %d\n',...
    sum(predictions(YTestclass==1)==-1)/sum(YTestclass==1));
```

```
Raw training error = 7.866042e-01
```

```
False Negative Percentage = 1.663685e-01
```

Published with MATLAB® R2015a

Appendix B: Codes Written

1. split_data.py
2. filter.c
3. merge_sort.c
4. Mean Compression
5. Interpolation Expansion
6. Ridge Regression
7. getTrainAndTestSets
8. Distribution Analysis
9. SVR Trainer

```
'''
Filename:    split_data.py
Name:        Computational Optimization Project - How Much Will It Rain
Description:    Split the large data file sever smaller files that can be fed into
  the filter.c program where each file will be filtered in a separate thread
Author:      Matt Poegel
Date:        04-13-2016
'''


import os, sys
from subprocess import call, Popen, PIPE
import re

DATA = './train.csv'
DATA_DIR = './debug2'
LINES = 100000
CARS = 100

def cleanLine(line):
  line = str(line).replace('b\'', '').replace("\\n'", '')
  return line.strip()

if (LINES == -1):
  x = Popen(['wc', '-l', DATA], stdout=PIPE)
  LINES = int(cleanLine( x.stdout.readline() ).split(' ')[0])

call(['mkdir', '-p', DATA_DIR])

header = Popen(['head', '-n', '1', DATA], stdout=PIPE)
header = cleanLine( header.stdout.readline() )

CAR_SIZE = LINES // CARS
start = 2
end = CAR_SIZE
for c in range(CARS):
  if (c == CARS - 1):
    end = LINES
  arg = str(start) + ',' + str(end) + 'p'
  x = Popen(['sed', '-n', arg, DATA],
            stdout=PIPE, stderr=PIPE, shell=False)
  start = end + 1
  end = start + CAR_SIZE
  fname = DATA_DIR + '/car' + str(c) + '.csv'
  with open(fname, 'w') as fp:
    fp.write( header + '\n' )
    for line in x.stdout.readlines():
      line = cleanLine(line)
      fp.write( line + '\n' )
```

```c
/**
 * Filename:    filter.c
 * Name:        Computational Optimization Project - How Much Will It Rain
 * Description: Read every line in every file in a given directory, reading each
 *   each file in a separate thread, and then throw out the garbage lines
 * Author:      Matt Poegel
 * Date:        04-13-2016
 */

#include <dirent.h>
#include <errno.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define INIT_ARRAY_SIZE 16
#define BUFFER_SIZE 512
#define EXPECTED_COLS 24
#define MAX_RAINFALL 305 /* world record for most rain in one hour */

#define FALSE 0
#define TRUE 1


struct FileTask {
  struct WordList* wordlist;
  struct dirent** file;
};

pthread_mutex_t indlock;
pthread_mutex_t arrlock;

struct WordList {
  char* header;
  struct Word** words;
  pthread_mutex_t indlock;
  pthread_mutex_t arrlock;
  unsigned int length;
  unsigned int allocated;
};

struct Word {
  char* text;
  char* source;
};


/**
 * Assign a block of n contigious indices
 * @param wordlist {struct WordList*} wordlist struct that is shared by all
 *  threads
 * @returns {unsigned int} the first index in the assigned block
 */
unsigned int getIndices(struct WordList* wordlist, int n)
{
  pthread_mutex_lock( &indlock );
  unsigned int start = wordlist->length++;
  while (--n) {
    wordlist->length++;
  }
  pthread_mutex_unlock( &indlock );
  return start;
}


/**
 * Insert the word at the index but first make sure that there's enough space
 *  and that no thread is in the process of reallocating. If reallocation is
 *  needed and no thread is doing so yet, then reallocate the array in wordlist
 * @param wordlist {struct WordList*} wordlist struct shared by all threads
 * @param word {struct Word* word} word to add to the array in wordlist
 * @param index {unsigned int} assigned index retrieved from getIndex(...)
 * @returns {void}
 */
void insertAtIndex(struct WordList* wordlist, struct Word* word,
  unsigned int index)
{
  if ( index >= wordlist->allocated ) {
    pthread_mutex_lock( &arrlock );
    if ( index >= wordlist->allocated ) {
      unsigned int new_size = wordlist->allocated * 2;
      wordlist->words = (struct Word**)realloc( wordlist->words,
        new_size * sizeof(struct Word*) );
      printf( "THREAD %u: Re-allocated array of %u character pointers.\n",
        (unsigned int)pthread_self(), new_size );
      wordlist->allocated *= 2;
    }
    pthread_mutex_unlock( &arrlock );
  }
  wordlist->words[index] = word;
}


/**
 * Determine if the given line should be thrown out
 * @param w {char*} string to check for validity
 * @returns {int} the id of the line if the line should be kept or -1 if the
 *  line should be thrown out
 */
int isGarbage(char* w)
{
  int count = 0;
  char* saveptr;
  char* tok = strtok_r(w, ",", &saveptr);
  int id = atoi( tok );
  char tmp[ BUFFER_SIZE ];
  while (tok) {
    count++;
    strcpy( tmp, tok );
    tok = strtok_r(NULL, ",", &saveptr);
  }
  float r = atof( tmp );
  if (r > MAX_RAINFALL) {
    return -1;
  }
  if (count < EXPECTED_COLS) {
    return -1;
  }
  return id;
}



/**
 * Read the input words from a file and store them in the shared wordlist
 * @param arg {void*} void* pointer that should be castable to a struct
 *  FileTask*
 * @returns {void*}
 */
void* readFile(void* arg)
{
  struct FileTask* ft = (struct FileTask*)arg;
  struct dirent* file = *(ft->file);
  struct WordList* wordlist = ft->wordlist;
  FILE* fp = fopen( file->d_name, "r" );
  if ( fp == NULL ) {
    fprintf( stderr, "THREAD %u: ERROR. Could not open file: %s\n",
       (unsigned int)pthread_self(), file->d_name );
    return NULL;
  }
  char buff[ BUFFER_SIZE ];
  struct Word* word_buffer[ BUFFER_SIZE ];
  int buff_ind = 0;
  int last_id = -1;
  fscanf( fp, "%s", buff );
  if (wordlist->header == NULL) {
    char* header = (char*)calloc( strlen(buff) + 1, sizeof(char) );
    strcpy( header, buff );
    wordlist->header = header;
  }
  /* Read in each line in the file */
  unsigned int count = 0;
  while ( fscanf( fp, "%s", buff ) == 1 ) {
    char* w = (char*)calloc( strlen(buff) + 1, sizeof(char) );
    char* s = (char*)calloc( strlen(file->d_name) + 1, sizeof(char) );
    if (w == NULL || s == NULL) {
      fprintf( stderr, " ERROR. Failed to allocate memory." );
      exit(EXIT_FAILURE);
    }
    strcpy(w, buff);
    strcpy(s, file->d_name);
    int id = isGarbage(w);
    if ( id == -1 ) {
      free(w);
      free(s);
      continue;
    }
    strcpy(w, buff);
    struct Word* word = (struct Word*)malloc( sizeof(struct Word) );
    word->text = w;
    word->source = s;
    if ( last_id == id || last_id == -1 ) {
      word_buffer[ buff_ind++ ] = word;
      last_id = id;
    } else if (buff_ind > 0) {
      unsigned int index = getIndices( wordlist, buff_ind );
      int k;
      for (k = 0; k<buff_ind; k++) {
        insertAtIndex( wordlist, word_buffer[k] , index + k );
        // printf( "THREAD %u: Added '%s' at index %u.\n",
        //   (unsigned int)pthread_self(), word_buffer[k]->text, index + k );
      }
      word_buffer[0] = word;
      buff_ind = 1;
      last_id = id;
    }
    count++;
  }
  if (buff_ind > 0) {
    unsigned int index = getIndices( wordlist, buff_ind );
    int k;
    for (k = 0; k<buff_ind; k++) {
      insertAtIndex( wordlist, word_buffer[ k ] , index + k );
      // printf( "THREAD %u: Added '%s' at index %u.\n",
      //   (unsigned int)pthread_self(), word_buffer[ k ]->text, index + k );
    }
  }
  if (! feof(fp) ) {
    fprintf( stderr, " ERROR. Failed to read entire file.\n" );
    exit(EXIT_FAILURE);
  }
  fclose( fp );
  free( ft->file );
  free( ft );
  return NULL;
}


/**
 * Search through the list of words and print out each word that contains the
 * given substring.
 * @param wordlist {struct WordList*} wordlist structure containing the list of
 *  words to search through
 * @param sub {char*} substring to find in the list
 * @returns {void}
 */
void printSubstringMatches(struct WordList* wordlist, char* sub)
{
  printf( "MAIN THREAD: Words containing substring \"%s\" are:\n", sub );
  unsigned int i;
  for (i=0; i<wordlist->length; i++) {
    if ( strstr( wordlist->words[i]->text, sub ) != NULL ) {
      printf( "MAIN THREAD: %s (from '%s')\n", wordlist->words[i]->text,
        wordlist->words[i]->source );
    }
  }
  return;
}


/**
 * Save the wordlist to a file
 * @param filename {char*} name of the file to write to
 * @param wordlist {struct WordList*} wordlist structure containing the list of
 *  words to save
 */
void saveToFile(char* filename, struct WordList* wordlist)
{
  FILE* fp = fopen( filename, "w" );
  if (fp == NULL) {
    fprintf( stderr, "ERROR: Could not open file %s\n", filename );
    exit( EXIT_FAILURE );
  }
  fprintf( fp, "%s\n", wordlist->header );
  unsigned int i;
  for (i=0; i<wordlist->length; i++) {
    fprintf( fp, "%s\n", wordlist->words[i]->text );
  }
  fclose( fp );
}


/**
 * MAIN
 */
int main(int argc, char* argv[])
{

  /* Check the user input */
  if (argc < 3) {
    fprintf( stderr, "ERROR: Invalid arguments\n" );
    fprintf( stderr, "USAGE: %s <directory> <output file>\n", argv[0] );
    return EXIT_FAILURE;
  }
  char* dirname = argv[1];
  char* output_filename = argv[2];

  /* initialize the wordlist structure */
  struct WordList* wordlist = (struct WordList*)
    malloc( sizeof(struct WordList) );
  wordlist->allocated = INIT_ARRAY_SIZE;
  pthread_mutex_init(&wordlist->indlock, NULL);
  pthread_mutex_init(&wordlist->arrlock, NULL);
  pthread_mutex_init(&indlock, NULL);
  pthread_mutex_init(&arrlock, NULL);
  wordlist->length = 0;
  wordlist->words = (struct Word**)calloc( sizeof(struct Word*),
    wordlist->allocated );
  wordlist->header = NULL;

  /* find all of the files in the given directory and start the threads */
  chdir( dirname );
  DIR* dir = opendir( "." );
  pthread_t* tid = (pthread_t*)calloc( sizeof(pthread_t), INIT_ARRAY_SIZE );
  unsigned int tid_size = INIT_ARRAY_SIZE;
  int rc;
  unsigned int t_count = 0;
  if ( dir == NULL ) {
    fprintf( stderr, "ERROR: Could not open directory %s", dirname );
    return EXIT_FAILURE;
  }
  while ( 1 ) {
    struct dirent** file = (struct dirent**)malloc( sizeof(struct dirent*) );
    *file = readdir(dir);
    /* stop condition */
    if ( *file == NULL ) {
      free( file );
      break;
    }
    /* only accept regular files */
    if ( (*file)->d_type == DT_REG ) {
      if (t_count >= tid_size) {
        tid_size *= 2;
        tid = (pthread_t*)realloc( tid, tid_size * sizeof(pthread_t) );
      }
      struct FileTask* ft = (struct FileTask*)malloc( sizeof(struct FileTask) );
      ft->file = file;
      ft->wordlist = wordlist;
      rc = pthread_create( &tid[t_count], NULL, readFile, ft );
      if ( rc != 0 ) {
        fprintf( stderr, "MAIN: Could not create child thread (%d)\n", rc );
        exit( EXIT_FAILURE );
      }
      printf( "MAIN THREAD: Assigned '%s' to child thread %u\n",
        (*file)->d_name, (unsigned int)tid[t_count] );
      t_count++;
    /* if it's not a regular file then we can immediately clean up the memory */
    } else {
      free( file );
    }
  }

  /* wait for the threads to finish reading the files */
  int i;
  for (i=0; i<t_count; i++) {
    unsigned int* x;
    pthread_join( tid[i], (void**)&x ); /* blocking */
    printf( "MAIN THREAD: Joined a child thread.\n" );
    free( x );
  }

  closedir( dir );
  free( tid );

  printf( "MAIN THREAD: All done (successfully read %u words from %u files).\n",
    wordlist->length, t_count );

  chdir( ".." );
  saveToFile( output_filename, wordlist );

  /* Clean up the mess on heap */
  unsigned int w;
  for (w=0; w<wordlist->length; w++) {
    free( wordlist->words[w]->text );
    free( wordlist->words[w]->source );
    free( wordlist->words[w] );
  }
  free( wordlist->header );
  free( wordlist->words );
  free( wordlist );

  return EXIT_SUCCESS;
}
```

```
/**
 * Filename:    merge_sort.c
 * Name:        Computational Optimization Project - How Much Will It Rain
 * Description: Sort all the data by ID using a multithreaded merge sort
  *  algorithm
 * Author:      Matt Poegel
 * Date:        04-19-2016
 */

#include <math.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define INIT_ARRAY_SIZE 16
#define BUFFER_SIZE 512
#define FALSE 0
#define TRUE 1
#define THREADING_THRESHOLD 1000


typedef struct {
  char* text;
  int id;
} Datum;

typedef struct {
  Datum** store;
  unsigned int alloc_size;
  unsigned int store_size;
} Data;


/**
 *
 */
Data* readFile(char* filename)
{
  FILE* fp = fopen( filename, "r" );
  if (fp == NULL) {
    fprintf( stderr, "ERROR: could not open file %s\n", filename );
    exit( EXIT_FAILURE );
  }
  Data* data = (Data*)malloc( sizeof(Data) );
  if (data == NULL) {
    fprintf( stderr, "ERROR: could not allocate memory\n" );
    exit( EXIT_FAILURE );
  }
  data->alloc_size = INIT_ARRAY_SIZE;
  data->store_size = 0;
  data->store = (Datum**)calloc( data->alloc_size, sizeof(Datum*) );
  char buffer[BUFFER_SIZE];
  while ( fscanf( fp, "%s", buffer ) == 1 ) {
    char* text = (char*)calloc( strlen(buffer) + 1, sizeof(char) );
    Datum* d = (Datum*)malloc( sizeof(Datum) );
    if (text == NULL || d == NULL) {
      fprintf( stderr, "ERROR: could not allocate memory\n" );
      exit( EXIT_FAILURE );
    }
    strcpy( text, buffer );
    d->text = text;
    /* capture the id of the line */
    char* saveptr;
    char* tok = strtok_r( buffer, ",", &saveptr );
    d->id = atoi(tok);
    data->store[ data->store_size++ ] = d;
    /* resize check */
    if (data->store_size >= data->alloc_size) {
      data->alloc_size *= 2;
      data->store = (Datum**)realloc( data->store,
        data->alloc_size * sizeof(Datum*) );
    }
  }

  fclose( fp );
  return data;
}


/**
 *
 */
Data* mergeData(Data* A, Data* B)
{
  unsigned int i = 0;
  unsigned int k = 0;
  Data* result = (Data*)malloc( sizeof(Data) );
  if (result == NULL) {
    fprintf( stderr, "ERROR: could not allocate memory\n" );
    exit( EXIT_FAILURE );
  }
  result->alloc_size = A->store_size + B->store_size;
  result->store = (Datum**)calloc( result->alloc_size, sizeof(Datum*) );
  while ( i < A->store_size && k < B->store_size ) {
    if ( A->store[i]->id < B->store[k]->id ) {
      result->store[i+k] = A->store[i];
      i++;
    } else {
      result->store[i+k] = B->store[k];
      k++;
    }
  }
  for (; i < A->store_size; i++) {
    result->store[i+k] = A->store[i];
  }
  for (; k < B->store_size; k++) {
    result->store[i+k] = B->store[k];
  }
  result->store_size = i+k;
  free( A->store );
  free( A );
  free( B->store );
  free( B );
  return result;
}


/**
 *
 */
```

```
void splitData(Data* data, Data** A, Data** B)
{
  *A = (Data*)malloc( sizeof(Data) );
  *B = (Data*)malloc( sizeof(Data) );
  if (*A == NULL || *B == NULL) {
    fprintf( stderr, "ERROR: could not allocate memory\n" );
    exit( EXIT_FAILURE );
  }
  (*A)->alloc_size = (data->store_size / 2) + (data->store_size % 2);
  (*B)->alloc_size = (data->store_size / 2);
  (*A)->store = (Datum**)calloc( (*A)->alloc_size, sizeof(Datum*) );
  (*B)->store = (Datum**)calloc( (*B)->alloc_size, sizeof(Datum*) );
  unsigned int a = 0;
  unsigned int b = 0;
  while ( (a+b)<data->store_size ) {
    if ( (a+b) % 2 == 0 ) {
      (*A)->store[a] = data->store[ a+b ];
      a++;
    } else {
      (*B)->store[b] = data->store[ a+b ];
      b++;
    }
  }
  (*A)->store_size = (*A)->alloc_size;
  (*B)->store_size = (*B)->alloc_size;
  free( data->store );
  free( data );
}


/**
 *
 */
Data* mergeSortSingle(Data* data)
{
  if (data->store_size <= 1) {
    return data;
  }
  Data* A = NULL;
  Data* B = NULL;
  splitData(data, &A, &B);
  A = mergeSortSingle(A);
  B = mergeSortSingle(B);
  return mergeData(A, B);
}


/**
 *
 */
void* mergeSortThreaded(void* arg)
{
  Data* data = (Data*)arg;
  /* base case */
  if (data->store_size <= 1) {
    return data;
  } else if (data->store_size <= THREADING_THRESHOLD) {
    pthread_exit( mergeSortSingle(data) );
  }
  /* split in two */
  Data* A = NULL;
```

```
    Data* B = NULL;
    splitData(data, &A, &B);
    /* and spawn recursive threads */
    pthread_t tid1;
    pthread_t tid2;
    int rc = pthread_create( &tid1, NULL, mergeSortThreaded, A );
    if ( rc != 0 ) {
      fprintf( stderr, "ERROR: Could not create child thread (%d)\n", rc );
      exit( EXIT_FAILURE );
    }
    rc = pthread_create( &tid2, NULL, mergeSortThreaded, B );
    if ( rc != 0 ) {
      fprintf( stderr, "ERROR: Could not create child thread (%d)\n", rc );
      exit( EXIT_FAILURE );
    }
    pthread_join( tid1, (void**)&A );
    pthread_join( tid2, (void**)&B );
    /* merge and return */
    pthread_exit( mergeData(A, B) );
    return NULL;
  }


  /*
   *
   */
  Data* mergeSort(Data* data)
  {
    pthread_t tid;
    int rc = pthread_create( &tid, NULL, mergeSortThreaded, data );
    if ( rc != 0 ) {
      fprintf( stderr, "ERROR: Could not create child thread (%d)\n", rc );
      exit( EXIT_FAILURE );
    }
    Data* result;
    pthread_join( tid, (void**)&result );
    return result;
    // return mergeSortSingle(data);
  }


  /*
   * MAIN
   */
  int main(int argc, char* argv[])
  {
    /* check user input */
    if (argc < 3) {
      fprintf( stderr, "ERROR: invalid usage.\n" );
      fprintf( stderr, "%s <input file> <output file>\n", argv[0] );
      return EXIT_FAILURE;
    }
    char* input_filename = argv[1];
    char* output_filename = argv[2];

    /* read the input data from the given file */
    Data* data = readFile( input_filename );

    /* run merge sort */
    data = mergeSort( data );
    /* save the sorted output to file */
    FILE* fp = fopen( output_filename, "w" );
    if ( fp == NULL ) {
      fprintf( stderr, "ERROR: could not open output file %s\n",
        output_filename );
      return EXIT_FAILURE;
    }
    unsigned int i;
    for (i=0; i<data->store_size; i++) {
      fprintf( fp, "%s\n", data->store[i]->text );
    }
    fclose( fp );

    /* free the data on the heap */
    for (i=0; i<data->store_size; i++) {
      free( data->store[i]->text );
      free( data->store[i] );
    }
    free( data->store );
    free( data );

    return EXIT_SUCCESS;
  }
```

Table of Contents

Squash the data corresponding to each ID down to the mean so that there every entry has exactly one row of data to use in the model

```
format long;
```

Load the data

```
input_filename = 'train_filtered_b.sorted.sample.csv';
D = csvread( input_filename, 1, 0 );
```

Squash the data

```
IDs = unique( D(:,1) );
num_IDs = size(IDs, 1);
Ds = zeros( num_IDs, size(D, 2) );
s = 1;
e = 1;
tic;
for i=1:num_IDs
    id = D(s,1);
    while e < size(D,1) && D(e,1) == id
       e = e + 1;
    end
    if (e - s == 1)
        Ds(i,:) = D(s,:);
    else
        means = mean( D(s:e-1,:) );
        Ds(i,:) = means;
    end
    s = e;
    e = e + 1;
end
toc
```

```
Elapsed time is 0.893025 seconds.
```

normalize the data by column, ignoring the IDs

```
Ds_ = Ds;
Ds_ = Ds_ - ones(size(Ds_,1),1) * mean(Ds);
```

```
Ds_(:,2:end) = Ds_(:,2:end) ./ repmat( max(Ds_(:,2:end)),
 size(Ds_,1),1 );
```

Save the squashed data to file

```
output_filename = 'train_filtered_b.sorted.squashed.norm.sample.csv';

fid = fopen( input_filename );
a = textscan( fid, '%s', 1 );
fclose( fid );

fid = fopen( output_filename, 'w' );
fprintf( fid, '%s\n', char( a{1} ) );
fclose( fid );

dlmwrite( output_filename, Ds_, 'delimiter', ',', '-append', ...
    'precision', 16 );
```

Published with MATLAB® R2015a

Table of Contents

Since we have a variable number of readings for each hour, we must somehow obtain the same number of readings for each hour and expand the feature space. We will do this by finding an interpolating function for the existing data and resampling the feature

```
format long;

% Size of expanded feature space (number of times the interpolated
 function
%  is sampled)
N = 5;
```

Load the data

```
input_filename = 'train_filtered_b.sorted.sample.csv';
D = csvread( input_filename, 1, 0 );
```

Interpolate using Newton's Divided Differences Method

```
IDs = unique( D(:,1) );
num_IDs = size(IDs, 1);
Ds = zeros( num_IDs, (size(D,2)-4)*N+3 );
sampleTimes = linspace(5,55,N);
s = 1;
e = 1;
tic;
for i=1:num_IDs
    id = D(s,1);
    while e < size(D,1) && D(e,1) == id
       e = e + 1;
    end

    Ds(i,1) = id;                % save the ID
    Ds(i,2) = D(s,3);            % save the radar distance as the same
    Ds(i,end) = D(s,end);        % save the y-value as the same
    if (e - s == 1)              % only one reading for the hour
        x0 = D(s,2);
        y0 = D(s,4:end-1);
    else                         % multiple readings in the hour
        x0 = D(s:e-1,2);         % minutes
        y0 = D(s:e-1,4:end-1);  % radar readings
    end

    % some hours have multiple readings at the same time, so we must
 first
    %  average them together before we can interpolate
    x0_ = unique(x0, 'stable');
    y0_ = zeros( size(x0_,1), size(y0,2) );
    for k = 1:size(x0,1)
        new_y = y0(k,:);
        for j = k+1:size(x0,1)
            if ( x0(k) == x0(j) )
                new_y(end+1,:) = y0(j,:);
            end
        end
        if (size(new_y,1) == 1)
            y0_(k,:) = new_y;
        else
            y0_(k,:) = mean(new_y);
        end
    end

    % interpolate and expand the feature space
    for k = 1:size(y0_,2)
        f = newtonDivDiff( x0_', y0_(:,k)' );
        for n = 1:N
            t = sampleTimes(n);
            Ds(i, 2 + N * (k - 1) + n) = f(t);
        end
    end

    s = e;
    e = e + 1;
end
toc
```

```
Elapsed time is 58.377413 seconds.
```

Normalize the data by column, ignoring the IDs

```
Ds_ = Ds;
Ds_(:,2:end) = Ds_(:,2:end) ./ repmat( max(Ds_(:,2:end)),
 size(Ds_,1),1 );
```

Save the interpolated data to file

```
output_filename
 = 'test_filtered_b.sorted.interpolated.norm.sample.csv';

dlmwrite( output_filename, Ds_, 'delimiter', ',', 'precision', 16 );
```

Published with MATLAB® R2015a

```
function [w] = RidgeRegression( X, y )

%
% Create a linear model using Ridge Regression using cross validation
%  and plot the y against Yhat
%

[n,m] = size(X);
```

```
Error using RidgeRegression (line 7)
Not enough input arguments.
```

analytical method for linear regeression, cross validation

```
lambda = 0:5:100;
dev_errors = zeros( size(lambda) );
eye_ = eye(m);
eye_(end,end) = 0;

cv_errs = zeros(size(lambda,2),1);
for i = 1:size(lambda,2)
    lam = lambda(i);
    Hlam = X* inv(X'*X + lam*eye_) * X';
    yhat = Hlam*y;
    for k = 1:n
        cv_errs(i) = cv_errs(i) + ((yhat(k)-y(k))/(1-Hlam(k,k)))^2;
    end
    cv_errs(i) = cv_errs(i) / n;
end

figure
hold on;
plot( lambda, cv_errs, 'b-' )
title( 'Cross Validation Error for Ridge Regression' )
xlabel('lambda')
ylabel('Cross Validation Error')
hold off;
```

Plot y vs yhat using lambda star

```
[min_cv, ind] = min(cv_errs);
lambda_star = lambda(ind)

w = inv(X'*X + lambda_star*eye_) * X' * y;
yhat = X * w;

figure
hold on;
plot(y, yhat, 'b.')
```

```
title('Ridge Regression Using Lambda Star - Training')
xlabel('y');
ylabel('y hat');
axis([0 1 0 1])
hold off;

end
```

Published with MATLAB® R2015a

```
function [ Xtrain, Ytrain, Xtest, Ytest, ntrain, mtrain, ntest,
 mtest ] ...
    = getTrainAndTestSets( M, max_n, ratio )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

M = M(randperm(size(M,1)),:);

id = M(:,1);
minutes = M(:,2);

Ytrain = M(1:max_n,end);
Xtrain = M(1:max_n,3:end-1);

Xtest  = M(max_n+1:max_n+1+max_n*ratio,3:end-1);
Ytest  = M(max_n+1:max_n+1+max_n*ratio,end);

Xtrain(:, end+1) = ones(size(Xtrain,1), 1);

Xtest(:, end+1) = ones(size(Xtest,1), 1);

[ntrain,mtrain] = size(Xtrain);
[ntest,mtest] = size(Xtest);

end
```

```
Error using getTrainAndTestSets (line 6)
Not enough input arguments.
```

Published with MATLAB® R2015a

Table of Contents

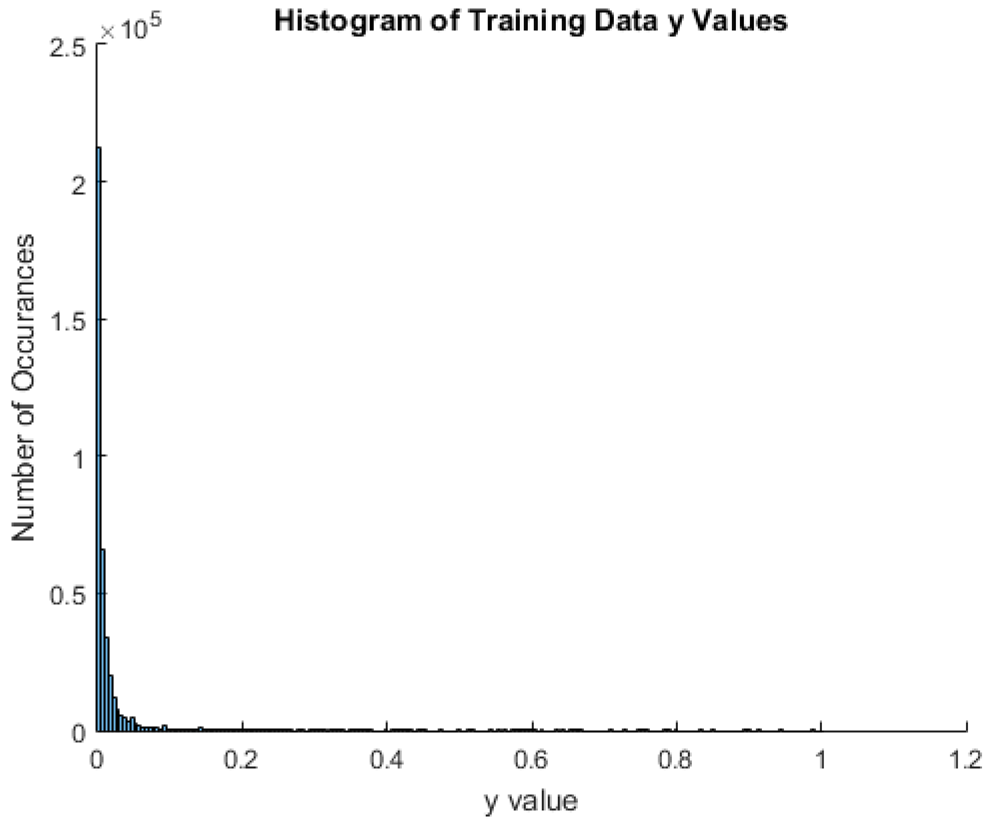
Do we have a heavy tailed distribution of our data?

Read the data

```
M = csvread('train_filtered_b.sorted.squashed.norm.csv', 1, 0);

y = M(:,end);
```

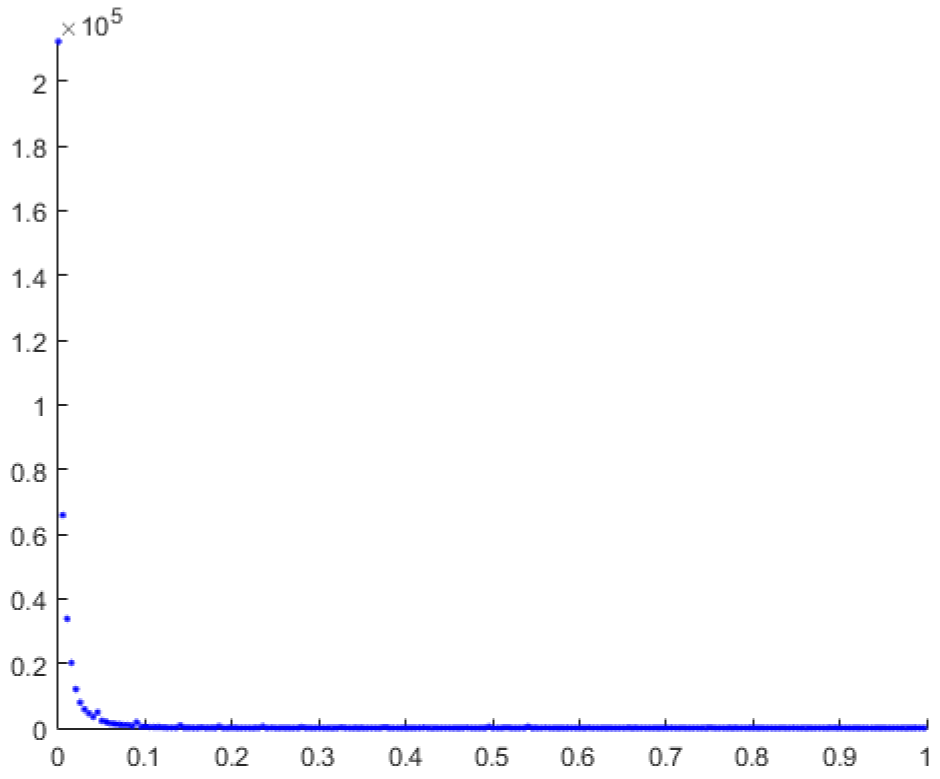
Plot a histogram of the y-values

```
figure
hold on;
numBins = 200;
h = histogram(y, numBins);
title('Histogram of Training Data y Values')
xlabel('y value')
ylabel('Number of Occurances')
hold off;
```



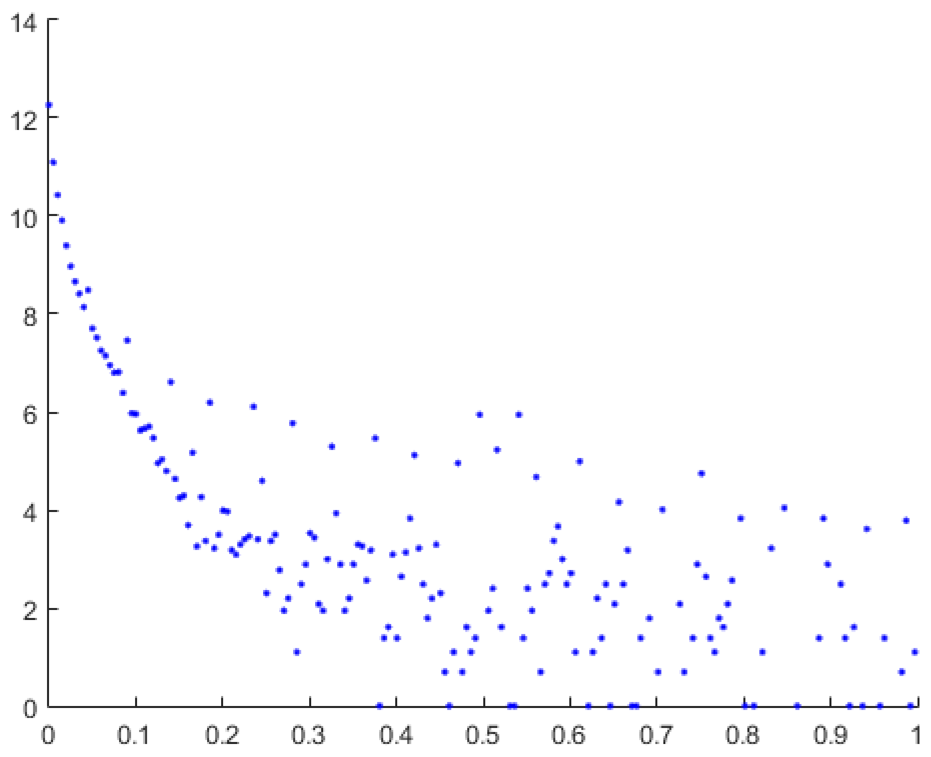
linear plot

```
figure
hold on;
plot( h.BinEdges(1:end-1), h.Values, 'b.' )
axis([0 1 0 max(h.Values)+1])
hold off;
```

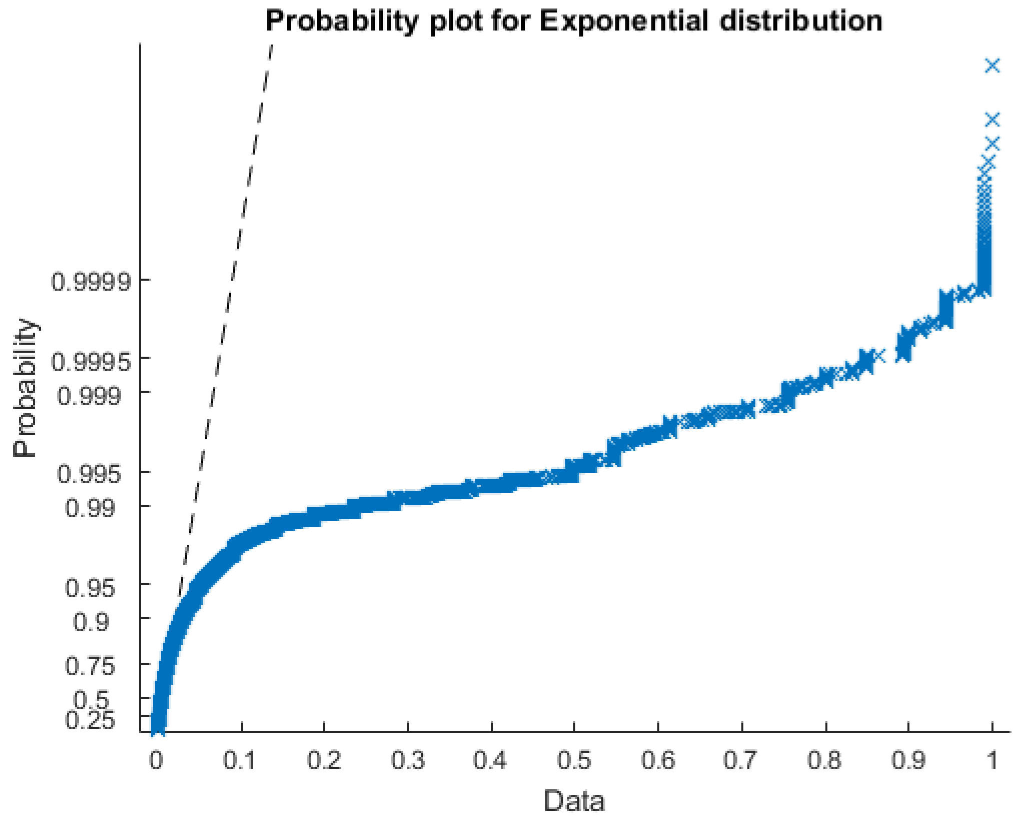


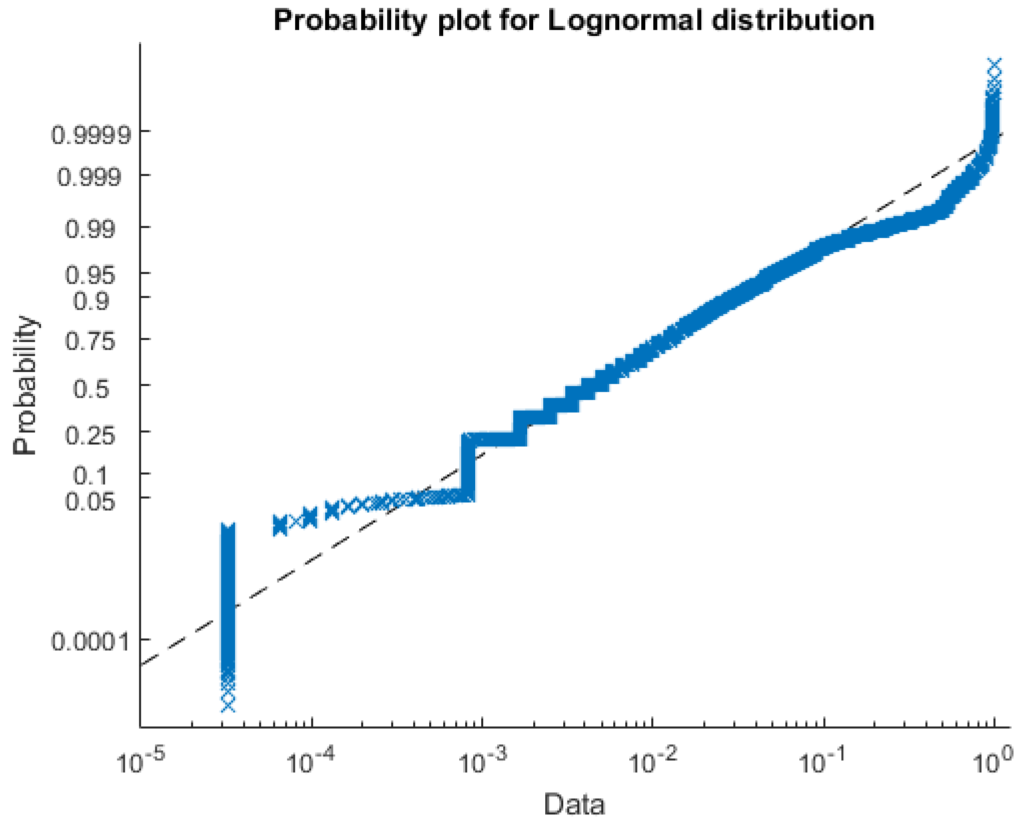
log-linear plot

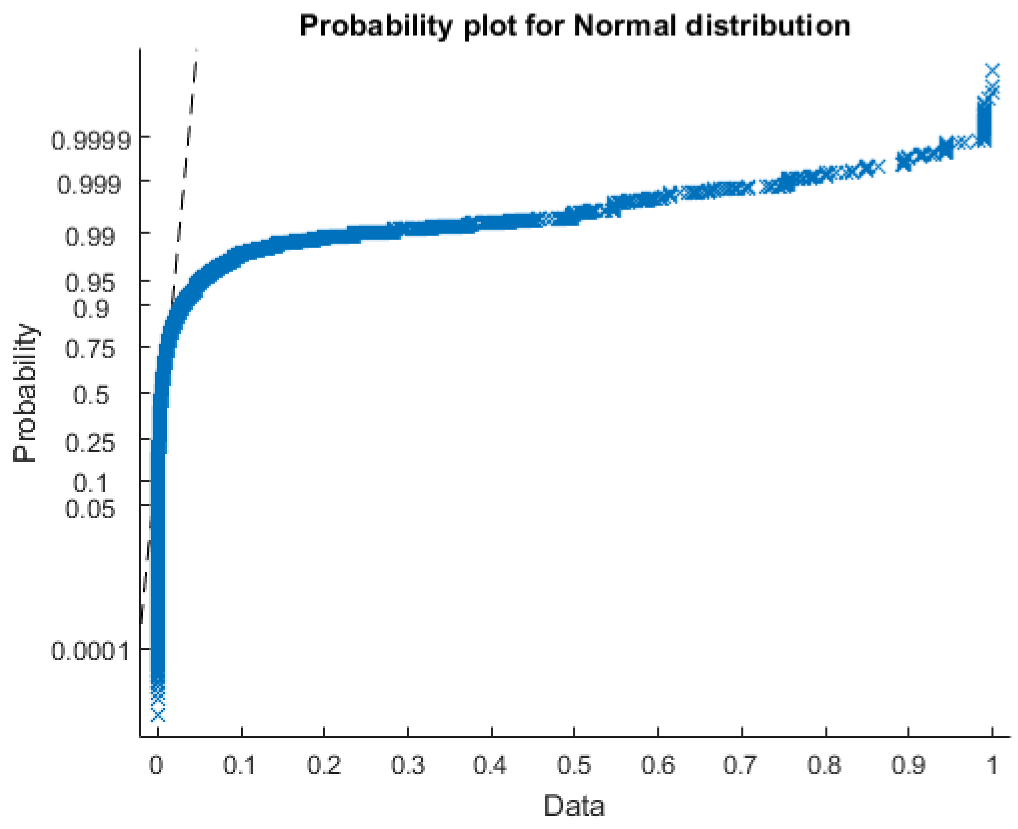
```
figure
hold on;
plot( h.BinEdges(1:end-1), log(h.Values), 'b.' )
% axis([0 1 0 1])
hold off;
```

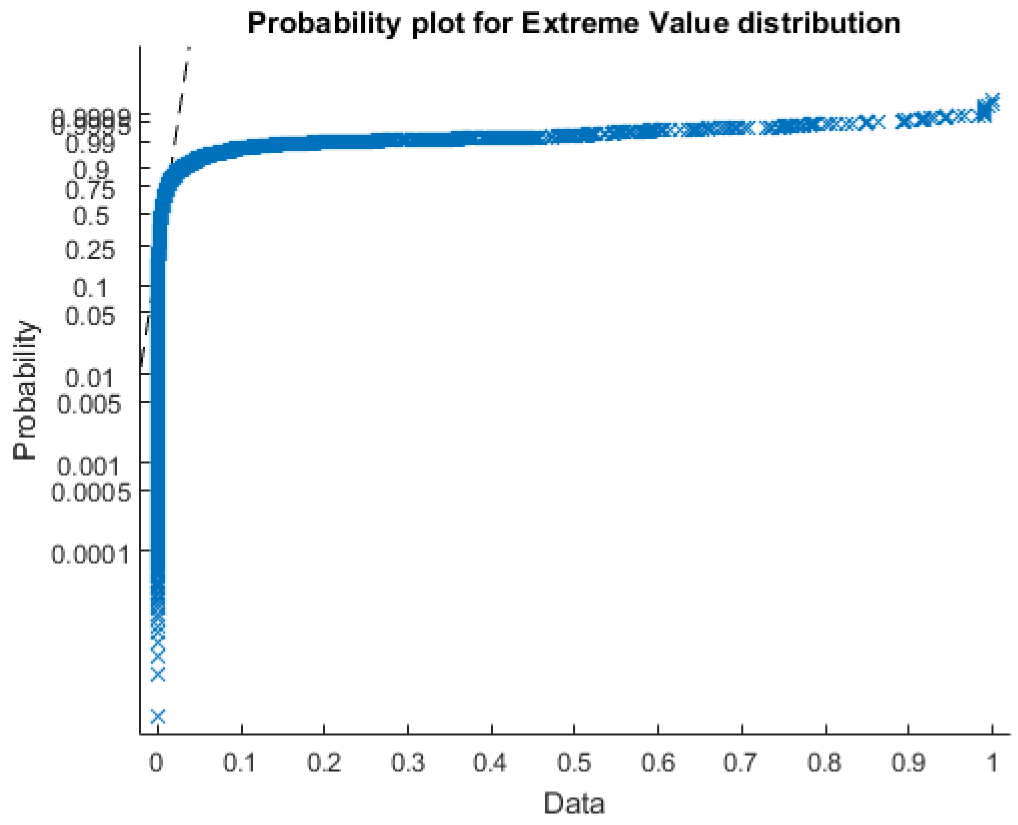


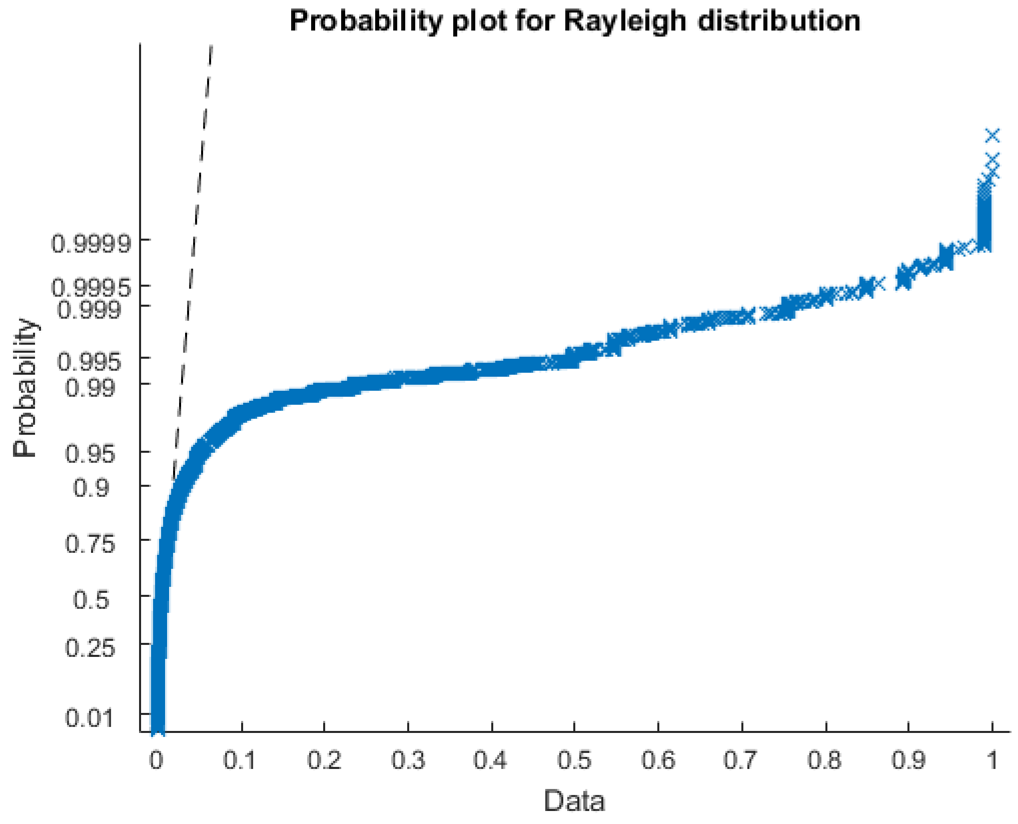
```
figure
probplot('exponential', y)

figure
probplot('lognormal', y)

figure
probplot('normal', y)

figure
probplot('extreme value', y)

figure
probplot('rayleigh', y)

figure
probplot('weibull', y)
```

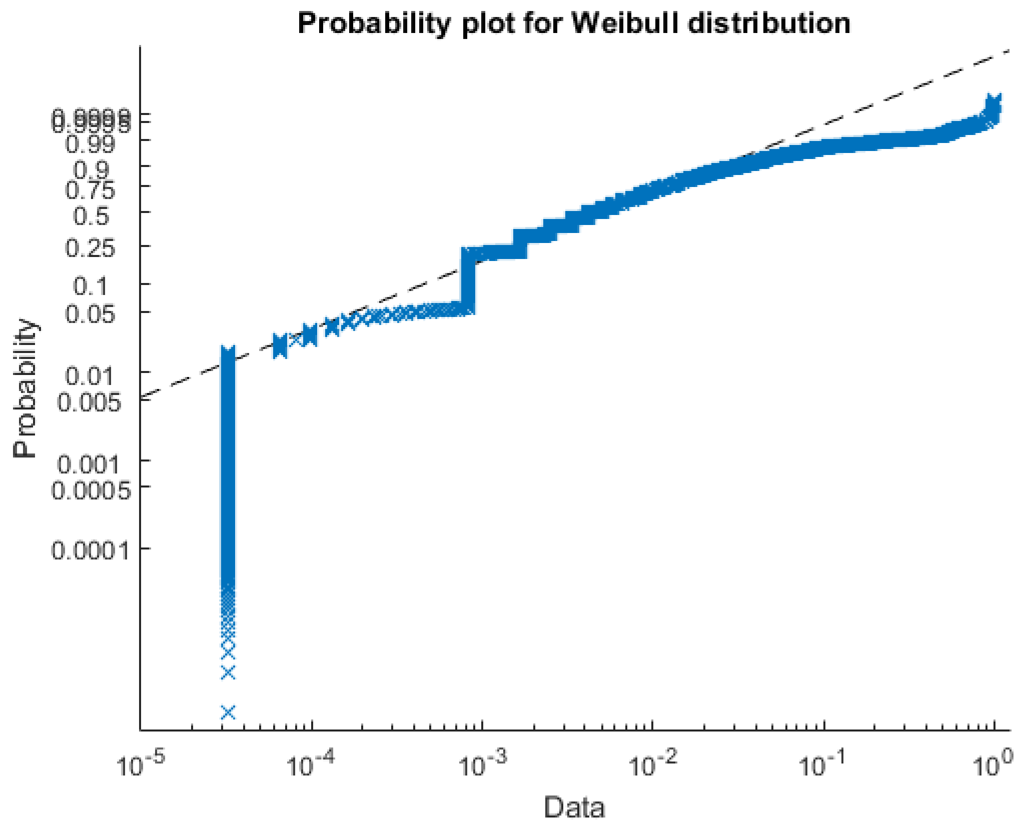

Probability plot for Exponential distribution
Probability
0.9999
0.9995
0.999
0.995
0.99
0.95
0.9
0.75
0.5
0.25
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Data


Probability plot for Lognormal distribution
Probability
0.9999
0.999
0.99
0.95
0.9
0.75
0.5
0.25
0.1
0.05
0.0001
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
Data


Probability plot for Normal distribution
Probability
0.9999
0.999
0.99
0.95
0.9
0.75
0.5
0.25
0.1
0.05
0.0001
Data
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

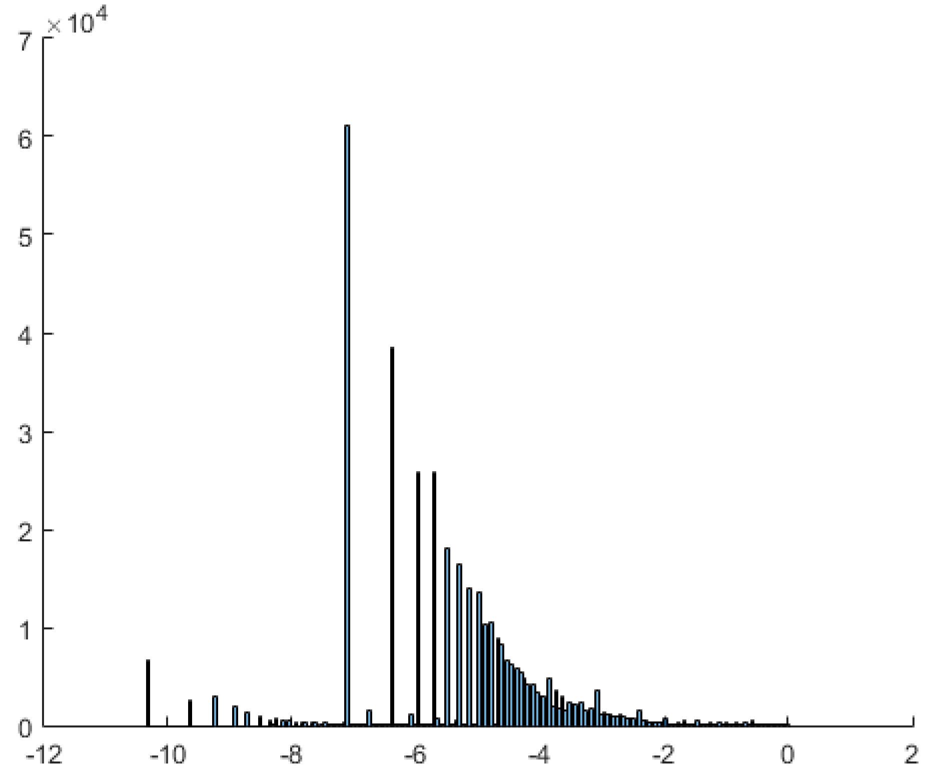

Probability plot for Extreme Value distribution
Probability
0.9999
0.9995
0.99
0.9
0.75
0.5
0.25
0.1
0.05
0.01
0.005
0.001
0.0005
0.0001
Data
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Probability plot for Rayleigh distribution
Probability
0.9999
0.9995
0.999
0.995
0.99
0.95
0.9
0.75
0.5
0.25
0.01
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Data


Probability plot for Weibull distribution
Probability
0.9999
0.9995
0.99
0.9
0.75
0.5
0.25
0.1
0.05
0.01
0.005
0.001
0.0005
0.0001
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
Data

Plot a histogram of the log of the y-values

```
figure
hold on;
numBins = 200;
h = histogram(log(y), numBins);

hold off;
```



Published with MATLAB® R2015a

```
function svrobj = svr_trainer(xdata,ydata, c, epsilon, kernel,
 varargin)
%
% Original Author: Ronnie Clark
% Modified by: Matt Poegel
%
% SVR  Utilises Support Vector Regression to approximate
%           the functional relationship from which the
%           the training data was generated.
%  Function call:
%
%    svrobj = svr_trainer(x_train,y_train,c,epsilon,kernel,varargin);
%    The training data, x_train and y_train must be column vectors.
%
%  Example usage:
%
%    svrobj =
 svr_trainer(x_train,y_train,400,0.000000025,'gaussian',0.5);
%    y = svrobj.predict(x_test);
%
    if strcmp(kernel,'Gaussian')
        lambda = varargin{1};
        kernel_function = @(x,y) exp(-lambda*norm(x.feature-
y.feature,2)^2);
    elseif strcmp(kernel,'Spline')
        kernel_function = @(a,b) prod(arrayfun(@(x,y) 1 + x*y
+x*y*min(x,y) ...
            -(x+y)/2*min(x,y)^2+1/3*min(x,y)^3,a.feature,b.feature));
    elseif strcmp(kernel,'Periodic')
        l = varargin{1};
        p = varargin{2};
        kernel_function = @(x,y) exp(-2*sin(pi*norm(x.feature-
y.feature,2) ...
            /p)^2/l^2);
    elseif strcmp(kernel,'Tangent')
        a = varargin{1};
        c = varargin{2};
        kernel_function = @(x,y) prod(tanh(a*x.feature'*y.feature+c));
    elseif strcmp(kernel,'Polynomial')
        d = varargin{1};
        kernel_function = @(x,y) (x.feature*y.feature' + 1)^d;
    end

    ntrain = size(xdata,1);

    alpha0 = zeros(ntrain,1);

    for i=1:ntrain
     for j=1:ntrain
            xi(i,j).feature = xdata(i,:);
            xj(i,j).feature = xdata(j,:);
        end
    end

    % *********************************
    % Set up the Gram matrix for the
    % training data.
    % *********************************
    M = arrayfun(kernel_function,xi,xj);
    M = M + 1/c*eye(ntrain);

    % *********************************
    % Train the SVR by optimising the
    % dual function ie. find a_i's
    % *********************************

    options = optimoptions('quadprog','Algorithm','interior-point-
convex');
    H = 0.5*[M zeros(ntrain,3*ntrain); zeros(3*ntrain,4*ntrain)];
    figure; imagesc(M);
    title('Inner product between training data ...(ie. K(x_i,x_j)');
    xlabel('Training point #'); ylabel('Training point #');

    lb = [-c*ones(ntrain,1); zeros(ntrain,1); zeros(2*ntrain,1)];
    ub = [ c*ones(ntrain,1); 2*c*ones(ntrain,1); c*ones(2*ntrain,1)];
    f = [ -ydata;
 epsilon*ones(ntrain,1);zeros(ntrain,1);zeros(ntrain,1)];
    z = quadprog(H,f,[],[],[],[],lb,ub,[],options);

    alpha = z(1:ntrain);
    figure; stem(alpha);
    title('Visualization of the trained SVR');
    xlabel('Training point #'); ylabel('Weight (ie. \lambda_i -
 \lambda_i^*)');
    % *********************************
    % Calculate b
    % *********************************
    for m=1:ntrain
        bmat(m) = ydata(m);
        for n = 1:ntrain
            bmat(m) = bmat(m) - alpha(n)*M(m,n);
        end
        bmat(m) = bmat(m) - epsilon - alpha(m)/c;
    end
    b = mean(bmat);

    % *********************************
    % Store the trained SVR.
    % *********************************
    svrobj.alpha = alpha;
    svrobj.b = b;
    svrobj.kernel = kernel_function;
    svrobj.train_data = xdata;
    svrobj.predict = @(x) cellfun(@(u) svr_eval(u),num2cell(x,2));

    % *********************************
    % Plot Y vs. Yhat
    % ********************************
    yhat = svrobj.predict(xdata);
    figure;
    hold on;
    plot( ydata, yhat, 'b.' );
    title( strcat('SVR with ', kernel, ' Kernel - Training') );
    xlabel( 'y value' );
    ylabel( 'y hat' );
    axis([min(ydata) 1 min(yhat) 1]);
    hold off;

function f = svr_eval(x)
    f = 0;
    n_predict = size(x,1);
    for i=1:n_predict
     sx(i).feature = x(i,:);
    end
    n_train = size(xdata,1);
    for i=1:n_train
     sy(i).feature = xdata(i,:);
    end

    for i=1:n_train
        f = f + svrobj.alpha(i)*kernel_function(sx(1),sy(i));
    end
    f = f + b;
    f = f/2;
end

end
```

```
Error using svr_trainer (line 19)
Not enough input arguments.
```

Published with MATLAB® R2015a

```
#CONSTANTS
param p;        # number of predictor variables
param n;        # total number of data points
param lambda;

set P:={1..p};  # indices of preditor variables
set N:={1..n};  # indices of observations

param X{N,P};   # predictor data
param Y{N};# response data


#decision variables
var w{P};
var b;


#OBJECTIVE FUNCTION for ridge regression model

minimize objective:
sum{i in N} ( sum {j in P} X[i,j]*w[j]+b-Y[i])^2 + lambda*sum{j in P} w[j]*w[j] ;
```

Appendix C: Data Files

Shortened URL:
`http://bit.ly/1rK5UaT`

Full URL:
`https://drive.google.com/folderview?id=0B0XdSwTDS3G-Y3hDQjQzUERxeGM&usp=sharing`