A comparison of three machine learning algorithms for predicting rainfall amount

Zachary DeStefano (zdestefa), Elmira Forouzmand (forouzme), Daniel Quang (daquang)

March 20, 2015

1 Introduction

We decided to explore three different machine learning algorithms: regression trees, Support Vector Regression (SVR), and neural networks, and compare their performance on predicting the amount of rainfall at a given location, using remote sensing (satellite) and numerical model data. Because this is a regression problem, we decided to use Mean Squared Error (MSE) as our error function for judging the performance of our algorithms.

2 Materials and Methods

2.1 Data

Training and testing data were downloaded from the in-class Kaggle competition, **How's the weather?**. The training dataset includes 60000 samples and the testing dataset includes 40000 samples. Labels are only available for training data, corresponding to non-negative rainfall amounts at various locations. Features are split into two sets: **X1** and **X2**. **X1** is a set of 91 primary features, and **X2** is a set of 441 features derived from raw IR3 local image patches.

2.2 Implementation

The SVR algorithm was implemented in Python with the relevant modules from scikit-learn, an open source machine learning library. The regression trees algorithm was implemented in MATLAB and used the treeRegress class provided from the class source code to make the decision trees.

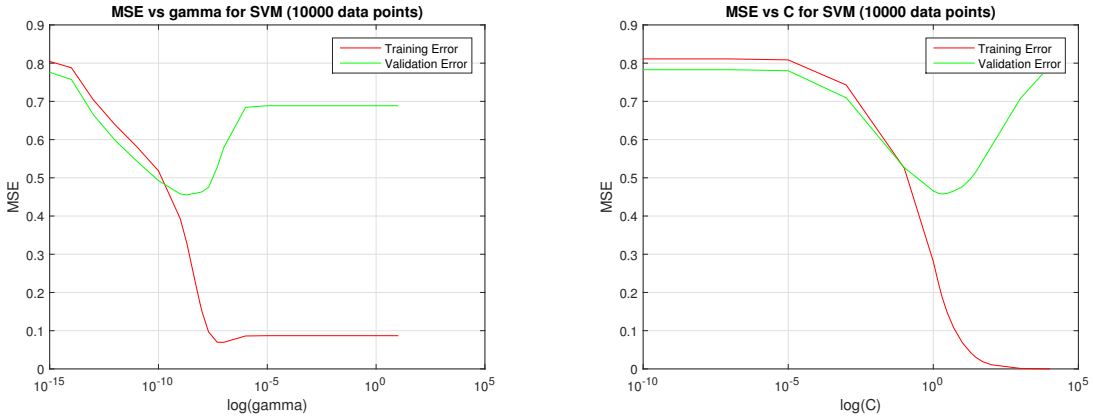
The backpropagation algorithm for neural network training was implemented in Pylearn2, a machine learning Python library. Pylearn2 is built on top of Theano, another Python library that optimizes and stabilizes mathematical expressions and then compiles them to a GPU backend.

3 Results

3.1 SVR

We first investigated the performance of the SVR algorithm. It is an optimization algorithm that works by minimizing the error while maximizing the margin. It has two hyperparameters to tune: C, the penalty for error, and γ, the threshold for tolerated error. Hyperparameter tuning was accomplished in a grid search fashion. Before training, **X1** and **X2** features are preprocessed by a Radial Basis Function Kernel which performed efficiently. However, the SVR algorithm is generally very inefficient because its running time grows quadratically with the number of training

samples. Therefore, we randomly selected 10000 training samples to train the model on, which we further split into a training and validation set in an 8:2 ratio. Based on the learning curves (**Figure 1**), we set C and γ to 1.2 and 5e-9, respectively. Evaluating this final model on the test dataset, we achieved an MSE of 0.444.



$C = 1.5$ $\gamma = 5e-9$

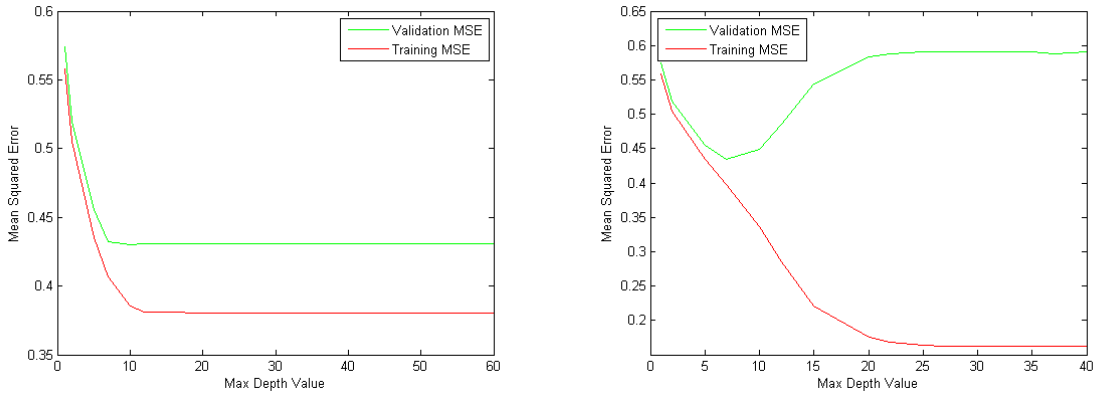
Figure 1: Learning curves for SVR showing the MSE (y-axis) on training data (red) and validation data (green) as a function of the hyperparameter γ (left) or C (right) (x-axis). Below each plot is the value of at which the other parameter is fixed.

3.2 Regression Trees

Regression trees, or decision trees, can capture non-linear relationships among features and naturally output non-negative values. To develop our tree model, we first tune the hyperparameters, minParent and maxDepth, of a single decision tree (**Figure 2**) by evaluating its performance on 20% of the training data we set aside as a validation set. The features include **X1** and the mean and standard deviation of the image patch features in **X2**.

In the work done in class, it was verified that the ideal value for minParent was 512. We verified the ideal value for maxDepth and found that once it reaches 20 (**Figure 2**), the training and validation error stay relatively constant. This is likely because after 20 layers, there is not enough data for more splits. To verify this, we lowered the minParent value to 32 and then tested the training and validation error with different maxDepth values. As it turns out, at 30 layers, the training error stops improving (**Figure 2**). Looking at the validation error as a function of maxDepth, it is clear that there is overfitting after about 7 layers. Hence having a low minParent value increases overfitting. In the end setting the max depth and minParent to 20 and 512, respectively, provides optimal performance on the validation set.

Although a single decision tree can perform well, an ensemble of decision trees can perform significantly better. We first investigate Random Forests, an ensemble method that averages outputs



minParent=512 minParent=32

Figure 2: Learning curves for training a single decision tree showing the MSE (y-axis) on training data (red) and validation data (green) as a function of the max depth value (x-axis). Below each plot is the value minParent is fixed at.

from decision trees trained on different feature subsets and random subsets of the training data (**Figure 3**). As can be observed in the figure, to make this work well, it is necessary to train on a large number of the features, suggesting that many of the features together help determine the outcome. The individual predictions still seem weak, so we need to combine a good number of learners also. In the end, our best Random Forest model (40 features and 30 weak learners) achieved an MSE of 0.363 on the test dataset.

We did investigate another popular ensemble method, Gradient Boosting. We tested the number of gradient boosted learners versus MSE and found that the validation error stays relatively constant after 100 learners are used. The training error does decrease, however. This suggests that the additional trees do fit to the training data but do not affect the validation predictions. In the end though, Gradient Boosting with 100 boosted learners did not do as well as Random Forests and achieved an MSE of 0.376.

Next, we explored whether we can improve performance by combining Random Forests with Gradient Boosting according to the following procedure:

1. Train decision trees on random sets of 40 features
2. Take the 30 best sets of features and train boosted decision trees on them
3. Ensemble the trees from step 2

Figure 4 shows the learning curves for training the Gradient Boosted Random Forest. The "best" hyperparameters were selected and the model was retrained on the entire training dataset. This final model achieved an MSE of 0.378 on the test data, which is not an improvement on our Random Forest model. This could suggest that boosting the trees based on the training data does

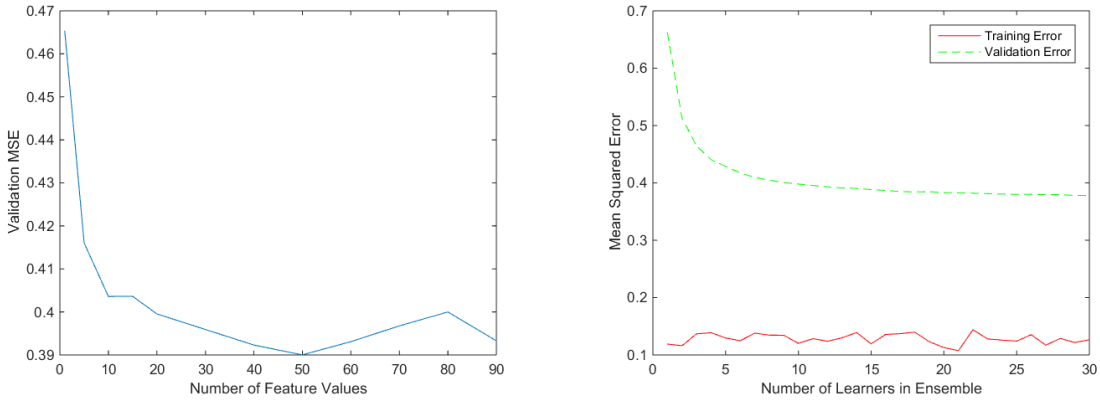


Figure 3: Learning curves for training a random forest model. Plots show the MSE as either the number of feature values (left) or the number of learners in the ensemble (right) are varied. Max depth and MinParent of individual decision trees were set to 20 and 512, respectively.

not work well for this case and efforts should instead be focused on making more detailed decision trees and averaging their results.

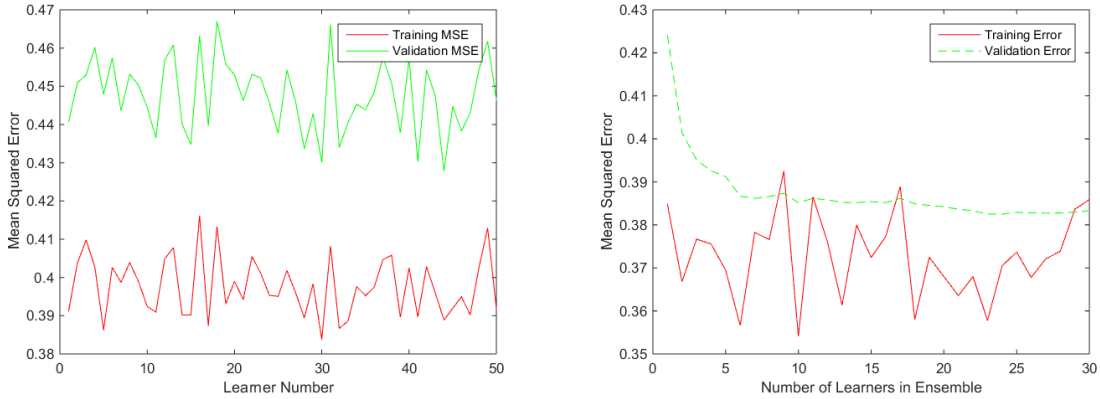


Figure 4: Learning curves for training a Gradient Boosted Random Forest. The left plot details gradient boosting and the right plot details the averaging of several models.

3.3 Neural Network

We next trained a neural network to predict rainfall amounts. This is advantageous for problems with many features because deep neural networks can infer the optimal set of features to use without any manual feature selection. Therefore, we trained neural networks on the full set of features, **X1** and **X2** (normalized to zero mean and unit variance).

To study how deeper networks perform, we trained neural networks with between 1 and 5 hidden layers (data for model with 4 hidden layers not shown). Due to the large number of parameters,

	SVR	Random Forest	Neural Network
Test MSE	0.444	0.363	0.364

Table 1: Summary of performance on testing data with three models.

neural networks have a tendency to overfit if trained for too many training steps or epochs. Therefore, we further split the training data into a training and validation set in a 9:1 ratio. Training is terminated if the validation MSE, which is evaluated at the end of each training epoch, fails to decrease after 10 training epochs. Each hidden layer contains 1000 hidden neurons and the learning rate is fixed at 0.001. Rectified linear units serve as the activation function for computing the values of the hidden neurons and the final output layer is a standard linear layer. All weights were initialized by randomly drawing from a Gaussian distribution with mean 0 and standard deviation 0.01. Weights were updated by stochastic gradient descent with a fixed minibatch size of 100.

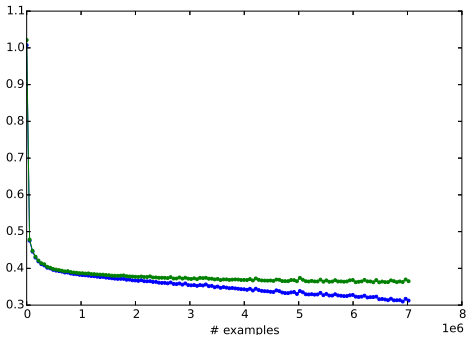
Learning curves for the neural network training clearly displays evidence of overfitting (**Figure 5**). Optimal performance on the validation was reached around 122 training epochs. Based on the performance on the validation set, we selected the neural network with 2 hidden layers as our "best" model. We retrained this neural network with the full training data for 122 epochs and evaluated. Because linear output layers are not strictly non-negative, we apply an additional preprocessing step of converting all negative predictions to 0 before evaluating the trained model on the testing set. Evaluating this model on the test data, we achieved an MSE of 0.364.

3.4 Summary of results on test data

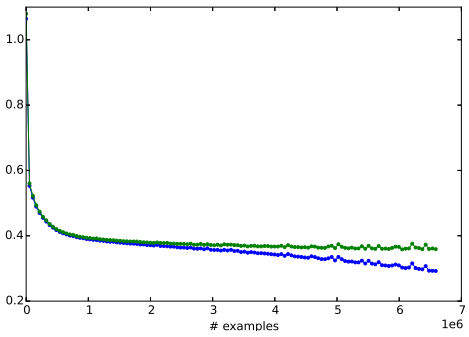
Table 1 summarizes the performance of the three models on the test dataset. We find that regression trees perform the best overall, but neural networks perform similarly well, while SVR performs the worst.

4 Discussion and conclusions

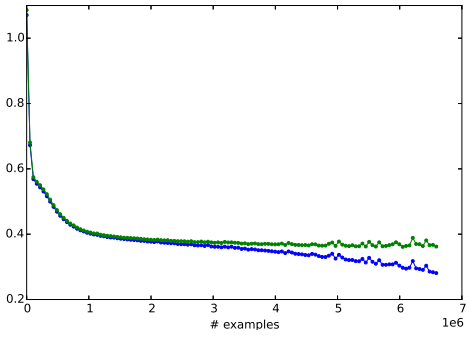
In this paper, we tuned and compared three different machine learning regression models and compared their performance on predicting rainfall amounts. Based on the MSE on the test data, SVR performs the worst, while regression trees and neural networks achieve very similar performance. We hypothesize this is because the Radial Basis Function makes some strong assumptions about the features, while regression trees and neural networks make relatively fewer assumptions and can capture non-linear relationships among the features. Future work can focus on improving the individual models. For SVR, one of the limiting factors was that we had to train on a subset of the data due to the long running time of the training algorithm. An obvious route to explore is to apply powerful GPU hardware to accelerate SVR training so that we can train an SVR model on



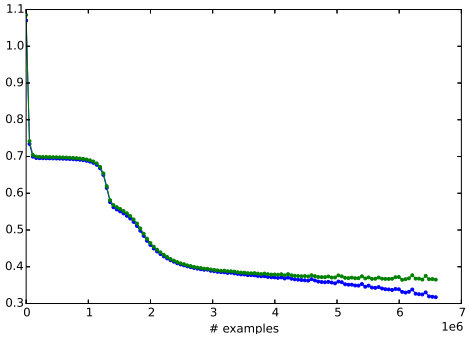
$N_{hidden} = 1$
Train MSE: 0.316
Validation MSE: 0.362



$N_{hidden} = 2$
Train MSE: 0.303
Validation MSE: 0.358



$N_{hidden} = 3$
Train MSE: 0.297
Validation MSE: 0.361



$N_{hidden} = 5$
Train MSE: 0.332
Validation MSE: 0.365

Figure 5: Learning curves for neural network training showing the MSE (y-axis) on training data (blue) and validation data (green) as a function of the number of training samples processed (x-axis). Below each plot is the number of hidden layers in the neural network, and the training and validation MSE of the model yielding the lowest validation MSE during training.

the entire training dataset. Based on the overfitting observed in the neural network training, we can explore how dropout improves performance. Applying momentum training, which adjusts the learning rate throughout training, is another option. Different ensemble strategies for regression trees, such as AdaBoost and Extra Trees, may also improve performance. Another avenue of future interest is to combine results from multiple models, including the 3 models considered in this paper, into an ensemble model.

5 Authors' contributions

All authors wrote the manuscript. ZD implemented regression trees. EF implemented SVR. DQ implemented neural networks. All authors read and approved the final manuscript.