Comp 540 Final Report:
Galaxy Zoo Challenge with Extremely Randomized Trees

Matt Delhey
matt.delhey@rice.edu

Abstract

The Galaxy Zoo Challenge asks participants to predict proportions of user classifications of galaxy morphologies. Using the Extra-Trees algorithm and data preprocessing and augmentation, an error rate of 0.12859 and rank of $145/329$ was achieved. The key idea behind the solution was to mimic the success of convolutional neural networks by implementing a large model with lots of randomness to combat overfitting. While this approach performed moderately well, a feature engineering focused approach would likely perform better given my computational limitations.

1 Introduction

1.1 Galaxy Zoo Challenge

The Galaxy Zoo challenge was a data mining competition hosted on Kaggle[1] that asked participants to reproduce the probability distributions derived from human classifications of galaxy morphologies taken from telescope images. The provided data has two components: (1) the telescope images[2] and (2) the probability distributions related to each image, generated by the citizen scientist volunteers. Both components of the data underwent significant filtering and preprocessing[3], which is documented in detail in the Galaxy Zoo 2 paper ([2] Willett, Kyle W., et al.).

The task, then, is to predict the user response distribution of the galaxy morphologies using the very same telescope images. This is distinct from just classifying the galaxies themselves: in our case, we are trying to predict user responses and not necessarily the actual galaxy morphologies.

Users provide responses to the galaxy morphologies through the help of a decision tree, which can be seen in Figure 1. The data represents the proportions of users who selected each possible response. In total, there are 11 tasks or probability distributions and 37 responses. For each galaxy image, we are either given the 37 responses (the training case), or asked to predict the 37 responses (the testing case).

The model is then evaluated using a standard root mean squared error (RMSE). I like to think of the RMSE as the standard deviation of the unexplained variance, giving an absolute quality of the model in terms of (i.e. on the same scale as) the response. Its formal characterization for this competition is:[4]:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2}$$

Where

[1] `https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge`

[2] All telescope images used in the Galaxy Zoo 2 dataset were taken from the Sloan Digital Sky Survey, which consists of approximately one million galaxy images. Our dataset consists of $61,578$ train images and $79,975$ test images.

[3] The preprocessing resulted in data that was rotation, scale, and translation invariant.

[4] `https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/details/evaluation`

- N is the number of galaxies times the total number of responses (e.g. for the training set there are $61,578$ images, so $N = 61,578 * 37 = 2,278,386$).
- p_i is the predicted value for each response.
- a_i is the actual value for each response.

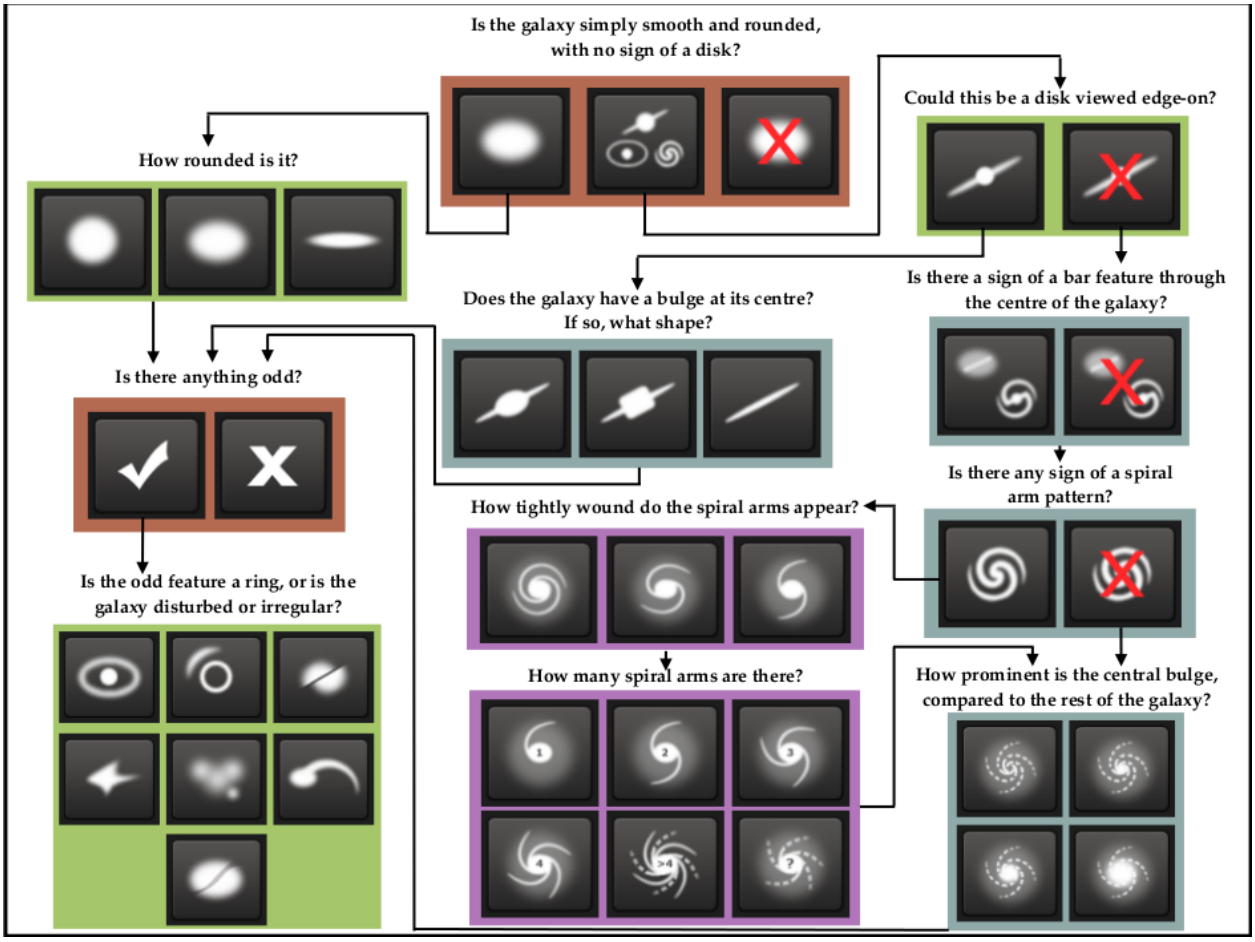

Figure 1: Provided flowchart for the user classification tasks for Galaxy Zoo volunteers. Each separate colored box represents its own probability distribution. Tasks are color-coded by their relative depths in the decision tree. Tasks outlined in brown are asked for every galaxy. Tasks outlined in green, blue and purple are one, two, or three steps below branching points in the decision tree. Flowchart taken from the Galaxy Zoo 2 paper ([2] Willett, Kyle W., et al.).

1.2 Interpreting the problem

I approached the Galaxy Zoo challenge as a multiple regression problem. While one could approach this problem from it's semantics, attempting to generate 11 probability distributions and taking into account the various constraints, treating the problem as multiple regression simplified the task. In my own experience, using the constraints imposed by the problem semantics did not decrease the overall error. This was confirmed at the end of the competition: most competitors also reduced the problem to regression.

I did, however, implement a hard constraint, forcing all predicted values to fall between 0 and 1, inclusively. So, for example, a predicted value of 1.1 would be converted to a prediction of 1. As my models improved, the forcing effect became negligible in the overall error of the model.

1.3 Background & related work

The first Galaxy Zoo challenge asked participants to construct a classifier to determine if each galaxy was either elliptical, spiral, or merger. This corresponds to the first question in the decision tree for our competition. The first competition was ran on a much smaller dataset, consisting of images that had been hard labeled by professional astronomers. The state of the art for this case was a misclassification rate of about 7%, using a neural network with two hidden layers and $2N$ nodes each[5], where N is the number of input parameters ([1] Banerji, Manda, et al). It is important to note that this model was trained using features that were not available in the Galaxy Zoo 2 competition, i.e. not extracted from only the images themselves.

Many of top scoring solutions to the Kaggle competition utilized large convolutional neural networks (CNNs). Indeed, the top three scoring solutions all used convolutional neural networks [6] and two of used Alex Krizhevsky's cuda-convnet library ([3] Krizhevsky, Sutskever, Hinton). The general consensus between these three solutions was to train a very large network and avoid overfitting using data augmentation, i.e. random transformations and perturbations of the training data.

1.4 My approach in context

While large convolutional networks seem to be the best performers for this problem, they have the primary drawback of being computational intensive. All three of the winning solutions, for example, measured their training time in days and had the aid of current generation GPU's. Rather than try to compete at the level of computation, I instead only used their conceptual essence: a large model with lots of randomization.

2 Methods

At a high level, my method was to use a large committee of extremely randomized decision trees trained on randomly manipulated data in order to reduce overfitting.

2.1 Preprocessing

I used the following preprocessing steps for the each of the images:

1. Crop the images to 200x200 pixels, retaining color. This cuts out the noise in the outer part of the galaxies.
2. Downsample the images to 64x64 pixels. This greatly reduces computation time.
3. Random data augmentation for the **training images**[7]:
    - Random angle rotation, chosen from $Normal(0, 0.3)$.

[5]The resulting architecture is N:2N:2N:3. The authors stated that additional nodes and hidden layers did not improve performance.
[6]The winning solutions can be found at:

1. `http://benanne.github.io/2014/04/05/galaxy-zoo.html`
2. `https://github.com/milakov/nnForge/blob/master/examples/galaxy_zoo/galaxy_zoo.pdf?raw=true`
3. `https://github.com/tund/kaggle-galaxy-zoo/blob/master/report/gz_report.pdf?raw=true`

[7]Adapted from sedielem's winning solution.

- Random scaling, chosen from $Uniform(0, 360)$.
- A chance to flip the image, chosen from $Bernoulli(0.5)$.

All of these potential augmentations are combined into a single affine transformation. Galaxy morphologies are scale and rotational invariant, so one could use this method to generate more "training" data as well.

4. Flatten the image by converting it to a $64 * 64$ by 3 matrix, where the three columns represent the RGB color.
5. Scale the image by dividing each element by 255.
6. Whiten the flat matrix, i.e. normalize on a per color basis where each color is divided by its standard deviation giving it unit variance.

The results of the preprocessing can be seen in Figure 2. The goal was to reduce overfitting to the training dataset by introducing randomness. Whatever features picked out from the augmented data would be the core features for predicting accurately on the test set.

2.2 Algorithm: Extra-Trees

I used the Extra-Trees (ET) algorithm, developed by Geurts, Ernst and Wehenkel ([4] Geurts, Ernst, and Wehenkel), for my final Kaggle submission. The implementation was done in python with the `scikit-learn` library[8].

The Extra-Trees algorithm is similar to Random Forests and other ensembles of trees. It's defining contrast to these other methods is its inclusion of even more randomization and its lack of bagging.

In a Random Forest, each tree is built from a bootstrap sample drawn from the training dataset, wherein each split is chosen from a random subset of features. This increases the bias of each individual tree but results in fewer correlated errors among the trees. If we consider these trees as a committee and take their average, the resulting model is of low variance. This often results in superior models relative to non-randomized trees.

As mentioned, Extra-Trees implements even more randomization into the ensemble process. Extremely randomized trees are all built using the original training dataset without bagging. For each split, both the subset of features (variable index) and the value at which to split (cut-point, variable splitting value) are randomized. The resulting committee is similar to Random Forests, except that variance sees slightly more reduction at the cost of a slightly greater increase in bias[9].

The intuition for preferring this algorithm for this particular problem was the goal of creating a model large enough to accurately predict the 37 responses with enough randomization to reduce overfitting. In a sense, this can be seen as an attempt to mimic the essence of the convolutional neural networks.

2.3 Model Parameters

There are two sets of parameters for Extra-Trees, those that describe the behavior of the each tree (`max_features` or K, `max_depth`, `min_samples_split` or n_{min}, `min_samples_leaf`) and the number of trees to use in the ensemble (`n_estimators`).

The parameter K determines both the number of features and the number of potential splits per feature. For each split in the trees, K random features are selected. For each of feature selected, K random splits are selected. Therefore, at the creation of each node during training the model must score K^2 potential splits.

[8]For more information regarding this implementation, see `http://scikit-learn.org/stable/modules/sklearn.ensemble.ExtraTreesRegressor.htm`

[9]For a full description of the Extra-Trees algorithm see the Geurts, Ernst, and Wehenkel paper.

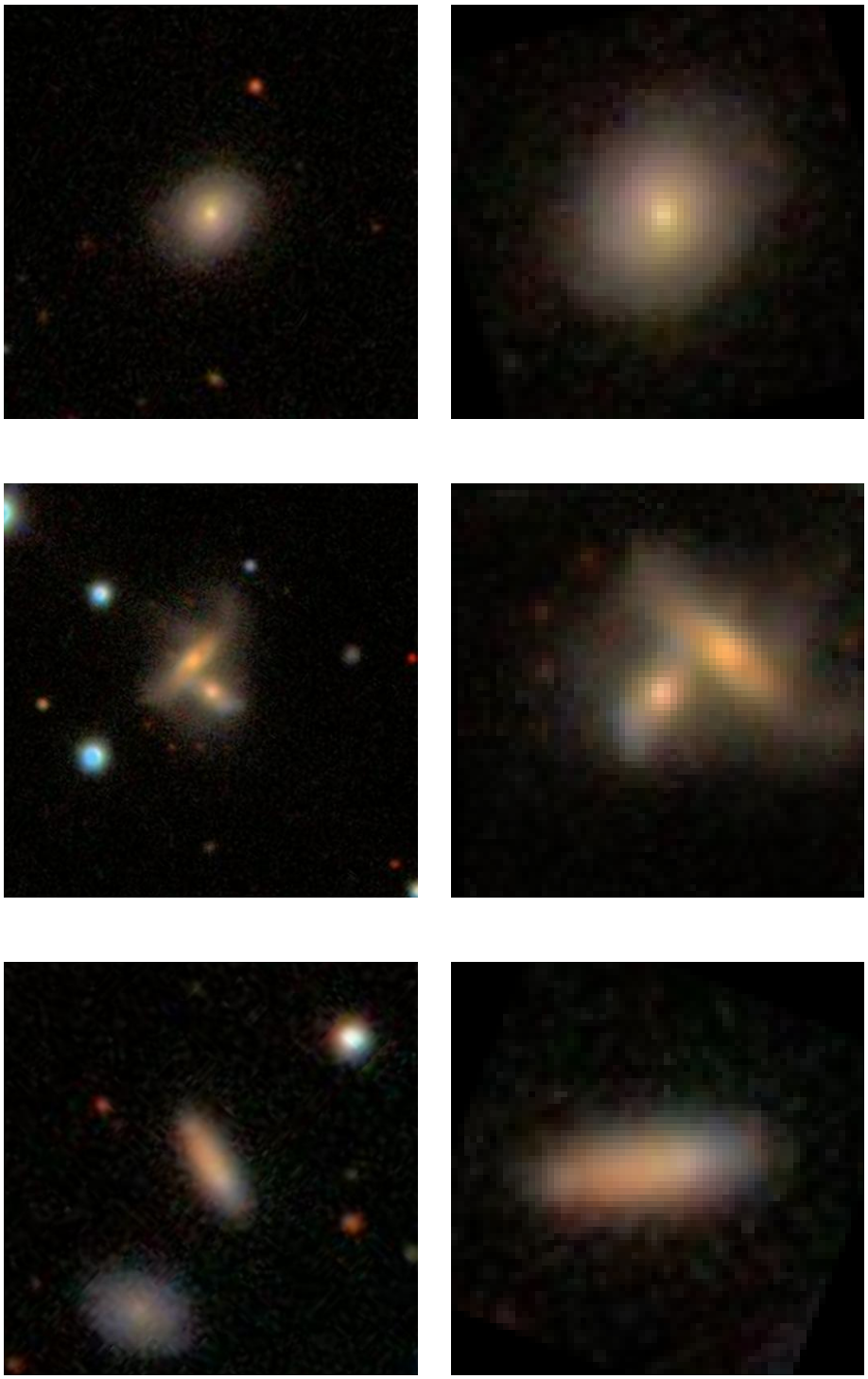

Figure 2: Three examples of the image preprocessing. The left side images are the originals, and the right side images are those used as the input to the learning algorithm.

The potential splits are scored by their relative variance reduction[10]. Predictions are made by pushing every new point through each tree and then averaging their predictions.

The other parameters that determine the behavior of the tree construction are primarily for prevent overfitting by giving extra conditions for stopping tree growth. In my experience, changing these conditions did not have a large effect on the overall error.

Model parameters were found using an incomplete grid search, using the validation framework described in the section below. I say incomplete because computational limitations forced me to limit the number of different parameters I could test. The model parameters that I used were:

parameter	description	value
`n_estimators`	the number of trees in the forest	500
`max_features` (K)	number of random features to consider per split and the number of random splits to consider per feature	$sqrt(64 * 64) = 64$
`max_depth`	maximum number of splits per tree	100
`min_samples_split` (n_{min})	minimum number of samples to split a node	2
`min_samples_leaf`	minimum number of samples required in each leaf for a split to be considered	1

2.4 Performance, Training and Validation

Model performance was primarily conducted using the standard RMSE, as described previously. Models were also analyzed using learning curves and error analysis, which are discussed in more detail in the results.

Model selection and model assessment was done locally using constant, randomly selected hold out evaluation and test sets. The split used was 60% for training, 20% for validation, and 20% for testing or evaluation. Therefore, each submitted model had associated train, validation, test, and leaderboard errors. Such a method ensured that no training data bias was created hyperparameters, but also left me vulnerable to potentially biased hold out sets. Therefore, I also calculated a random hold out error, which randomly split the data into 70% train and 30% test and then evaluated.

I relied primarily on my testing hold out error and random hold out error to determine which models to upload to Kaggle. During the competition, there was also a public leaderboard which was calculated on approximately 25% of the test data. Because Kaggle allowed for 5 submissions every 24 hours, I used the Kaggle leaderboard to compare with my own hold out errors for most of my models.

3 Results

Fitting the Extra-Trees model on the entire training dataset results in an RMSE of 0.12859 on the Kaggle private leaderboard[11], which is the final standings of the competition.

The learning curve of a smaller, yet very similar model trained using 32x32 pixel images can be seen in Figure 3. The learning curve for the full model is not shown due to limited computation time. An analysis of this learning curve gives us an idea of the limitations of the method implementation.

The model appears to be both overfitted and underfitted. On the one hand, the test error is significantly greater than the train error indicating overfitting. On the other hand, even the overfitted train error is

[10] $Score(s, S) = \frac{Var[Y|S] - \frac{|S_1|}{|S|} Var[Y|S_1] - \frac{|S_2|}{|S|} Var[Y|S_2]}{Var[Y|S]}$. Where Y is the response associated with the subset S and S_1 and S_2 represent the potential splits.

[11] `http://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/leaderboard/private`

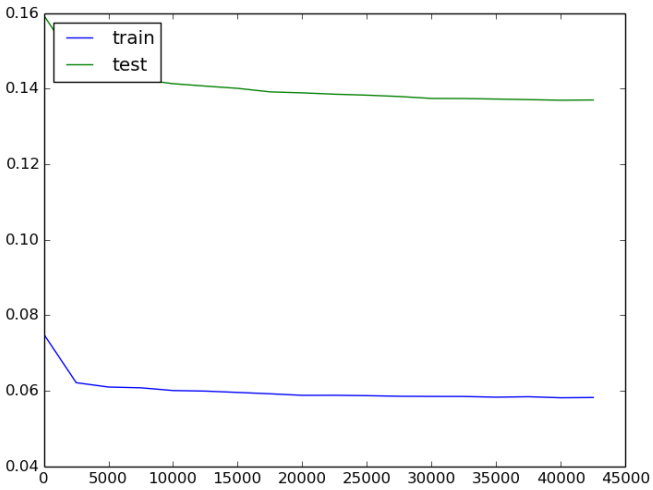

Figure 3: The learning curve of the Extra-Trees model, generated using all of the training data.

around the level of the winning solutions indicating that my features are likely not rich enough for achieving competitive accuracy. This curve, then, is a testament to the difficulty of the problem. Indeed, multiple response regression and object recognition are hard tasks on their own. Combined, these two problems require a model that is systematically both (1) rich enough to accurately fit many continuous responses and (2) regularized or randomized in some way to avoid overfitting. The learning curve demonstrates that my method, in at least a limited sense, failed in both of these dimensions.

3.1 Feature importance

Figure 4 shows two visualizations of the feature importance in the ET model. The feature heatmap shows that the most useful features were those in the center of the image. This isn't an entirely shocking finding, but invites one to wonder how a contour approach to feature engineering would work out. The bar chart shows that blue was the most important color in the model, whereas red is significantly less important.

One possible explanation for the relatively low importance allocated to red is the redshift phenomena, documented by Willett, Kyle W., et al. The general idea is that, due to distortions in space, the wavelength of light increases thus shifting its colors to become more "red". Although images were processed to take in account this effect, it is possible that the problem still effected the dataset. Because redshift is not related to the galaxy morphology, this could help explain why it was found to be relatively noisy.

3.2 Error analysis

In Figure 5, I've included the top 4 galaxies that saw the largest evaluation error in the model. These galaxies are examples of those in which responses to the first question were split about evenly between "smooth" and "features or disk" (e.g. 0.46 to 0.51, or 0.55 to 0.45). The response to this question leads the user down different decision trees. This isn't inherently a problem, except that there are class imbalances between "smooth" and "features or disk" galaxies and very few cases with approximately even splits. This causes the model to treat these cases as weak "smooth" galaxies as opposed to a toss up with "features or disk". It is somewhat comforting, however, that the Extra-Trees model struggles on the very same cases that are difficult for human beings.

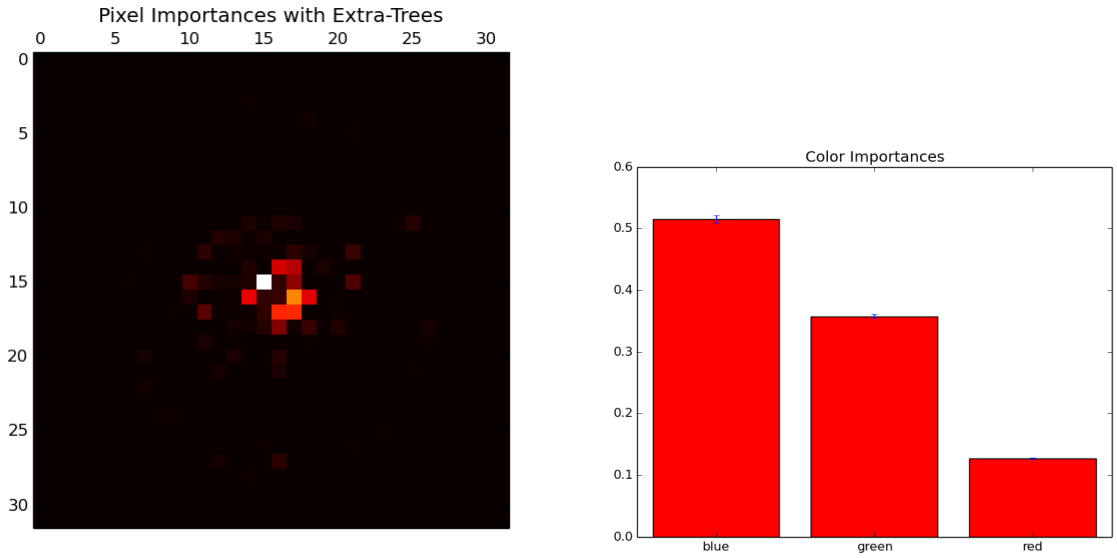

Figure 4: (Left) Feature heatmap, where color represents the relative pixel position importance. The brighter the color, the more important the pixel. White is the highest importance. This heatmap was generated using the 32x32 model. (Right) Bar chart of the relative importances of each image color. This ratio of importance held constant over most images.

3.3 What didn't work

The list of methods and approaches that I tried that did not improve performance is much longer than the list of those that did. Here are a few of the most noteworthy:

- feature engineering: contours, histograms of gradients, image deskewing, color perturbation.
- least squares regression: with and without $L1$ and $L2$ regularization
- dimensionality reduction: PCA and non-negative matrix factorizations
- other models: random forest regression, k-nearest neighbors regression, small neural networks, kernel neural networks, kernel SVM

3.4 How I placed

During the competition, my best error was 0.13950 putting me in 195^{th} out of 329. This model was linear regression trained on cropped but otherwise raw 32x32 images. After the competition, my best error was 0.12859 which would have put me in 145^{th} place. This result was reached using the methods described herein. To put this performance in perspective, the top three winning solutions were between 0.07492 and 0.07869. In order to place in the top 10%, you had to beat 0.09470 and to place in the top 100, an error of 0.11827.

4 Discussion and Conclusions

The Galaxy Zoo competition was, on my analysis, a challenging machine learning task. Here I would like to discuss what I found to be the two most significant sources of difficulty: computation and feature engineer-

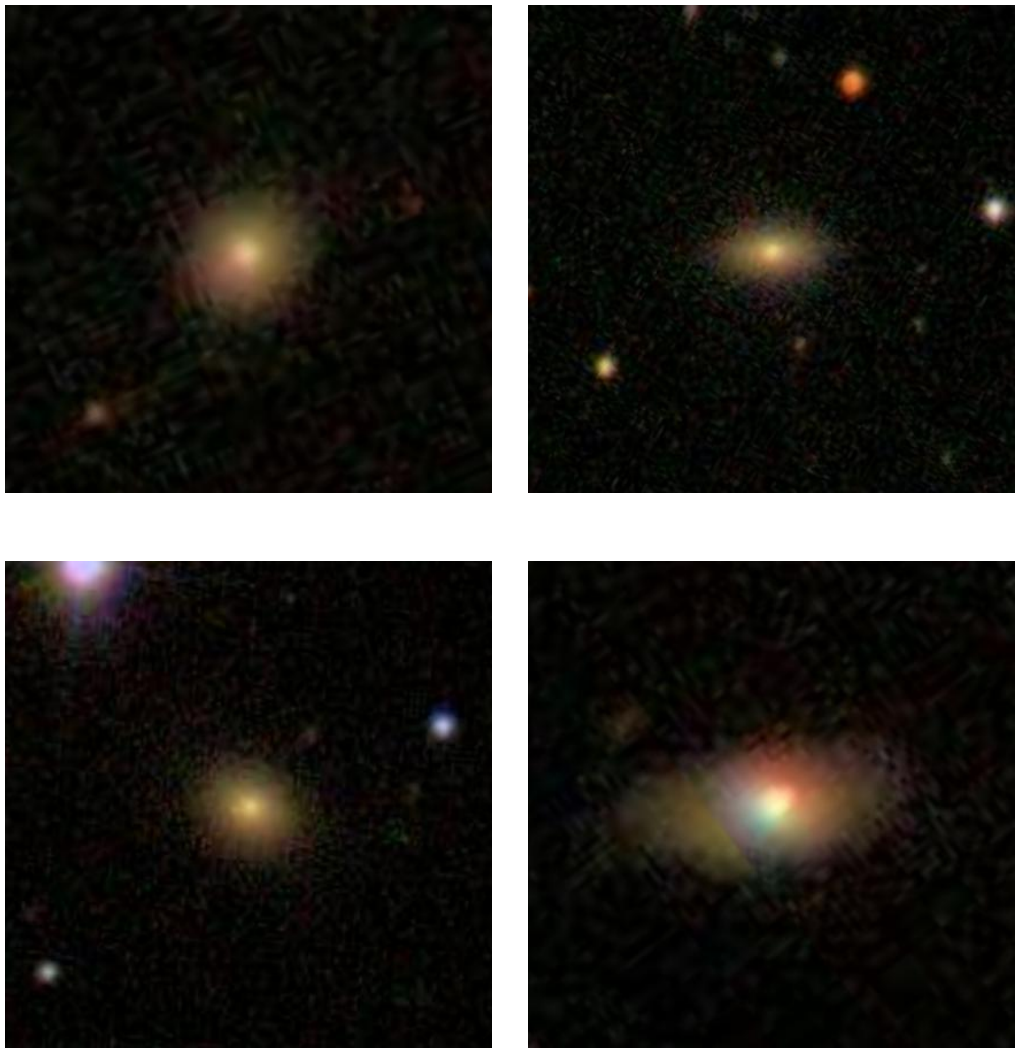

Figure 5: The top 4 galaxies that saw the largest evaluation error in the model. The RMSE validation errors, starting at the top: 0.37696, 0.36980, 0.36511, 0.35606.

ing.

Computing on the Galaxy Zoo dataset was itself a problem. While the dataset wasn't "webscale" large, it was, for example, impossible to fit the entire, unaltered training dataset into memory. Even after reducing the dimensionality of the images, the training and test datasets where still in the 8 gigabyte range. However, this problem wasn't entirely fatal at the data stage, and could be mitigated by processing the images one-by-one.

Fitting models to the dataset, however, was another story. As mentioned previously, many of the best scoring solutions utilized convolutional neural networks (Deep Learning) implemented in massive parallel on GPU's. The requirement of a GPU alone killed any chance I had at competing in the computation game. However, opting for Extra-Trees, a so called "shallow" learning algorithm, did not solve my computational problems.

Extra-Trees, and ensemble methods like it, are inherently parallel tasks as each tree can be trained independently of the others. Each tree, however, can be costly to construct. Given the nature of Extra-Trees, constructing many trees is important for good prediction accuracy. The computational constraints of using a large ensemble of large trees with large feature vectors made it difficult to iterate on my models and run parameter optimization.

Feature engineering was the other major headache. My final model did not include any features that were explicitly extracted from the images, despite my best efforts to include these features. Every feature vector that I created that differed from the raw RGB pixels simply did worse. I know that this is not the case for every participant. On the contrary, many high scoring participants claimed that regularized linear regression, perhaps with the addition of boosting, and well curated features resulted in error rates as low as 0.11.

Why did my attempts at feature engineering fail? I think that I needed to create a robust methodology for testing new features and crafting features by error analysis. During the competition, I attempted to extract features ad-hoc and often tested them in addition to the raw feature vectors. This made it difficult to determine how well these features were performing.

Looking back at the the computational problem, I would take a reasonable subset of the training and validation data and do my model iterations off this subset. I have in mind a sample of about $10,000$ galaxy images. Looking back at the feature engineering problem, I would make feature engineering a priority given my limited computational budget.

My final takeaway for similar problems, i.e. large scale regression using images where accuracy and only accuracy is important, would be to either train a large model with randomization to reduce overfitting such as a convolutional neural network with data preprocessing or to instead use hardcore feature selection if you don't have the computational resources. Overall, I believe that I now have a much more sophisticated understanding of image processing and object recognition, ensemble methods, and convolutional neural networks.

References

[1] Banerji, Manda, et al. "Galaxy Zoo: reproducing galaxy morphologies via machine learning." *Monthly Notices of the Royal Astronomical Society* 406.1 (2010): 342-353.

[2] Willett, Kyle W., et al. "Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey." *Monthly Notices of the Royal Astronomical Society* 435.4 (2013): 2835-2860.

[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." *NIPS*. Vol. 1. No. 2. 2012.

[4] Geurts, Pierre, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees." *Machine learning* 63.1 (2006): 3-42.