# CS63 Spring 2017
# Lab 10: Machine Learning Project
# Bike Sharing Demand

Matt Baer & Matt Parker

May 9, 2017

## 1 Introduction

For this project, we tackled the Bike Sharing Demand challenge from Kaggle.com. The goal of this challenge, given in 2015, was to "forecast bike rental demand in the Capital Bikeshare program in Washington, D.C." The challengers were provided with hourly renting data over the course of 2 years and had to provide an accurate algorithm able to predict the number of bikes at any year given the variables measured in the data set, such as the weather and date/time. We ran a number of regression algorithms to find the one that generated the best predictions.

## 2 Method and Details

This data set was provided by Hadi Fanaee Tork using data from Capital Bikeshare and consisted of the following variables:

- datetime - hourly date + timestamp
- season, notated by
  1. Spring
  2. Summer
  3. Fall
  4. Winter
- holiday - whether the day is considered a holiday
- workingday - whether the day is neither a weekend nor holiday
- weather, where the numbers are notated by:
  1. Clear, Few clouds, Partly cloudy, Partly cloudy
  2. Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  3. Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4. Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

- temp - temperature in Celsius
- atemp - "feels like" temperature in Celsius
- humidity - relative humidity
- windspeed - wind speed
- casual - number of non-registered user rentals initiated
- registered - number of registered user rentals initiated
- count - number of total rentals

This information is given for each item in the training set. Each row in the training set represents the value of these columns for one hour in the two-year recording period. The training data set contains the data from the first 20 days of each month within the data set, while the test data set contains the data from the rest of the month. The training set has both continuous inputs, such as windspeed, temperature, and humidity, and discrete elements, such as weather, working day vs. holiday, and hour at which the bike was rented. In this context, ideally certain conditions correlate with cyclists' demand for rentals. This data set has properties ideal for regression.

**Pre-processing:** For preprocessing, we determined it was necessary to remove the "casual" and "registered" columns from our training set, and we used the "count" column as our Y data set. It was sufficient to allow most of our variables to remain in integer or float form - the format was easy to work with, as all qualifiable variables had already been quantified. From the "datetime" column, we generated another integer variable to use in regression - the hour (from 0 to 23) when the data point was compiled.

**Training:** After preprocessing our data, we generated an 80/20 train/test split, which we then used to train and test several different regressors. Kaggle suggested using a random forest for a benchmark regressor, so we began by using sk-learn's implementation of a random forest. We then used several other regressors to make predictions, and used a short code snippet found on Kaggle to calculate root mean squared logarithmic error (RMSLE), the accuracy metric used to grade this competition. Our goal was to minimize that error.

After a couple test runs, we started to modify the parameters of some regressors, namely, sklearn's Gradient Boosting Regressor and Random Forest Regressor to see if we could improve their performance. We changed the Gradient Boosting Regressor's maximum depth to 7 (the maximum number of features that still had reasonable computation time), and observed increased accuracy.

At this point, we realized that the rules specified by the contest mandated that all predictions for test set members be made using only data obtained before that test point. We modified our code so that each each test point gathered a data set of points before it, and used a regressor trained only on those data points to make our predictions. While this had the effect of vastly increasing computation time and increasing RMSLE, the classifiers still performed similarly to one another. After we had made these modifications, we began testing different regressors in earnest, implementing multiple kNN regressors as well as a Decision Tree Regressor. Furthermore, we implemented an AdaBoost regressor which used our best-performing regressors to that point as base regressors.

| Regressor used | Description |
|---|---|
| **GradientBoostingRegressor** | This regressor is trained by essentially performing gradient descent on test data. A gradient boosting regressor creates a number of arbitrary estimating functions given input, and with each step up to a specified maximum depth, some term is added on to each function to improve its accuracy. The learning rate of this regressor determines how much the most accurate functions affect the rest over each iteration. |
| **DecisionTreeRegressor** | This regressor creates a decision tree, similarly to a decision tree classifier. However, the feature splits are picked in order to minimize a given loss function for members of the data set. If data set members have different values for the same path, the average of their values is taken. |
| **kNN Regression** | kNN regression works similarly to kNN classification. The $k$ nearest neighbors to a test point are found, and their values averaged to give a prediction for the the test point. Similar to kNN classification, the $k$ nearest points can have values weighted by distance. |
| **Random Forest** | A random forest regressor is built by creating a large number of random decision trees, each built with a subset of the training set. The average value of each of those decision trees for a given test point is returned. |
| **AdaBoost with a base regressor** | In AdaBoost, a base regressor is fitted to a given dataset, and subsequent regressors are fitted with the goal of improving poorly predicted values of the previous regressor. This continues for a set number of iterations, at which point the average value is then returned. |

## 3 Results

While we originally used a 60%/40 train/test split, the computational load was impractical for some of our regressors. The Gradient Boosting Regressor gave us the closest results to the random forest regressor, as did the Decision Tree Regressor. They both performed better than unweighted KNN regression. Weighted KNN regression performed slightly better than unweighted, but not significantly. Scikit-learn has an implementation of AdaBoost which can take any given regressor as a base regressor. We used the two best-performing regressors as bases, the Gradient Boosting Regressor and the random forest benchmark. Using these as base regressors for the AdaBoost regressor actually resulted in worse metrics compared to the standard usage of the regressor.

| Algorithm | RMSLE |
|---|---|
| **GradientBoostingRegressor**<br>100 estimators<br>Learning Rate = 0.1<br>Max Depth = 7 | 0.464693140324 |
| **DecisionTreeRegressor**<br>Default Parameters | 0.49733777689 |
| **Weighted kNN**<br>Weighed by distance<br>Training algorithm modified to require $k$ data points, so smaller test set used | 0.939299722866 |
| **Unweighted kNN**<br>Train/Test Split: 80/20<br>same modifications as above | 0.966813405105 |
| **Random Forest**<br>Default Parameters | 0.45199596815 |
| **AdaBoost - GBR Base**<br>Max Depth = 7 | 0.4835489606794 |
| **AdaBoost - Random Forest Base**<br>Default Parameters | 0.485281434633 |

The random forest regressor's top accuracy ranking on our test set would be among the top 25% of entrants in the Kaggle competition, although our results may have varied as we did not submit an entry to the competition using the test data.

## 4 Conclusions

After implementing several different regressors, we found we were unable to beat the mark set by the default Random Forest implementation in SK-Learn with our data set. We can see this as a testament to the usefulness of random forests in regression problems. After implementing the regressors, we used the Gradient Boosting Regressor and Decision Tree Regressor with AdaBoost and found that we were still unable to beat the random forest regressor, and moreover, actually achieved worse metrics than the straightforward implementations of the base regressors.

In future classification experiments, it might be useful to extract more information while pre-processing the data set - in this case, it would be useful to see if more insights could be gleaned from the 'datetime' data point. Nevertheless, the application of multiple regressors here allowed us to gain an understanding of how the randomness of ensemble learning enables it to compete with more easily observable learning algorithms. Other areas for inquiry could include AdaBoost's effects on a range of regression algorithms to find the most improved metrics between a regressor and the AdaBoost regressor using it as a base.