Matthew Burns
William Fedus
Bobak Hashemi
April 17, 2015

CSE 291 Project 1

An application of linear regression to predict housing prices

Problem Definition

We are given a list of m "features" and the sale price for N homes. The features are decimal numbers which characterize physical information about the property. We assume that the features have been picked carefully enough that they contain the necessary information to estimate the value of each home. Under this model, the sale price of the home is

$$s = h(\vec{x}) + \epsilon \tag{1}$$

where $h(\vec{x})$ is the intrinsic value of the property, a function of the chosen features. ϵ is the perturbation to the intrinsic value, which is a function of a broad number of additional variables that are random in nature and can be modeled as Gaussian noise with $\sigma << |h|$. The simplest approximation to $h(\vec{x})$ is a linear function of the form

$$h(\vec{x}) \approx \omega_0 + \omega_1 x_1 + ... + \omega_n x_n \tag{2}$$

In the following sections we will document a set of implemented scripts which find the "best" approximation to $h(\vec{x})$, in Matlab, given that we restrict ourselves to a linear function of the features.

Preprocessing

We shall denote the vector of the weights ω_i as $\vec{\theta}$, and the vector of features for the ith property as $\vec{x}_i$. We construct the $N \times (m+1)$ dimensional matrix, X, using the definition

$$X = \begin{bmatrix} 1 & \vec{x}_1 \\ ... & \\ 1 & \vec{x}_N \end{bmatrix}, \tag{3}$$

so that $h(\vec{x}_i)$ is the ith row of $X\vec{\theta}$. Finally, the sale prices for the ith house y_i are combined in the N dimensional vector $\vec{y}$.

MinFunc Implementation

Minfunc is a Matlab package for optimizing real valued multi-variate functions. The minfunc function call takes in $\vec{\theta}$, $\vec{y}$, X, and a custom function that returns a tuple $[f, g]$, where f is the objective function and g is the gradient of f with respect to $\vec{\theta}$. minfunc returns an optimized $\vec{\theta}$, in ex1a_linreg.m:

```
[alltheta{1}] = minFunc(@linear_regression, theta, options, train.X, train.y);
```

We have named our custom function linear_regression.m in this example.
The next step is to define a simple objective function, one choice is the squared 2-norm:

$$f = \left\| X\vec{\theta} - \vec{y} \right\|_2^2 \tag{4}$$

$$g = \nabla_{\vec{\theta}} f = 2\vec{\theta}^T X^T X - 2\vec{y}^T X \tag{5}$$

These definitions are implemented in linear_regression.m:

```
e=y'-X'*theta;
f=e'*e; % Euclidean norm squared between targets and guess
g=2*(X*X')*theta-2*(y*X')'; % Gradient of the objective
```

Gradient Descent

Gradient descent is a first order linear optimization algorithm which finds a local minima in a function f by stepping in the direction of the negative gradient of the function. In particular, the update rule is given by Equation 6,

$$\vec{\theta_{t+1}} = \vec{\theta_t} - \eta(t)\nabla f \tag{6}$$

where $\vec{\theta}$ is the parameter vector we seek which is a local minima of f and $\eta(t)$ is the step size at iteration t. In a simplest format, η_t may be chosen to be a constant value, that is, the algorithm will take a constant step size at each iteration. However, for step sizes too small, convergence may be excessively slow and for step sizes too large, convergence may be an issue since the algorithm may overstep the equilibrium. In order to solve this issue, we use the backtracking algorithm to reduce η_t as we perform gradient descent.

Closed Form Solution

f is quadratic function of the parameters ω that is always greater than 0. These two facts guarantee the existence of a unique minimum for E with respect to $\vec{\theta}$. We define the optimal $\vec{\theta'}$ as the one which minimizes the expression

We know this objective function f is convex because it is a norm, so any stationary point is guaranteed to be a local minimum. Setting $g = 0$ in (5) and solving for $\vec{\theta}$:

$$\hat{\vec{\theta}} = (X^T X)^{-1} X^T \vec{y} \tag{7}$$

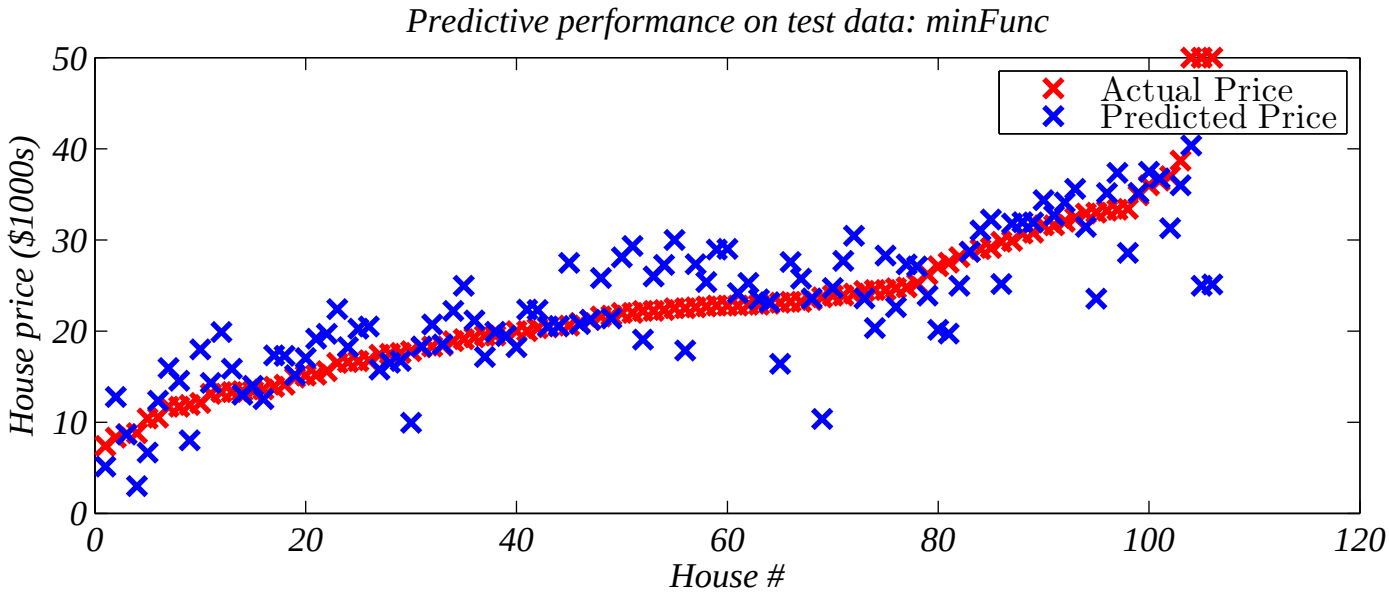
```
function [theta, f, g, exitflag ] = closed_form( X, y )
theta=X*X'\X*y';
end
```

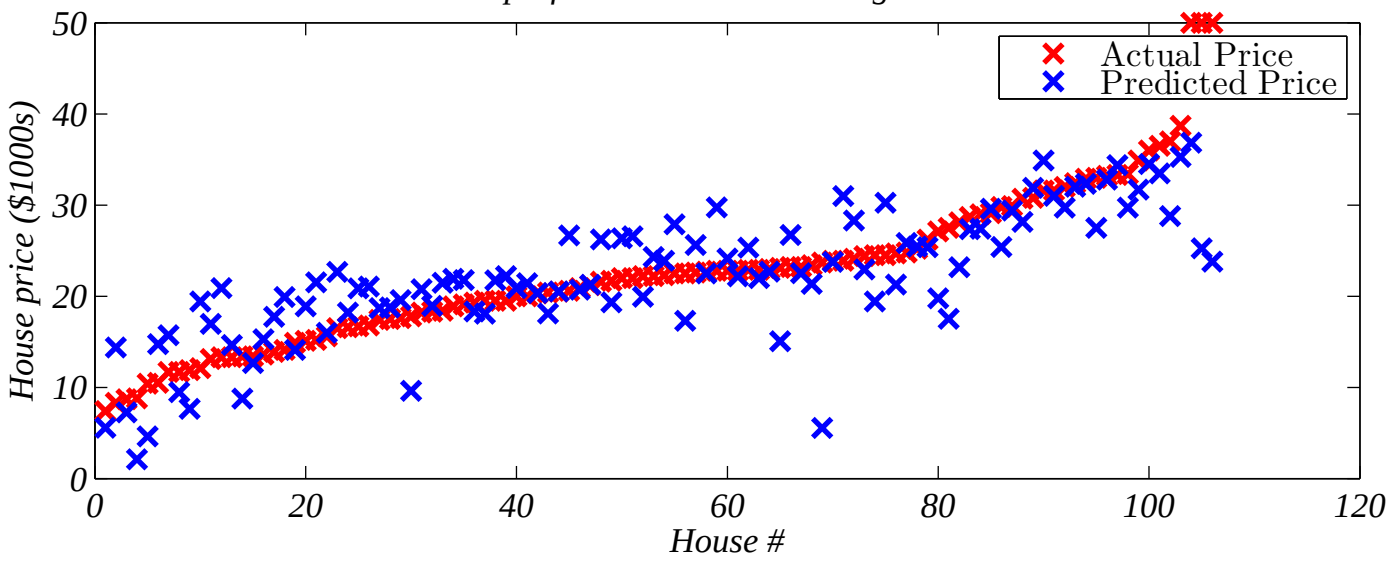
Results

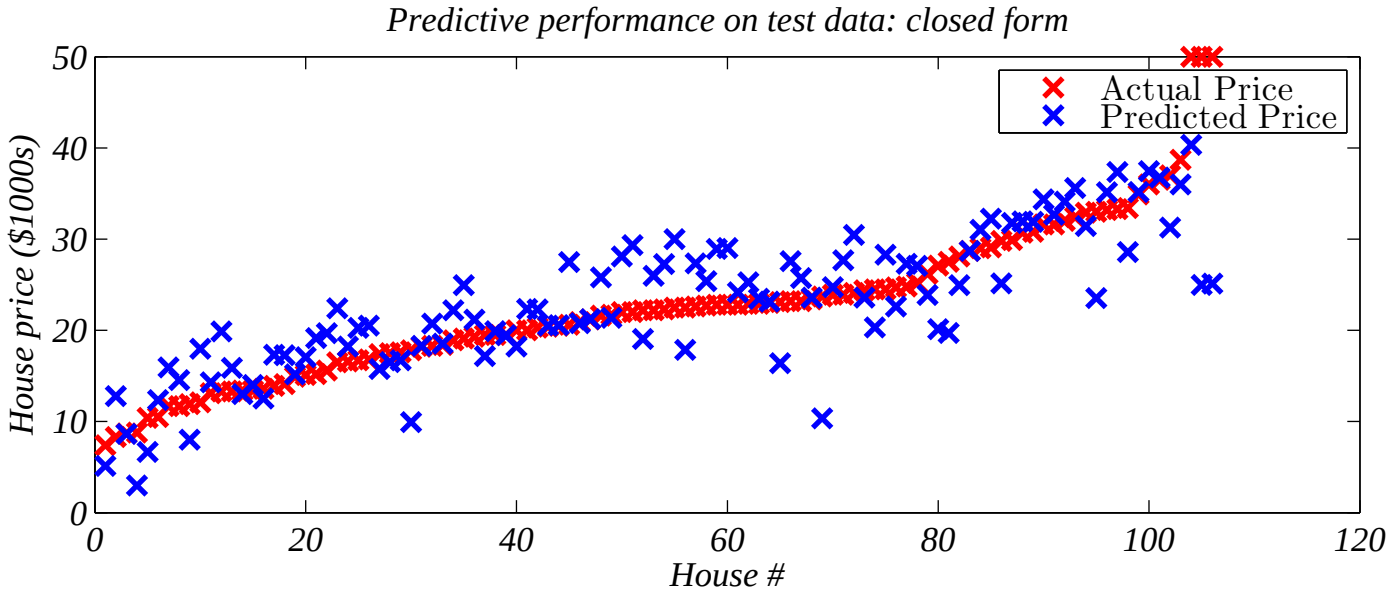
All three methods converged, with the gradient descent taking the most iterations (10,000). We compare the performance of the three methods via Root Mean Squared Error over the partitioned test set. We display the results of the three methodologies in Table 1.

Measure	RMSE
MinFunc	5.03
Gradient Descent	5.81
Closed Form Solution	5.03

Table 1: Performance of our three approaches as measured by RMSE on the partitioned test set.







Who did what?

Matt: Closed form solution, solution and code
William: Gradient descent, solution and code
Bobak: Code manager, LaTeX, theoretical development

References

[1] Matthew Burns, Bobak Hashemi, William Fedus, Computer Code, extending the original work of Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository. URL: *https://github.com/bth5032/CreamCastle_CSE291/tree/master/project%201*

[2] Schmidt, M. (2005). MinFunc. Retrieved April 14, 2015, from http://www.cs.ubc.ca/ schmidtm/Software/minFunc.html