**COMP 455   
Machine Learning Project**

Predicting the amount of Pectin in orange juices

Presented to Gabriel Murray

Prepared by William Gilbert Go

December 3, 2013

**Project Description**

The quality of the orange juice produced by a manufacturer is constantly monitored. There are numerous sensory and chemical components that combine to make the best-tasting orange juice. One manufacturer has developed a quantitative level of the sweetness of orange juice. This means a higher level indicates a sweeter orange juice. From the data gathered during 24 productions that were sampled at a juice-manufacturing plant, each **sweetness level** has a corresponding amount of water-soluble **pectin** (measured in parts per million).

Pectin is a type of soluble fiber contained in various fruits like citrus fruits. It has many beneficial uses such as a stabilizer in fruit juices and has many pharmaceutical applications. The many health benefits of pectin include improvement in digestive health, regulation of bowel movements and prevention of heart disease. However, consuming large amounts of high-fiber products such as pectin may result in certain unpleasant effects on health such as appetite loss, abdominal pains and cramps, diarrhea.

Determining the amount of pectin in the orange juices is important for the health of consumers. The content of this type of fiber in orange juices should be observed. Otherwise, orange juices with very high level of pectin amount can be health-problematic to consumers.

Therefore, the project aims to predict the amount of pectin – a type of soluble fiber present in the orange juices based on the sweetness level develop as the quantitative measure of the sweetness of orange juice. The underlying question on this project is that “**Can the sweetness level of orange juices be used to predict the amount of pectin contained in the juices?**”

**Data Testing and Interpretations**

Because the project aims to predict real-valued output based on the description of the problem and data variables, the problem is classified as regression. By implementing the machine learning method/algorithm I’ve learned throughout the course like linear regression, we should be able to predict the amount of pectin (ppm) in the orange juices indicated by the sweetness level. In this project, I will make use of the Octave codes from the previous lab activities that calculate the *Cost Function* and perform *Gradient Descent* to the cost function.

The cost function ***J(Θ0, Θ1)*** is based on the squared error of all the data points in the training set formulated as *J(Θ0, Θ1) = 1/2m mΣi=1 (hΘ(x(i)) – y(i))2*. For each data point *x(i)* – *the sweetness level*, we calculate the difference between the prediction for that data point *hΘ(x(i))* and its actual label *y(i)*– *the* *pectin amount* and square the difference. All the squared differences of the training examples are summed and then multiplied by *1/2m*.After calculating, we see the initial cost is about 34190, which is a large value that we want to minimize.

The cost function for the linear regression is always a *bowl-shaped function or convex function*, which has a **global minimum**. The idea of Gradient Descent is to lower the cost function *J(Θ0, Θ1)* by slightly changing the ***theta*** parameters iteratively until the cost function converges to the global minimum of the bowl-shaped function. Each *theta* parameter is adjusted according to **α** (*alpha),* the learning rate that affects the step size at each iteration, and a partial derivative of the cost function with respect to the current parameter value.

The following are the files that we need to run and make predictions on Octave:

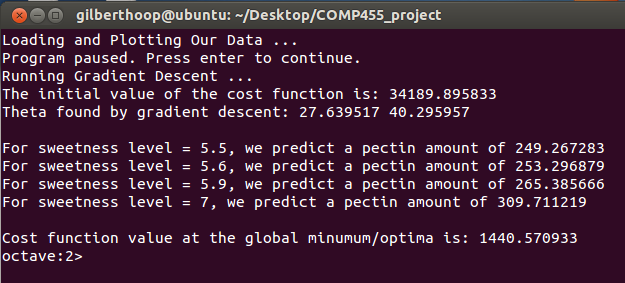
* juiceData.txt – contains the dataset of the sweetness level and pectin amount variables
* plotData.m – plots the data points x and y
* computeCost.m – calculates the cost function for linear regression
* gradientDescent.m – performs Gradient Descent to learn theta parameters
* runLinReg.m – performs linear regression on the dataset and implements plotData, computeCost, and gradientDescent functions

After loading testing the dataset *juiceData.txt* into the function *runLinReg.m* and running the function on Octave, we generated the following values:

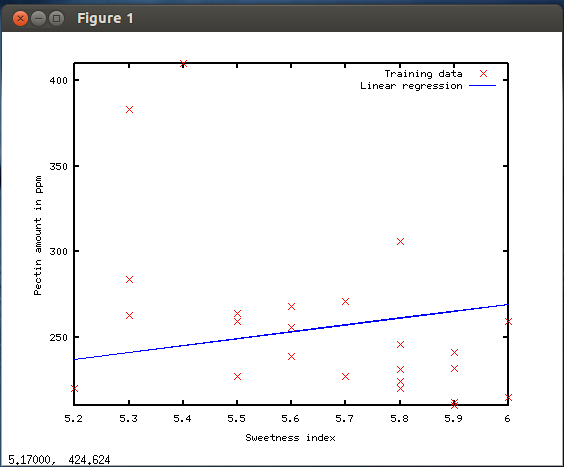
* Cost function = 34190 (initial value)
* HΘ(x) = 27.64 + 40.30 x

Where:

* α (alpha) = 0.01
* Gradient Descent iterations = 1500
* x = sweetness level
* y = pectin amount (parts per million)
* Training samples = 24



The screenshot of data testing on Octave above shows the initial value of the cost function, theta parameters for the linear regression, prediction results, and the final value of the cost function at the global minimum after Gradient Descent.

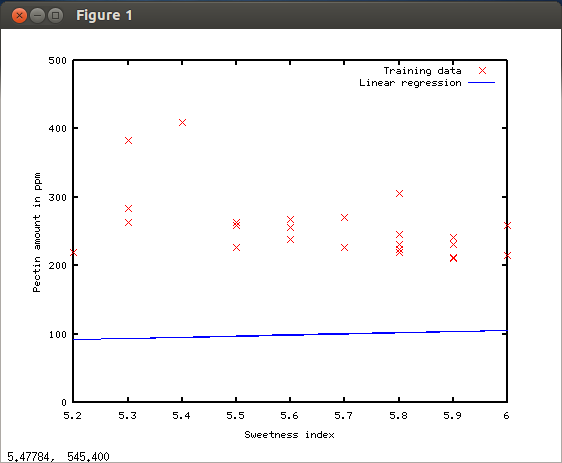


The graph above generated on Octave indicates that our linear regression algorithm learns the model from the training data in a positive slope. This means that the sweetness level of orange juice is proportion to the amount of pectin in it. Let’s make some predictions:   
 *For sweetness level of* ***5.5****, we predict a pectin amount of* ***249.27****.  
 For sweetness level of 5.****6****, we predict a pectin amount of* ***253.30****.  
 For sweetness level of* ***5.9****, we predict a pectin amount of* ***265.40****.  
 For sweetness level of* ***7****, we predict a pectin amount of* ***309.71****.*

**Conclusion**

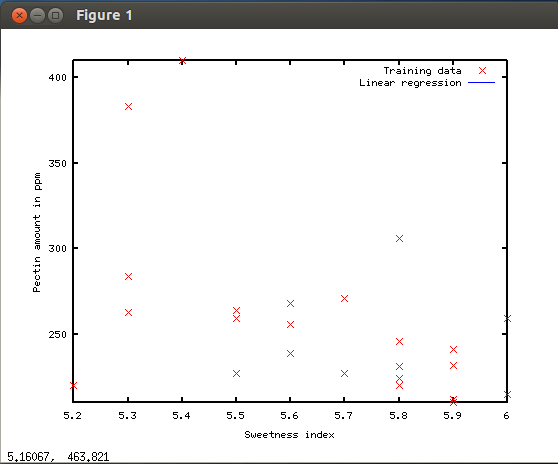
In conclusion, we can use the sweetness level to approximately measure or predict the amount of pectin (ppm) present in orange juices.

**Unexpected Outputs during Data Testing**



α (alpha) = 0.00001 (too small)

When the learning rate or alpha is too small, Gradient Descent takes very small step sizes to reach the global optima or minimum. However, in this graph above, Gradient Descent with very small learning rate failed to reach and converge to the global minimum because it would need more than 1500 iteration to converge.



α (alpha) = 0.1 (too big)

When the learning rate or alpha is too big, Gradient Descent takes very large step sizes to reach the global optima or minimum. The graph above shows that if the learning rate is very big, Gradient Descent can overshoot the global minimum and the learning rate tends to diverge from the global minimum as a result that happened in the graph.