Forecasting Item Price Changes in Runescape using a Neural Network

Victor M. Reyes Espinoza

August 3, 2018

Abstract

Items in the online game Runescape have direct ingredient-product relationships. Price changes in the ingredients may correlate to an increase in the product. Using a neural network, this relationship may be coerced out. Since Runescape has a commission free exchange, the model may be able to make a profit. We explore the model's performance in regards to a specific item: Saradomin Brew.

1 Runescape

1.1 The Game

Runescape is a browser-based Massive Multiplayer Online Role Playing Game (MMORPG). Gameplay mostly revolves around the player defeating monsters and enemies to level up and acquire weapons and/or gold pieces (gp). Some enemies award the player with valuable weapons which the user can sell or use for themselves.

1.2 The Items

Items in Runescape have various uses and have an extremely large range in value. Some of the most valuable items cost 2147 million gp. Most items are used to train secondary, non-combat skills such as potion-making (Herblore). Potions require ingredients, which make direct relationships between multiple items. This distinct relationship between items makes this class of item particularly interesting as price changes in the ingredient may indicate an upcoming price change in the potion. This is explored further in the case study.

1.3 The Grand Exchange

The game contains a central item exchange called the Grand Exchange (GE). At the GE, players can buy and sell items through limit orders at zero commission. A zero-commission exchange is particularly interesting to forecasting because a model need only beat the average slightly in order to be profitable.

2 The Data

2.1 Acquisition

I acquired item price data by writing a parser for the Runescape Wiki's item price history. The parser creates a url to visit, made using the item's name, and downloads the relevant html tag that contains the data.

2.2 Feature Engineering

The raw information is parsed and converted into a single Pandas DataFrame with three columns as shown below:

Date	Item_Name_Price	Item_Name_Trend

The trend column is made differently for different items. For "feature items", items being used as data to forecast with, the trend column is created by the following formula:

$$trend_t = (price_{t-1} - price_t)/price_{t-1}$$

For the "target item", the item whose price change is being forecast, the trend column is:

$$trend_t = (price_{t+1} - price_t)/price_t$$

Effectively, we are trying to forecast the percent change of the target item.

3 The Model

3.1 Neural Network Structure

Our neural network consists of a 3 layers: an input layer, a hidden layer, and an output layer. The input layer has a number of neurons equal to the number of features given to the model. The hidden layer has N neurons given by the equation: TODO

equal to the result of a formula found on the internet for optimum hidden layer neurons. The output layer has only one neuron. The output of this neuron indicates the model's prediction for tomorrow's price change in percentage.

4 Case Study: Saradomin Brew

4.1 Choosing an Item

To test the model, I decided to try to predict the price changes for Saradomin Brew. Saradomin Brew is widely used around Runescape for combat and is a product of training the Herblore skill. It has a price of around eight thousand gold pieces and has a buy limit of 1000 per 24 hours, or 8 million gold pieces per day.

4.2 Features

Saradomin Brew requires multiple items to create: Toadflax herb, a crushed nest, and a vial of water. By looking at the price changes in its ingredients, we hope to be able to predict the potion's next price change. Furthermore, I included mahogany planks, a top 100 most traded item in order to include a general "market movement" element to the model.

4.3 Parameters

The best testing loss of the model was given when epochs was set to 40 and the batch size was set to 32. Using a GPU, the training went by very quickly.

4.4 Testing

In addition to the testing data set, the model was run 5 times a day, averaged out, and an investment strategy was used that assumed the model was completely correct. In my testing, the model achieved an 80% accuracy rate (where accuracy is measured if it correctly predicted the sign of the percent change).