Building a Price Predictor for an Auctioning Website

C. Muthu[1] and M. C. Prakash[2]
[1]Associate Professor, Dept of Statistics, St. Joseph's College, Tiruchirappalli
[2]PG Student, Bharathidasan Institute of Management, Tiruchirappalli

1. Introduction

The Techniques that are related to Big Data Analytics have great impact on the productivity and profitability of big Organizations[1]. The Hadoop ecosystem is now extensively used for successfully implementing the advanced statistical algorithms on the big data [2]. The preferable approach for determining the price predictor for an item is to find a few of the most similar items and assume that the prices will be roughly the same. By finding a set of items similar to the item that interests us, the K-nearest Neighbours algorithm can average their prices and make a guess at what the price should be for this item.

2. Data needed for Study

Shalom InfoTech is at present developing an Auctioning Module for its UK-based Client Sherwood Tinnings. The data needed for predicting the prices of Artifacts auctioned by Sherwood Tinnings through its website were collected from its customers by way of conducting an online survey. The data thus collected were made available by Shalom InfoTech for analyzing through an XML-based API. This API was used to perform item searches and get detailed item information. The XML-based API provided by Sherwood Tinnings website required sending of values of appropriate parameters in XML format for every request. The API returned an XML document that could be parsed with the *parseString* function from the *minidom* library. The Python function *sendRequest* was written to open a connection to the server, post the parameters' XML and parse the result. The function *sendRequest* was added to *Sherwoodpredict.py*. As DOM parsing was found to be a tedious process, a simple convenience method named *getSingleValue* was created, which easily found a node and then returned its contents.

3. Performing a Search

Performing a search involved the creation of the XML parameters for the *GetSearchResults* API call and passing them to the previously defined *sendRequest* function. The XML parameters were in the following form:

```
<GetSearchResultsRequest xmlns = "urn:sherwood:apis :
  sherwoodBaseComponents">
<RequesterCredentials> <sherwoodAuthToken> token
</sherwoodAuthToken> </RequesterCredentials>
<parameter1> value </parameter1>
<parameter2> value </parameter2>
</GetSearchResultsRequest>
```

The following two parameters were passed to the *GetSearchResults* API call:

i) **Query**: This is a string containing the search terms. Using this parameter is exactly like typing in a search from the Sherwood Tinnings home page.

ii) **Category ID**: This is a numerical value specifying the category we wish to search. Sherwood Tinnings has a large hierarchy of categories, which we can request with the *Getcategories* API call. This can be used alone or in combination with Query.

The Python *doSearch* function took these two parameters and performed a search. It then returned a list of the item IDs, which were used with the *GetItem* call, along with their descriptions and current prices. The *doSearch* function was added to *sherwoodpredict.py*.

In order to use the *category* parameter, a function was needed to retrieve the category hierarchy. This was another straight forward API call, but the XML file for all the category data was found to be very large, took a longtime to download, and was very difficult to parse. Because of this reason, the category data were limited to the *Statue* category, for which the price is to be predicted.

The *getCategory* function took a string and a parent ID and returned all the categories containing that string within that top-level category. This function was added to *sherwoodpredict.py*. This function was subsequently used in the following Python session in order to list the *statue* category items:

```
>>> import sherwoodpredict
>>> statues = sherwoodpredict.doSearch ('statue', categoryID = 511480)
>>> statues [0 : 10]
```

4. Getting Details for an Item

The listing in the above search results gave the title and id, and it was possible to extract details such as the metal with which the *statue* was made and the year in which the *statue* was made from the XML text of the title. Sherwood Tinnings website also provided attributes specific to different item types. Each *statue* was listed with attributes like statue's age, its metal and level of craftsmanship. In addition to these details, it was also possible to get details such as the seller's rating, the number of bids, and the starting price.

In order to get the above details, an API call was made to *GetItem*, passing the item's ID as returned by the *doSearch* function. To do this, a function called *getItem* was added to *sherwoodpredict.py*. This function retrieved the item's XML using the *sendRequest* function and then parsed out the interesting data. Since attributes were different for each item, they were all returned in a dictionary. This function was used to display the attributes of the category statues [7][0].

```
>>> reload (sherwoodpredict)
>>> ebeypredict.getItem (statues [7][0]
```

{'attributes' : { u '12' : u '2', u '25710' : u 'India' u '26444' : u '45',
u '26446' : u 'Bronze'}, 'price' : u '515.0', 'bids' : 'u '28', 'feedback' : u '2797',
'title' : u 'Lord Nataraja Statue-India'
}

From the above output, it was known that the attribute 26444 represented statue's age, 26446 represented metal type, 12 represented level of craftsmanship and 25710 represented country of origin. The seller rating, the number of bids, and the starting price, which were provided by the above output, helped us to build the required Price Predictor.

5. Building a Price Predictor

The Python function *makeStatueDataSet* was made to call the *doSearch* function to get a list of statues, and then request each one individually. Using the attributes determined in the previous section, the function created a list of numbers that could be used for prediction, and put data in the structure appropriate for the K-nearest neighbours (KNN) function named *knnestimate*, which would provide the price estimate. The *makeStatueDataset* function was added to *sherwoodpredict.py.*

This function ignored any items that did not have the necessary attributes. Downloading and processing all the results took some time, but an interesting dataset of real prices and attributes was provided. The *knnestimate* function acted on this dataset and provided the price estimate,as shown below:

This function ignored any items that did not have the necessary attributes. Downloading and processing all the results took some time, but an interesting dataset of real prices and attributes was provided. The *knnestimate* function acted on this dataset and provided the price estimate,as shown below:

```
>>> reload (sherwoodpredict)
>>> set! = sherwoodbaypredict.makeStatueDataset()
>>> numpredict.knnestimate (set1, (40, bronze, 2, India))
$ 667.89 9999 9999 9998
```

References

1. Jacques Bughin (2016). *Big Data, Big Bang?*, *Journal of Big Data*, 3(2), pp. 1-14.
2. Muthu, C. and Prakash, M. C. (2015). *Impact of Hadoop Ecosystem on Big Data Analytics*, *International Journal of Exclusive Management Research - Special Issue(2015)*, pp. 88 - 90.
3. Wes Mckinney (2012). *Python for Data Analysis*. O'Reilly Press, USA.