Living Costs and Food Survey: Can Big Data help reduce respondent burden and make efficiency savings?

Theodore Manassis, Alessandra Sozzi, Iva Spakulova, Leone Wardman

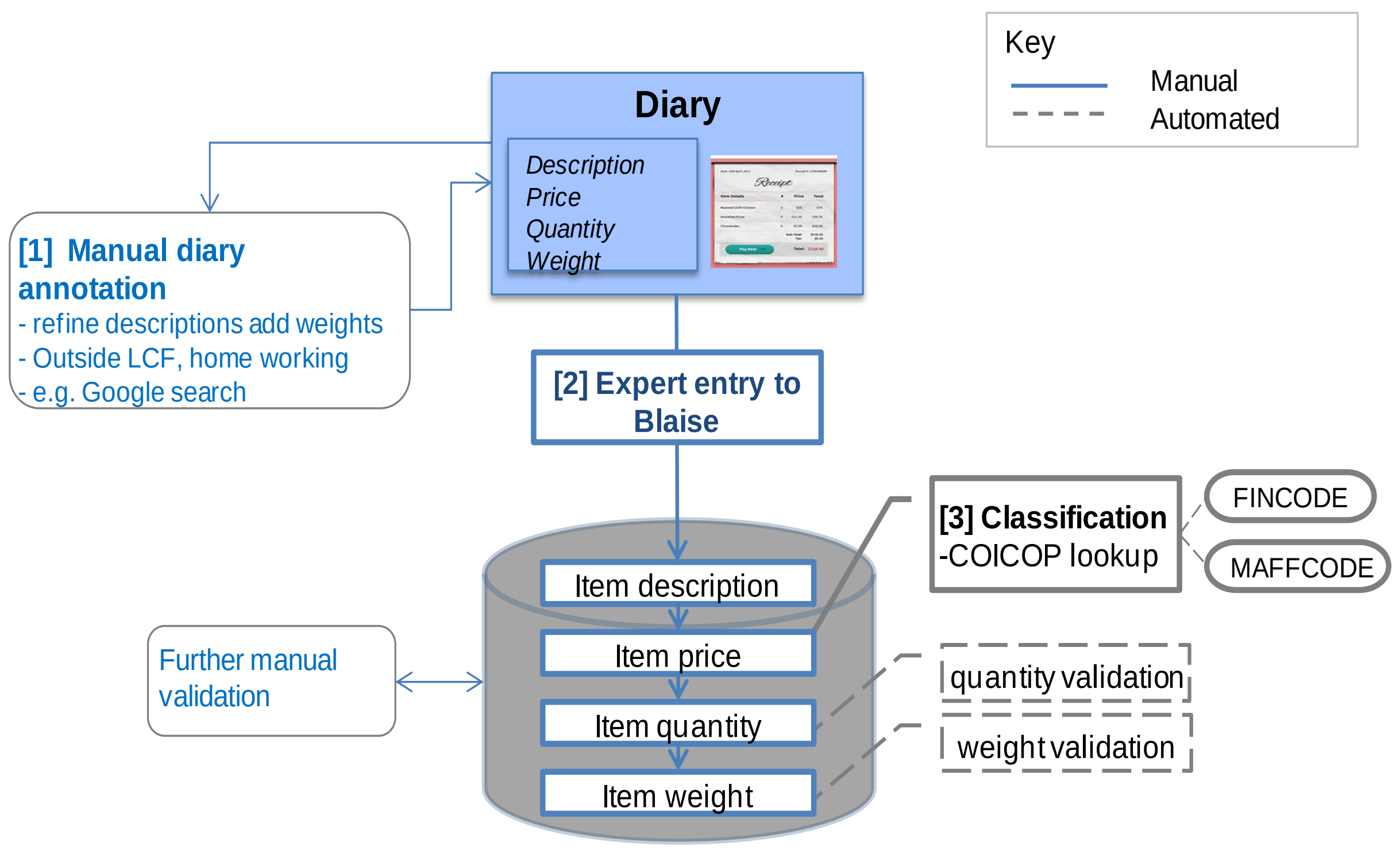
Big Data Team

Overview of diary entry process

The Figure on the right shows a flow diagram of the current process. LCF data are collected in the form of diaries and then entered into **Blaise**, a software package designed for survey data collection and processing.

The "Manual diary annotation" step happens outside of the LCF team and aims to complete the data by manually ascertaining and hand-writing missing information. Nevertheless, further manual checks are often required when coded into Blaise.

A significant proportion of coding effort is spent filling in **missing information**, most commonly the product amount/weight. Coders manually look to external sources, such as internet searches, to find the missing information for a given product description.



Improving LCF diary entry process

The most resource intensive parts of the process, where automation would be most beneficial, are:

- Classification of products by **COICOP categories** (Classification of Individual Consumption According to Purpose)
- Estimation of the weights for items that are **missing amount/weight information** on the receipt or diary at the point of data entry

The COICOP classification or weight/amount information can be provided based on:

- **Previously validated LCF data**
- Web scraped or purchased **commercial data** from major supermarkets

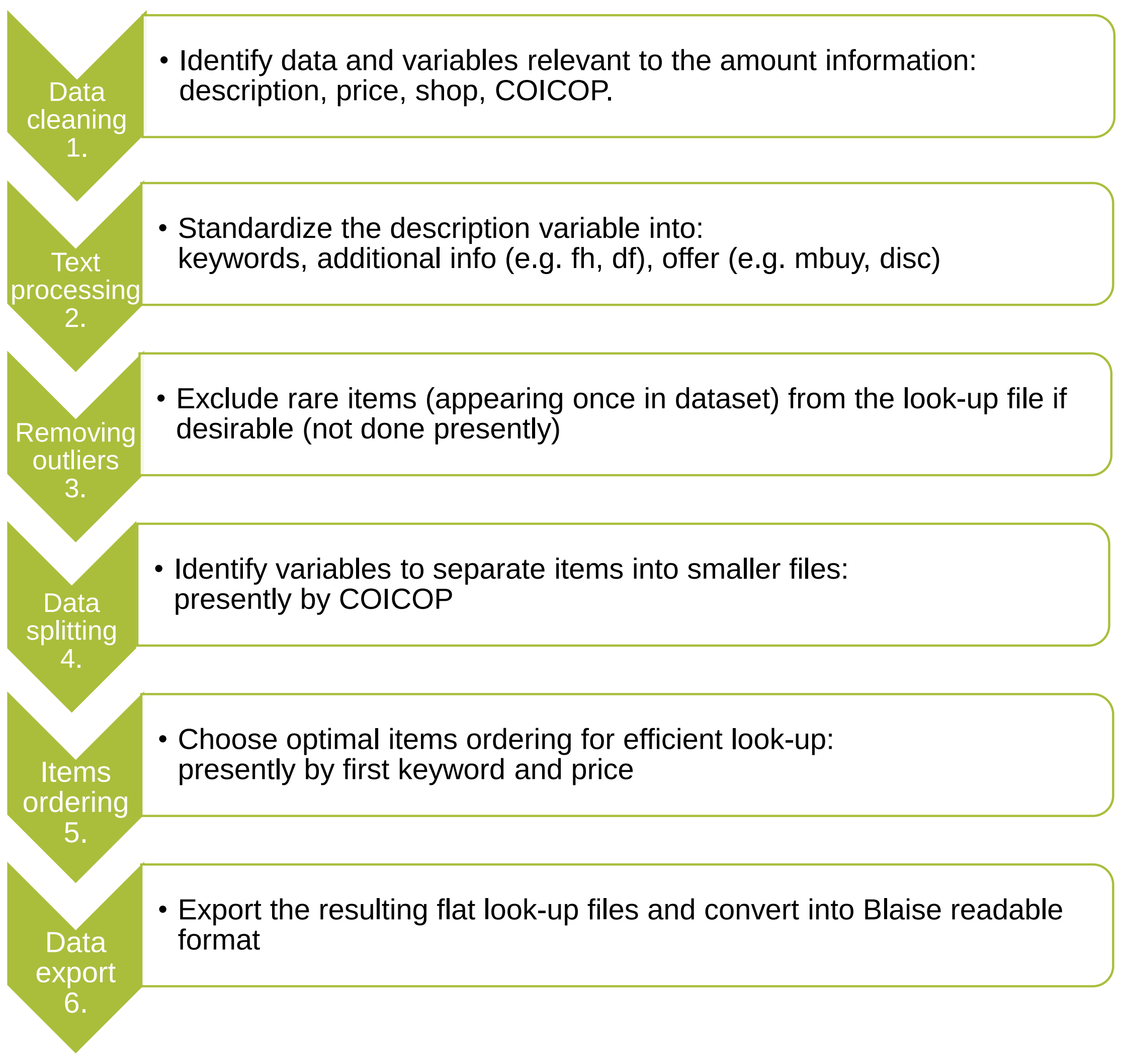
Ultimately, any solution needs to be compatible with Blaise and the secure environment within which LCF data are kept. Therefore, we explored several different options along side each other, and here we present three approaches using previously validated LCF data.

Three streams of work

Flat look-up files

When a coder encounters a product which is the same as another, that has previously been recorded, such a product should be available as a 'donor' for the amount/weight information.

Within Blaise, the coder would be presented with a short list of identical or similar previously entered items. The list needs to be populated with items which are the closest match according to previously entered variables: shop, price, description, COICOP (if available). As per our knowledge, Blaise, the currently used system, takes data for the drop-down list from a fixed look-up file stored locally. Creating such a list in runtime would be preferable (see Solr option). However, due to security restrictions, for the time being, we created flat look-up files, based on historic data , via the following data processing pipeline:



Data cleaning 1.
- Identify data and variables relevant to the amount information: description, price, shop, COICOP.

Text processing 2.
- Standardize the description variable into: keywords, additional info (e.g. fh, df), offer (e.g. mbuy, disc)

Removing outliers 3.
- Exclude rare items (appearing once in dataset) from the look-up file if desirable (not done presently)

Data splitting 4.
- Identify variables to separate items into smaller files: presently by COICOP

Items ordering 5.
- Choose optimal items ordering for efficient look-up: presently by first keyword and price

Data export 6.
- Export the resulting flat look-up files and convert into Blaise readable format

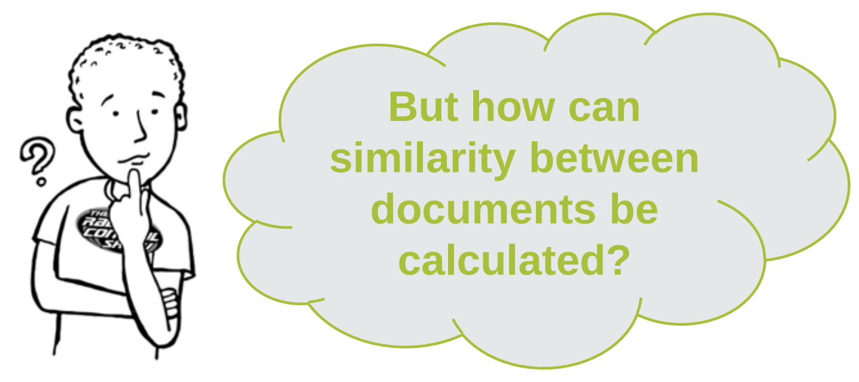
How well the flat look-up tables serve their purpose can be evaluated only by the users, but as an indication:

- For **82%** of items, the correct amount of the product is available in the look-up file within proximity of the cursor (within 10 lines).
- For **42%** of items, the correct amount of the product equals to the value which appears most frequently in the proximity of the cursor in the look-up file.

Solr/Lucene Technology

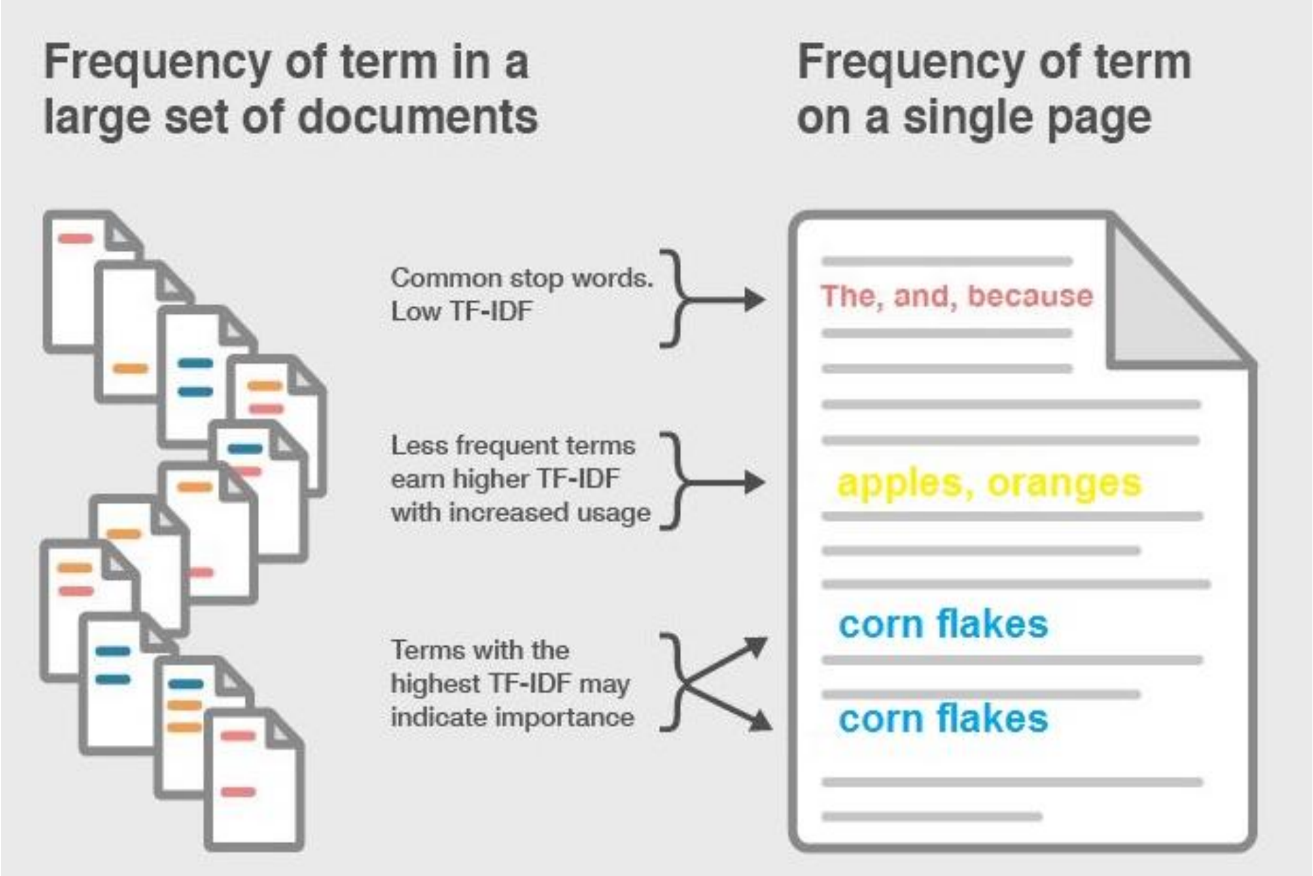
Solr is an open source search server with a web based interface (a bit like an internal Google Search Engine).

Stored records/documents are "indexed" so that they can be retrieved quickly based on requested criteria. You are then able to run a query to receive back a ranked result of documents, generating an accompanying score for each document, calculated based on the document's similarity to the search query.



Underneath, SOLR uses a modified TF-IDF method to calculate the similarity score, and queries can be run against all available historical LCF data. It also has the capability to do fuzzy searches.

Term frequency-inverse document frequency (TF-IDF) measures the importance of a keyword phrase by comparing it to the frequency of term in a large set of documents.


Frequency of term in a large set of documents
Frequency of term on a single page
Common stop words. Low TF-IDF
Less frequent terms earn higher TF-IDF with increased usage
Terms with the highest TF-IDF may indicate importance
The, and, because
apples, oranges
corn flakes
corn flakes

Using the Solr system, we were able to achieve an overall **98%** accuracy in the COICOP classification.
The most powerful aspect of this technology is that it is **scalable** i.e. the response time will remain very short even when the amount of documents, against which each query is run, is very large.

COICOP classification

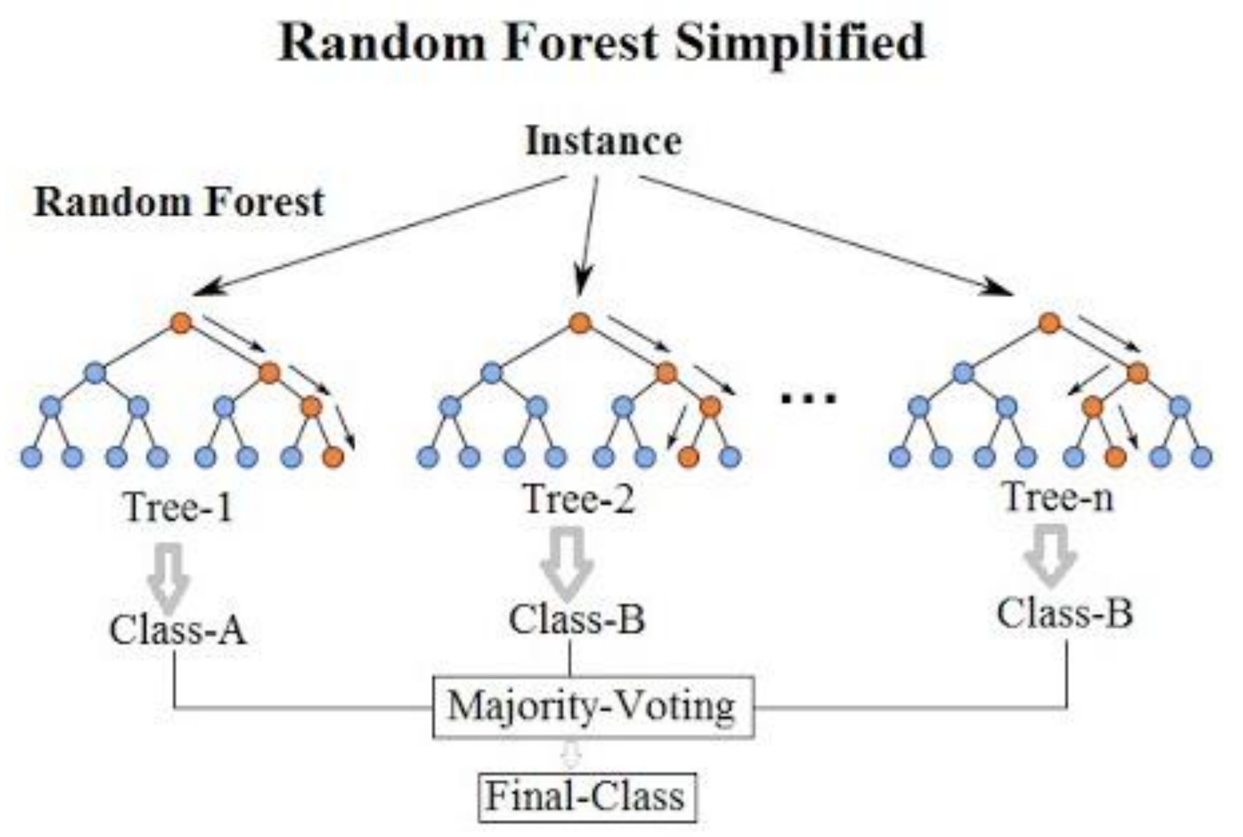
For COICOP classification, we also explored the possibility of using additional data science techniques, where the models used are trained based on historical LCF data, and then incorporated in the system to automatically estimate the missing value.

A classifier takes some input x, for example a textual product description. It then runs it through a model to output some value y, in this case the COICOP code that we are trying to predict.

We tested three different type of classifiers:

- Naïve Bayes classifier
- Support Vector Machine
- Random Forests.

The Random Forests classifier achieved similar accuracy as the Solr system. In the training part of the algorithm, multiple decision trees are created (based on the training data) and the resulting classification of a new data (or instance) is obtained based on a majority-vote.



Overall, implementing a Random Forest algorithm achieves an accuracy of **98%**.
While this is a promising result, the challenge of this approach is maintaining an updated model whilst inputting new data.

Future directions

All three options will be assessed together with the Social Survey Division to identify the most suitable solution for long term improvements in coding time and efficiency. Other related work regards using what we have built so far based on historical LCF data to repeat the same classification task, such the COICOP classification, for data coming from different sources, such as web scraped data. On the other hand, web scraped data can help LCF coders for information such as the weight, commonly found in online product descriptions. We have found that several projects here in ONS have similar aspects to what we have tried to solve. We are in touch with other business areas to see whether this can be implemented looking at a wider ONS. If you feel you are one of those, please let us know.

REFERENCES

❖Manning et al. (2008). *Introduction to information retrieval*

❖ Pedregosa et al. (2011). *Scikit-learn: Machine learning* in Python