CS838 DATA SCIENCE PROJECT

Data Merging

Tuan Dinh
dinh5@wisc.edu

Sam Gelman
sgelman2@wisc.edu

Varun Sah
varun.sah@wisc.edu

April 16, 2017

CONTENTS

1 Data

The objective was to match electronics products belonging to several categories listed on Amazon and Newegg.

- From Amazon, we had data of 17518 products.
- From Newegg, we had data of 4160 products.

2 Original Schema

Data from the two sources was cleaned and standardized to have the same schema:

- ID: Product's ASIN on Amazon and RID, NID on Newegg
- NAME: Product's title on both Amazon and Newegg
- CATEGORY: Product's bnode on Amazon and category on Newegg (both mapped to standardized strings specified in category_maps.csv)
- PRICE: Product's price on both Amazon and Newegg
- NUM_REVIEWS: Number of reviews on the product's pages on both Amazon and Newegg
- BRAND: Product's brand on both Amazon and Newegg
- INFO: Product's specifications on Amazon and model number on Newegg

3 Blocking and Matching

We ran the SVM based matcher from the entity matching stage on the 4938 tuple pairs within blocked_pairs.csv. The matcher classified the 4938 tuple pairs as a match (1) or not (0) and stored all the classifications in predicted.csv. Only positive classifications (matches) of tuple pairs were stored in matched_tuples.csv

4 Merging

4.1 Combining Strategy

We combined attributes NAME and PRICE using custom combination rules. Attributes CATEGORY and BRAND were simply taken from the Amazon table since their values are always same for a matched tuple pair. Attributes ID, INFO and NUM_REVIEWS were not merged. A detailed explanation of these combining strategies is provided in the following subsections.

4.1.1 NAME

Our goal with the NAME attribute was to preserve as much information as possible. Rather than selecting one of the names to use in the final schema, we combined the two names in the following way. We selected the longer name and added any tokens from the shorter name that were not already present in the longer name. For example, let's say the Amazon name was "Microsoft Laser Gaming Mouse SKU-123" and the Newegg name was "Microsoft Gaming Mouse 600 DPI SKU-123". Our combination function takes the longer of the two names as the base. In this case, the Newegg name is longer so it is the base. Next, we tokenize the Amazon name into ["Microsoft", "Laser", "Gaming", "Mouse", "SKU-123"]. We see that of these tokens, "Laser" is not present in the Newegg name. This token is appended to the Newegg name to produce the final combined name of "Microsoft Gaming Mouse 600 DPI SKU-123 | Laser".

4.1.2 PRICE

If one of the prices was missing, we used the other. If both prices were available, we took the average.

4.1.3 CATEGORY

We combined attribute CATEGORY based on the fact that for two tuples to match, they must have equivalent values for the CATEGORY attribute (blocking criteria). Thus we simply picked the Amazon value for the CATEGORY attribute, which is the same as the Newegg value for these attributes.

4.1.4 BRAND

We combined attribute BRAND based on the fact that for two tuples to match, they must have equivalent values for the BRAND attribute (blocking criteria). Thus we simply picked the Amazon value for the BRAND attribute, which is the same as the Newegg value for these attributes.

4.1.5 ID

For the ID attribute, we retained both Newegg and Amazon values by creating new attributes ASIN (from Amazon table), NID (from Newegg table), and RID (from Newegg table). We did this because merging the ID fields would make no sense. Also, all three ID fields would be required in the data analysis stage.

4.1.6 INFO

For the INFO attribute, we kept both Newegg and Amazon values by creating new attributes AMAZON_INFO and NEWEGG_INFO. We decided not to merge the INFO fields of the source tables because they contain very different information.

4.1.7 NUM_REVIEWS

For the NUM_REVIEWS attribute, we kept both Newegg and Amazon values by creating new attributes AMAZON_NUM_REVIEWS and NEWEGG_NUM_REVIEWS. We decided not to merge the NUM_REVIEWS field because it made no physical sense to merge the values from the two sources. Also, both the NUM_REVIEWS fields would be required in the data analysis stage.

4.2 Issues

We needed to remove commas from the price values in order to convert them to floating point numbers and take the average. Everything else went smoothly.

4.3 Additional Data

We used the two product tables from the previous stages (as described in sections 1 to 3) without adding any additional tables.

5 Combined Table Statistics

5.1 Schema

The schema for the combined table includes the following attributes:

- ASIN
- NID
- RID
- NAME
- PRICE
- CATEGORY
- BRAND
- AMAZON_INFO
- NEWEGG_INFO
- AMAZON_NUM_REVIEWS
- NEWEGG_NUM_REVIEWS

5.2 Size

There are 971 tuples in the combined table.

ASIN	*NID*	*RID*	*NAME*	*PRICE*	*CATE-GORY*	*BRAND*	*AMA-ZON_INFO*	*NEWEGG INFO*	*AMA-ZON NUM REV-IEWS*	*NEW-EGG NUM REV-IEWS*
B00005ARK3	9SIA0AJ0NA7913	33-124-002	linksys befw11s4 wireless router ieee 802.3/3u ieee 802.11b | cisco-linksys wireless-b cable/dsl	49.98	Wire-less Routers	linksys	8.2 x 7.2 x 2.1 inches 2.5 pounds 2.5 pounds b00005ark3 befw11s4 3.4 out of 5 stars may 2 2006 none none	befw11s4	1108	69
B00080DSEM	9SIA1N82853897	26-105-166	microsoft compact optical mouse 500 - black	11.59	Mice	mi-crosoft	3.4 x 1.9 x 1 inches 4 ounces 4 ounces b00080dsem u81-00009 4.1 out of 5 stars june 6 2005 none none	u81-00009	83	26

ASIN	*NID*	*RID*	*NAME*	*PRICE*	*CATE-GORY*	*BRAND*	*AMAZON_INFO*	*NEWEGG INFO*	*AMA-ZON NUM REVI-EWS*	*NEW-EGG NUM REV-IEWS*
B004RI5EHA	9SIA1UH50Y2009	17-182-023	rosewill rv350-2 350w atx 12v v2.2 power supply | 350-watt atx12v	27.27	Power Sup-plies	rosewill	b004ri5eha 3.8 out of 5 stars 7.3 pounds item can be shipped within u.s. this item is not eligible for international shipping. december 17 2010 rosewill rv350-2 3.5 pounds 5.5 x 5.9 x 3.4 inches 5.51 x 5.91 x 3.35 inches silver 220 volts none	rv350-2	74	142
B00603QXPM	9SIAAEE5B48720	19-116-492	intel core i7-3930k sandy bridge-e 6-core 3.2ghz 3.8ghz turbo lga 2011 130w bx80619i73930k desktop processor | hexa-core 3.2 ghz 12 mb cache -	507.82	CPUs	intel	4.9 x 2 x 1.8 inches 3.2 ounces 4.8 ounces item can be shipped within u.s. this item is not eligible for international shipping. b00603qxpm bx80619i73930k 4.5 out of 5 stars september 10 2011 none none	bx80619-i73930k	135	356

6 PYTHON CODE

```
""" combines data into a single set """

import utils
import csv

# filename constants
AMAZON_PRODUCTS_FN = "data/amazon_products.csv"
NEWEGG_PRODUCTS_FN = "data/newegg_products.csv"
MATCHES_FN = "data/matched_tuples.csv"
OUTPUT_FN = "output/combined_products.csv"


def product_dictionary(products):
    """ creates a product dictionary key'd on the first item """

    product_dict = {}
    for product in products:
        product_dict[product[0]] = product

    return product_dict


def combine_name(amazon_name, newegg_name):
    """ combines the names """

    # select the shorter and longer names
    if len(amazon_name) >= len(newegg_name):
        longer = amazon_name
        shorter = newegg_name
    else:
        longer = newegg_name
        shorter = amazon_name

    # split each name into tokens
    longer_tokens = longer.split()
    shorter_tokens = shorter.split()

    # check which parts of the shorter name need to be appended to the longer name
    need_appending = []
    for shorter_token in shorter_tokens:
        if shorter_token not in longer_tokens:
```

```
            need_appending.append(shorter_token)

    if need_appending:
        # append the shorter tokens to the longer name
        longer += "␣|"
        for token in need_appending:
            longer += "␣{}".format(token)

    return longer


def combine_price(amazon_price, newegg_price):
    """ combines the prices """

    # cases where one or both of the prices is missing
    if amazon_price == "":
        return newegg_price
    elif newegg_price == "":
        return amazon_price
    else:
        newegg_price = float(newegg_price)
        amazon_price = float(amazon_price)
        avg = round((newegg_price + amazon_price) / 2, 2)

        # output expecting a string
        avg = str(avg)
        return avg


def combine_products(amazon_product_dict, newegg_product_dict, matches):
    """ combines all the matching newegg and amazon tuples """

    combined_products = []

    # loop through match and create a new row for the combined table
    for asin, nid in matches:

        # place each of the output data in its own variable for readability
        rid = newegg_product_dict[nid][1]
        name = combine_name(amazon_product_dict[asin][1], newegg_product_dict[nid][2])
        price = combine_price(amazon_product_dict[asin][3], newegg_product_dict[nid][4])
        category = amazon_product_dict[asin][2]
        brand = amazon_product_dict[asin][5]
        amazon_info = amazon_product_dict[asin][6]
```

```
            newegg_info = newegg_product_dict[nid][7]
            amazon_num_reviews = amazon_product_dict[asin][4]
            newegg_num_reviews = newegg_product_dict[nid][5]

            # put into a vector along with asin and nid
            combined_product = [asin, nid, rid, name, price, category, brand, amazon_info,
                                newegg_info, amazon_num_reviews, newegg_num_reviews]

            # add to list
            combined_products.append(combined_product)

    return combined_products


def main():

    # amazon header (for reference)
    # ASIN, NAME, CATEGORY, PRICE, NUM_REVIEWS, BRAND, INFO

    # newegg header (for reference)
    # NID, RID, NAME, CATEGORY, PRICE, NUM_REVIEWS, BRAND, INFO

    # load amazon products
    amazon_products = utils.load_csv(AMAZON_PRODUCTS_FN, skip_first=True)
    amazon_product_dict = product_dictionary(amazon_products)

    # load newegg products
    newegg_products = utils.load_csv(NEWEGG_PRODUCTS_FN, skip_first=True)
    newegg_product_dict = product_dictionary(newegg_products)

    # load matches
    matches = utils.load_csv(MATCHES_FN, skip_first=True)

    # combine the products
    combined_products = combine_products(amazon_product_dict, newegg_product_dict,
    matches)

    # output schema header
    header = ["ASIN", "NID", "RID", "NAME", "PRICE", "CATEGORY", "BRAND",
              "AMAZON_INFO", "NEWEGG_INFO", "AMAZON_NUM_REVIEWS",
              "NEWEGG_NUM_REVIEWS"]

    # save to a csv file
    with open(OUTPUT_FN, "w") as out_f:
```

```
        writer = csv.writer(out_f, delimiter=",")
        # write the header row
        writer.writerow(header)
        # write the products
        for product in combined_products:
            writer.writerow(product)


if __name__ == "__main__":
    main()
```