Matching Unstructured Product Offers to Structured Product Specifications

Anitha Kannan
Search Labs
Microsoft Research
ankannan@microsoft.com

Inmar E. Givoni*
PSI Group
University of Toronto
inmar@psi.toronto.edu

Rakesh Agrawal
Search Labs
Microsoft Research
rakesha@microsoft.com

Ariel Fuxman
Search Labs
Microsoft Research
arielf@microsoft.com

ABSTRACT

An e-commerce catalog typically comprises of specifications for millions of products. The search engine receives millions of sales offers from thousands of independent merchants that must be matched to the right products. We describe the challenges that a system for matching unstructured offers to structured product descriptions must address, drawing upon our experience from building such a system for Bing Shopping. The heart of our system is a data-driven component that learns the matching function off-line, which is then applied at run-time for matching offers to products. We provide the design of this and other critical components of the system as well as the details of the extensive experiments we performed to assess the readiness of the system. This system is currently deployed in an experimental Commerce Search Engine and is used to match all the offers received by Bing Shopping to the Bing product catalog.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Management Database applications Subjects: Data mining

General Terms

Experimentation, Measurement, Performance

Keywords

Matching, Structured data, Unstructured data, Commerce search

*Work done while author was an intern at Search Labs

KDD '11 August 21-24 2011, San Diego, California, USA.

1. INTRODUCTION

With the increasing widespread use of the Internet, there has been tremendous growth in the amount of commerce conducted over the web. A recent ComScore study reports record-breaking $43.4 billion in Q4 2010 U.S. retail e-commerce spending, which is up 11% from the previous year [1]. Nearly seven out of ten consumers say that the Internet has become important in providing them with information to help make buying decisions. More than 70% of consumers are likely to shop online before making an offline purchase [14].

A comprehensive product catalog is a prerequisite for the effectiveness of an e-commerce search service. Such a catalog at web-scale contains information about every product as well as sales offers from various merchants. For instance, the Bing Shopping catalog (shopping.bing.com) has information on more than five million products and more than ten million offers from upwards of tens of thousands of merchants. The product information consists of various attributes and their corresponding values, stored in a structured record comprised of attribute ⟨name, value⟩ pairs. Many products do not have universal unique identifiers. The product information is obtained from multiple product aggregators (*e.g.,* CNET, PriceGrabber), each providing only partial, but different, information. Information from different aggregators is combined to arrive at the specification for a product. However, the catalog can still have multiple data records corresponding to the same product, each somewhat different from the other.

Similarly, offers come from multiple merchants (*e.g.,* buy.com, gadgettown.com). Generally, there is very little structure in the offers. Typically, an offer consists of a concise textual description of the product for which the offer is being made. Embedded in the description are some attribute values and sometimes attribute names along with other terms, which the merchant presumes might be sufficient for the offer to be matched to the intended product. The text of the description is often not grammatically well-formed. Different merchants often use different names for the same attribute. Many offers have no identifier that could be used for matching the offer to the corresponding product. The matching is currently done using rules written manually – a costly and brittle process. Consequently, many offers are matched incorrectly and millions of offers go unmatched.

When a consumer searches for a product, the search en-

Attribute Name	Attribute Value
category	digital camera
brand	Panasonic
product line	Panasonic Lumix
model	DMC-FX07
sensor resolution	7 megapixel
color	silver
weight	132 g
width	9.4 cm
height	5.1 cm
depth	2.4 cm
display: type	LCD display
display: technology	TFT active matrix
display: diagonal size	2.5 in
audio input type	none
flash memory: form factor	memory stick
flash memory: storage capacity	8 MB
video input: still image format	JPEG
video input: digital video format	MPEG-1
lens system: optical zoom	3.6
...	

Unstructured Text (Offer-1)

Panasonic Lumix DMC-FX07 digital camera [7.2 megapixel, 2.5", 3.6x optical zoom, LCD monitor]

Unstructured Text (Offer-2)

Panasonic DMC-FX07EB digital camera silver

Unstructured Text (Offer-3)

Lumix FX07EB-S, 7.2 MP

Figure 1: Structured product record for 'Panasonic DMC-FX07 digital camera' and textual descriptions from three matching offers.

gine also shows offers paired with the product. Thus, the ability to correctly match offers to their corresponding products is of paramount importance for the success of the search engine.

Fig. 1 shows part of the structured record for a Panasonic DMC-FX07 digital camera as well as three merchant offers for this product as they appear in the Bing Shopping catalog. We observe the following:

- While Offer-1 is the most detailed one, it still contains only a small part of the information in the structured record. The phrase 'Panasonic Lumix' indicates both the brand (Panasonic) as well as the product line (Panasonic Lumix). Some of the attribute values only match approximately (7.2 megapixel *vs.* 7 megapixel, LCD monitor vs. LCD display). The only attribute name present in the offer is optical zoom (called 'lens system: optical zoom' in the structured record), with corresponding values 3.6x and 3.6, respectively.

- Information provided in Offer-2 is largely a subset of what is provided in Offer-1. This offer provides the values of the category and brand, but the value of the model has an extra suffix. It additionally provides the value of the color attribute.

- Offer-3 is even more interesting. It provides part of the value of the product line (Lumix) and a somewhat different value for the sensor resolution (7.2 MP *vs.* 7 megapixel) as well as for the model (FX07EB-S *vs.* DMC-FX07). It neither provides the category nor the brand information.

- With respect to Offer-3, note further that Panasonic also makes other 7.2 megapixel Lumix digital cameras (*e.g.,* DMC-TZ3K, DMC-LZ6, and DMC-FX12). Moreover, there is also a field controller product with model number FX07.

Clearly, we have on our hands a hard problem of matching free-text descriptions of products to their structured records for which it is desirable to have an algorithmic solution. The matching system must have high precision. Our enquiry revealed that for the system to be deployable, the precision must be at least 85%. Since a large number of offers are received on a daily basis and only a fraction of them can be shown, lower recall is not as detrimental as lower precision. In this paper, we present the design and performance of an experimental system that satisfies this desiderata.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 gives the design of our system. Section 4 describes the datasets and metrics we used for evaluating the performance of the system. We also present experimental results in this section. We conclude with a summary and directions for future work in Section 5.

2. RELATED WORK

The problem of matching records has been studied under various topics including record linkage [13, 21, 23, 27], duplicate detection [12, 24], entity resolution [2, 3, 26], and merge/purge [15]. While our solution continues this rich lineage of work, there are distinguishing traits in our setting that call for fresh approaches and techniques. For instance, while the work of Newcombe [21] (later formalized by Fellegi and Sunter in [13]) pioneered the probabilistic approach to matching, their work (and much of the subsequent record linkage literature) tacitly assumes that the data to be matched consists of properly structured records with a well-defined schema. The work on duplicate detection, merge/purge, and entity resolution is also targeted at structured and properly segmented records.

At the other end of the spectrum, the work in natural language processing [19] focuses on the detection of mentions of the same entity in free text. In contrast, in matching offers to products, there are components from both bodies of work: the offers consist of only free text, while the products are properly structured under a given schema.

Much of the prior work has relied on the presence of values for all attributes in the data records, and the goal has been to design a similarity metric either at the entire record level [9, 20] or at the attribute level that are subsequently combined to measure record level match [5, 8]. This assumption is not valid in our setting. Since offers consist of free text, their tokens need to be mapped to attributes. However, some tokens may not map to any attribute (*e.g.,* the token 'monitor' in Offer-1 of Fig. 1), and when they do map, they can be ambiguously mapped to multiple attributes (*e.g.,* the token 'Panasonic' in Offer-1 of Fig. 1). So, unlike previous settings, the matching algorithm needs to disambiguate among the multiple possible interpretations

of the offers. These problems also arise in the context of understanding queries [22, 25]. In [25], a probabilistic model is introduced to identify the annotation of a query which corresponds to best explanation of that query. In our work, we propose the notion of optimal parse which is defined with respect to a product the offer will be matched against.

The design of a matching function in prior work has focused primarily on the computation of weights for value matches and mismatches for the different fields of a record [5, 8], but the explicit modeling of missing values has not received much attention. An exception is [11], wherein a comparison feature vector is augmented to encode presence or absence of values. However, that approach does not explicitly penalize mismatches at the attribute level, and therefore does not leverage a strong signal for matching offers to products.

Additional related work includes research in assigning database entities to text documents [7, 10]. These approaches assume that the documents consist of well-formed English sentences. Closest to us is the work of Michelson and Knoblock [17], who also consider the problem of matching concise textual descriptions to structured records. Their techniques differ from ours in how blocking as well as matching are done. For blocking, they learn rules for constructing the blocks whereas we use the natural organization of products into categories for this purpose. For matching, they employ the entire unstructured text, whereas our matching function uses only those segments of the text that are deemed relevant on a per-attribute basis.

Specifically in the commerce domain, Bilenko et al. [4] proposed techniques for clustering merchant offers, but assume that the offers have structured information. Many of the challenges of matching unstructured offers to structured product specifications disappear when clustering structured offers.

3. SYSTEM DESIGN

We have a database $\mathcal{S}$ of product descriptions, represented as structured records. Every structured record $s \in \mathcal{S}$ consists of a set of attribute ⟨name, value⟩ pairs, denoted by Q_s. The attributes can be numeric or categorical. We receive as input an unstructured offer, u, which is a concise free-text description that specifies values for a subset of the attributes in $\mathcal{S}$ in an arbitrary manner. The text may also contain additional words and may be grammatically ill-formed. Our objective is to match u to one or more structured records in $\mathcal{S}$. We use the metric of precision and recall for judging the quality of the matching system.

We take a probabilistic approach and find the product $s \in \mathcal{S}$ that has the largest probability of match to the given offer, u. Our matcher is learned in an offline stage (see Algorithm 1). For this, we postulate a small training set $\mathcal{U}$ of unstructured offers. Each $u \in \mathcal{U}$ has been matched to one structured record in $\mathcal{S}$ (set $\mathcal{M}$). We also have mismatched records from $\mathcal{S}$, one for every $u \in \mathcal{U}$ (set $\mathcal{N}$).

In the subsequent online stage (Algorithm 2), new offers are matched one at a time. We first do candidate selection, and then choose the best matched product amongst the candidates by applying the learned model.

We next describe the key components required of our system:

3.1 Semantic Parsing

Our matching algorithm is based on understanding the semantics of the offer descriptions and using these semantics to aid matching. Thus, the first step in matching is the semantic parsing step. This step is performed in a three stage process consisting of *tagging* the offer with attributes, identifying *plausible parsings* based on the tags, and finally obtaining an *optimal parse*. We describe each of these steps:

Tagging: The tagging step discovers attribute names present in the offer and associates corresponding strings from the offer to them. This step is performed as follows. Let $\mathcal{A}$ represent all the attributes present in structured data available in product descriptions $\mathcal{S}$. We first build an inverted index $\mathcal{D}$ from $\mathcal{S}$ such that $\mathcal{D}(v)$ returns a set of attribute names (in $\mathcal{A}$) associated with string v. Note that multiple attributes can be associated with the same value. We perform standard preprocessing such as unit conversion and name synonymization, while building the dictionaries. Let $\mathcal{Z}_u$ represent the set of all n-grams (up to n = 4) present in u. Then, the tagging step identifies the set of all attribute ⟨name,value⟩ pairs in u:

$$\mathcal{R}_u = \left\{ \left\langle a, \{v | v \in \mathcal{Z}_u, a \in \mathcal{D}(v)\} \right\rangle | a \in \mathcal{A} \right\} \quad (1)$$

Fig. 2(a) shows an offer for digital camera, and the output of the tagging step. A portion of the output from this step is { {brand, {‘panasonic’}}, {product line, {‘lumix’, ‘panasonic lumix’}}, where brand and product line are the two of the identified attributes, with brand having a single value ‘panasonic’ and product line having a set of two values {‘lumix’, ‘panasonic lumix’}. In the same example, the value ‘3.6x’ is associated with both optical and digital zoom: { {optical zoom, {‘3.6x’}}, { {digital zoom, {‘3.6x’}}

Plausible parse: Given the tagging, a plausible parse of an offer is defined to be a particular combination of all attributes identified in the tagging step such that each attribute is associated with exactly one value. Thus, if each attribute a has k_a values, then there are $\prod_a k_a$ plausible parses in the offer. Typically, k_a is small and thus, only a small number of parses are plausible.

The example in Fig. 2(a) has a single value for six of the seven identified attributes and the ‘product line’ attribute has two values. Thus, this offer has two plausible parses, one parse in which ‘lumix’ is the ‘product line’ and another in which ‘panasonic lumix’ is the ‘product line’.

Multiple plausible parses arise because of ambiguities in the data. Therefore, we maintain these plausible parses until the offer is paired with a product which gives rise to the optimal parse of the offer with respect to that product.

Optimal parse: When an offer u is paired with a product s, we use the parse of u in which the maximum number of attributes agree in their values with s. We call this plausible parse *the optimal parse* of the offer with respect to the product:

$$\mathcal{R}^*_{u,s} = \left\{ \langle a, v \rangle | \langle a, v \rangle \in \mathcal{R}_u \text{ AND } \left(\langle a, v \rangle \in Q_s \right. \right. \\ \left. \left. \text{OR } a \text{ is numeric OR } s.val(a) = \emptyset \right) \right\}, \quad (2)$$

where $s.val(a)$ is the value of attribute a in product s. Note that in addition to keeping those attribute ⟨name, value⟩ pairs in u that match with attribute ⟨name, value⟩ pairs in s, we also retain those attribute ⟨name, value⟩ pairs for which there is no corresponding pair in s (because s has a missing

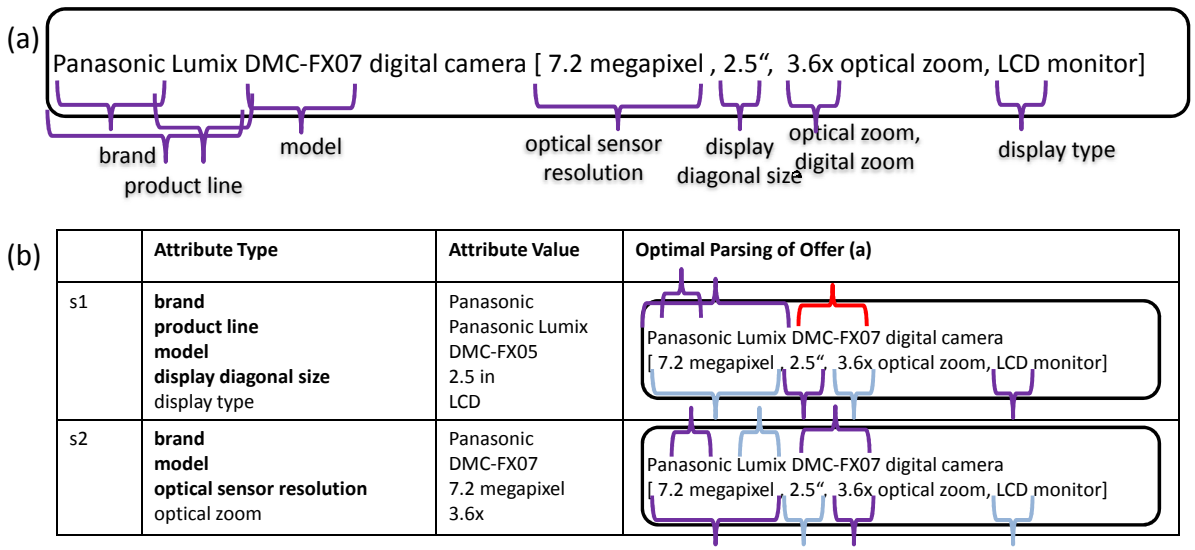


Figure 2: (a) An offer u with all its plausible parses, one in which 'panasonic lumix' is the product line and the other in which 'lumix' is the product line(b) Two products $s1$ and $s2$ with their structured information. Note that the optimal parse of u is different depending on the product that u is paired with.

value for that attribute) or if the attribute is numeric. As we will see momentarily, the missing values and numeric values are treated differently when we define the similarity function between offer and product. Observe that optimal parse is defined only with respect to a particular product. Different products can give rise to a different choice of plausible parse to be optimal. Continuing with our example using Fig. 2(b), the optimal parse of u corresponding to product $s1$ is the plausible parse with 'Panasonic Lumix' as the product line. When u is paired with $s2$, both plausible parses are optimal since $s2$ does not have the product line specified.

3.2 Similarity Feature Vectors

The similarity between an offer and a product is defined in terms of their similarity on the values of the attributes present in them. We consider all attributes $\mathcal{K} \subset \mathcal{A}$ that are present in at least one offer. We would like the similarity function to take into account not only the match in values of certain attributes, but also reflect mismatches or missing values in either products or offers. The function should penalize mismatches differently from missing values; in fact, a mismatched value is a stronger indicator of the corresponding offer and product mismatch. In addition, an attribute that is frequently missing reflects its lower importance for the purposes of matching.

The similarity feature vector we define in the sequel satisfies the above design considerations. Let $\mathcal{R}^*_{u,s}$ (Eq. 2) represent the optimal parse of offer u with respect to product s. Then, for the pair $\langle u, s\rangle$, we compute the similarity feature vector $\mathbf{f}$ by determining similarity levels between u and s for each attribute $k_j \in \mathcal{K}$:

$$f_j = \begin{cases} 0 & \text{if } \mathcal{R}^*_{u,s}(k_j) = \emptyset \text{ OR } s.val(k_j) = \emptyset \\ (-1)^{\mathrm{I}[\frac{|\mathcal{R}^*_{u,s}(k_j) - s.val(k_j)|}{\max(\mathcal{R}^*_{u,s}(k_j), s.val(k_j))} > \lambda]} & \text{if } k_j \text{ is numeric} \\ (-1)^{\mathrm{I}[\mathcal{R}^*_{u,s}(k_j) = s.val(k_j)]} & \text{otherwise} \end{cases} \tag{3}$$

where I[z] is the indicator function. Note that when either the optimal parse of the offer or the product has a missing value for an attribute, the corresponding feature value is 0. In addition, when the attribute values mismatch, the corresponding feature value is -1. This definition enables penalizing the matching score differently when u or s is missing an attribute value than if they disagree on that attribute.

For categorical attributes, we use binary loss rather than a string similarity measure (*c.f.* [6]). The reason for this decision was that while string similarity measure can be useful in certain situations (*e.g.,* product lines 'canon powershot' and 'powershot' are similar), we found it to be detrimental more often (*e.g.,* model numbers 'a1000' and 'a100' of Canon powershot cameras are close in string similarity but refer to distinct products). Numeric attributes, on the other hand, frequently have imprecise values because of round-off errors (*e.g.* 7 MP vs. 7.2 MP) or difference in conversion factors (1GB = 1000 MB or 1GB =1024 MB). Hence, we allow for approximation in measuring their similarity. After some experimentation, we set λ to 0.1.

3.3 Matching Function

We would like a matching function that can provide a probabilistic score of match between an offer and a product. In addition, since the number of attributes in $\mathcal{S}$ is large, and not all attributes are present in the offers, the function needs to automatically infer attributes that are required to be matched and also learn the relative importance among them.

The binary logistic regression lends itself to satisfy these criteria. Given labeled data of good and bad matches, and features that measure similarity between the attributes, it can automatically learn the relative importance between the attributes, and in turn provide a function that measures the match between an offer and product in terms of a probabilistic score. We use binary logistic regression of the form:

$$\mathcal{F}(\mathbf{w}, \mathbf{f}) = P(y = 1|\mathbf{f}, \mathbf{w}) = \frac{1}{1 + \exp\{-(b + \mathbf{f}^T\mathbf{w})\}} \tag{4}$$

Algorithm 1 Off-line Training

Input:
$\mathcal{U} = \{u_1 \dots u_N\}$ - a set of offers
$\mathcal{S} = \{s_1 \dots s_M\}$ - a set of structured product descriptions, $M >> N$
$\mathcal{M} = \{\langle u_i, s_j \rangle_i\}_{i=1}^{N}, (u_i \in \mathcal{U}, s_j \in \mathcal{S})$ - pairs of correctly matched records, one for every u_i.
$\mathcal{N} = \{\langle u_i, s_k \rangle_i\}_{i=1}^{N}$ - similarly, pairs of mismatched records.
Output:
$\mathcal{D}$ - dictionaries,
$\mathbf{w}$ - algorithm parameters
Preprocess:
$\mathcal{D} \Leftarrow$ CreateAttributeDictionaries($\mathcal{S}$) - build dictionaries of attributes and their values using $\mathcal{S}$
Train:
for all $u \in \mathcal{U}$ **do**
  $u \Leftarrow$ SemanticParsing($u, \mathcal{D}$) - *Extract plausible parses* (Section 3.1)
end for
for all pairs $\in \mathcal{M}$ and pairs $\in \mathcal{N}$ **do**
  $\mathbf{f}_i^{\mathcal{M}} \Leftarrow$ ExtractSimFeatures(pair$_i$)
  $\mathbf{f}_j^{\mathcal{N}} \Leftarrow$ ExtractSimFeatures(pair$_j$) - *Construct similarity feature vector for matched and mismatched pairs* (Section 3.2)
end for
$\mathbf{w}^* \Leftarrow \arg\max_{\mathbf{w}}$ LearnToMatch($\mathcal{F}(\mathbf{w}, \mathbf{f}), \{\mathbf{f}_i^{\mathcal{M}}\}, \{\mathbf{f}_j^{\mathcal{N}}\}$) - *Train a function that maps feature vectors to match probability,* $\mathcal{F}(\mathbf{w}, \mathbf{f}) : \mathbf{f} \to [0, 1]$ (Section 3.3)
Return: $\mathbf{w}^*, \mathcal{D}$

The logistic regression learns a mapping from the similarity feature vector $\mathbf{f}$ to a binary label y, through the logistic function. The parameter $\mathbf{w}$ is the weight vector wherein each component w_j measures the relative importance of the feature f_j for predicting the label y.

In addition to the human-labeled matched training pairs, we construct mismatched pairs as follows. We pair each offer in the matched training pair with a small number of products(10) that are matched to offers, other than the offer under consideration. This addition of mismatched pairs introduces variability among the mismatched pairs. Some are completely different (*e.g.,* 'Canon EOS 40D camera' mismatched to 'Olympus MJU 300') while others are highly overlapping products (*e.g.,* 'Canon EOS 40D camera' mismatched to 'Canon 50D camera').

Having the matched and mismatched training pairs, let $\mathbf{f}_i = [f_{i1}, f_{i2}, \dots, f_{i|\mathcal{K}|}]$ be the feature vector for pair i. Let $\{\mathbf{F}, \mathbf{Y}\} = \{(\mathbf{f}_1, y_1), \dots, (\mathbf{f}_T, y_T)\}$ be the set of feature vectors along with their corresponding binary labels. Here, $y_i = 1$ indicates that the ith pair is a match, otherwise $y_i = 0$. Logistic regression maximizes an objective function which is the conditional log-likelihood of the training data:

$$\arg\max_{\mathbf{w}} \log P(\mathbf{Y}|\mathbf{F}, \mathbf{w}) = \arg\max_{\mathbf{w}} \sum_{i=1}^{T} \log P(y_i|\mathbf{f}_i, \mathbf{w}), \quad (5)$$

where $P(y_i = 1|\mathbf{f}_i, \mathbf{w})$ is defined by Eq. 4. Note that a feature with positive weight will affect the score by increasing the probability of match for a pair with agreement on the feature, by decreasing the score in the case of a mismatch, and by leaving the score unaffected in the case of a missing value.

Algorithm 2 Online Matching

Input:
u - offer
$\mathcal{S}, \mathcal{D}, \mathbf{w}$
Output: s^* - best matching $s \in \mathcal{S}$
$u \Leftarrow$ SemanticParsing($u, \mathcal{D}$) - (Section 3.1)
Blocking:
$k_i \Leftarrow$ Top attributes with largest weights in $\mathbf{w}$
$\mathcal{S}^* \Leftarrow$ Subset of $\mathcal{S}$ with $\cup_i (u.val(k_i) = s.val(k_i)$
for all $s_i \in \mathcal{S}^*$ **do**
  $\mathbf{f}_i \Leftarrow$ ExtractSimFeatures($\langle u, s_i \rangle, \mathcal{K}$) - (Section 3.2)
  $P(\text{match}(s_i, u)) \Leftarrow \mathcal{F}(\mathbf{w}, \mathbf{f}_i)$ - *Matching score of a pair* (Section 3.4)
end for
Return: $s^* = \arg\max_{s_i} P(\text{match}(u, s_i))$ - *Best Matching score of all pairs* (Section 3.4)

3.4 Online Matching

During the online phase, we are given a previously unseen offer u, and the goal is to identify the best matching product $s \in \mathcal{S}$.

The scoring function learned during the offline phase provides the probability of match for a pair $\langle u, s \rangle$. Naively, we can find the best match by pairing u with every $s \in \mathcal{S}$, calculating the pair match score, and choosing the s^* that results in the highest score. However, such naive pairing will cost $O(|\mathcal{S}|)$ operations for each offer.

Instead, we design a staged blocking strategy [15, 16, 27]. We note that the products are categorized into a taxonomy. Therefore, in the first stage, we use a classifier trained on product data to categorize the given offer into a category node in this taxonomy[18]. This reduces the candidate set to only those products that belong to the offer category.

Further, within the category, we would like to reduce the number of candidate products to match against the offer. For that, we make the following observation. The goal of the matching process is to match the offer to the product that has the largest matching score. To obtain this large score, a product needs to agree with the offer, especially on the values of the attributes that contribute large weights to the matching function. Using this insight, in the second stage, we further reduce the candidate set by identifying those top weighing attributes that can potentially give a matching score of at least θ in Eq. 4. Specifically, after identifying attributes in the offer using method in Section 3.1, we choose subset of attributes, $\{k_j\} \in \mathcal{K}$, in the descending order of weights until the following condition is satisfied:

$$\sum_j f_j w_j \geq \log \frac{\theta}{1-\theta} - b \quad (6)$$

Here, θ corresponds to $P(y = 1|\mathbf{f}, \mathbf{w})$. This equation can be derived from Eq. 4 by rearranging terms and taking the log. This set of attributes $\{k_j\}$ are then used to retrieve products such that the retrieved products match on the value of at least one of the attributes in $\{k_j\}$. The union of all these products becomes the candidate set of products. Note that this candidate set is a superset of products that can potentially match to offer since we consider all products that have at least one matching value within this attribute set.

4. EVALUATION

The central goal of our work is to develop a system that can be deployed to match offers to products on a daily basis. The main requirement for us is to have, for every deployed category, at least 85% precision in matching. In general, this is a tall order for any system to achieve. In our setting, we find two main sources of difficulty that further compound this problem: First, some categories lack sufficient structured information required for matching. This prevents the matcher from performing semantic analysis effectively, and thus in turn, affects the learning of the matching function. Second, the offers are not correctly specified by the merchants, thereby contributing to reduction in matching precision. Thus, identifying error-prone categories become important. This identification enables either augmenting such a category with more structured information or reporting to the merchants about poor quality of their offers.

In this section, we discuss how we select deployable categories and describe the experiments we conducted to study the performance of our technique.

4.1 Algorithms Studied

Variants of our Algorithm

We defined the following variants of our matching function in order to study its characteristics:

1. **Equal Weights (EW):** This is the simplest version where the number of agreeing attributes between the offer u and the product s is used as the predictor for matching. The product s having the largest number of attributes in agreement with u is taken to be the best match.
2. **Learned Weights (LW):** In this version, we learn the relative importance between the attributes, but we treat missing and mismatched attributes equally. For attribute k, the value of feature $\mathbf{f}_k$ is -1 when either the values are missing or when the values are mismatched.
3. **LW with distinction between Mismatched and Missing (LWMM):** Here, relative importance between attributes is learned taking into account whether attribute values are mismatched or missing. For attribute k, the value of feature f_k for missing values is 0, and it is -1 for mismatched values. This version implements the full functionality of Algorithm 1.

Baseline

Inspired by the work in record linkage [4, 9, 20], we employ a baseline, which we call TFIDF. It uses the tf-idf weighted cosine similarity as the measure of agreement between the offers and products. Each token is associated with the tf-idf score defined as log(TF(token)+1) log(IDF(token)) [9]. Here, TF(token) is the frequency of the token in the offer/product and IDF(token) is its corresponding inverse document frequency; IDF(token) is computed across all the products in the category. Before matching, the structured specifications of a product are converted into a string representation by concatenating the content of the record.

TFIDF, instead of treating all tokens equally, weighs tokens inversely to their popularity. Thus, a token such as '40D' (corresponding to the model number of a 'Canon EOS 40D' digital camera) will have higher tf-idf score than 'digital camera' that is ubiquitous in the digital camera category. Typically, tokens that are unique such as model numbers and brands are the ones that are useful in matching. Since TFIDF can choose such unique tokens, it provides a reasonable baseline to our approach.

4.2 Performance Metrics

For evaluation purposes, we have access to a test set of offers, $u \in \mathcal{T}$. We also know the correctly matched product s^* for every u. The matcher has no knowledge about the matching product, but instead predicts the best matched product $\tilde{s}$ with probabilistic score $\gamma_{u,\tilde{s}}$. By best matched we mean that there is no other s that can match u with a higher score. Thus, instead of a standard classification task, we are considering a harder task of the matcher finding the best matching s for every offer u. We require the match score to be at least some $\eta \in [0, 1]$ before calling out a match. We define precision and recall at threshold level η as:

$$\text{Precision}(\eta) = \frac{\sum_{u \in \mathcal{U}} I[(\gamma_{u,\tilde{s}} > \eta) \text{ AND } (s^* = \tilde{s})]}{\sum_{u \in \mathcal{U}} I[\gamma_{u,\tilde{s}} > \eta]} \quad (7)$$

$$\text{Recall}(\eta) = \frac{\sum_{u \in \mathcal{U}} I[(\gamma_{u,\tilde{s}} > \eta) \text{ AND } (s^* = \tilde{s})]}{|\mathcal{T}|}, \quad (8)$$

where I[z] is the indicator function. We also combine precision and recall values into the F-measure, defined as:

$$\text{F-measure}(\eta) = \frac{2\text{Precision}(\eta)\text{Recall}(\eta)}{\text{Precision}(\eta) + \text{Recall}(\eta)}. \quad (9)$$

4.3 Dataset

The dataset comprised of 54 categories related to electronics (*e.g.,* televisions, mp3 players), computing (*e.g.,* desktop computers, laptops) and cameras and accessories (*e.g.,* digital cameras, camera accessories). We had a labeled set of 20,000 offers from these categories, each labeled with the corresponding matched products. There were on average 425 offers/category; the smallest category had 50 offers.

For each category, we randomly sampled 100 offers (20 if the number of offers is less than 200) and used them for training the matcher. We used whatever offers were left within a category as the test set for that category. Thus, the training set for each category had at most 100 samples. The test set size varied from category to category as we made use of all the available samples, but in no case was the test set smaller than 30 samples. Results are obtained using 5-fold cross validation.

Note that other than using a separate training set for each category, we do not use any category-specific features and the same code is used for different categories. We use small training set (100 offers per category), which can be curated easily from successful matches (*e.g.,* high click throughs) in a running system. We also do not have any parameters that needs to be tuned. These characteristics are critical for a solution to work at web scale.

4.4 Selecting Categories for Deployment

We use a small validation set to compute precision of matches, and select only those categories that meet our precision criteria of at least 85% precision. This enables us to do two important things required for deployment: (a) this technique can be applied at regular intervals to ensure quality control, (b) excluded categories can be analyzed for rectification, either by obtaining additional structured data

Offer Description	Products
imation 4gb nano usb 2.0 flash drive - 4gb - usb - external	imation nano flash drive usb flash drive - 4 gb imation 2gb usb 2.0 clip flash drive

Table 1: Sample offer and two products from the memory cards category.

or reporting to merchants about their data quality. In addition, after the issues are fixed, we can re-train matchers for these categories and decide whether or not to deploy.

By using the validation set, LWMM was able to select 37 categories for immediate deployment, each having the precision of at least 85% and we were able to initiate corrective action for others. We found that there was a correlation between the economic value of a category and the data quality; the higher the economic value of products in a category, the higher the data quality. Many of the categories that did not pass our filter correspond to low economic value categories (*e.g.,* accessories). An example is the 'computer memory card' category. Table 1 shows example offer and products from this category. As we can see from this example, an important attribute for this category is the capacity of the memory card. However, as this attribute is missing in all but two products (out of 7500 products), the matcher learned to ignore this attribute, and hence was not able to learn the function that can find appropriate products.

We also computed the precision achieved on the excluded categories using TFIDF. The numbers were average precision of 51% (largest precision of 70%) and 32% recall, reinforcing further that these categories are indeed error-prone.

4.5 LWMM Performance

We present precision and recall values for deployed categories. Fig. 3 shows the scatter plot of precision-recall values that the LWMM algorithm exhibits. Each circle corresponds to a category, and the area of the circle is proportional to the test set size. We have labeled some of the categories. The macro average precision is 93% while the macro average recall is 54% [1]. The corresponding micro average numbers are 93% precision and 41% recall. We studied some of these categories. The 'laptops' category has high precision and low recall. The main reason for low recall is that many of the offers have a large number of attributes that are missing, and hence the matcher decides not to match these offers to any product, so as to maintain high precision. In contrast, the 'printers' category has both high precision and recall, because a very large fraction of offers specify the attributes for matching. This difference partly arises from the nature of the categories - 'laptops' require a more detailed specification than 'printers' and hence it is easier for the merchants to include information needed for matching 'printers'.

By way of comparison, the average precision and recall values for TFIDF are 64% and 11% respectively. Fig. 4 shows the relative precision of LWMM and TFIDF. We see that LWMM performs better than TFIDF for all categories. The reason is that TFIDF weighs the relative importance

[1]The macro precision (or recall) is obtained by averaging the precision (or recall) of every categories, while the micro precision(recall) is computing the precision (recall) of all offers across all categories.

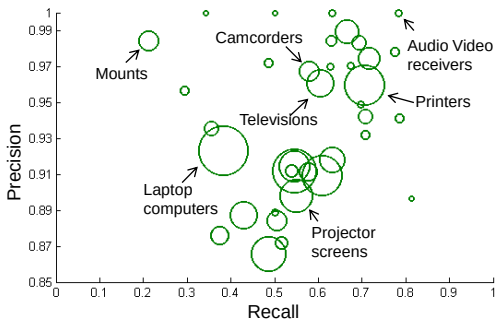


Figure 3: Performance of LWMM

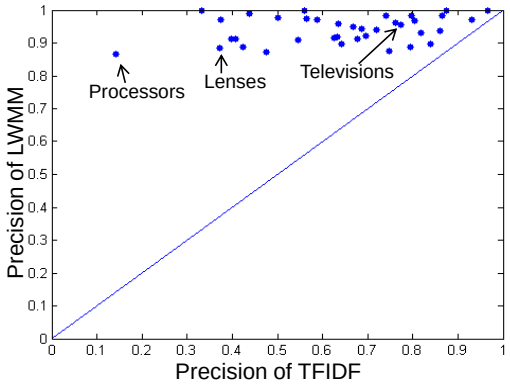


Figure 4: LWMM *v.s.* TF-IDF

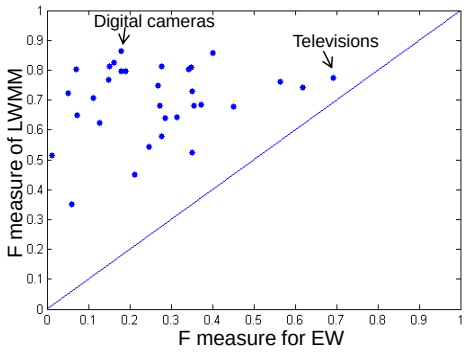
of the tokens, but only as measured by the frequency of their presence in the product collection. It is not cognizant of what tokens are semantically important. As an example, the token 'canon' gets down-weighted in the 'digital cameras' category because there are many 'Canon' cameras. However, 'canon' as the brand is a very good indicator of a product identity.

4.6 Importance of Learning Weights

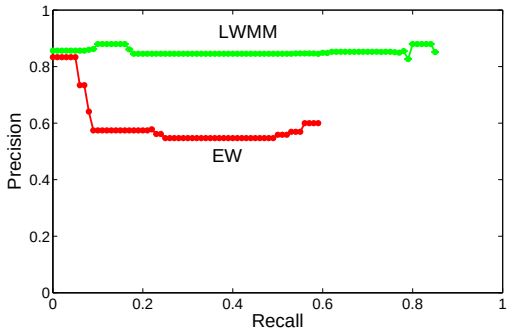
Fig. 5(a) shows the ratio of the LWMM's F-measure over EW's F-measure. Clearly, learning weights makes matching better. We notice that the gains are much larger for some categories than others. To understand this difference, we focus on two categories – digital cameras and televisions.

Fig. 5(b) shows the precision-recall values for digital cameras. For this category, there were seven attributes present in at least one offer during the offline training phase. These attributes were brand, model, product line, color, resolution, optical zoom, viewfinder type and video input type. EW weighs these attributes equally. At low recall, EW insists on all key attributes to agree on their values, and hence the precision becomes high. However, as we increase recall by reducing the number of agreements in attributes, precision drops. It is as expected, since certain combinations of attributes provide spurious matches (*e.g.,* agreement on color and resolution of a camera). On the other hand, by learning the relative importance of the attributes, LWMM matches a larger fraction of offers without sacrificing much precision.

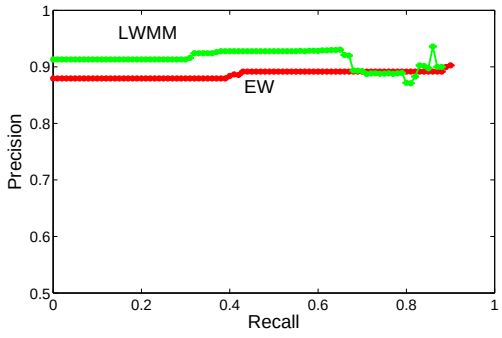
The precision-recall values for televisions are shown in



(a) LWMM *v.s.* EW



(b) LWMM *v.s.* EW (Digital Cameras)



(c) LWMM *v.s.* EW (Televisions)

Figure 5: Importance of learning weights.

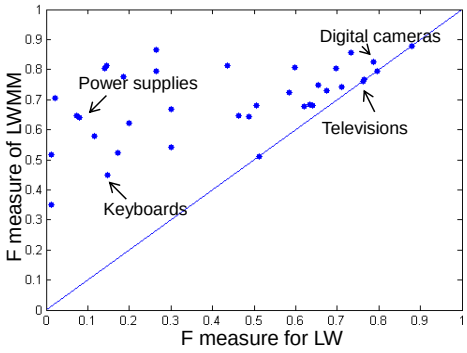


Figure 6: Treating mismatching attributes differently from missing attributes.

Fig. 5(c). For this category, only five attributes (brand, model, product line, diagonal screen size, projection display technology) are present in the offers during the offline training phase. We found that these attributes were often all present in the data and their values were generally correct. Hence, the performance of EW is quite high and it performs almost as well as its learned counterpart, LWMM.

4.7 Importance of Treating Mismatching and Missing Values Differently

Fig. 6 shows the scatter plot comparing the F-measures between LWMM and LW. We can see that there is a gain in F-measure for all categories, when we treat missing attributes differently from mismatched values. The gain is less pronounced for high economic value categories such as digital cameras than for low economic value categories such as keyboards and power supplies. The reason, we speculate, is that in the case of the former, the merchants have the incentive to provide better offer descriptions containing all the necessary attributes needed for matching. Hence, missing attributes becomes less of an issue.

4.8 Scalability

We perform staged blocking at the time of matching. In the first stage, offers are classified into categories, and then they are matched to products within that category. This substantially reduces the number of products to compare against. Subsequently, as described in Section 3.4 only a viable subset of products is selected to be matched against the given offer. Fig. 7 shows the number of candidates considered for each category. Each stem corresponds to a particular category; the length of the stem is one standard deviation of the average number of candidates across the offers in that category. We can see that the candidate set size is much smaller than the number of products in most of the categories.

5. SUMMARY AND FUTURE WORK

We described a system for matching unstructured textual descriptions to structured data records that arises in the context of matching sales offers to product specifications in e-commerce websites. The highlights of our solution include:

1. Semantic understanding of offers by leveraging structured information in the database:

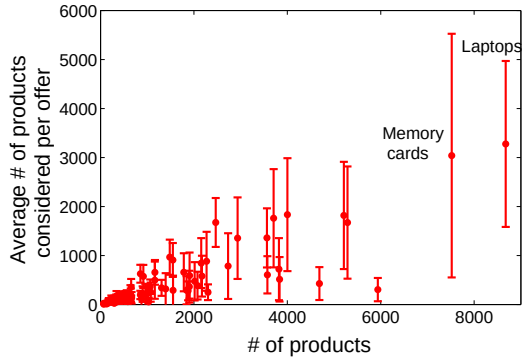


Figure 7: Scalability contributor during online matching

- The semantics is defined using plausible parses
- The same offer has different optimal parse with respect to different products, and thus no hard decision is made about the attribute ⟨name, value⟩ pairs in the offer until it is paired with a product

2. A match function based on the semantics is learned to find the product that has the largest match probability to the given offer:
   - Takes into account matches as well as mismatches in attribute values between offer-product pairs
   - Differentiates between missing attribute values and mismatch of attribute values.
   - Infers the relative importance of different attributes in the matching.
3. Built-in strategies for the solution to work at web scale:
   - Avoiding domain-specific features in the matching system
   - Reducing the candidate set of products that can potentially match a given set of offers

Extensive experiments using the Bing Shopping catalog demonstrate the effectiveness of our solution.

While presented in the context of commerce search, our system is applicable to any vertical search domain where there is the notion of an offer and a repository of structured data. As an example, in travel search there are offers for hotels (from various travel agencies) and a database with information about the hotels. Similarly, in local search, the system can be applied to match deals at restaurants against a restaurant database. As future work, we plan to apply our system to such domains.

6. REFERENCES

[1] Comscore reports record-breaking 43.4 billion in Q4 2010 U.S. retail e-commerce spending, up 11 percent vs. year ago. `http://www.comscore.com/Press_Events/Press_Releases/2011/2/comScore_Reports_Record-Breaking_43.4_Billion_in_Q4_2010_U.S._Retail_E-Commerce_Spending`, Accessed Feb 15 2011.

[2] R. Agrawal, R. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. Auditing compliance with a hippocratic database. In *VLDB*, pages 516–527, 2004.

[3] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, 18(1):255–276, 2009.

[4] M. Bilenko, S. Basu, and M. Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *ICDM*, pages 58–65, 2005.

[5] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[6] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.

[7] V. Chakaravarthy, H. Gupta, P. Roy, and M. Mohania. Efficiently linking text documents with relevant structured information. In *VLDB*, pages 667–678, 2006.

[8] M. Cochinwala, V. Kurien, G. Lalk, and D. Shasha. Efficient data reconciliation. *Information Sciences*, 137(1-4):1–15, 2001.

[9] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD*, pages 202–212, 1998.

[10] N. Dalvi, R. Kumar, B. Pang, and A. Tomkins. Matching reviews to objects using a language model. In *EMNLP*, pages 609–618, 2009.

[11] N. S. D. DuBois. A solution to the problem of linking multivariate documents. *Journal of the American Statistical Association*, 64(325):163–174, 1969.

[12] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, 2007.

[13] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[14] G. Fulgoni. State of the U.S. Retail Economy in Q2 2009. Technical report, Comscore, August 20 2009.

[15] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[16] A. Mccallum, K. Nigam, and H. L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *In Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[17] M. Michelson and C. Knoblock. Creating relational data from unstructured and ungrammatical data sources. *Journal of Artificial Intelligence Research*, 31:543–590, 2008.

[18] T. Mitchell. *Machine Learning*. McGraw Hill Higher Education, 1997.

[19] R. Mitkov. *Anaphora Resolution*. Longman, 2002.

[20] A. Monge and C. Elkan. The field-matching problem: algorithm and application. In *KDD*, 1996.

[21] H. B. Newcombe, M. J. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, October 1959.

[22] K. Q. Pu and X. Yu. Keyword query cleaning. In *PVLDB*, pages 909–920, 2008.

[23] P. Ravikumar and W. W. Cohen. A hierarchical graphical model for record linkage. In *UAI*, 2004.

[24] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.

[25] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *SIGMOD*, pages 771–782, 2010.

[26] S. Singh, K. Schultz, and A. McCallum. Bi-directional joint inference for entity resolution and segmentation using imperatively-defined factor graphs. In *ECML-PKDD*, pages 414–429, 2009.

[27] W. E. Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Census, 2006.