19th International Conference on Web Engineering (ICWE2019), Daejeon, South Korea, 11-14 June, 2019

Merging Intelligent API Responses using a Proportional Representation Approach

Tomohiro Ohtake[1] [0000−0002−7675−3100], Alex Cummaudo[2] [0000−0001−7878−6283], Mohamed Abdelrazek[1] [0000−0003−3812−9785], Rajesh Vasa[1] [0000−0003−4805−1467], and John Grundy[3] [0000−0003−4928−7076]

[1] Faculty of Science, Engineering and Built Environment, Deakin University, Geelong, Australia
{ tomohiro.otake, mohamed.abdelrazek, rajesh.vasa }@deakin.edu.au
[2] Applied Artificial Intelligence Institute, Deakin University, Geelong, Australia
ca@deakin.edu.au
[3] Faculty of Information Technology, Monash University, Caulfield, Australia
John.Grundy@monash.edu

Abstract. Intelligent APIs, such as Google Cloud Vision or Amazon Rekognition, are becoming evermore pervasive and easily accessible to developers to build applications. Because of the stochastic nature that machine learning entails and disparate datasets used in their training, the output from different APIs varies over time, with low reliability in some cases when compared against each other. Merging multiple unreliable API responses from multiple vendors may increase the reliability of the overall response, and thus the reliability of the intelligent end-product. We introduce a novel methodology – inspired by the proportional representation used in electoral systems – to merge outputs of different intelligent computer vision APIs provided by multiple vendors. Experiments show that our method outperforms both naive merge methods and traditional proportional representation methods by 0.015 F-measure.

Keywords: Application programming interfaces · Web services · Data integration · Artificial intelligence · Supervised learning

1 Introduction

With the introduction of intelligent web services that make machine learning (ML) more accessible to developers [8, 20], we have seen a large growth of intelligent applications built using such APIs [5, 14]. For example, consider the advances made in computer vision, where objects are localised within an image and labelled with associated categories. Cloud-based computer vision APIs (e.g., [1–3, 6, 10, 11, 15, 23]) utilise machine-learning techniques to achieve image recognition via a remote black-box approach, thereby reducing the overhead for application developers to understand how to implement intelligent systems from scratch. Furthermore, as the processing and training of the machine-learnt algorithms is offloaded to the cloud, developers send simply send RESTful API requests to do the recognition, making it more accessible to them. There are, however, inherit differences and drawbacks between traditional APIs and intelligent APIs, which we describe with the motivating scenario below.

1.1 Motivating Scenario: Intelligent APIs vs Traditional APIs

An application developer, Tom, wishes to develop a social media Android and iOS app that catalogues photos of him and his friends, common objects in the photo, and generates brief descriptions in the photo (e.g., all photos with his husky dog, all photos on a sunny day etc.). Tom comes from a typical software engineering background with little knowledge of computer vision and its underlying concepts. He knows that intelligent computer vision web APIs are far more accessible than building a computer vision engine from scratch, and opts for building his app using these cloud services instead.

Based on his experiences using similar cloud services, Tom would expect consistency of the results from the same API and different APIs that provide the same (or similar) functionality. As an analogy, when Tom writes the Java substring method `"doggy".substring(0, 2)`, he expects it to be the same result as the Swift equivalent `"doggy".prefix(3)`. Each and every time he interacts with the substring method using either API, he gets `"dog"` as the response. This is because Tom is used to deterministic, rule-driven APIs that drive the implementation behind the substring method.

Tom's deterministic mindset results in three key differentials between a traditional API and intelligent API:

(1) **Given similar input, results differ between similar intelligent APIs.** When Tom interacts with intelligent APIs, he is not aware that each API provider trains their own, unique ML model, both with disparate methods and datasets. These intelligent APIs are, therefore, nondeterministic and data-driven; input images—even if they contain the same conceptual objects—often output different results. Contrast this to the substring method of traditional APIs; regardless of what programming language or string library is used, the same response is expected by developers.

(2) **Intelligent responses are not certain.** When Tom interprets the response object of an intelligent API, he finds that there is a 'confidence' value or 'score'. This is because the ML models that power intelligent APIs are inherently probabilistic and stochastic; any insight they produce is purely statistical and associational [18]. Unlike the substring example, where the rule-driven implementation provides certainty to the results, this is not guaranteed for intelligent APIs. For example, a picture of a husky breed of dog is misclassified as a wolf. This could be due to adversarial examples [22] that 'trick' the model into misclassifying images when they are fully decipherable to humans. It is well-studied that such adversarial examples exist in the real world unintentionally [4, 12, 19].

(3) **Intelligent APIs evolve over time.** Tom may find that responses to processing an image may change over time; the labels he processes in testing may evolve and therefore differ to when in production. In traditional APIs, evolution in responses is slower, generally well-communicated, and usually rare (Tom would always expect `"dog"` to be returned in the substring example). This has many implications on software systems that depend on these APIs, such as confidence in the output and portability of the solution. Currently, if Tom switches from one API provider to another, or if he doesn't regularly test his app in production, he may begin to see a very different set of labels and confidence levels.

1.2 Research Motivation

These drawbacks bring difficulties to the intended API users like Tom. We identify a gap in the software engineering literature regarding such drawbacks, including: lack of best practices in using intelligent APIs; assessing and improving the reliability of APIs for their use in end-products; evaluating which API is suitable for different developer and application needs; and how to mitigate risk associated with these APIs. We focus on improving reliability of intelligent APIs for use in end-products. The key research questions in this paper are:

RQ1: Is it possible to improve reliability by merging multiple intelligent API results?
RQ2: Are there better algorithms for merging these results than currently in use?

Previous attempts at overcoming low reliability include triple-modular redundancy [13]. This method uses three modules and decides output using majority rule. However, in intelligent APIs, it is difficult to apply majority rule: these APIs respond with a list of labels and corresponding scores. Moreover, disparate APIs ordinarily output different results. These differences makes it hard to apply majority rule because type of outputs are complex and disparate APIs output different result for the same input. Merging search results is another technique to improve reliability [21]. It normalises scores of different databases using a centralised sample database. Normalising scores makes it possible to merge search results into a single ranked list. However, search responses are disjoint, whereas they are not in the context of most intelligent APIs.

In this paper, we introduce a novel method to merge responses of intelligent APIs, using image recognition APIs as our motivating example. Section 2 describes naive merging methods and requirements. Section 3 gives insights into the structure of labels. Section 4 introduces our method of merging computer vision labels. Section 5 compares precision and recall for each method. Section 6 presents conclusions and future work.

2 Merging API Responses

Image recognition APIs have similar interfaces: they receive a single input (image) and respond with a list of labels and associated confidence scores. Similarly, other supervised-AI-based APIs do the same (e.g. detecting emotions from text and natural language processing [9, 24]. It is difficult to apply majority rule on such disparate, complex outputs. While the outputs by *multiple* AI-based API endpoints is different and complex, the general format of the output is the same: it follows a list of labels and associated scores.

2.1 API Façade Pattern

To merge responses from multiple APIs, we introduce the notion of an API façade. It is similar to a metasearch engine, but differs in their external endpoints. The façade accepts the input from one API endpoint (the façade endpoint), propagates that input to all user registered concrete (external) API endpoints simultaneously, then ‘merges’ outputs from these concrete endpoints before sending this merged response to the API user. We demonstrate this process in fig. 1.

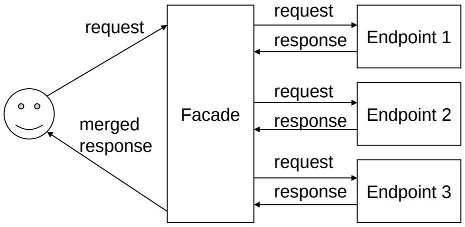

Fig. 1. The user sends a request to the façade; this request is propagated to the relevant APIs. Responses are merged by the façade and returned back to the user.

Although the model introduces more time and cost overhead, both can be mitigated by caching results. On the other hand, the façade pattern provides the following benefits:

- **Easy to modify:** It requires only small modifications to applications, e.g. changing each concrete endpoint URL.
- **Easy to customise:** It merges results from disparate concrete APIs according to user's preference.
- **Improves reliability:** It enhances reliability of the overall returned result by merging results from different endpoints.

2.2 Merge Operations

The API façade is applicable to many use cases. However, this paper focuses on APIs that output a list of labels and scores, as is the case for many image recognition APIs. Merge operations involve the mapping of multiple lists and associated scores, produced by multiple APIs, to just one list. For instance, an image recognition API receives a bowl of fruit as the input image and outputs `[[apple, 0.9], [banana, 0.8]]`, where the first item is the label and the second item is the score. Similarly, another computer vision API outputs `[[apple, 0.7], [cherry, 0.8]]` for the same image. Merge operations, therefore, merges these two lists into just one.

Naive ways of merging results could make use of max, min, and average operations on the confidence scores. For example: (i) *max* merges results to `[[apple, 0.9], [banana, 0.8], [cherry, 0.8]]`; (ii) *min* merges results to `[[apple, 0.7]`; (iii) *average* merges results to `[[apple, 0.8], [banana, 0.4], [cherry, 0.4]]`. However, object labels in the results are natural language words in many cases; thus, max, min, and average operations do not exploit label semantics – the conceptual meanings of these labels – when conducting label merging. To improve the quality of the merged results, we consider the meaning of these labels, as we describe below.

2.3 Merging Operators for Labels

Merge operations on labels are n-ary operations that map R^n to R, where $R_i = \{(l_{ij}, s_{ij})\}$ is a response from endpoint i, and contains pairs of labels (l_{ij}) and scores (s_{ij}). Merge operations on labels have the following properties:

- *Identity* defines that merging single response should output same response. That is $R = \text{merge}(R)$ is always true.
- *Commutativity* defines that the order of operands should not change the result. That is $\text{merge}(R_1, R_2) = \text{merge}(R_2, R_1)$ is always true.
- *Reflexivity* defines that merging multiple same responses should output same response. That is $R = \text{merge}(R, R)$ is always true.
- *Additivity* defines that, for a specific label, the merged response should have higher or equal score for the label if a concrete endpoint has a higher score. Let $R = \text{merge}(R_1, R_2)$ and $R' = \text{merge}(R'_1, R_2)$ be merged responses. R_1 and R'_1 are same, except R'_1 has a higher score for label l_x than R_1. The additive score property requires that R' score for l_x should be greater than or equal to R score for l_x.

Max, min, and average operations in section 2.2 follow each of these rules as all operations calculate the score by applying these operations on each score.

3 Graph of Labels

Image recognition APIs typically return a lists of labels (in most cases, an English word or words) and associated scores. Lexical databases, such as WordNet [16], can therefore be used to describe the ontology behind these labels' meanings. Figure 2 is an example of graph of labels and synsets. A synset is a grouped set of synonyms for the input word. We label red nodes as labels from Endpoint 1, yellow nodes as labels from Endpoint 2, and blue nodes as synsets. As actual graphs are usually more complex, fig. 2 is a simplified graph to illustrate the usage of associating labels from two concrete sources to synsets.

3.1 Labels and synsets

The number of labels depends on input images and concrete API endpoints used. Table 1 and fig. 3 show how many labels are returned from Google Cloud Vision [6], Amazon Rekogition [1] and Microsoft Azure Computer Vision [15] image recognition APIs, using 1,000 images from Open Images Dataset V4 [7] Image-Level Labels set.

Table 1. Number of labels

Endpoint	Average number of labels	Has synset	No synset
Amazon	11.42 ± 7.52	10.74 ± 7.10 (94.0%)	0.66 ± 0.87
Google	8.77 ± 2.15	6.36 ± 2.22 (72.5%)	2.41 ± 1.93
Microsoft	5.39 ± 3.29	5.26 ± 3.32 (97.6%)	0.14 ± 0.37

Labels from Amazon and Microsoft tend to have corresponding synsets. That means these endpoints return common words that are found in WordNet. On the other hand, Google's labels have less corresponding synsets. Examples of labels without corresponding synsets are car models and dog breeds. Google tries to identify objects in greater detail.

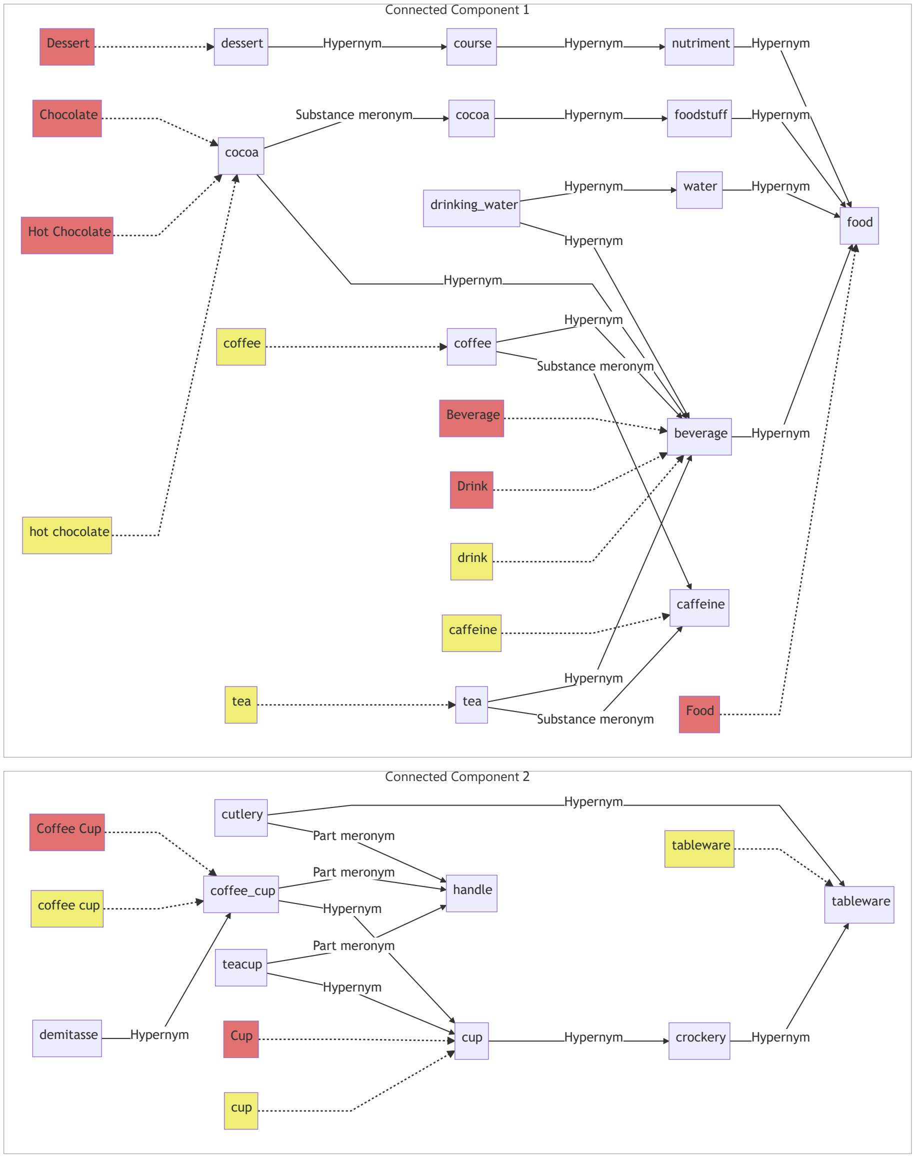

Fig. 2. Graph of labels from two concrete endpoints (red and yellow) and their associated synsets to related both words.

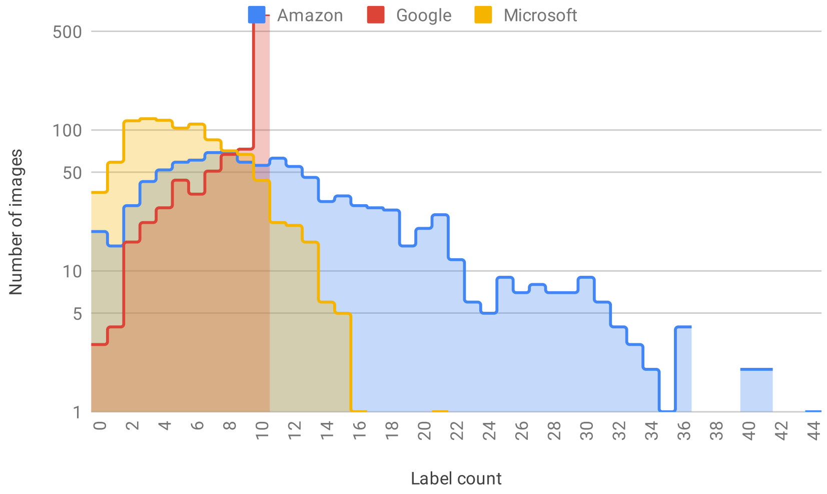

Fig. 3. Number of labels responded from our input dataset to three concrete APIs assessed.

3.2 Connected Components

A connected component (CC) is a subgraph in which there are paths between any two nodes. In graphs of labels and synsets, CCs are clusters of labels and synsets with similar meanings. For instance, there are two CCs in fig. 2. CC 1 in fig. 2 has *beverage*, *dessert*, *chocolate*, *hot chocolate*, *drink*, and *food* labels from the red first endpoint and *coffee*, *hot chocolate*, *drink*, *caffeine*, and *tea* labels from the yellow second endpoint. Therefore, these labels are related to *drinks*. On the other hand, CC 2 in fig. 2 has *cup* and *coffee cup* labels from the first red endpoint and *cup*, *coffee cup*, and *tableware* labels from the yellow second endpoint. These labels are, therefore, related to *cups*.

Figure 4 shows a distribution of number of CCs for 1,000-image label detections on Amazon, Google, and Microsoft APIs. The average number of CCs is 9.36 ± 3.49. The smaller number of CCs means that most of labels have similar meanings, while the larger number means that the labels are more disparate.

4 API Results Merging Algorithm

Our proposed algorithm to merge labels consists of four parts: (1) mapping labels to synsets, (2) deciding the total number of labels, (3) allocating the number of labels to CCs, and (4) selecting labels from CCs.

4.1 Mapping Labels to Synsets

Labels in responses are words in natural language and do not identify their intended meanings. For instance, a label *orange* may represent the fruit, the colour, or the name of the longest river in South Africa. To identify the actual meanings behind a label,

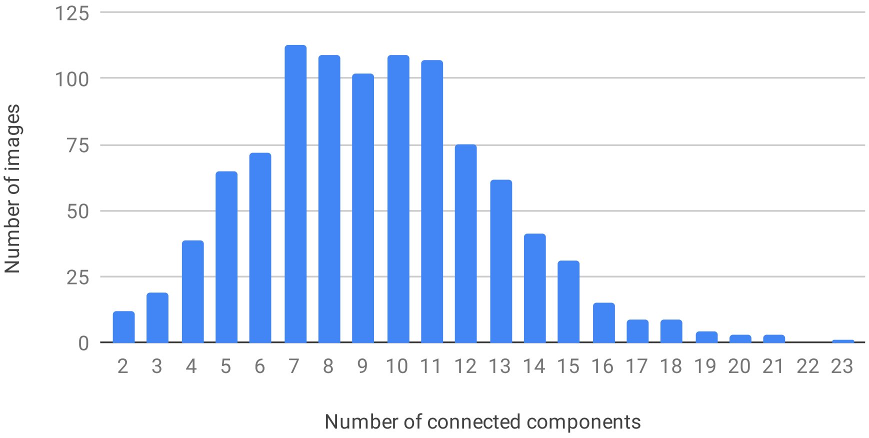

Fig. 4. Number of connected components

the façade enumerates all synsets corresponding to labels. It then finds the most likely synsets for labels by traversing WordNet links. For instance, if an API endpoint outputs the *orange* and *lemon* labels, the façade regards *orange* as the fruit. If an API endpoint outputs *orange* and *nile* labels, the façade regards *orange* as the river.

4.2 Deciding Total Number of Labels

The number of labels in responses from endpoints vary as described in section 3.1. The façade decides the number of merged labels using the numbers of labels from endpoints. A simple equation about number of labels is established.

$$\min_i(|R_i|) \leq \frac{\Sigma_i |R_i|}{n} \leq \max_i(|R_i|) \leq \Sigma_i |R_i|$$

Where $|R|$ is number of labels and scores in response, and n is number of endpoints. In case of naive operations in section 2.2, equations following are true.

$$\begin{aligned} |\text{merge}_{\max}(R_1,\ldots,R_n)| &\leq \min_i(|R_i|) \\ \max_i(|R_i|) \leq &|\text{merge}_{\min}(R_1,\ldots,R_n)| \leq \Sigma_i |R_i| \\ \max_i(|R_i|) \leq &|\text{merge}_{\text{average}}(R_1,\ldots,R_n)| \leq \Sigma_i |R_i| \end{aligned}$$

The proposal uses $\lfloor \Sigma_i |R_i| / n \rfloor$ to conform the necessary condition in section 4.3.

4.3 Allocating Number of Labels to Connected Components

The graph of labels and synsets is then divided into several CCs. The façade decides how many labels are allocated for each CC. In fig. 5, there are three CCs. Square-shaped nodes are labels in responses from endpoints. Text within these label nodes describe which endpoint outputs the label and score, for instance, "L-1a, 0.9" is label *a* from endpoint *1* with a score *0.9*. Circle-shaped nodes represent synsets, where the edges between the label and synset nodes are the relationships between them. Edges between synsets are links in WordNet.

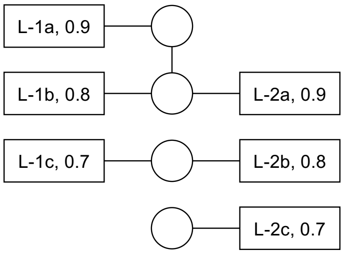

Fig. 5. Allocation to connected components.

Allegorically, allocating the number of labels to CCs is similar to proportional representation in a political voting system, where CCs are the political parties and labels are the votes to a party. Several allocation algorithms are introduced in proportional representation, for instance, D'Hondt method and Hare-Niemeyer method [17]. However, there are differences from proportional representation in a political parties context. For label merging, labels have scores and origin endpoints. This information may improve the allocation algorithm. For instance, CCs supported with more endpoints should have a higher allocation than CCs with fewer endpoints, and CCs with higher scores should have a higher allocation than CCs with lower scores. We introduce an algorithm to allocate number of labels to CCs. This allocates more to a CC with more supporting endpoints and higher scores. The steps of the algorithm are:

1. Sort scores separately for each endpoint.
2. If all CCs have an empty score array or more, remove one, and go to step 2.
3. Select highest score for each endpoint. Calculate product of highest scores.
4. A CC with the highest product score receives an allocation. This CC removes every first element from score array.
5. If requested number of allocation has been done, quit allocation. Otherwise, go to step 2.

Tables 2 to 5 are examples of allocation iterations. In *table 2*, the façade sorts scores separately for each endpoint. For instance, the first CC in fig. 5 has scores of 0.9 and 0.8 from endpoint 1 and 0.9 from endpoint 2. All CCs have a non-empty score array or more, so the façade skips step 2. The façade then picks the highest scores for each endpoint and CC. CC 1 has the largest product of highest scores and receives an allocation. In *table 3*, the first CC removes every first score in its array as it received an allocation in table 2. In this iteration, the second CC has largest product of scores and receives an allocation. In *table 4*, the second CC removes every first score in its array. At step 2, all the three CCs have an empty array. The façade removes one empty array from each CC. In *table 5*, the first CC receives an allocation. The algorithm is applicable if total number of allocation is less than or equal to $\max_i(|R_i|)$ as scores are removed in step 2. The condition is a necessary condition.

Table 2. Allocation iteration 1

Scores	Highest	Product	Allocated
[0.9, 0.8], [0.9]	[0.9, 0.9]	0.81	0+1
[0.7], [0.8]	[0.7, 0.8]	0.56	0
[], [0.7]	[NA, 0.7]	NA	0

Table 3. Allocation iteration 2

Scores	Highest	Product	Allocated
[0.8], []	[0.8, NA]	NA	1
[0.7], [0.8]	[0.7, 0.8]	0.56	0+1
[], [0.7]	[NA, 0.7]	NA	0

Table 4. Allocation iteration 3

Scores	Highest	Product	Allocated
[0.8], []			1
[], []			1
[], [0.7]			0

Table 5. Allocation iteration 4

Scores	Highest	Product	Allocated
[0.8]	[0.8]	0.8	1+1
[]	[NA]	NA	1
[0.7]	[0.7]	0.7	0

4.4 Selecting Labels from CCs

For each CC, the façade applies average operator in section 2.2, and takes labels with n-highest scores up to allocation in section 4.3.

4.5 Conformance to properties

Section 2.3 defines four properties: identity, commutativity, reflexivity, and additivity. Our proposed method conforms to these properties: **identity:** the method outputs same result if there is one response; **commutativity:** the method does not care about ordering of operands; **reflexivity:** the allocations to CCs are same to number of labels in CCs; and **additivity:** increases in score increases or does not change the allocation to the corresponding CC.

5 Evaluation

5.1 Evaluation Method

To evaluate the merge methods, we merged image label detection results from three representative image analysis API endpoints and compared these merged results against human-verified labels. Images and human-verified labels are sourced from 1,000 randomly-sampled images from Open Images Dataset V4 [7] Image-Level Labels test set.

The first three rows in table 7 are the evaluation of original responses from each API endpoint. Precision, recall, and F-measure in table 7 do not reflect actual values:

for instance, it appears that Google performs best at first glance, but this is mainly because of Google's label is similar to that of the Open Images Dataset label set.

The Open Images Dataset V4 uses 19,995 classes for labelling and human-verified labels for the 1,000 images of the test set contain 8,878 of these classes. Table 6 shows the correspondence between each APIs' labels and the Open Images Dataset classes. For instance, Amazon outputs 11,416 labels in total for 1,000 images. There are 1,409 unique labels in 11,416 labels. 1,111 labels out of 1,409 can be found in Open Images Dataset classes. Amazon's labels matches to Open Images Dataset classes at 78.9% ratio, while Google has an outstanding matched percentage of 94.1%. This high match is likely due to the Open Images Dataset also being provided by Google. An endpoint with higher matched percentage has a more similar label set to the Open Images Dataset classes. However, a higher matched percentage does not mean imply better quality of an API endpoint; it will increase apparent precision, recall, and F-measure only.

The true and false positive (TP/FP) label averages as well as the TP/FP ratio is shown in table 7. Where the TP/FP ratio is larger, the scores are more reliable. It is possible to increase the TP/FP ratio intentionally by adding more false labels with low scores. On the other hand, it is impossible to increase F-measure intentionally, because increasing precision will decrease recall, and vice versa. Hence, the importance of the F-measure statistic is critical here.

Let R_A, R_G, and R_M be responses from Amazon, Google, and Microsoft, respectively. There are four sets of operands, i.e., (R_A, R_G), (R_G, R_M), (R_M, R_A), and (R_A, R_G, R_M). Table 7 shows evaluation of each operands set. Table 8 shows averages of four operands sets. Figure 6 shows comparison of F-measure of methods.

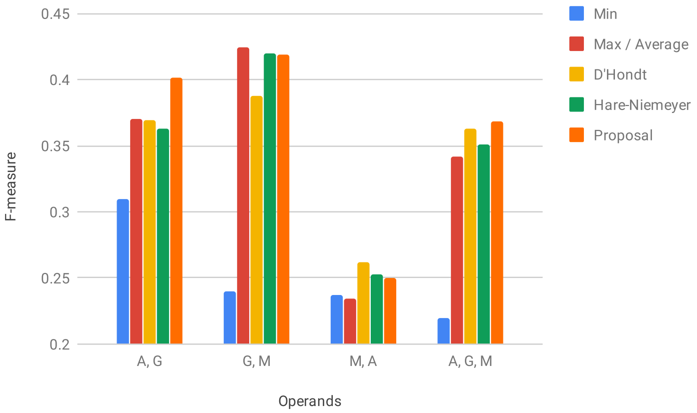

Fig. 6. F-measure comparison

5.2 Naive Operators

Results of *min*, *max*, and *average* operators are shown in tables 7 and 8 and fig. 6. The *min* operator is similar to *union* operator of set operations, and outputs all labels

Table 6. Matching to human-verified labels

Endpoint	Total	Unique	Matched	Matched %
Amazon	11,416	1,409	1,111	78.9
Google	8,766	2,644	2,487	94.1
Microsoft	5,392	746	470	63.0

Table 7. Evaluation result

Operands	Operator	Precision	Recall	F-measure	TP average	FP average	TP/FP ratio
A		0.217	0.282	0.246	0.848 ± 0.165	0.695 ± 0.185	1.220
G		0.474	0.465	0.469	0.834 ± 0.121	0.741 ± 0.132	1.126
M		0.263	0.164	0.202	0.858 ± 0.217	0.716 ± 0.306	1.198
A, G	Min	0.771	0.194	0.310	0.805 ± 0.142	0.673 ± 0.141	1.197
A, G	Max	0.280	0.572	0.376	0.850 ± 0.136	0.712 ± 0.171	1.193
A, G	Average	0.280	0.572	0.376	0.546 ± 0.225	0.368 ± 0.114	1.485
A, G	D’Hondt	0.350	0.389	0.369	0.713 ± 0.249	0.518 ± 0.202	1.377
A, G	Hare-Niemeyer	0.344	0.384	0.363	0.723 ± 0.242	0.527 ± 0.199	1.371
A, G	Proposal	0.380	0.423	0.401	0.706 ± 0.239	0.559 ± 0.190	1.262
G, M	Min	0.789	0.142	0.240	0.794 ± 0.209	0.726 ± 0.210	1.093
G, M	Max	0.357	0.521	0.424	0.749 ± 0.135	0.729 ± 0.231	1.165
G, M	Average	0.357	0.521	0.424	0.504 ± 0.201	0.375 ± 0.141	1.342
G, M	D’Hondt	0.444	0.344	0.388	0.696 ± 0.250	0.551 ± 0.254	1.262
G, M	Hare-Niemeyer	0.477	0.375	0.420	0.696 ± 0.242	0.591 ± 0.226	1.179
G, M	Proposal	0.414	0.424	0.419	0.682 ± 0.238	0.597 ± 0.209	1.143
M, A	Min	0.693	0.143	0.237	0.822 ± 0.201	0.664 ± 0.242	1.239
M, A	Max	0.185	0.318	0.234	0.863 ± 0.178	0.703 ± 0.229	1.228
M, A	Average	0.185	0.318	0.234	0.589 ± 0.262	0.364 ± 0.144	1.616
M, A	D’Hondt	0.271	0.254	0.262	0.737 ± 0.261	0.527 ± 0.223	1.397
M, A	Hare-Niemeyer	0.260	0.245	0.253	0.755 ± 0.251	0.538 ± 0.218	1.402
M, A	Proposal	0.257	0.242	0.250	0.769 ± 0.244	0.571 ± 0.205	1.337
A, G, M	Min	0.866	0.126	0.220	0.774 ± 0.196	0.644 ± 0.219	1.202
A, G, M	Max	0.241	0.587	0.342	0.857 ± 0.142	0.714 ± 0.210	1.201
A, G, M	Average	0.241	0.587	0.342	0.432 ± 0.233	0.253 ± 0.106	1.712
A, G, M	D’Hondt	0.375	0.352	0.363	0.678 ± 0.266	0.455 ± 0.208	1.492
A, G, M	Hare-Niemeyer	0.362	0.340	0.351	0.693 ± 0.260	0.444 ± 0.216	1.559
A, G, M	Proposal	0.380	0.357	0.368	0.684 ± 0.259	0.484 ± 0.200	1.414

Table 8. Average of evaluation result

Operator	Precision	Recall	F-measure	TP/FP ratio
Min	0.780	0.151	0.252	1.183
Max	0.266	0.500	0.344	1.197
Average	0.266	0.500	0.344	1.539
D’Hondt	0.361	0.335	0.346	1.382
Hare-Niemeyer	0.361	0.336	0.347	1.378
Proposal	0.257	0.242	0.360	1.289

of operands. The precision of the *min* operator is always greater than any precision of operands, and the recall is always lesser than any precision of operands. *Max* and *average* operators are similar to *intersection* operator of set operations. Both operators output intersection of labels of operands. There is no clear relation to precision and recall of operands. Since both operators have same precision, recall, and F-measure, fig. 6 groups them into one. The *average* operator performs well on TP/FP ratio. Most of same labels from multiple endpoints are true positives. In any cases of four operand sets, all naive operators' F-measures are between F-measures of operands. None of naive operators improve results by merging responses from multiple endpoints.

5.3 Traditional Proportional Representation Operators

There are many existing allocation algorithms [17] in proportional representation, e.g., D'Hondt and Hare-Niemeyer methods. These methods may be replacements of those in section 4.3. Other steps, i.e. sections 4.1, 4.2 and 4.4, are same as for our proposed technique. Tables 7 and 8 and fig. 6 show result of these traditional proportional representation algorithms. Averages of F-measures by traditional proportional representation operators are almost equal to that of *max* and *average* operators. It is worth noting that merging M and A results in a better F-measure than each F-measure of M and A individually. Because endpoints M and A are not biased to human-verified labels, situations in the real world should, therefore, be similar to the case of M and A. So, RQ1 is true.

5.4 New Proposed Label Merge Technique

As shown in table 8, our proposed new method performs best in F-measure. Instead, TP/FP ratio is less than average, D'Hondt, and Hare-Niemeyer. As described in section 5.1, we argue that F-measure as more important than TP/FP ratio in this case. Therefore, RQ2 is true. Shown in table 7, our proposed new method improves the results when merging M and A non-biased endpoints. It is similar to traditional proportional representation operators, but performs less well than them. However, it performs better on other operand sets, and performs best on overall as shown in fig. 6.

5.5 Performance

We used AWS EC2 m5.large instance (2 vCPUs, 2.5 GHz Intel Xeon, 8 GiB RAM); Amazon Linux 2 AMI (HVM), SSD Volume Type; Node.js 8.12.0. It takes 0.370 seconds to merge responses from three endpoints. Computational complexity of the algorithm in section 4.3 is $O(n^2)$, where n is total number of labels in responses. The estimation assumes that the number of endpoints is a constant. Complexity of step 1 in section 4.3 is $O(n \log n)$, because the worst case is that all n labels are from one single endpoint and all n labels are in one CC. Complexity of step 2 to 5 is $O(n^2)$ because number of CCs is less than or equal to n and number of iterations are less than or equal to n. As table 1 shows, the averaged total number of three endpoints is 25.58. Most of time for merging is consumed by looking up WordNet synsets (section 4.1). The API façade calls APIs on actual endpoints in parallel. It takes about 5 seconds, which is much longer than 0.370 seconds taken for the merging of responses.

6 Conclusions and Future Work

In this paper, we propose a method to merge responses from intelligent APIs. Our method merges API responses better than naive operators and other proportional representation methods (i.e., D'Hondt and Hare-Niemeyer). The average of F-measure of our method marks 0.360; the next best method, Hare-Niemeyer, marks 0.347. Our method and other proportional representation methods are able to improve the F-measure from original responses in some cases. Merging non-biased responses results in 0.250 of F-measure, while original responses have an F-measure between 0.246 and 0.242. Users can improve their applications' precision by small modification, e.g. changing endpoint URL from API endpoints to façades. Performance impact by applying façades is small, because overhead in façades is much smaller than API invocation. The proposal method conforms identity, commutativity, reflexivity, and additivity properties. These properties are advisable for integrating multiple responses.

Our idea of a proportional representation approach can be applied to other intelligent APIs. If response type is a list of entity and score, and if there is a way to group entities, a proposal algorithm can be applied. The opposite approach is to improve results by inferring labels. Our current approach picks some of the labels returned by endpoints. intelligent APIs are not only based on supervised machine learning. Thus to cover a wide range of intelligent APIs, it is necessary to classify and analyse APIs, and establish a method to improve results by merging. Currently graph structures of labels and synsets (fig. 2) are not considered when merging results. Propagating scores from labels could be used, losing the additivity property but improving results for users. There are many ways to propagate scores. For instance, setting propagation factors for each link type would improve merging and could be customised for users' preferences. It would be possible to generate an API façade automatically. APIs with same functionality have same or similar signatures. Machine-readable API documentation, for instance, OpenAPI Specification, will help a generator to build an API façade.

References

[1] AWS: Amazon rekognition, https://aws.amazon.com/rekognition/
[2] Clarifai, Inc: Clarifai, https://www.clarifai.com
[3] Deep AI, Inc: Image Recognition API, https://deepai.org/ai-image-processing
[4] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust Physical-World Attacks on Deep Learning Visual Classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1625–1634 (2018)
[5] FileShadow: Fileshadow delivers machine learning to end users with google vision api, https://www.businesswire.com/news/home/20180723005503/en/FileShadow-Delivers-Machine-Learning-Users-Google-Vision
[6] Google: Google cloud vision api, https://cloud.google.com/vision/
[7] Google: Open images dataset v4, https://storage.googleapis.com/openimages/web/index.html

[8] Hwang, K.: Cloud Computing for Machine Learning and Cognitive Applications: A Machine Learning Approach. MIT Press (2017)

[9] IBM: Tone analyzer, https://www.ibm.com/watson/services/tone-analyzer/

[10] IBM: Watson Visual Recognition, https://www.ibm.com/watson/services/visual-recognition/

[11] Imagga: Imagga's API, https://imagga.com

[12] Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. arXiv.org (Jul 2016)

[13] Lyons, R.E., Vanderkulk, W.: The use of triple-modular redundancy to improve computer reliability. IBM Journal of Research and Development **6**(2), 200–209 (April 1962). https://doi.org/10.1147/rd.62.0200

[14] Media, G., Communications: Mapillary and amazon rekognition collaborate to build a parking solution for us cities through computer vision, https://www.geospatialworld.net/news/mapillary-and-amazon-rekognition-collaborate/

[15] Microsoft: Microsoft azure computer vision api, https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/

[16] Miller, G.A.: Wordnet: A lexical database for english. Commun. ACM **38**(11), 39–41 (Nov 1995). https://doi.org/10.1145/219717.219748, http://doi.acm.org/10.1145/219717.219748

[17] Niemeyer, H.F., Niemeyer, A.C.: Apportionment methods. Mathematical Social Sciences **56**(2), 240 – 253 (2008). https://doi.org/https://doi.org/10.1016/j.mathsocsci.2008.03.003, http://www.sciencedirect.com/science/article/pii/S0165489608000358

[18] Pearl, J.: The Seven Tools of Causal Inference with Reflections on Machine Learning (2018)

[19] Pezzementi, Z., Tabor, T., Yim, S., Chang, J.K., Drozd, B., Guttendorf, D., Wagner, M., Koopman, P.: Putting Image Manipulations in Context: Robustness Testing for Safe Perception. IEEE International Symposium on Safety, Security, and Rescue Robotics SSRR pp. 1–8 (Apr 2018)

[20] Ribeiro, M., Grolinger, K., Capretz, M.A.M.: MLaaS: Machine Learning as a Service. In: 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). pp. 896–902. IEEE (Dec 2015)

[21] Si, L., Callan, J.: A semisupervised learning method to merge search engine results. ACM Trans. Inf. Syst. **21**(4), 457–491 (Oct 2003). https://doi.org/10.1145/944012.944017, http://doi.acm.org/10.1145/944012.944017

[22] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv:1312.6199 (2013)

[23] Talkwalker: Image Recognition for Visual Social Listening, https://www.talkwalker.com/image-recognition

[24] TheySay: Sentiment analysis api, http://www.theysay.io/sentiment-analysis-api/