Exploiting Equivalence to Efficiently Enhance the Accuracy of Cognitive Services

Aabhas Bhatia[1], Shuangyi Li[2], Zheng Song[2] and Eli Tilevich[2]

[1]Microsoft, Redmond, USA
[2]Software Innovations Lab, Dept. of Computer Science, Virginia Tech, USA
aabhatia@microsoft.com,{amnos,songz,tilevich}@vt.edu

***Abstract*—Equivalent services deliver the same functionality with dissimilar non-functional characteristics, including latency, accuracy, and cost. With these dissimilarities in mind, developers can exploit the combined execution of equivalent services to increase accuracy, shorten latency, or reduce cost. However, it remains unknown how to effectively combine equivalent services to satisfy application requirements. With the recent surge in popularity of machine learning, different vendors offer a plethora of equivalent services, whose characteristics are mostly undocumented. As a result, developers cannot make an informed decision about which service to select from a set of equivalent services. To address this problem, we explore different service combination strategies (i.e., majority voting, weighted-majority voting, stacking, and custom) to ascertain their impact on non-functional characteristics. In particular, we study how these strategies impact the accuracy, cost, and latency of the face detection task and validate our findings on the sentiment analysis task. We consider the combined executions of commercial web services, deployed in the cloud, and open-source implementations, deployed as edge services. Our evaluation reveals that the combined execution of equivalent services is most effective for improving cost and latency. Informed by our experimental results, we formulate practical guidelines to help developers identify the best execution strategy for a given set of services.**

***Index Terms*—Equivalent Services , Combined Use, QoS Optimization**

I. Introduction

Due to the proliferation of the service-oriented architecture (SOA), both web services, provided by different cloud vendors, and edge services, hosted on local computing resources can satisfy the same functional requirements. For example, competing cloud vendors (e.g., Google, IBM, Amazon, and etc.) provide machine learning (ML) web services that include face detection, speech recognition, language translation, and event detection. Numerous popular open-source libraries, hosted on in-house devices and exposed as edge services, also provide the same functionality. *Equivalent* services satisfy the same functional requirement, but can differ widely in their non-functional characteristics, thus presenting an implementation choice to the developer.

Quality of service (QoS) is a meta-descriptor that encompasses various non-functional characteristics, including latency, cost, and accuracy. When exposing a functionality as a service, developers face a difficult dilemma: which equivalent service would best meet given QoS requirements. A straightforward procedure for identifying the optimal service is to compare the available equivalent services, and select the one whose actual performance is the closest to the requirements. Meanwhile, prior research [1] indicates that combining multiple equivalent services can better meet the QoS requirements. Nevertheless, none of the prior works have focused on assessing the impact of combined service execution on the overall accuracy of the combination. That is, existing works focus on either enhancing accuracy itself via majority voting, stacking, or weighted voting, or optimizing reliability, cost, and latency via service composition. As a result, developers lack actionable insights for optimally leveraging the combined executions of equivalent service to efficiently improve the accuracy, especially for cognitive tasks.

Without such insights, developers mostly rely on their intuition, which can be misleading. For example, the intuition would suggest that accuracy would improve if *the majority voting* strategy were applied to the results of executing multiple equivalent services [2]. However, our findings indicate that this intuition is in-fact incorrect. Meanwhile, we observe that applying the *stacking* strategy reduces both latency and cost. This paper bridges the knowledge gap of how developers can exploit the combined use of equivalent services to better meet the QoS requirements.

We experimentally study the combined execution of equivalent services and the resulting QoS impact. We choose `face detection` as an exploration use case, and `sentiment analysis` as a validation use case. For both use cases, we derive the performance characteristics of five equivalent services. Three of them are hosted by major cloud providers; the other two are locally hosted edge services based on open-source implementations. The `use case 1` studies the accuracy, cost, and latency of different combined executions of these services to form practical guidelines for finding optimal execution strategies. The `use case 2` validates and bolsters our guidelines. In particular, we apply the majority voting, weighted-majority voting, stacking, and our custom execution strategies to ascertain their performance impact on accuracy and QoS.

The contribution of this paper is three-fold:

1) **An experimental study** that explores how the combined

This research was performed when the first author was with the Software Innovations Lab at Virginia Tech.

execution strategies above impact non-functional characteristics.

2) **A customized execution strategy**, a new approach that combines the execution of equivalent services to fulfill the QoS requirements better than the existing strategies.
3) **Practical guidelines** that inform developers about which service or a combination thereof would meet the QoS requirements.

The rest of this paper is organized as follows: Section II introduces the conceptual foundations for our study. Section III introduces our study's experimental setup. Section IV presents our observations and guidelines. Section V validates and generalizes our guidelines. Section VI expands on the experimental caveats of our evaluation. Section VII discusses the limitations of our approach and the future work. Section VIII summarizes the related state of the art. Section IX presents concluding remarks.

II. Conceptual Foundations

This work's conceptual foundations are grounded in SOA and the strategies used to select, fuse, and execute services. We introduce these concepts in turn next.

A. Services Composition

Microservices, an SOA variant, has become immensely popular in enterprise software development [3]. An application is structured as a collection of loosely coupled microservices, with each providing a distinct functionality. The problem of composing microservices to create new functionalities by aggregating different services has been studied extensively [34]. Prior research proposes and evaluates approaches that select microservices based on their QoS characteristics. Some approaches select microservices based on their execution semantics; they define a QoS ontology and extend QoS attributes to select microservices [4], [5]. Other approaches compute the QoS characteristics of microservices to meet the user's overall requirements [6]–[10]. Some strategies compute the microservice characteristics using advanced computational techniques that include Integer Programming [11], Genetic Algorithms [6], [8], and Constraint Programming [12]. Some prior research [1], [35] focuses on combining equivalent microservices to improve QoS, and provide different approaches to estimate the cost, latency, and reliability of equivalent microservice combinations. However, none of these existing works apply service composition to improve QoS with accuracy. In this work, we exploit the combined execution of web services with accuracy as one of the major QoS concerns.

B. Service Fusion Strategies

We expand the traditional notion of QoS with the accuracy characteristic, which is specific to the ML domain. To the best of our knowledge, the accuracy characteristic has not yet been explored in concert with the traditional QoS characteristics for the problem of selecting services. Furthermore, we draw inspiration from a large body of prior works in the domain of classifier fusion [13]–[17]. These works study how various fusion methods applied to a classifier improve the overall accuracy for the same classification task. They also explore how the diversity of the classifiers impacts the overall accuracy of their combination. In this work, we choose to explore three fundamental effective fusion strategies, described in turn next:

1) Majority Voting: The majority voting fusion methods work as follows. From a set of N classifiers, the combined execution of N services predicts "c" as the class label if "c" is the most frequently predicted label. In other words, we look at the individual predictions of each service and choose the label which was predicted the largest number of times. Prior research shows how majority voting applied to individual classifiers can improve overall accuracy [17], [18]. In our study, we enumerate all possible combinations of 5 services for $N = \{2, 3, 4, 5\}$. 5 services can be combined in $\binom{5}{2} + \binom{5}{3} + \binom{5}{4} + \binom{5}{5} = 10 + 10 + 5 + 1 = 26$ different ways. Further, we execute the services comprising a combination in parallel and calculate the cost, accuracy, and latency of each combination. The cost of a combination is calculated by summing up the costs incurred by each comprising service. The latency of the combination is calculated by measuring the maximum latency among the comprising services. The accuracy is calculated by comparing the predicted outcome of the combination against the ground truth value.

2) Weighted Majority Voting: In vanilla majority voting, we assume that all services are equally accurate. For example, when fusing three services, an agreement between any two of them is sufficient to determine the outcome. It may be the case that two equally inaccurate services would agree more often, thus bringing down the overall accuracy of the combined execution. Therefore, we also examine the strategy of weighted majority voting, in which weights are assigned to the predicted outcome of each service. In our experiments, we assign weights by exhaustively enumerating all possible weight combinations with the step size of 0.05. The weight distribution determines how accurate a given combination is expected to be. For classification tasks, the predicted label which has the highest weight attached to it becomes the combination's prediction. Table I illustrates five services A-E, their respective weights, and their respective predictions for one data point. The maximum weight is attached to label "-1" i.e. 0.40. Hence, the combination predicts "-1" as the label for this setting. The characteristics of cost, latency, and accuracy are calculated similarly to Majority Voting.

Service	A	B	C	D	E
Weights	0.20	0.20	0.10	0.35	0.15
Predicted Label	-1	-1	0	1	0

TABLE I
Weighted Majority Voting

3) Stacking: The Ensemble methods execute multiple ML algorithms to improve the predictive performance. A typical Ensemble method involves training constituent algorithms using the outcome of previously executed algorithms. In contrast, the Stacking Ensemble method trains a meta-algorithm on the predictions of individual learning algorithms. The predicted values from each algorithm are used as the feature set for the meta-learner. The new model shares the same prediction objective with the constituent algorithms. Stacking has become a popular fusion strategy, as it strikes a good balance between its ease of implementation and effectiveness. Simple statistical models, such as logistic regression, are often used as the meta-learner to improve overall accuracy. Consider three classifiers C1, C2, C3. For an incoming picture x_i, the predicted labels of C1, C2, C3 are y_1, y_2, y_3. The set $\{y_1, y_2, y_3\}$ is passed as input to a logistic regression model for predictions. Again, we consider all possible 26 enumerations for 5 services and train the vanilla logistic regression model with default parameters. We choose default parameters and only have one layer of stacking, so as to keep the process simple, avoiding any unnecessary engineering overhead. The characteristics of cost, latency, and accuracy are calculated similarly to majority voting.

C. Our Custom Service Execution Strategy

The aforementioned fusion strategies process the results of individual services and process them differently to reach the final prediction. An issue that is orthogonal to processing the results is how these services are invoked. A set of services can be invoked in sequence, in parallel, or in combination thereof. These execution strategies have a strong impact on various non-functional characteristics. In practical applications, accuracy can rarely be treated as the only relevant concern. Hence, we further consider how to balance different QoS characteristics while maximizing the overall utility for the combined execution of services. Next, we define our custom execution strategy.

Our execution strategy comprises a sequence of steps. Each step executes a set of services in parallel, with their execution results evaluated to see if they agree. The execution proceeds until the current step's result determines the final outcome. For our running example, the outcome of a step is determined via majority voting: if over 50% of the executed services agree on one category (i.e., detecting a face or not), the agreed upon category becomes the final outcome. Otherwise, if one half of the results agrees on one category, while the other half agrees on the other category, the final outcome cannot be determined, so the execution proceeds to the next step.

By combining services in such sequential steps, the execution strategy above can greatly reduce the incurred costs as compared with having to execute all services at once. To reduce the overall latency, services with longer response time can be executed in the last steps. Although executing all services at once would yield a higher accuracy, custom execution strategies can improve the overall utility by reducing the execution cost and latency.

In the subsequent discussion, we use the following notation. "$*$" denotes services executed in parallel, and "$-$" denotes consecutive sequential groups of services. For a set of services s_1, s_2, s_3, s_4, s_5, $s_1 * s_2 - s_3 * s_4 * s_5$, $s_1 * s_2 - s_3 * s_4 - s_5$, and $s_1 * s_2 * s_3 * s_4 * s_5$ represent possible strategies. Our custom strategy can be generally expressed as $(2n) * -(2n+1)$, where n is a positive integer. The last group must have an odd number of services, while every other group must have an even number of individual services.

III. Experimental Setup

In this section, we discuss our experimental dataset and the evaluated services used in our analysis.

A. Dataset and Services

1) Face Detection: We first study the task of detecting a human face in an image: determine if a given image contains a face. This task is widely performed in social media analytic. In general, facial recognition is used to infer user demographics to study trends and target advertisements. Traditionally requiring tedious manual processing, facial recognition has become automated due to recent advances in ML [19]. In fact, we found several public web services that offer this functionality. Specifically, in our experiments, we use cloud-based services provided by IBM [20], Microsoft [21], and Face++ [22]. We also deploy two edge services that offer the same functionality, implemented by means of OpenCV [23] and Deep Learning [24], popular open-source software packages. We choose these particular packages due to their high stargazer counts on GitHub.

For our experimental evaluation, we construct a large balanced dataset of 8,000 images. 4,000 images come from the IMDB-Wiki dataset [25], picked at random. Each image has at least one face appearing in it. The rest of 4,000 images are chosen at random from the Caltech 101 dataset [26], with none of the images containing a face.

2) Sentiment Analysis: We validate our findings on the task of analyzing the sentiment in a body of text: determine if a given body of text's sentiment is positive, neutral, or negative. This functionality is commonly applied in e-commerce for product recommendations based on customer reviews. The sheer volume of customer reviews posted online defeats manual analysis, making automatic sentiment analysis a valuable mechanism in modern e-commerce applications. In fact, we found several cloud-based services that offer this functionality, provided by IBM [27], Microsoft [28], and Google [29]. In addition, we locally deploy two edge services that offer the same functionality by means of TextBlob [30] and Vader [31], popular open-source packages. We select these packages because they have the highest startgaze counts across all Python implementations. We choose the Twitter dataset [32], comprised of labelled tweets about various airlines. We choose 1,945 tweets for each label (i.e., positive, neutral, and negative). Unfortunately, we could not identify a larger labelled dataset for sentiment analysis, as larger datasets were only labelled with the positive and negative sentiments. Choosing a stratified data set (i.e., in which all classes are equally represented) is essential to minimize evaluation bias.

B. Individual Service Characteristics as the Benchmark

We empirically evaluate the undocumented non-functional characteristics for each single service (i.e., accuracy and

Service	IBM	MS	Face++	DL	OpenCV
Cost($)/1000 Requests	4	1	1	0.01	0.01
Latency(in ms)	737.51	94.99	95.73	65.6	56.44
Accuracy(%)	96.07	86.43	95.51	82.16	88.02

TABLE II
NON-FUNCTIONAL CHARACTERISTICS OF FACE DETECTION SERVICES

Service	IBM	MS	Google	Blob	Vader
Cost($)/1000 Requests	3	2	1	0.01	0.01
Latency(in ms)	155.19	73.53	126.93	1.08	13.25
Accuracy(%)	66.74	62.51	61.92	54.25	58.44

TABLE III
NON-FUNCTIONAL CHARACTERISTICS OF SENTIMENT ANALYSIS SERVICES

latency) by invoking each service to record its classification outcome and execution time. Thereafter, we average all the data points to compute the overall accuracy and the average latency. Vendors always explicitly specify the cost of invoking a cloud service. We approximate the cost of invoking an edge service as two orders of magnitude less than the cheapest cloud service for the same functionality. This cost model accounts for the local deployment of an edge service consuming resources, such as CPU processing and power. Tables II and III summarize our empirical findings for the images and tweets data sets, respectively.

Observation 1. Compared to commercial cloud services, open-source edge services offer lower latency, cost, and accuracy.

Even given the numbers in Table II, a developer would still find it difficult to select the best service for an application scenario. For example, IBM offers the maximum accuracy, but it also incurs the maximum cost and latency. We use the numbers reported in Table II and III as benchmarks to evaluate how well service combinations perform. Since in both use cases IBM offers the best accuracy, our goal is to find combinations that outperform IBM.

IV. FORMULATING GUIDELINES: USE CASE 1

By carrying out this use case, we formulate our service combination guidelines.

A. Exploring Impact on Accuracy

For ML services, accuracy is a major non-functional characteristic and an important target of our study. We evaluate and explore how each of the following four strategies impacts the accuracy of combined executions in this use case.

1) Majority Voting: Counter-intuitively, the majority voting strategy offers no increase in accuracy. Trying to understand why, we notice that the services are correlated and dependent. That is, if one service is observed to return incorrect result for a data point, it is highly likely that the others will do the same for the said data point. Fig 1 depicts a sample of images that all services misclassify.

This correlation is further revealed by calculating the joint probabilities of the services. The probability of a service being correct is the same as its accuracy. We define $P(A)$ as the probability of classifier A being correct, and $P(AB)$ as the probability of classifiers A and B being both correct. Theoretically, we expect accuracy to improve if the services are not correlated. For example, when using majority voting, the combined accuracy for three equivalent services A, B, and C is: $P(AB)$ + $P(BC)$ + $P(AC)$ - $2P(ABC)$. If we plug in $P(A)$, $P(B)$, and $P(C)$ with $P(IBM) = 96.07\%$, $P(MS)$ = 86.43%, and $P(Face++)$ = 95.51% as shown in Table II. Assuming the IBM, MS, and Face++ are not correlated, their expected combined accuracy would be 98.72%, a significant improvement over the accuracy of each individual service; however, the actual combined accuracy is only 96.02% due to the services being strongly correlated.

Observation 2. Correlation/dependence among the individual services limits majority voting from offering higher overall accuracy.

2) Weighted Majority Voting: Our initial motivation for exploring weighted majority voting was to reduce the negative impact of inaccurate services by assigning them lower weights. After observing that the services are correlated, we want to reduce the impact of the correlation as well by experimenting with the weights.

For our running example of face detection, we observe an increase in accuracy. However, this increase is accompanied by an increase in cost. We do find a select few combinations that both improve accuracy and reduce cost as compared to IBM. However, the improvement in accuracy for such combinations ranges between 0-2%. Table IV reports the combinations, their respective weights, and the accuracy improvements.

3) Stacking: As expected, stacking increases the overall accuracy for the combined execution. Similar to weighted majority voting, the increase in accuracy is accompanied by an increase in cost and the combinations with comparable cost report an insignificant (i.e., 0-2%) increase in accuracy. Table V reports the combinations, their respective weights, and the accuracy improvements.

Observation 3. Irrespective of correlation, weighted majority voting and stacking can increase the overall accuracy by adding services to a combination.

4) Our Custom Execution Strategy: Our custom execution strategy provides no combination with improved overall accuracy. Similar to aforementioned strategies, it does provide combinations with comparable accuracy while reducing the incurred cost and latency. Table VI reports some of the combinations found by our custom execution strategy. Algorithm 1 below describes how the characteristics of our custom strategy are computed.

Three out of the four strategies offer us combinations with comparable overall accuracy with reduced incurred cost and latency. Hence, we further explore in search of combinations offering better overall utility. Following section summarizes our exploration.

Observation 4. Irrespective of correlation, weighted majority voting, stacking, and our custom execution strategy can reduce

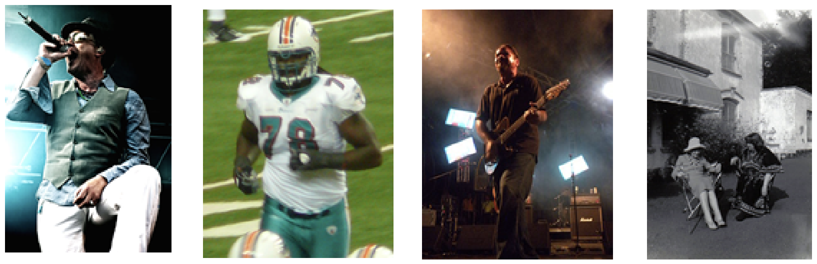
Fig. 1. Images with faces misclassified by all services

Combination	Weights	Accuracy(%)	Cost($)	Latency(ms)
IBM	1	96.07	4	737.51
MS & Face++	0.50,0.50	97.19	2	95.37
Face++ & OpenCV & DL	0.50,0.45,0.05	96.31	1.02	95.37
Face++ & OpenCV	0.50,0.50	96.37	1.01	95.37
Face++ & DL	0.50,0.50	96.31	1.01	95.37

TABLE IV
INCREASE IN ACCURACY WITH WEIGHTED MAJORITY VOTING

Combination	Accuracy(%)	Cost($)	Latency(ms)
IBM	96.07	4	737.51
MS & Face++ & OpenCV	97.19	2.01	95.37
MS & Face++ & DL	97.19	2.01	95.37
Face++ & OpenCV	96.37	1.01	95.37
Face++ & DL	96.31	1.01	95.37

TABLE V
INCREASE IN ACCURACY WITH STACKING

cost and latency while offering comparable accuracy.

B. Exploring QoS Balance

1) Calculating the Utility Index: Our goal is to find a combination that improves the overall utility. Utility is defined as the overall performance of a service, used as a singular metric for selecting services. Based on prior work [33], we use Eq. 1 to define the utility $u(e)$ of a combined execution e as follows:

$$u(e) = w_a * \frac{a_e - a_l}{a_{range}} + w_c * \frac{c_h - c_e}{c_{range}} + w_l * \frac{l_h - l_e}{l_{range}}, \quad (1)$$

where w_a, w_c, and w_l denote the weights of accuracy, cost, and latency, respectively, while a_{range}, c_{range}, and l_{range} denote their ranges. c_h and l_h denote the higher bound of cost and latency, while a_l denotes the lower accuracy bound. The ranges, higher, and lower bounds are derived by using the evaluation values of individual services. In general, to improve the utility of a combined execution, a developer can increase its overall accuracy and/or decrease its cost and latency.

2) Estimating QoS Characteristics for Custom Execution Strategy: Algorithm 1 details the procedure, which estimates the cost, latency, and accuracy of our custom execution plan. $P(E, a, b)$ denotes the possibility that for a set E of equivalent services, a of them would generate accurate results, and b of them would generate inaccurate results.

3) Results: Through an exhaustive search, we find an optimal execution strategy for a given set of QoS weights. We estimate the QoS characteristics of all possible execution strategies using Algorithm 1 and calculate their respective utilities according to Eq. 1. For a given set of weights, we retain the execution strategy with the highest $U(e)$.

Algorithm 1 Estimating the QoS of Our Custom Execution Strategy

```
Input Execution plan $EP = E_0 - E_1 - .... - E_n$, where $E_n$ denotes the $n$th parallel group, i.e., $E_n = e_n^0 * e_n^1 * ... * e_n^m$.
$|E_n| = m + 1$, number of services in a parallel group
For $E_0, E_1, ...E_{n-1}$, $(m+1)\%2 == 0$;
For $E_n$, $(m+1)\%2 == 1$.
Output Accuracy $Acc$, Cost $C$, Latency $Lat$;
1:  Acc, C, Lat ← 0;
2:  for $i \leftarrow 0$ to n do
3:      $P_{previousFail} = \prod_{j=0}^{i-1} P(E_j, |E_j|/2, |E_j|/2)$;
4:      $currCost = \sum_{k=0}^{|E_i|} Cost(e_i^k)$;
5:      $currLat = \max(Lat(e_i^k)) : k \in \mathbb{Z} \wedge 1 \leq k \leq |E_i|)$;
6:      if $i! = n$ then
7:          $P_{iSucceed} = \sum_{k=|E_i|/2+1}^{|E_i|} P(E_i, k, |E_i| - k)$;
8:      else
9:          $P_{iSucceed} = \sum_{k=(|E_i|+1)/2}^{|E_i|} P(E_i, k, |E_i| - k)$;
10:     end if
11:     Acc $+ = P_{previousFail} * P_{iSucceed}$;
12:     C $+ = P_{previousFail} * currCost$;
13:     Lat $+ = P_{previousFail} * currLat$;
14: end for
```

We vary the weights of the QoS characteristics to simulate different concerns and evaluate the execution strategies that can satisfy them. We allot the highest weight to one characteristic at a time, demarcating it as the major QoS concern. Table VI shows the generated execution strategies that offer the maximum utility for different QoS concerns. We observe that by combining edge and cloud web services, a developer can balance the QoS characteristics and achieve a higher overall utility than by invoking individual services.

Observation 5. Custom execution strategy offers the most

Weight (a, c, l)	generated strategy	(acc(%), cost($), latency(ms))
(high, low, low)	$MS * FacePlusPlus - IBM$	(94.2,2.31,156)
(low, high, low)	$DL * OpenCV - FacePlusPlus$	(92.6,0.14,76)
(low, low, high)	$DL * OpenCV - FacePlusPlus$	(92.6,0.14,76)

TABLE VI
QoS CONCERNS AND GENERATED PLANS

Combination	Weights	Accuracy(%)	Cost($)	Latency(ms)
IBM	100	66.77	4	155.19
MS & Google & Blob	15,45,40	64	3.01	126.93
MS & Google & Vader	40,35,25	65.48	3.01	126.93

TABLE VII
IMPROVING COST AND LATENCY WITH WEIGHTED MAJORITY VOTING

optimal combination in terms of the utility provided for varying QoS concerns.

C. Guidelines

Based on our observations, we recommend that developers follow a systematic approach to select and combine services. Our guidelines are as follows:

1) If accuracy is the only concern and should be improved, utilize the stacking strategy. Combining a higher number of individual services is expected to provide higher overall accuracy.
2) If the cost and latency need to be considered as well, utilize either weighted majority or stacking. Both strategies offer combinations with comparable accuracy, with reduced cost and latency.
3) If the overall utility is the deciding criterion, our custom execution strategy offers combinations with the most optimal overall utility.

V. VALIDATING THE GUIDELINES: USE CASE 2

In this section, we check whether our findings of the face detection in the use case 1 apply to the unrelated use case 2. We choose the domain of sentiment analysis. If both use cases yield similar insights, this outcome would bolster our claims, generalizing our guidelines.

A. Impact on Accuracy

By evaluating the service probabilities, we find that the expected probability of a combination formed via majority voting differ from the observed probability. Hence, the evaluated services are again correlated. Therefore, no useful combination via majority voting can increase the overall accuracy, thus confirming our prior findings.

For weighted majority voting and stacking, as long as combinations exclude IBM, none of them is more accurate than IBM, possibly due to the low size of the training data set. However, both strategies offer combinations, whose overall accuracy is comparable, but cost and latency are lower. Table VII reports the combinations and weights for the weighted majority voting strategy that improve cost and latency, without significantly decreasing accuracy. Similarly, Table VIII reports the combinations for the stacking strategy.

Combination	Accuracy(%)	Cost($)	Latency(ms)
IBM	66.77	4	155.19
MS & Google & Blob	67.01	3.01	126.93
MS & Google & Vader	66.4	3.01	126.93
MS & Blob & Vader	66.3	1.02	73.53

TABLE VIII
IMPROVING COST AND LATENCY WITH STACKING FOR SENTIMENT ANALYSIS

B. QoS Balance

Table IX shows the most optimal strategies for given sets of weights assigned to QoS characteristics. For each set, our custom execution strategy offers the highest overall utility, thus confirming the guideline 3 above.

Compared to the most accurate single service, our custom execution strategy provides a combination with: 0.003% increase in accuracy, 68.6% decrease in cost, and 40% decrease in latency, when accuracy is the primary concern; 0.011% decrease in accuracy, 89% decrease in cost, and 47% decrease in latency, when cost is the primary concern; and 0.003% increase in accuracy, 68.6% decrease in cost, and 40% decrease in latency, when latency is the primary concern.

VI. EXPERIMENTAL HEURISTICS

In this section, we touch upon some of the most important heuristics we developed to overcome the caveats of our problem domain.

A. Class Labels for Sentiment Analysis

While evaluating the accuracy of services for sentiment analysis, we found that four of the services return back a score and not a label. The fifth service, IBM, returns a label and a score. Meanwhile, out of the four services returning scores, only one of them specifies how to interpret the score. Vader in its documentation instructs to treat a body of text with the score above 0.05 as containing a positive sentiment, one with the score below -0.05 as containing a negative sentiment, and one with the score between -0.05 and 0.05 as containing a neutral sentiment. For the remaining three services, we lack any specifications for the treatment of the returned scores. This omission presents a complication, as without specific labels we are unable to gauge the accuracy of these services.

We optimize the classification ranges for these three services through the following procedure. Find a value x that can be used to interpret the returned score. A score value above x is assigned a positive label, one below $-x$ a negative label, and one between $-x$ and x a neutral label. We also observe

Weight (a,c,l)	generated strategy	(acc(%), cost($), latency(ms))
(high, low, low)	$Blob * Vader - IBM$	(67, 1.25, 93)
(low,high,low)	$Blob * Vader - Google$	(66, 0.44, 81)
(low,low,high)	$Blob * Vader - MS$	(65, 0.85, 59)

TABLE IX
QoS concerns, Generated Plans, and Execution Results

that while the Microsoft service returns scores between $[0, 1]$, the other ones return scores between $[-1, 1]$. We reconcile this discrepancy by redefining Microsoft's scores greater than $0.50 + x$ as positive, ones smaller than $0.50 - x$ as negative, and ones in between as neutral. We derive x for each service by minimizing the categorization error as follows. We vary x from 0.01 to 0.99 with a step size of 0.01, calculating the classification error over a training data set chosen at random from the original data set, without replacement.

B. Reducing the Number of Combinations to Explore

In all of our strategies, we combine services by exhaustively enumerating them. This practice is only feasible for combinations of up to about 5 services. However, for larger combinations, this approach can introduce a prohibitively large overhead. Hence, we introduce a heuristic that reduces the number of explored combinations, required to find the optimal combinations. In the three utilized fusion strategies, the cost of the combined execution comes from summing the costs of the comprising services. The latency of the combined execution comes from the maximum of the latency of the comprising services. A combination's cost and latency are always higher than those of any of its comprising services. A combination can offer a higher utility only if the overall accuracy is higher than that of any of its comprising services. Moreover, without achieving any increase in accuracy, the extra effort required to explore a combined execution cannot be justified. Hence, we filter out all the combinations whose overall accuracy is smaller than the maximum accuracy of the comprising services. This filtering heuristic reduces the number of combinations that need to be evaluated, making our experiments manageable.

VII. Limitations and Future Work

Although our custom execution achieves the best overall utility, we were unable to find a strategy that increases accuracy while keeping the remaining QoS characteristics in check. We are intrigued by the ubiquitous presence of correlation among the individual equivalent services. Hence, we plan to further study that correlation in order to devise a strategy that leverages its presence. One possible direction is to utilize the correlation coefficient between services to develop a combined execution strategy in which the constituent services augment each other, thus increasing the overall accuracy.

Having been meticulous in our experiments, we validated our findings with a completely unrelated use case to be able to generalize our guidelines. We now plan to empirically validate them against additional use cases, especially those with a high number of classes. To that end, we are on the lookout for the right mix of web services and labelled data. Because different services return class labels inconsistently, the current strategies cannot predict the label for a combination.

Lastly, we plan to explore how our research findings can be applied to service composition. One direction is leveraging our insights for choosing services for each sub-task. Further, we would evaluate if we can improve the QoS offered by the composition as a whole and compare the improvement with existing approaches.

VIII. Related Work

Various aspects of the service selection problem have been studied extensively. Liu et al. [7] presented an open, fair, and dynamic QoS model for selecting web services by implementing a QoS registry in a market place application that provisions phone services. Alrifai et al. [33] study service composition, in which component services are selected to satisfy end-to-end QoS requirements (e.g., availability, response time, cost) by combining global and local optimization. Huang et al. [34] efficiently select services in two different contexts: single QoS-based service discovery and QoS-based optimization of service composition. Hiratsuka et al. [1] reduce the costs of combinational use when selecting services to satisfy QoS requirements. Their experimental results show that the computational costs of effective service combinations can be reduced irrespective of the number of services and their QoS values. However, prior research has not considered accuracy or aimed at achieving QoS balance. Combinations of equivalent services rather than a single service have not been explored either.

Classifier fusion has been used to increase accuracy. Kittler et al. [13]'s theoretical framework combines classifiers using distinct pattern representations. By experimentally comparing various combinations, they discover that the sum-rule and its derivatives (developed under the most restrictive assumptions) offer the best performance. Woods et al. [14] combine classifiers by estimating each individual classifier's local accuracy in small regions of the feature space surrounding an unknown test sample, putting forward a method that determines how to best mix individual classifiers. Ruta et al. [15] revise the classifier selection methodology and evaluate the practical applicability of diversity measures in the context of combining classifiers by majority voting. Furthermore, their novel design of multiple classifier systems recurrently applies selection and fusion to a population of the best combinations of classifiers rather than to the single best ones. Kuncheva et al. [16] derive upper and lower limits on the majority vote accuracy with respect to individual accuracy p, the number of classifiers in the pool (L), and the pairwise dependence between classifiers, measured by Yules Q statistic. Whitaker et al. [17] show, by means of an enumerative example, how combining classifiers can increase or decrease accuracy as compared to each single classifier.

Service composition [1], [35], [36] optimize reliability, cost, and latency by the combined use of equivalent services. These approaches estimate the reliability, cost, and latency of various service combinations and apply different algorithms to find an optimal combination. However, none of these approaches have taken accuracy into consideration. For cognitive tasks, no services can be 100% accurate, thus service composition for enhancing accuracy is an important unaddressed problem. Different from other QoS attributes, the accuracy of different equivalent services may be correlated, rendering the existing QoS estimation algorithms directly inapplicable to estimate the accuracy of a service combination.

Our work draws inspiration from these lines of works. We apply the findings from accuracy improvement efforts to those that select services based on QoS. By exploring combinations of equivalent services to better meet the QoS requirements, which include accuracy, we increase the overall utility of cognitive services.

IX. Conclusion

To help developers choose from a set of equivalent services, we study the impact of the combined execution of equivalent services on non-functional characteristics. Three widely used strategies process/combine the results of each service to predict outcome of the combination. We also offer a novel custom execution strategy that helps better meet the QoS requirements. Two case studies explore and validate the QoS satisfaction of these strategies, respectively. No strategy can significantly improve accuracy without increasing cost and/or latency. However, some combinations in both use cases reduce cost and/or latency while providing comparable accuracy, as compared to the most accurate single service. Based on our findings, we suggest the following developer guidelines for selecting services:

1) In the absence of a labelled training data, intuitively select the best service from a set of equivalent services.
2) Otherwise, record the predictions and the time taken for each equivalent service over the training data.
3) Check if the services are correlated by evaluating their joint probabilities
4) In the absence of correlation, apply majority voting to exploit combinations and try to improve accuracy with lower cost and latency.
5) Otherwise, apply stacking and weighted majority voting, in order. These strategies can help find combinations that provide comparable accuracy with lower cost and latency. Hence, the combination can better meet non-functional requirements.
6) To achieve QoS balance, apply our custom execution strategy that offers combinations with the most optimal utility values.

References

[1] N. Hiratsuka, F. Ishikawa and S. Honiden. Service Selection with Combinational Use of Functionally-Equivalent Services. 11' IEEE International Conference on Web Services, Washington, DC, pp. 97-104

[2] J. Vuurens, A. de Vries, C. Eickhoff. 11'. How much spam can you take? an analysis of crowdsourcing results to increase accuracy. ACM SIGIR Workshop on Crowdsourcing for Info Retrieval. 21-26.

[3] Dragoni N. et al. (2017) Microservices: Yesterday, Today, and Tomorrow. In: Mazzara M., Meyer B. (eds) Present and Ulterior SE.

[4] Wang X., Vitvar T., Kerrigan M., Toma I. A QoS-Aware Selection Model for Semantic Web Services. In: Dan A., Lamersdorf W. (eds) Service-Oriented Computing ICSOC 2006. ICSOC 2006

[5] D. Tsesmetzis, I.G. Roussaki, I.V. Papaioannou, M.E. Anagnostou. QoS awareness support in Web-Service semantics. 06' 128- 128

[6] Gerardo Canfora , Massimiliano Di Penta (2004). A lightweight approach for QOS aware service composition. Proceedings of the 2nd ICSOC

[7] Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. QoS computation and policing in dynamic web service selection. In Proceedings of the 13th WWW 04', pp. 66-73.

[8] C.W. Zhang, S. Su, J.L. Chen (2006). Genetic algorithm on web services selection supporting QoS. Jisuanji Xuebao/Chinese Journal of Computers. 29. 1029-1037.

[9] M. Kerrigan. Web Service Selection Mechanisms in the Web Service Execution Environment. SAC 06

[10] G. Canfora, M. Di Penta, R. Esposito and M. L. Villani. QoS-aware replanning of composite Web services. (ICWS'05) pp. 121-129

[11] Liangzhao Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, QoS-aware middleware for Web services composition. in IEEE Transactions on Software Engineering

[12] Pascal Van Hentenryck. 1989. Constraint Satisfaction in Logic Programming. MIT Press

[13] J. Kittler, M. Hater and R. P. W. Duin. Combining classifiers. Proceedings of 13th International Conference on Pattern Recognition, 1996, pp. 897-901

[14] K. Woods, W. P. Kegelmeyer and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. In IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 405-410, 97'.

[15] D. Ruta, B. Gabrys, Classifier selection for majority voting, Information Fusion, 05', pp. 63-81

[16] Kuncheva, L., Whitaker, C., Shipp, C. et al. Pattern Anal Appl 03'

[17] Whitaker, C.J., Kuncheva, L.I., and Bangor, B. (2003). Examining the Relationship Between Majority Vote Accuracy and Diversity in Bagging and Boosting.

[18] Mu Zhu. When is the majority-vote classifier beneficial? arXiv 13'

[19] Hyoung Woo Lee, SeKee Kil, Younghwan Han and SeungHong Hong. Automatic face and facial features detection. ISIE 2001. pp. 254-259

[20] https://www.ibm.com/watson/services/visual-recognition/

[21] https://azure.microsoft.com/en-us/services/cognitive-services/face/

[22] https://www.faceplusplus.com/face-detection/

[23] https://github.com/ageitgey/face_recognition

[24] https://github.com/ShiqiYu/libfacedetection

[25] Soon-gyo Jung, Jisun An, Haewoon Kwak, Joni Salminen, and Bernard Jansen, "Assessing the Accuracy of Four Popular Face Recognition Tools for Inferring Gender, Age, and Race," in International AAAI Conf. on Web and Social Media

[26] L. Fei-Fei, R. Fergus and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. Workshop on Generative-Model Based Vision.

[27] https://www.ibm.com/watson/services/natural-language-understanding/

[28] https://azure.microsoft.com/en-us/services/cognitive-services/text-analytics/

[29] https://cloud.google.com/natural-language/

[30] https://github.com/sloria/TextBlob

[31] https://github.com/cjhutto/vaderSentiment

[32] https://www.kaggle.com/crowdflower/twitter-airline-sentiment

[33] Mohammad Alrifai and Thomas Risse. 2009. Combining global optimization with local selection for efficient QoS-aware service composition. In Proceedings of the 18th international conference on World wide web.881-890

[34] Angus F. M. Huang, Ci-Wei Lan, and Stephen J. H. Yang. 2009. An optimal QoS-based Web service selection scheme. 3309-3322

[35] Cardellini Valeria, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. Moses: A framework for QoS driven runtime adaptation of service-oriented systems. IEEE Transactions on Software Engineering 38, no. 5 (2011): 1138-1159.

[36] Guo, Yan, Shangguang Wang, Kok-Seng Wong, and Myung Ho Kim. Skyline service selection approach based on QoS prediction. International Journal of Web and Grid Services 13, no. 4 (2017): 425-447.