11 Evaluation of NLP Systems

PHILIP RESNIK AND JIMMY LIN

1 Introduction

As the engineering branch of computational linguistics, natural language processing is concerned with the creation of artifacts that accomplish tasks. The operative question in evaluating an NLP algorithm or system is therefore the extent to which it produces the results for which it was designed. Because NLP encompasses an enormous range of different tasks, each with its own particular criteria for assessing results, a single chapter on evaluation cannot hope to be comprehensive. In this chapter, therefore, we have selected a number of basic issues, laying out some fundamental principles of NLP evaluation, describing several of the most common evaluation paradigms, and illustrating how the principles and paradigms apply in the context of two specific tasks, word-sense disambiguation and question answering. For a comprehensive treatment, we refer the reader to Galliers and Jones (1995).[1]

It must be noted that the design or application of an NLP system is sometimes connected with a broader scientific agenda; for example, cognitive modeling of human language acquisition or processing. In those cases, the value of a system resides partly in the attributes of the theory it instantiates, such as conciseness, coverage of observed data, and the ability to make falsifiable predictions. Although several chapters in this volume touch on scientific as well as practical goals (e.g., the chapters on computational morphology, unsupervised grammar acquisition, and computational semantics), such scientific criteria have fallen out of mainstream computational linguistics almost entirely in recent years in favor of a focus on practical applications, and we will not consider them further here.

In addition, like other types of computer software, NLP systems inherit a wide range of evaluation concerns, criteria, and measures from the discipline of software quality evaluation. When assessing any software product, for example, it is typical to consider such issues as cost, support, efficiency, reliability, scalability, interoperability, security, and so forth. Many relevant issues are covered by the

ISO 9126 standard for software product evaluation (ISO 1991), and specific extensions to that standard have been created specifically for the purpose of evaluating language technology (EAGLES 1996).

To introduce some of the ideas we will be looking at in greater detail below, consider evaluation in the context of a 'semantic' search engine that utilizes NLP components.[2] For purposes of illustration, suppose that the search engine matches users' questions with facts that appear in documents, and that the underlying method involves a component that analyzes sentences to produce normalized subject–relation–object dependency tuples. The use of normalized dependency tuples has the potential to allow more specific concept-level matches; for example, the question *When was the light bulb patented by Edison?*, can match *Thomas Edison's patent of the electric light bulb* via the tuple [Thomas Edison, patented, bulbs].[3]

How might the quality of dependency tuple analysis be evaluated? One time-tested approach that deserves mention would be to skip formal evaluation of this component altogether, and instead perform a *demonstration* of the search engine for the project's investors or other target audience. Well-designed demos are often simple to execute, easy to understand, and surprisingly powerful in making a compelling case for a system's value. However, demos can also be quite misleading because they rarely exercise the full range of a system's capabilities and can be carefully orchestrated to hide known flaws.

Another approach would be to perform a standard *intrinsic* evaluation of the dependency extraction component. In a case like this, one would typically create a test set that contains a sample of test sentences for input, along with the *ground truth*, i.e., an 'answer key' in the form of tuples that the system is expected to create for each test sentence. The design of the evaluation would quantify the system's agreement with the ground truth, much as a teacher counts up the agreements and disagreements between a student's multiple-choice test answers and the answer key. In this case, each test sentence has a *set* of tuples that together comprise the correct answer, and the goal is to produce all and only the tuples in that set. In settings like this, one would usually calculate *recall*, measuring the extent to which *all* the tuples were produced, *precision*, capturing the extent to which *only* correct tuples are included in the output, and *F-measure*, a score that combines recall and precision into a single figure of merit (see Section 3.2). One might compare the F-measure for the current tuples analyzer against an earlier version (a *formative* evaluation, measuring progress and informing new development) or against competing techniques (a *summative* evaluation, concerned with the outcome of development).

The intrinsic evaluation helps to assess the quality of the tuples analyzer, but how do we know that improvements in the analyzer actually make a difference in the overall quality of the system (that is, better search results)? One answer is to perform an *extrinsic* evaluation, which measures the quality of the analyzer by looking at its impact on the effectiveness of the search engine. In this case, we would create a test set not for the analyzer, but for *the search engine as a whole*. The test items in this case would therefore be user questions, and the ground truth for each question would be the answers that should be produced by the system.

Precision, recall, and F-measure could be used again here, but this time they would be used to quantify the extent to which the *search engine* produces all and only the correct answers to the test questions. Crucially, the quality of the dependency tuple analyzer is measured only indirectly, by evaluating the whole system with and without it, or by swapping out this analyzer and substituting another. To put this another way, the extrinsic evaluation treats the analyzer as an *enabling technology*, whose value is not intrinsic but rather resides in its contribution to a larger application (Resnik 2006).

Even an extrinsic evaluation may still be too far removed from quality in the real world, however. One way to address this would be to conduct a laboratory test involving real users employing the search engine to perform a standardized task, e.g., finding the answers to a set of test questions. This is, in effect, a variation on the extrinsic evaluation in which the 'system as a whole' actually includes not only the search engine but the user as well, and the setup makes it possible not only to measure the quality of the answers produced, but also to look at factors such as time taken and user satisfaction.

Ultimately, however, laboratory experimentation cannot completely predict how a technology will fare in a real-world environment with real users performing real tasks. For this, a system must be deployed, and observations made *in situ* of the system, the users, and their interaction. Learning how the technology does in the real world comes at the cost of experimental controls and replicability, but for many pieces of real-world technology, the final test is not the laboratory measures, but the usefulness of the tools, the satisfaction of users, and their willingness to come back and use that technology again.

2 Fundamental Concepts

2.1 Automatic and manual evaluations

Perhaps the most basic dichotomy in evaluation is that between automatic and manual evaluation. Often, the most straightforward way to evaluate an NLP algorithm or system is to recruit human subjects and ask them to assess system output along some predetermined criteria. In many cases, this is the best approach for finding out whether a system is actually useful and whether users are pleased with the system – a criterion that goes beyond whether or not the system meets predetermined requirements or specifications. Note that manual evaluations are the norm in many fields, for example, user studies in human–computer interaction.

Unfortunately, manual evaluations have two significant limitations: they often generate inconsistent results and they are slow. Human beings are notoriously inconsistent in their judgments about what is 'good' (both across multiple subjects and sometimes with themselves), and even with the adoption of standard best practices in study design (e.g., counterbalanced presentation of experimental conditions, calibration for learning effects and environmental settings, etc.),

it is difficult to control for unanticipated factors. In addition, manual evaluations are time-consuming and laborious. In addition to time for the actual experiment, human subjects must be recruited, scheduled, and trained. To arrive at statistically significant findings, dozens of subjects are often necessary. All of these factors conspire to make well-designed manual evaluations a large investment of resources.

Modern NLP has evolved into an empirical, evaluation-driven discipline, which means that researchers have little patience for long turnaround between successive experiments. Thus, automatic evaluation methods are favored today by most. The development of evaluation algorithms that mimic the behavior of human assessors is an important subfield in NLP, and such research can be readily found in the pages of conference proceedings and journal articles in the field. However, recognizing that manual evaluations remain valuable, it is common practice to periodically conduct studies that establish a correlation between results of automatic and manual evaluations. If such a correlation is demonstrated, then researchers have a degree of confidence that improvements according to automatic evaluations will translate into *meaningful* improvements for users.

2.2 Formative and summative evaluations

The distinction between formative and summative evaluations is best summed up with a quote: "When the cook tastes the soup, that's formative; when the customer tastes the soup, that's summative."[4] Formative evaluations typically occur during the development of NLP systems – their primary purpose is to inform the designer as to whether progress is being made towards the intended goals. As such, formative evaluations tend to be lightweight (so as to support rapid evaluation) and iterative (so that feedback can be subsequently incorporated to improve the system). In contrast, summative evaluations are typically conducted once a system is complete (or has reached a major milestone in its development): they are intended to assess whether intended goals of the system have been achieved.

In NLP research, there is a tendency for formative evaluations to be automatic, so that they can provide rapid feedback in the development process. In contrast, summative evaluations often involve human judges, in order to assess the usefulness of the system as a whole for users.

2.3 Intrinsic and extrinsic evaluations

Intrinsic and extrinsic evaluations form another contrast that is often invoked in discussions of evaluation methodologies. In an intrinsic evaluation, system output is directly evaluated in terms of a set of norms or predefined criteria about the desired functionality of the system itself. In an extrinsic evaluation, system output is assessed in its impact on a task external to the system itself. Evaluation of document summarization systems serves to illustrate this distinction. In an intrinsic evaluation, we would ask questions such as the following about system-generated

summaries: How fluently does the summary read? Does the summary contain coverage of key ideas? On the other hand, extrinsic evaluations consider tasks in which a document summarization system may be useful – for example, as a component of a search engine that summarizes results and presents short snippets to help users decide whether or not a document is relevant (i.e., worth reading). In the context of this particular task, we might ask: How accurately can a user make such relevance judgments, compared to having access to the entire document? How much more quickly can such judgments be made with summaries?[5]

In NLP research, at least, there is an affinity between intrinsic, formative, and automatic evaluations on the one hand, and extrinsic, summative, and manual evaluations on the other. The characteristics of these different approaches naturally explain these associations. Since extrinsic evaluations must be couched within the context of a user task, it is difficult to avoid having human subjects. It is usually easier to develop automatic techniques for intrinsic evaluations since only the system output needs to be considered.

2.4 Component and end-to-end evaluations

Most NLP systems today are not monolithic entities, but rather consist of distinct components, often arranged in a processing pipeline. For example, identification of semantic role (e.g., agent, patient, theme) depends on syntactic parsing, which in turn depends on part-of-speech tagging, which in turn depends on tokenization. We could choose to evaluate each component individually, or instead consider multiple components at once. For example, when evaluating the accuracy of a syntactic parser that requires part-of-speech tags as input, one could assess the quality of the parse-trees based on the output of a real tagger that may contain errors (an end-to-end evaluation), or based on 'gold standard' part-of-speech tags supplied by a human (a component evaluation).[6]

Both component and end-to-end evaluations are useful, but for different purposes. Obviously, end-to-end evaluations provide a more meaningful quantification of system effectiveness under real-world circumstances. However, measuring system characteristics under ideal circumstances may also be useful, since it isolates the system from errors in other components. With component evaluations, it is possible to artificially manipulate input and observe the impact on system effectiveness. For example, in evaluating a syntactic parser one could start with gold standard part-of-speech tags and then artificially degrade tagging accuracy in a controlled manner. Doing so would allow a researcher to understand the input–output characteristics of the component, i.e., sensitivity of the parser to tagging errors.

In most cases, it is desirable to conduct both component and end-to-end evaluation of NLP systems since components often interact in non-obvious ways. For some systems, the effects of errors are multiplicative: that is, since each component depends on the previous, errors propagate down a processing pipeline, so that the final output may be quite poor despite high effectiveness for each of the components (consider the pipeline for identifying semantic roles described above).

For other systems, the overall effectiveness is much higher than one would expect given individual component-level effectiveness – these represent cases where the components are able to compensate for poor quality. One example of this is in cross-language information retrieval (CLIR), where the user issues a query in one language to retrieve documents in another language. Abstractly, one can think of CLIR systems as having a translation component and a search component. In even the most sophisticated systems, the translation component is little better than word-for-word translation – the quality of which is quite poor by human standards. Yet CLIR systems are about as effective as monolingual IR systems – since the inherent redundancy in documents and queries (i.e., the presence of multiple words referring to the same concepts) compensates for the poor translation quality.

2.5 *Inter-annotator agreement and upper bounds*

In many NLP evaluation settings – particularly intrinsic, component-level evaluations – the task being evaluated is to 'annotate' (tag, label) text. For example, part-of-speech taggers assign grammatical category tags to words, named entity extractors assign category labels (e.g., PERSON, ORGANIZATION) to phrases, and parsers can be viewed as assigning constituent labels like NP or VP to spans of text within a sentence. It is common practice in such cases to compare the performance of multiple *human* annotators, for two reasons. First, if human beings cannot reach substantial agreement about what annotations are correct, it is likely either that the task is too difficult or that it is poorly defined. Second, it is generally agreed that human inter-annotator agreement defines the upper limit on our ability to measure automated performance; Gale et al. (1992: 249) observe that "our ability to measure performance is largely limited by our ability [to] obtain reliable judgments from human informants." As a well-known case in point, the WordNet lexical database includes sense tags that are notoriously fine-grained, e.g., distinguishing verb sense *chill* (make cool or cooler) from *chill* (lose heat) because the former involves *causing* a change and the latter *undergoing* a change in temperature (Palmer et al., 2007). Could we really expect any word-sense disambiguation algorithm to achieve 95 percent agreement with human-selected WordNet sense tags, for example, if the human taggers themselves can only agree 75 percent of the time when doing the task (Snyder & Palmer 2004)? For this reason, human agreement is generally viewed as the *upper bound* on automatic performance in annotation tasks.[7]

One way to measure agreement between two annotators is simply to measure their observed agreement on a sample of annotated items. This, however, may not constitute an accurate reflection of the true difficulty or upper bound on the task, because, for any given task, some agreements may occur according to chance. Consider a simple example: if the annotation task were to sense-tag instances of the word *bank* as either RIVERBANK or FINANCIALBANK, and two annotators make their choices independently by flipping a coin, they could be expected to agree 50 percent of the time. Therefore, in order to establish the validity of a coding scheme, as well as define upper bounds for the annotation task, it is now common practice to compute a measure of *chance corrected* agreement

(Artstein & Poesio 2008). Correction for chance is captured by measures that generally take the form

$$(1) \quad \frac{A_0 - A_e}{1 - A_e}$$

where A_0 is the observed agreement (total agreements divided by total number of items), and A_e is an estimate of chance agreement varying according to the specific measure. Cohen's kappa is in widespread use for this purpose, but Artstein and Poesio (2008) provide a thorough discussion of its limitations and of alternative measures, as well as in-depth consideration of detailed issues, including, e.g., measuring agreement among three or more annotators, weighting some disagreements more heavily than others, and the interpretation of agreement coefficient values.

2.6 *Partitioning of data used in evaluations*

Within most NLP settings, system development and evaluation involves partitioning the available data into the following disjoint subsets:

- **Training data.** This term most often refers to a data set where input items are paired with the desired outputs, often as the result of manual annotation (cf. Section 2.5). It usually refers to the input for supervised learning algorithms, but it can refer more broadly to any data used in the process of developing the system's capabilities prior to its evaluation or use.[8]
- **Development (dev) data.** Some systems include parameters whose settings influence their performance. For example, a tagger might choose its output based on a weighted vote $\sum \lambda_i p_i(\text{input})$, where each p_i is a different method of prediction and the λ_i are weights for the different methods. Rather than choosing weights or parameters arbitrarily, it is common to hold out some subset of the training data as a *development set*. A search for good values for λ_i is conducted, either in an ad hoc manual fashion or using an optimization technique such as expectation-maximization. In either case, performance on the development data measures the 'goodness' of the parameter choices.
- **Development-test (devtest) data.** Typically one or more data sets are also held out for use in formative evaluation (Section 2.2) as the system is developed. A devtest set is just a test set that is being used during the cycle of system development and improvement.
- **Test data.** This term describes the data that will be used to evaluate the system's performance after development has taken place, i.e., at the point of a summative evaluation.[9]

It is typical to reserve as much data as possible for training and development. For example, one might split the available data into 70, 20, and 10 percent for training, held-out (i.e., dev and devtest), and test data respectively.

Disjointness of the subsets is crucial, because a fundamental principle in NLP evaluation is that the technology being evaluated *cannot be informed by the test data*. In its purest form, this means that test data should remain entirely untouched and unseen by the researcher or developer until system development is frozen just prior to evaluation. The reasoning behind this stricture is simply that evaluations are intended to help predict a system's performance on future unseen data, i.e., to generalize. In machine learning, the error rate when testing on the training data is referred to as the *resubstitution error rate*, and it is regarded as a severe underestimate of the true error rate, as a result of overfitting. Performance on the training data can be a useful reality check, of course, since something is probably wrong if it is not quite good. But it cannot be relied upon to predict future performance.

Moreover, it should be noted that evaluations can be overly optimistic even when test data are kept properly disjoint from data used in system development. For example, it is typical to assume that systems will be evaluated (or run) on the same kind of data that were used during system development, e.g., language that is similar in genre and topic. It is widely recognized, however, that system performance will suffer when this assumption is not met (e.g., Escudero et al., 2000; Gildea, 2001).

It is also worth noting that there are, in fact, some uses of the test data that are generally regarded as valid. The following are some examples:

- For research on machine translation systems, using the test set to automatically filter the phrase table, so that it contains only entries that are relevant for the given test set. This is a common way to reduce the size of the model so that it fits in memory (Lopez 2008a). Note that this affects the efficiency of a system, but does not fundamentally alter its behavior.
- Using the test set automatically for model adaptation (e.g., Kim & Khudanpur 2003).
- Performing error analysis prior to moving on to a fresh test set – i.e., the current test set becomes devtest data.
- Looking just at performance numbers on test data, without examining the system's output. For example, parsing researchers test over and over again using Section 23 of the *Wall Street Journal* in the Penn TreeBank, and MT researchers test repeatedly on test sets from the NIST machine translation evaluation exercises.

2.7 Cross validation

Employing a single partition of the available data is common, but it does present two potential problems. First, regardless of whether the evaluation results are good or bad, one has to wonder whether the results reflect a particularly fortuitous (or infortuitous) selection of test data. More precisely, a single split provides only a point estimator for whatever measure or measures are used for evaluation, as opposed to an interval estimate such as a 95 percent confidence interval. Second, as a practical matter, even a 70 percent allocation may not produce a large

enough training set for automatic learning methods if only a small quantity of annotated data is available.

Within NLP, the most common solution to these problems is *k-fold cross-validation*. Instead of creating a single split, the full set of available data is partitioned into k pieces, or folds, $\{f_1 \ldots f_k\}$.[10] Then evaluation is conducted as follows:

for i from 1 to k
  Let TEST = f_i be the test set
  Let TRAIN = $\cup_{j \neq i}\{f_j\}$ be used as the training set
  Compute m_i, the evaluation measure, by training on TRAIN and testing on TEST
Compute statistics, e.g., the mean and standard deviation, over the $\{m_i\}$.

If held-out data is needed for parameter tuning, TRAIN is subdivided into training and dev data.

K-fold cross-validation ensures that every item in the full data set gets used for both training and testing, while at the same time also ensuring that no item is used simultaneously for both purposes. Therefore it addresses the concern that the evaluation results only reflect a particularly good or particularly bad choice of test set. Indeed, addressing the second concern, the set $\{m_1 \ldots m_k\}$ can be used to compute not only the mean, as a scalar figure of merit, but also the standard deviation, enabling the computation of confidence intervals and tests of statistical significance when alternative algorithms or systems are compared. Finally, although values of k are typically between 4 and 10 – e.g., training uses from 75 percent to 90 percent of the available data – it is possible to use data even more efficiently by employing a larger number of folds. At the extreme, one can set k equal to the number of items N in the full data set, so that each fold involves $N - 1$ items used for training and one item for testing. This form of cross-validation is known as *leave-one-out*, and is similar to the jackknife estimate (Efron & Gong 1983).

2.8 Summarizing and comparing performance

All quantitative evaluation paradigms make use of at least one *figure of merit*, sometimes referred to as an *evaluation measure* or *evaluation metric*, to summarize performance on a relevant property of interest. Some of the most important paradigms, and their associated evaluation measures, are discussed in Section 3.

Table 11.1 shows the typical structure for reporting results in NLP evaluations. The first column of the table identifies the 'conditions,' i.e., variant approaches taken to the task. A single evaluation measure might be reported (e.g., accuracy, Section 3.1). Or there might be columns for multiple measures, often trading off against each other, with some single metric representing their combination (e.g., recall, precision, and F-measure, Section 3.2).

Turning to the rows, a results table almost always includes at least one *baseline* condition. The role of a baseline is similar to the control condition in an experiment studying the effectiveness of a new drug: in order for the study to successfully

Table 11.1 Structure of a typical summary of evaluation results

Condition	*Measure 1*	*Measure 2*	*Combined Measure*
Baseline 1	M_1^{B1}	M_2^{B1}	M_c^{B1}
Baseline 2	M_1^{B2}	M_2^{B2}	M_c^{B2}
Variation 1	M_1^{V1}	M_2^{V1}	M_c^{V1}
Variation 2	M_1^{V2}	M_2^{V2}	M_c^{V2}
Upper Bound	M_1^{U}	M_2^{U}	M_c^{U}

demonstrate that a drug is effective, patients taking the drug must show benefits over and above that experienced by patients taking a placebo. Similarly, the baseline condition in an NLP experiment defines the performance that must be improved upon in order for the study to deliver a positive result. One category of baselines can be defined independently of prior work in the literature; for example, choosing an answer at random, or always selecting an item's most frequent label in the training data, or applying something else that is equally obvious and simple.[11] Another kind of baseline is the effectiveness of some prior approach on the same data set. Generally the first category can be viewed as a 'reality check': if you cannot beat one of these baselines, most likely something is fundamentally wrong in your approach, or the problem is poorly defined (see Section 2.5).

The 'upper bound' for a task defines the highest level of performance one could expect to attain in this experiment. Typically, upper bounds are defined by human inter-annotator agreement (as discussed in Section 2.5). Sometimes, alternative upper bounds are defined by allowing the system to use knowledge that it would not have access to in a fair evaluation setting. As an example, a machine translation system might be permitted to produce its 1000-best hypotheses for each input sentence, and the *oracle upper bound* would be defined as the score of the hypothesis that performs best when compared against the reference translations. In practice, of course, an MT system cannot choose its single-best output by looking at correct translations of the input. But the oracle upper bound helps to quantify how much better the system could potentially get with better ranking of its hypotheses (Och et al., 2004).

When comparing system results against baselines, upper bounds, or across variations, it is important to recognize that not all differences between scores matter. One question to consider is whether or not an apparent difference is *statistically significant*; that is, if the difference is unlikely to have occurred as a result of chance variation. As a simple example, suppose we are selling a coin-flipping machine that will (we claim) make a fair coin more likely to land heads up. If we test the machine by performing an experiment with 10 flips, and the coin comes up heads 6 times instead of 5, should a potential buyer be convinced that our machine works as advertised, compared to typical, unassisted coin flips? Probably not: even with normal, 'unimproved' coin flipping, someone doing a large number of 10-flip experiments could be expected to get the same result, exactly

6 heads out of 10 flips, in fully 20 percent of those experiments, just by chance. So getting that particular result in this experiment could easily have happened even if the machine just flipped coins in the usual way. By convention, an experimental outcome is not usually considered 'statistically significant' unless the likelihood of its having occurred by chance is less than 5 percent, often written $p < .05$.[12]

Even if a difference in experimental conditions is statistically significant, however, it is essential to recognize that the result may not be large, important, or even meaningful. The 'significance' of an experimental improvement (in the ordinary sense of the word) is usually calibrated as a matter of folklore or common wisdom within a particular experimental community. In information retrieval, for example, a system might meet accepted criteria for a meaningful result by achieving a .05 absolute improvement in average 'precision at 10' (the precision computed using the ten most highly ranked hits in response to a query), with an improvement of .10 being considered substantial.[13] In machine translation, researchers might expect to see around a one-point improvement in BLEU score (Papineni et al., 2002) on one of the recent NIST evaluation data sets, with a gain of two points or more being considered substantial.[14]

However, the bottom line is that no hard-and-fast rule involving evaluation numbers or statistical significance can tell the full story when comparing alternative approaches to a problem. Ultimately, the potential value of a new contribution in NLP depends also on the relevance of the evaluation task, the representativeness of the data, the range of alternative approaches being compared, and a host of other more tangible and less tangible factors.

Finally, it is worth noting that sometimes the relative difference in performance metrics provides more insight than their absolute difference. On the combined measure, the *relative improvement* of Variation 1 over Baseline 1 in Table 11.1 is

$$(2) \quad \frac{M_c^{V1} - M_c^{B1}}{M_c^{B1}}$$

For example, improving accuracy from $M_c^{B1} = 35\%$ to $M_c^{V1} = 40\%$ is only a 5 percent improvement in absolute terms, but the relative improvement defined in (2) is more than 14 percent. To make the point more dramatically, if a system improves accuracy from 98 percent to 99 percent, this may seem like only a small accomplishment, only one percentage point. But the picture changes a great deal if the results are expressed in terms of error rate (100 percent − accuracy): the improved system cuts the number of errors in half, which can make a huge difference if the number of inputs is very large.

3 Evaluation Paradigms in Common Evaluation Settings

At the most basic level, NLP systems are designed to accomplish some task, which can be characterized by input–output characteristics. Thus, evaluation boils down

to a conceptually simple question: for a set of inputs, to what extent does system output correspond to outputs that are *correct* or *desirable*? This question helps organize common paradigms in evaluation, discussed in this section. In some cases, there is a one-to-one correspondence between input and the correct output (e.g., part-of-speech tagging, word-sense disambiguation). In other cases, multiple outputs are desirable (e.g., information retrieval, parallel text alignment), the output takes the form of text (e.g., machine translation, document summarization), or the output contains complex structure (e.g., parsing). Finally, the output may involve values on a scale (e.g., language modeling, semantic similarity).

3.1 One output per input

The most straightforward evaluation paradigm in NLP is one in which each input produces a single output – a nominal value that can be considered a category or label (Stevens 1946) – and that output is compared against a single correct answer. This is analogous to multiple-choice tests, although for many NLP tasks the number of possible answers for any input can be quite large.

Classic word-sense disambiguation tasks fall into this category: each word, in its context, represents an input, and the possible outputs are defined by an enumeration of sense labels. For example, suppose that the task involves deciding whether instances of the word *ash* are being used in the sense ASH_1, 'a tree of the olive family,' or in the sense ASH_2, 'the solid residue left when combustible material is burned' (Lesk 1986).[15]

To evaluate disambiguation performance in settings like this, one would run the system on inputs $\{a_1, \ldots, a_n\}$ (where each a_i is an instance of the word *ash* in context), producing single-best output label decisions $\{l_1, \ldots, l_n\}$ (where each $l_i \in \{\text{ASH}_1, \text{ASH}_2\}$), and then compare those decisions to 'ground truth' human-annotated sense labels $\{t_1, \ldots, t_n\}$ ($t_i \in \{\text{ASH}_1, \text{ASH}_2\}$). The primary figure of merit would be the percentage of agreement with the true labels, i.e., the *accuracy*:

$$(3) \quad A = \frac{\sum_{i=1..n} \text{agr}_i}{n} = \frac{\text{number correct}}{n}$$

where agr_i is 1 if $l_i = t_i$ and 0 otherwise.[16] Sometimes the inverse of accuracy, or *error rate*, is reported instead: $1 - A$.

Sometimes a system is not required to produce any answer at all for some inputs; that is, l_i could remain undefined. In some tasks, it might be preferable for the system to remain silent than to risk being wrong. In those cases, we can define d to be the number of defined answers, and d replaces n in the denominator when accuracy is calculated. We then define *coverage* as d/n, in order to measure the extent to which answers were provided. By default, one can assume $d = n$, i.e., coverage is 100 percent, when accuracy is presented alone as the figure of merit. When $d < n$, accuracy and coverage can be traded off against each other – for example, a system can obtain high accuracy by providing an answer for an input only when it is very confident, at the expense of coverage. This is quite similar to the trade-off between precision and recall discussed below in Section 3.2. Indeed, the

Table 11.2 Contingency table for a document retrieval task

	relevant	¬*relevant*
retrieved	r	n
¬retrieved	R	N

$$\text{Precision} = r/(r+n) = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{retrieved}|}$$
$$\text{Recall} = r/(r+R) = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{relevant}|}$$

terms 'precision' and 'recall' are sometimes used, in our view somewhat confusingly, to refer to accuracy and coverage in task settings where each input has only a single output.[17]

In a common variant of this paradigm, the desired output for each input is a sequence $y_1 \ldots y_k$. For example, in part-of-speech tagging, each input a_i would be a whole sentence, i.e., a sequence of tokens $x_1 \ldots x_k$, and the output label would be a sequence $y_1 \ldots y_k$ of grammatical category tags. When the output sequence stands in one-to-one correspondence with the input sequence, as in this example, it is most common simply to evaluate as if each input token comprises its own single-token labeling problem, even if that's not really how the output was produced. This is equivalent to concatenating all the output sequences to produce one long sequence of length n, and then computing A as defined above.

When the output sequence can differ in length from the input sequence, the situation becomes a bit more complicated; we treat that case as a variant of structured output in Section 3.2.

3.2 Multiple outputs per input

For many NLP tasks, there is no single correct answer; multiple outputs are sought. Information (document) retrieval is perhaps the best illustration of this general evaluation paradigm. Given an information need expressed as a query (e.g., 'gardening in arid soil'), the task of the system is to return the set of documents that are relevant and, in most cases, there are multiple satisfactory documents. Formalizing the task abstractly in terms of set membership – a document is either retrieved by the system or it is not, and a document is either relevant to the information need or it is not – is an imperfect approximation of the real-world task, where documents may be relevant only to a greater or lesser extent, and systems may estimate degrees of confidence. But this abstraction makes it possible to define the quality of a system's set of retrieved documents in terms of two extremely useful and intuitive concepts: *precision* and *recall*. The contingency table in Table 11.2 illustrates how these are computed. Precision is the fraction of system output that is relevant, or $r/(r+n)$; recall is the fraction of relevant documents that is retrieved, or $r/(r+R)$.[18]

Notice that the numerator is the same in both cases: r counts the number of documents that were both relevant and retrieved by the system. For precision, we are interested in comparing that with the total number of documents retrieved by the system, hence $r + n$ in the denominator. If $n = 0$, i.e., no irrelevant documents were retrieved, then precision is perfect. For recall, we are interested in comparing r with the total number of documents that *should* have been retrieved, hence $r + R$ in the denominator. If $R = 0$, i.e., every relevant document was retrieved, then recall is perfect.

High precision is easy to obtain, at the expense of recall: just return the single document most likely to be relevant, or, more generally, do not return documents unless the system's confidence in their relevance is very high. This keeps n close to 0, but of course it also increases R, so recall suffers. Similarly, perfect recall is easy to achieve (just return all the documents in the collection, so $R = 0$), but at the expense of precision, since n is then likely to be large.

To balance the need for both precision and recall, F-measure (or F-score) is often reported:

$$F(\beta) = \frac{(\beta^2 + 1) \times P \times R}{\beta^2 \times P + R} \tag{4}$$

The F-measure computes a harmonic mean between precision and recall, where the relative emphasis on the two components is controlled by the β parameter (higher values of β place more emphasis on recall). The choice of β depends a lot on the task. For example, a person searching the web for gardening information does not want to slog through lots of irrelevant material, and does not require every single gardening article that is out there, so precision is a high priority. In contrast, a complex legal argument can be undermined by even a single court ruling overturning a previous precedent, so systems for legal research can be expected to place a heavy emphasis on recall.

F-measure is frequently used to compare different systems. In addition to combining two measures into a single figure of merit, F-measure has the attractive property of incurring a penalty in performance when precision and recall are very different from each other, thereby discouraging an emphasis on one at the expense of the other. Other common metrics in information retrieval (e.g., mean average precision, R-precision) derive from this set-based formalism, with the addition of other concepts such as document ranking. It is worth noting that all these metrics are intrinsic in nature, in that they do not measure how *useful* the retrieved documents are in a real-world task, e.g., writing a report, answering a complex question, making a decision, etc.

3.3 Text output for each input

Frequently in NLP, the task of the system is to produce text in response to the input. Machine translation is the most obvious example of the paradigm, since an output text in the target language is produced for the source-language input.

Text summarization is similar, producing an output text that condenses pertinent information from a set containing one or more input documents.

Evaluating text output introduces a difficult challenge: how do we account for the fact that the desired information can be expressed correctly in many different ways? Testing for exact equality is necessary when computing agreement in Section 3.1, or when computing the intersection |relevant∩retrieved| in Section 3.2, but string equality hardly seems appropriate as a way of evaluating whether or not two texts are saying the same thing. One solution to this problem is to rely on human judges to compare system outputs with correct answers (see Section 3.5), but that solution is extremely labor-intensive. It would generally be impractical, for example, to collect human judgments on a weekly basis in order to track progress during system development.

Most ways of dealing with this challenge involve two elements. First, texts being compared are broken down into units that *can* be compared via exact matching, e.g., word *n*-grams. Then a bag of *n*-grams from the system output can be compared with the *n*-grams present in the human 'gold standard' reference, quantifying the relationship using measures derived from precision and/or recall. In essence, the *n*-grams in gold standard references define the 'relevant' elements of the desired response to the input, and *n*-grams in the system output constitute what the system has 'retrieved.' This idea has been operationalized, for example, in the BLEU metric for machine translation (Papineni et al., 2002) and the ROUGE metric for text summarization (Lin & Hovy 2003), both of which are widely used in their respective communities, albeit not without some controversy.

A second useful strategy is to define multiple correct references for each input. For example, it is not uncommon in MT evaluations to provide anywhere from two to ten correct translations for each test input. The evaluation measure is then generalized to take into account correct or 'relevant' units from multiple valid outputs. For example, consider a system that produces the English sentence *my dog is always hungry*, with reference translations

my dog is hungry all the time
my pup is always famished

Using the BLEU metric, which focuses on precision, the system would get credit for having produced 'relevant' unigrams *my*, *dog*, *is*, *always*, and *hungry*; bigrams *my dog*, *dog is*, and *is always*; and the trigram *my dog is*. Notice that it is getting some credit for having conveyed both *always* and *hungry*, even though no single reference translation conveys the combined meaning *always hungry* using both of those words.

3.4 Structured outputs

The paradigm in Section 3.3 also provides a way of thinking about evaluation settings in which *structured* outputs are expected, whether or not multiple references are available. The basic idea is the same: to break the structured representations

up into bags of smaller units and then compute precision and/or recall over those smaller units. Metrics like BLEU and ROUGE apply this concept to sequences (since sentences are sequences of tokens), which are broken up into bags of *n*-grams. But the idea is significantly more general; for example, the PARSEVAL measures (Abney et al., 1991) evaluate parsers by computing precision and recall over constituents.[19]

Another way to compare structured outputs, particularly sequences, is *edit distance* or its variants. For example, speech recognition researchers compute the *word error rate* between the system's output and the reference transcription:

$$\text{WER} = \frac{S + D + I}{n} \tag{5}$$

where S, D, and I are, respectively, the number of substitutions, deletions, and insertions in a minimum-cost edit transforming the system output into the reference. In machine translation, translation edit rate (TER) has gained currency. TER "measures the amount of editing that a human would have to perform to change a system output so it exactly matches a reference translation" (Snover et al., 2006: 223).[20]

3.5 *Output values on a scale*

Some tasks involve producing a value on a measurement scale, e.g., the traditional nominal, ordinal, interval, and ratio scales of Stevens (1946).[21] Producing values on nominal scales can be viewed simply as assigning a label or category to each input (one can only meaningfully ask about equality, but not relative ordering or magnitude). Comparisons of nominal outputs are addressed in Sections 3.1 and chance-corrected agreement is discussed in Section 2.5.

Ordinal scales capture a common situation in which desired outputs represent ratings, e.g., performing opinion analysis in order to assign a rating from one to five stars given the text of a movie review. Output values are ordered with respect to each other, but the intervals on the scale are not necessarily comparable. One cannot assume that the 'distance' between a one-star and two-star review represents the same difference in quality as the distance between a four-star and a five-star review – the number of stars merely tells you how the movies are ranked relative to each other. In these situations, it is common to compare a system's output ratings against human ratings by computing the Spearman rank order correlation coefficient, r_s (sometimes ρ), over the set $\{(o_i, t_i)\}$ of system outputs paired with human 'ground truth' ratings.

Interval scales are similar to ordinal measurements, with the additional assumption that differences between values constitute equivalent intervals. On the Celsius scale, for example, the difference in temperature between 1°C and 2°C is the same as the difference between 101°C and 102°C. Within NLP, comparisons of system scores against human ratings often assume this interpretation of the ratings scale is valid, and use the Pearson product–moment correlation (Pearson's r) over $\{(o_i, t_i)\}$

as a figure of merit. In machine translation, the validity of automatic evaluation metrics like BLEU (Papineni et al., 2002), TER (Snover et al., 2006), and METEOR (Banerjee & Lavie 2005) is sometimes supported by comparing automatic scores with human ratings of accuracy and fluency, using Pearson's r. Similarly, automatic measures of semantic similarity are often evaluated via correlation with human similarity ratings, using pairs of words (e.g., *furnace*, *stove*) as the items for which similarity is being computed (Resnik 1999; Pedersen et al., 2007).

Ratio scales assume that there is meaning not only for sums and differences on the scale but also for products and ratios. Within NLP, the most common quantitative output on a ratio scale would be the assignment of probabilities to inputs, often in the context of language modeling. For example, when evaluating a trigram language model p_{tri}, the test set consists of a text $T = w_1 \dots w_N$, and we measure either the cross entropy

$$(6) \quad H = -\frac{1}{N} \sum_{i=1}^{N} \log_2 p_{\text{tri}}(w_i | w_{i-2} w_{i-1})$$

or, more commonly, the perplexity, 2^H. Notice that whenever the model makes an accurate prediction in the test data, i.e., when the probability $p_{\text{tri}}(w_i|w_{i-2}w_{i-1})$ is high for an observed instance of w_i preceded by $w_{i-2}w_{i-1}$ in T, the contribution to H is small. Intuitively, perplexity is measuring the extent to which the model p_{tri} correctly reduces ambiguity, on average, when predicting the next word in T given its prior context. To put this another way, on average we are 'k-ways perplexed' about what the next word will be, with k ranging from 1 to the vocabulary size $|V|$.[22] In the worst case, the model might be no better than rolling a fair $|V|$-sided die, yielding perplexity $k = 2^{-\frac{1}{N} N \times \log \frac{1}{|V|}} = |V|$, meaning that the model provides no value at all in narrowing down the prediction of the next word. At the other extreme, a model that always predicts the next word perfectly (giving it a probability of 1 and therefore zero probability to all alternatives) would have a perplexity of $k = 2^0 = 1$.

Sometimes evaluation involves comparing output in the form of a probability distribution with ground truth that is also a distribution. In such cases, it is common to use Kullback–Leibler distance (also known as KL divergence or relative entropy) to compare the two distributions:

$$(7) \quad D(p||m) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{m(x)}$$

where p is the true probability distribution and m is the model being evaluated. Kullback–Leibler distance is zero when m is identical to p, and otherwise it is always positive. Its value can be interpreted as the cost, measured in bits of information, of encoding events in $\mathcal{X}$ using the imperfect model m rather than the truth p.[23]

4 Case Study: Evaluation of Word-Sense Disambiguation

Word-sense ambiguity is one of the earliest challenges singled out by researchers interested in automatically processing natural language. Some well-known early discussions of the problem include Weaver's (1949) memorandum on automatic translation, Bar-Hillel's (1960) argument that automatic high-quality translation requires comprehensive world knowledge in order to resolve lexical ambiguity, and Wilks's (1975) 'preference semantics' approach to semantic interpretation. *Word-sense disambiguation* (WSD) is conventionally regarded as the task of identifying which of a word's meanings (senses) is intended, given an observed use of the word and an enumerated list of its possible senses. In this section, we briefly review how approaches to WSD have been evaluated, with reference to the concepts we introduced earlier in the chapter. For informative general treatments of WSD, see Ide and Véronis (1998) and Agirre and Edmonds (2006), and for a more comprehensive discussion of recent WSD evaluation, see Palmer et al. (2006).

4.1 Pre-Senseval WSD evaluation

From the earliest days, assessing the quality of WSD algorithms has been primarily a matter of intrinsic evaluation, and "almost no attempts have been made to evaluate embedded WSD components" (Palmer et al., 2006: 76). Only very recently have extrinsic evaluations begun to provide some evidence for the value of WSD in end-user applications (Resnik 2006; Carpuat & Wu 2007). Until 1990 or so, discussions of the sense disambiguation task focused mainly on illustrative examples rather than comprehensive evaluation. The early 1990s saw the beginnings of more systematic and rigorous intrinsic evaluations, including more formal experimentation on small sets of ambiguous words (Yarowsky 1992; Leacock et al., 1993; Bruce & Wiebe 1994).[24]

Since word-sense disambiguation is typically defined as selecting one sense among a number of possibilities, it is naturally regarded as a classification problem involving the labeling of words in context (Edmonds & Agirre 2008). Thus evaluation has required answering six main questions.[25]

How do you define the 'sense inventory,' i.e., the set of possible sense labels for a word? Early efforts involved a wide variety of answers to this question – for example, Roget's thesaurus, various paper dictionaries, and various machine readable dictionaries. By the mid-1990s, WordNet (Fellbaum 1998) had emerged as a standard, easily available lexical database for English, and WordNet's 'synonym sets' provided a widely used enumeration of senses for content words (nouns, verbs, adjectives, and adverbs).

How do you select input items? Early experimentation focused on identifying a small set of 'interesting,' highly ambiguous words, e.g., *line* and *interest*, and

collecting a sample of those words within their sentential contexts. Cowie et al. (1992) represent a notable exception, tackling the problem of disambiguating all the content words in a sentence simultaneously.

How do you obtain labels ('ground truth') for items in the data set? In early studies it was not uncommon for experimenters to label their own test sets, e.g., Yarowsky (1992); Leacock et al. (1993); Bruce and Wiebe (1994). Miller et al. (1993) and Ng and Lee (1996) introduced large-scale manual sense labeling of corpora using WordNet, laying the groundwork for WSD approaches involving supervised learning techniques.

How do you compare system output against ground truth? In a setting where one correct label is assumed per input, the most natural figure of merit is accuracy, possibly accompanied by coverage if the system is permitted to abstain from labeling some inputs. Measures derived from cross-entropy (equation 6) can be used to give partial credit to systems that assign a probability distribution over senses (Resnik & Yarowsky 1999; Melamed & Resnik 2000).

What constitutes a lower bound on performance? An obvious but overly generous lower bound is chance, selecting randomly among a word's senses according to a uniform distribution. A more sensible lower bound is defined by tagging each instance of a word with its most frequent sense.[26] It is also not uncommon to compare WSD algorithms against easily implemented dictionary-based techniques, e.g., Lesk (1986) or variants.

What constitutes an upper bound on performance? Word-sense disambiguation is a classic example of a task where human inter-annotator agreement, and particularly chance-corrected agreement, are used to define the limits on what can be expected from automated algorithms (Artstein & Poesio 2008).

4.2 Senseval

In April 1997, a workshop entitled "Tagging Text with Lexical Semantics: Why, What, and How?" was held in conjunction with the Conference on Applied Natural Language Processing (Palmer & Light 1999). At the time, there was a clear recognition that manually annotated corpora had revolutionized other areas of NLP, such as part-of-speech tagging and parsing, and that corpus-driven approaches had the potential to revolutionize automatic semantic analysis as well (Ng 1997). Kilgarriff (1998: 582) recalls that there was "a high degree of consensus that the field needed evaluation," and several practical proposals by Resnik and Yarowsky (1997) kicked off a discussion that led to the creation of the Senseval evaluation exercises.

The Senseval-1 exercise involved 'lexical sample' tasks for English, French, and Italian (Kilgarriff 1998), essentially a community-wide version of evaluations previously conducted by individual researchers for words like *line* (Leacock et al.,

1993) and *interest* (Bruce & Wiebe 1994). As a community-wide figure of merit, Resnik and Yarowsky (1997) had suggested using cross-entropy rather than accuracy in order to accommodate systems with probabilistic output, thereby allowing a system $\mathcal{A}$ to obtain partial credit for word w_i even if the correct sense cs_i was not deemed most probable (cf. equation 6):

$$(8) \quad H = -\frac{1}{N}\sum_{i=1}^{N} \log_2 p_A(\text{cs}_i|w_i, \text{context}_i)$$

Senseval-1 adopted a variant of this suggestion proposed by Melamed and Resnik (2000), which accounted for fine- to coarse-grained distinctions in a sense hierarchy, and also permitted human annotators to specify a disjunction of correct answers in ground truth sense labelings.[27]

Following Senseval-1, other Senseval exercises continued for some time as the primary forum for evaluation of word-sense disambiguation. Senseval-2 dramatically expanded the scope of the exercise to include ten languages, using WordNet-based sense inventories. It also introduced 'all-words' tasks, requiring systems to assign a sense label to every content word within a document. Senseval-3 (Mihalcea & Edmonds 2004) continued lexical sample and all-words tasks, and added new semantic annotation tasks including semantic role labeling (Gildea & Jurafsky 2002), creation of logical forms, and sense disambiguation of the words in WordNet's definitional glosses. More recently, Senseval has become Semeval, a series of evaluation exercises for semantic annotation involving a much larger and more diverse set of tasks (Agirre et al., 2009).

5 Case Study: Evaluation of Question Answering Systems

In response to a short query representing an information need, a search engine retrieves a list of 'hits,' or potentially relevant results. The user must then manually examine these results to find the desired information. Given the amount of information available today on the web and in other electronic formats, typical queries retrieve thousands of hits. Question answering (QA) aims to improve on this potentially frustrating interaction model by developing technologies that can understand users' needs expressed in natural language and return only the relevant answers. From an algorithmic standpoint, question answering is interesting in that it combines term-level processing techniques (from information retrieval) with rich linguistic analysis. This section provides a case study on the evaluation of question answering systems.

The earliest question answering systems focused on fact-based questions that could be answered by named entities such as people, organizations, locations, dates, etc. A few examples of these so-called 'factoid' questions are shown below:

- What position did Satchel Paige play in professional baseball?
- What modern country is home to the ancient city of Babylon?

- Who was responsible for the killing of Duncan in *Macbeth*?
- What Spanish explorer discovered the Mississippi River?

For several years, the locus of question answering evaluation has resided at the Text Retrieval Conferences (TRECs).[28] TREC is a yearly evaluation forum, organized by the US National Institute of Standards and Technology (NIST), which brings together dozens of research groups from around the world to work on shared information retrieval tasks. Different 'tracks' at TREC focus on different problems, ranging from spam detection to biomedical text retrieval. Question answering occupied one such track from 1999 to 2007. During this time, the TREC QA tracks were recognized as the de facto benchmark for assessing question answering systems. These annual forums provide the infrastructure and support necessary to conduct large-scale evaluations on shared collections using common test sets, thereby providing a meaningful comparison between different systems. The TREC model has been duplicated and elaborated on by CLEF in Europe and NTCIR in Asia, both of which have introduced cross-language elements. This case study focuses specially on the TREC evaluations, recognizing, of course, that it merely represents one of many possible evaluation methodologies.

The TREC QA tracks occurred on an annual cycle. Several months in advance of the actual evaluation, the document collection to be used in the evaluation was made available to all participants, as well as results from previous years (to serve as training data). The actual evaluation occurred during the summer: participants were required to 'freeze' their systems (i.e., to conclude system development) before downloading the official test data. Results were due before a subsequent deadline (typically, about a week). System results were evaluated manually by a team of human assessors at NIST during the late summer or early fall. Each TREC cycle concluded with a workshop in November where all participants were invited to discuss their results and plan for next year.

One might think that evaluating answers to factoid questions would be straightforward, but even such a seemingly simple task has many hidden complexities. First is the issue of granularity: although the goal of a question answering system is to directly identify the answer, it might seem a bit odd if the system returned only the *exact answer* (i.e., a short phrase). Consider the question 'Who was the first person to reach the South Pole?' A response of 'Roald Amundsen' might not be very helpful, since it provides the user with little context (Who was he? When was this feat accomplished? etc.). Giving the user a sentence such as 'Norwegian explorer Roald Amundsen was the first person to reach the south pole, on December 14, 1911' would seem to be preferable – indeed, Lin et al. (2003a) present results from a user study that confirms this intuition.[29] In evaluating answers to factoid questions, what exactly should be assessed? Short phrases? Sentences? A case can certainly be made for evaluating answers 'in context,' but requiring exact answers makes the task more challenging and helps drive forward the state of the art. TREC eventually chose the route of requiring short, exact answers, accompanied by a document from which that answer was extracted.

Another issue is the notion of support: the document from which an answer derives should provide justification for the answer. Consider a sample question, 'What Spanish explorer discovered the Mississippi River?' An answer of 'Hernando de Soto,' extracted from a document that reads 'the sixteenth-century Spanish explorer Hernando de Soto, who discovered the Mississippi River ...' would be considered correct. However, the same answer extract from a document that says 'In 1542, Spanish explorer Hernando de Soto died while searching for gold along the Mississippi River ...' would be considered *unsupported*, since the passage does not actually answer the question. Of course, what counts as evidence varies from assessor to assessor.

Finally, for a number of questions there are simply differences in interpretation. A well-known example is the question 'Where is the Taj Mahal?' In addition to the famous structure in Agra, India, there is the Taj Mahal casino in Atlantic City, New Jersey. Whether or not the latter location was acceptable as an answer stirred quite a debate among both TREC assessors and participants. Such questions are actually not uncommon, especially since many noun phrases have ambiguous referents. This was resolved, somewhat arbitrarily, by instructing assessors to interpret such questions as always referring to the 'most famous' version of an entity.

Judging the correctness of system responses is the most difficult and time-consuming aspect of TREC QA evaluations. Once the appropriate label has been assigned (e.g., *correct*, *inexact*, *unsupported*, *incorrect*), computing scores for system runs is relatively straightforward. The official metric varied from year to year, but the most basic method to quantify system effectiveness is through accuracy – of all the questions, how many were answered correctly.[30]

The summative nature of the TREC QA evaluations provides a fair, meaningful comparison across a large number of systems. Official results from TREC are viewed as authoritative, and the best systems are often used as yardsticks for assessing the state of the field. There are, of course, downsides to the TREC question answering tracks. Organizing and administering each evaluation consumes a significant amount of resources and represents a significant investment from both NIST and the participants (in choosing to participate). The other obvious drawback of the TREC QA evaluations is the rigid yearly cycle.

To support formative evaluations for system development between each TREC event, researchers have developed regular expressions for answers that mimic the behavior of assessors, so that system output can be informally assessed without the need for human intervention.[31] The regular expressions were created by manually examining actual system outputs and assessors' judgments to capture correct answers for each question. However, these answer patterns were simultaneously too permissive and too restrictive. They were too restrictive in not being able to capture all variants of correct answers – it was very difficult to predict a priori all variant forms of correct answers, e.g., different ways of writing numbers and dates. At the same time, the regular expressions were too permissive, in giving credit to system responses that happened to coincidentally contain words in the answer (without actually answering the question). Furthermore, the answer patterns did not address the problem of support: although it was possible to use the

list of relevant documents from the manual assessment as a guide, the space of possible answer sources exceeded the number of documents that were assessed manually, making the automatic assessment of support problematic.

Despite inadequacies with using regular expression answer patterns, they were nevertheless useful for system development and for providing researchers with rapid experimental feedback – which is exactly the purpose of formative evaluation tools. The combination of annual summative evaluations at TREC and formative evaluations in between helped drive the state of the art in factoid question answering.

6 Summary

Evaluation plays a crucial role in the development of language technology. In this chapter, we have presented a set of fundamental evaluation concepts, descriptions of the most widely used evaluation paradigms, and two case studies drawing on the authors' experiences with evaluation of word-sense disambiguation and question answering systems.

NOTES

1 That book is a revised version of an earlier technical report (Galliers & Jones 1993). See also Palmer et al. (1990).

2 This illustration is modeled on the NLP-enabled Wikipedia search engine introduced by San Francisco-based Powerset in May 2008 (Auchard 2008). However, neither author of this chapter has any connection with Powerset, and our examples should not be relied on as an accurate depiction of its technology.

3 Notice that this example illustrates not only dependency tuple extraction but also one way to do normalization of dependency tuples. In particular, observe how a definite generic noun phrase *the . . . bulb* has been represented by a plural generic *bulbs*; cf. the semantic equivalence between *The dodo is extinct* and *Dodos are extinct* (McCawley 1993: 263ff). See Katz and Lin (2003) and references therein for additional discussion of search using dependency tuples.

4 This quote is attributed to "evaluation theorist Bob Stake" in Westat (2002: 8).

5 See Dorr et al. (2005) and references therein for actual studies along these lines.

6 This distinction is similar in some ways to *black-box* versus *glass-box* evaluation. The former is restricted to external measurements such as quality of output and speed, while the latter can take into account internal characteristics such as the quality of knowledge resources, as well as run-time internal workings of the system.

7 Note that it is conventionally assumed that the human annotators are working *independently*. In a post hoc review of one annotator's work, a second annotator is likely to give the first annotator's choice the benefit of the doubt if there is a gray area, even though she might well have made a different choice when annotating independently. Chapter 10 in this volume, LINGUISTIC ANNOTATION, discusses in detail a wide variety of annotation projects, as well as general principles and processes for annotation.

8 Supervised learning, a paradigm that currently dominates NLP systems development, requires the availability of annotated training data. 'Unsupervised' systems do not require annotated training data, but they are nonetheless evaluated using annotated test data. Today NLP evaluation is rarely done in the absence of annotated test material, though at times clever tricks have been used to automatically create 'pseudo-annotated' test items without requiring an investment in actual annotation (e.g., Gale et al., 1992a).

9 Recall from Section 2.2 that a summative evaluation can take place after a system has been completely developed, or at some milestone in its development. In NLP research, such milestones typically occur a few days (or a few hours, or a few minutes) before a conference submission deadline.

10 Other forms of non-point estimation are also sometimes used, e.g., bootstrapping, which involves sampling with replacement rather than creating a partition (Efron & Gong 1983; Yeh 2000b; Krymolowski 2001).

11 Though see comments on the most frequent baseline in Section 4.

12 For a highly approachable introduction to statistical hypothesis testing, we recommend Gonick and Smith (1994).

13 According to Doug Oard (personal communication), Karen Sparck Jones advocated this threshold because it has a clear interpretation in terms of the user's experience. He comments: ".10 corresponds roughly to finding one more document near the top of the list, and 0.05 corresponds roughly to finding one more document near the top of the list about half the time. These are usually applied to MAP [mean average precision] in practice, where the correspondence is quite approximate to those moves, but in precision at 10 the correspondence is perfect."

14 In contrast to the IR example, it must be noted that these absolute gains have no direct interpretation in terms of improvements in the user experience.

15 See discussion of the Senseval 'lexical sample' paradigm in Section 4.

16 This notation, based on Artstein and Poesio (2008), makes it easy to generalize from simple agreement to chance-correct agreement as discussed in Section 2.5. Our A is equivalent to their observed inter-annotator agreement A_0 between annotators, where one annotator is the NLP system and the other is the ground truth.

17 For example, Palmer et al. (2006) define *precision*, *recall*, *accuracy*, and *coverage* in such a way that accuracy and recall are synonymous. Other closely related concepts include misses versus false alarms, sensitivity versus specificity, and Type I versus Type II errors.

18 The list of relevant documents forms an essential component of a *test collection*, a standard experimental tool in information retrieval research. Test collections are typically constructed through large-scale system evaluations, such as the Text Retrieval Conferences (TRECs) (for more details, see Harman 2005).

19 The original version of the metrics considered constituents to match in the gold standard and system-output parse-trees as long as they bracketed the same span of tokens, regardless of constituent label. Requiring the non-terminal symbols to also match is a straightforward and rather stricter variant. The PARSEVAL metrics also included a third measure, 'crossing brackets,' that penalizes irreconcilable differences between the system parse and the gold standard; for example, *(the (old men) and women)* can be reconciled with *(the ((old men) and women))*, since it is just missing one pair of brackets, but there is no way to reconcile it with *(the old (men and women))*.

20 Every necessary edit constitutes an error, and so the acronym TER is also sometimes expanded as 'translation error rate.' Snover et al. (2006) use 'edit,' but Mathew Snover et al. (2005) used 'error.'

21 The use of these data types is not without controversy (see, e.g., Velleman & Wilkinson 1993).
22 This nice connection between the formal definition and the everyday idea of being perplexed is suggested in the Wikipedia page for *Perplexity*, June 2009.
23 Notice that in contrast to Kullback–Leibler distance, the computation of perplexity did not require knowing the 'true' distribution (for discussion, see Jurafsky & Martin 2009: 116ff.)
24 Yarowsky (1992: 458) observes that most previous authors had "reported their results in qualitative terms." He also cites a half dozen exceptions starting with Lesk (1986).
25 We have structured this discussion roughly following Palmer et al. (2006).
26 It must be noted that selecting the most frequent sense is best viewed as a supervised approach, and therefore an unfairly rigorous lower bound for unsupervised techniques, since accurately computing sense frequencies requires labeled training data. McCarthy et al. (2004) introduced an unsupervised method for finding predominant word senses in untagged text.
27 See Artstein and Poesio (2008) for an insightful analysis of this evaluation metric in the context of measuring inter-coder agreement.
28 Details about TREC can be found at http://trec.nist.gov/
29 Note that this finding illustrates potential differences between intrinsic and extrinsic evaluations.
30 See TREC QA tracks overview papers for details of different evaluation metrics that have been adopted.
31 For many years, Ken Litkowski headed up this effort.