University of Alberta

Leveraging supplemental transcriptions and transliterations via re-ranking

by

Aditya Bhargava

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2011
Edmonton, Alberta

Abstract

Grapheme-to-phoneme conversion (G2P) and machine transliteration are important tasks in natural language processing. Supplemental data can often help resolve difficult ambiguities: existing transliterations of the same word can help choose among a G2P system's candidate output transcriptions; similarly, transliterations from other languages can help choose among candidate transliterations in a given language. Transcriptions can be leveraged in this way as well. In this thesis, I investigate the problem of applying supplemental data to improve G2P and machine transliteration results. I present a unified method for leveraging related transliteration or transcription data to improve the performance of a base G2P or machine transliteration system. My approach constructs features with the supplemental data, which are then used in an SVM re-ranker. This re-ranking approach is shown to work across multiple base systems and achieves error reductions ranging from 8% to 43% over state-of-the-art base systems in cases where supplemental data are available.

Acknowledgements

> Gratitude is not only the greatest of virtues, but the parent of all others.
>
> *Pro Plancio*
> *Cicero*

First and foremost, I am deeply grateful to my supervisor, Greg Kondrak. To him I owe a great debt of gratitude for his guidance, mentorship, reference letters, and for putting up with me in general. Under his direction I've completed several projects, published a number of papers, and learned a great deal more than I ever thought I could. Thank you, Greg; I look forward to collaborating in the future.

Thanks are also due to a number of graduate students that were at some point or another collocated with me at the University of Alberta's Department of Computing Science. To name the guilty (in alphabetical order, of course), I extend my sincere thanks to Shane Bergsma, for the ideas and collaboration; to Ken Dwyer, for always giving me a fun challenge, both professionally and otherwise; to Sittichai Jiampojamarn, for the great discussions and feedback; and to Bernardo Ávila Pires, for looking over the SVM section of this thesis and the endless philosophizing.

As well, I am grateful to my thesis committee members David Beck, Vadim Bulitko, and of course Greg: thanks for managing to trudge through this thesis and for giving me some challenging questions reminiscent of my Google interviews. Thanks also to the committee chair Davood Rafiei for sitting through the whole thing, including my 40-minute presentation!

Finally, a decidedly non-finite amount of thanks is due to my family for the endless love, support, and for keeping me well-fed (despite my wishes). Thanks particularly to my parents, Sudeep and Abha, as well as my great (in both applicable senses of the word!) aunt Gura. And of course, thanks to my sister Supriya, my best buddy for life, without whom life would just not be as much fun—or any fun at all.

Contents

List of figures

List of tables

Chapter 1

Introduction

Imagine that you have been tasked with reading a news article aloud. Of the words in the document that you have not seen before, many are names, and you are not sure how to pronounce them. If you are having trouble thinking of such a name, consider "Eyjafjallajökull", the name of a volcano in Iceland whose eruption in 2010 was heavily reported due to the ash cloud that massively disrupted much European air traffic. Are the 'j' letters pronounced as in "Jack", "Jacques", "Jaeger", or is there some other pronunciation for them? What about the double-'l' groups? And what does that trema (umlaut) over the 'o' do?

In addition to being difficult for humans, pronouncing new words and names can be difficult for computational methods as well. Handling such cases gracefully is becoming increasingly important as we turn to speech synthesis systems to speak text aloud for us automatically. For example, *The Economist* provides podcasts for each issue, wherein each article is read aloud by a human. Ultimately, we would like to supplant the need for a human and use a speech synthesis system instead, especially if we do not have a similar magnitude of resources available as does *The Economist*.

Since we cannot hope to create an exhaustive pronunciation database of all possible words and names, speech synthesis systems will need to be able to make reasonable guesses for new words and names; this presents the same types of issues for computer programs as we saw for humans above. A *human* may have the option of asking an Icelandic friend to help, but a computer doesn't: to provide a pronunciation requires specific linguistic expertise. Even relying on pronunciation databases for other languages can be difficult due to differences in formatting and the dearth of such databases for many languages that have fewer speakers (and even with a suitable resource, it will also suffer from being unable to cover all possibilities).

While it is difficult at best to rely upon the computational equivalent of an Icelandic friend, we can instead take advantage of information that we already have. For example, assume that you can read not only English but Japanese and Russian as well. Resourceful person that you are, you turn to the Web and look for translations of the original news item. You find that in Japanese, "Eyjafjallajökull" is written as "エイヤフィヤトラヨークトル" and appears as "Эйяфьядлайёкюдль" in Russian. These help you resolve some of the ambiguities, which you can combine with your own knowledge of English to help your initial pronunciation of the name. Importantly, you do not follow the Japanese or Russian pronunciations exactly, as that would

sound strange to a native English speaker. Even if your pronunciation is not true to the original Icelandic, it will at least be more informed than if you had simply guessed. Similarly, a computer program can find foreign-language versions of new names from the Web and use them to help guide its own pronunciation.

Such translated versions of names are known as transliterations. Generating transliterations automatically is another task that has been researched from a computational perspective. In this case we can also refer to other transliterations to learn how to write out a difficult name; seeing the Russian equivalent of Eyjafjallajökull can help write out the Japanese version. Similarly, if we know the pronunciation of a name in some form that provides pronunciation information perfectly, we can use that to help as well.

The commonality between the various above cases is that related external data are being consulted to inform a base task. In this thesis, I present a unified method for leveraging such supplemental data to improve the performance of a base system. The base tasks that I explore are grapheme-to-phoneme conversion and machine transliteration, to each of which I apply supplemental transcriptions and transliterations.

1.1 Base task definitions

1.1.1 Grapheme-to-phoneme conversion

A **grapheme** is a written unit that represents a spoken sound; we refer to these more specifically in English as letters. **Phonemes** are more abstract: they are psychological representations of spoken sounds, a core unit of speech. Phonemes provide a way of dealing with the pronunciation of a word, while graphemes are simply the written representation of it. As it is not always clear what the pronunciation of a word is from its written representation, representing it with phonemes allows this information to be communicated in a written manner. This is known as a **transcription**. For example, I use the International Phonetic Alphabet (IPA)[1] to represent phonemes and transcribe words throughout this thesis.

Grapheme-to-phoneme conversion (G2P) (also known as **letter-to-phoneme conversion (L2P)** or **letter-to-sound (LTS)**), then, is the task of converting an input word to its phonetic representation, or transcription. While G2P is something that humans do without thinking, it is much more difficult to describe algorithmically for implementation in a computer program, where it is needed for speech synthesis (among others): G2P would tell a speech synthesis system which phonemes to produce. Simply storing transcriptions in a dictionary will not work, as there will always be new words that have not been seen previously, necessitating the development of general G2P systems.

[1] The casual (or else otherwise IPA-illiterate) reader can consult *Wikipedia* to get an idea of the sound represented by individual IPA characters. For example, the "English phonology" article provides examples for the various IPA characters used to represent English sounds, and also links to a concise chart.

1.1.2 Machine transliteration

To **transliterate** a name means to rewrite it in another orthography (language script) while trying to preserve the original pronunciation as closely as possible. Transliterating a name (or, more generally, a named entity) is important when translating documents, as we generally do not translate names by meaning. My own first name, for example, is a transliteration: the meaning is "sun", but my name in English is not "Sun"; instead, we try to write it such that the original Hindi pronunciation is maintained to whatever degree possible.

As we turn to computers to do our translation for us, we must enable them to transliterate as well. This task is known as **machine transliteration**, and has much in common with G2P: both are **string transduction** tasks, in which an input string is to be converted into another format; and for both tasks, preservation of pronunciation is paramount.

1.2 Leveraging supplemental data

I focus on applying supplemental data to these base tasks. Consider again the G2P case: our base G2P system is given a name that is difficult to pronounce. In this case, we can turn to transliterations, which can be mined from the Web. (Wikipedia, for example, often has copies of articles in multiple languages.) This is particularly true of news items, which are often reported around the world, especially if they are of global (or at least international) significance.

My objective is to use available transliterations to improve the performance of existing G2P systems. Many state-of-the-art G2P systems are able to provide multiple candidate transcriptions, ranked by the score that the system gives each of them; this is known as an n**-best output list**. This list presents an opportunity to re-order it according to the transliteration.

To provide a more concrete (and readable) example, consider the ambiguity in English of the phoneme associated with the letter 'G', which usually represents either /ɡ/ as in "Gertrude" or /d͡ʒ/ as in "Gerald". When considering a list of candidate transcriptions provided by a G2P system for "Gershwin", we may find that the top choice mis-predicts the initial phoneme as /d͡ʒ/ instead of /ɡ/. The correct transcription may still appear in the output list, however; existing transliterations can help indicate which of the transcriptions in the output list is indeed the correct choice.

It may seem simple conceptually to use transliterations to disambiguate transcriptions, but doing so computationally is non-trivial. Different languages have different phonologies, or sets of spoken sounds, and mapping sounds between languages is usually a complex affair. For example, the Hindi /ʋ/ phoneme, similar to the English /v/, is usually represented with a 'w' in English. Similarly, English words and names containing 'w' are written with the character for /ʋ/, so in a Hindi transliteration of an English or other foreign name such as "Gershwin", the /w/ phoneme is lost. This indicates that considerations must be made for the specific circumstances under which a transliteration from another language will be helpful in pronouncing a name. Even without such an issue, converting transliterations into transcriptions is often complicated; although few orthographies are as incon-

sistent as that of English, this is effectively the G2P task for the particular language in question (which is an active research area for many languages).

This scenario of applying transliterations to G2P can work for machine transliteration as well. When transliterating from English to Hindi, for example, Japanese or Russian transliterations can help disambiguate certain characters. Similarly, transcriptions could also help machine transliteration, and transcriptions from one **corpus** (data set) could be used to inform G2P systems trained on another. These are the four cases on which I focus.

The key, as noted above, is that the base systems are all capable of providing output lists. I apply machine learning methods to leverage multiple supplemental data sources, such as transliterations from multiple languages or transcriptions, to re-order the output lists of existing base systems for G2P and machine transliteration, achieving a substantial error reduction for all cases and across multiple base systems; my approach can operate on any base system that provides an n-best output list. Notably, I am able to achieve these results using similar information as is included in the base systems, but applied in a new way across supplemental data.

Chapter 2

SVM re-ranking

This chapter provides a brief introduction to Support Vector Machines and their application to re-ranking. Note that the focus of this thesis is not Support Vector Machines; rather, they are used as a tool for accomplishing another task. I therefore do not survey them in-depth and instead focus on the background aspects relevant to this thesis, namely their fundamental concepts and how they are applied to re-ranking.

2.1 Support Vector Machines

In their simplest and most commonly used formulation, Support Vector Machines (SVMs) (Cortes and Vapnik, 1995) are binary classifiers: given an input, they predict to which of two classes the input belongs. Such binary classification can obviously be applied to simple yes/no questions, such as whether a given brain scan indicates the presence of a tumour or not, or (to provide a more NLP-based example) whether a given text is in English or not. This latter example is known as language identification, an instance of the task of text categorization.

As with most machine learning approaches to classification, each input must be represented by a set of numbers called **features**; these form the feature vector for the input, denoted by $\mathbf{x}$. Each feature is representative of some aspect of the input; in the language identification example, we may have a binary feature for every character. Then, for a given input (e.g., a document, sentence, or word), every character in the input would "activate" the corresponding feature: a feature value of 1 would indicate that the character occurs in the input, while 0 would indicate that it does not. Alternatively, the features might not be binary and instead indicate the number of times the character occurs in the text; the value of this kind of feature can indicate the degree to which a certain property is "expressed", to borrow a biological term. In this example, it should be noted that single characters do not provide enough information to discriminate between languages; groups of n characters, or n**-grams**, must be counted. As we increase n, our number of features drastically increases; machine learning methods allow us to use potentially millions of criteria (in the form of features) to inform the classification decision, although they do have their own limits. Each feature can be thought of as a dimension (axis) in a multi-dimensional hyperspace, in which case each feature vector would be a point in this space.

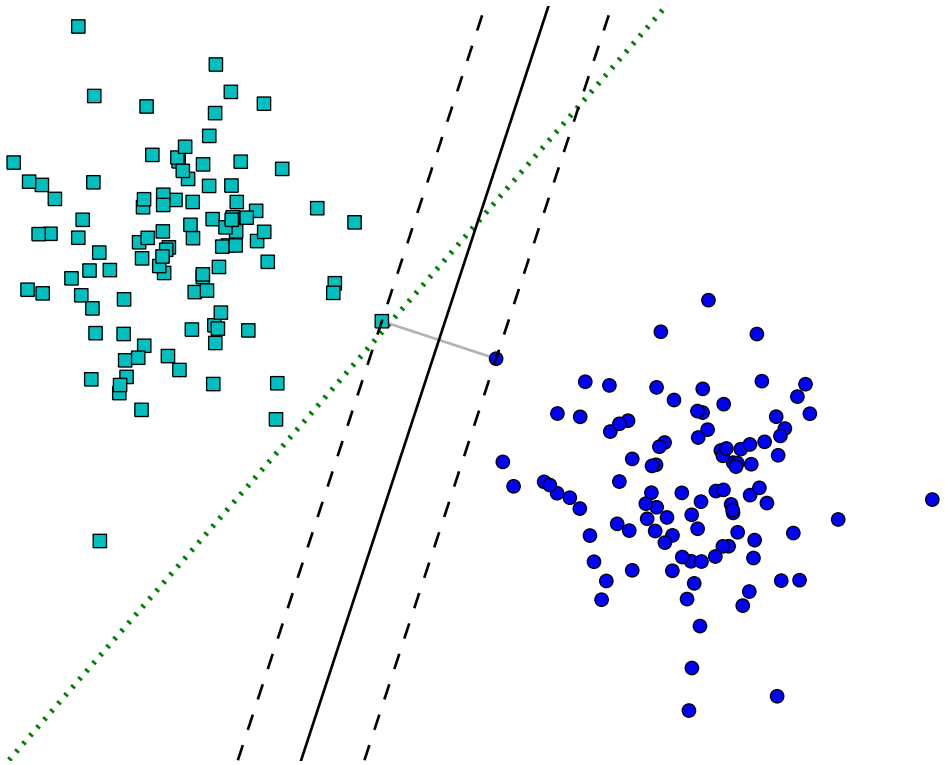

Figure 2.1: The decision boundary found using SVMs (solid line) as compared to that found using perceptrons (dotted line). The perceptron finds a separating hyperplane, but not one that maximizes the margin between the hyperplane and the data points. In this case, there are two dimensions in the feature vector (so the hyperplane is a line); points belonging to the 1 class are shown as squares, while points belonging to the -1 class are shown as circles. The SVM decision boundary has the equation $\lambda \cdot \mathbf{x} - b = 0$ and the margins (shown as dashed lines) have the equations $\lambda \cdot \mathbf{x} - b = \pm 1$. The support vectors are the points that are on the margins, also indicated by the grey lines between the points and the SVM decision boundary.

The actual output class of a given input $\mathbf{x}$ is denoted by y; the predicted class is denoted by $\hat{y}$. In binary classification, the outputs are usually defined such that $y \in \{-1, 1\}$; -1 is used to represent one class, while 1 represents the other. SVMs are linear classification methods: they model the output as a linear combination of the input features, with each feature having a corresponding weight in a weight vector λ. Formally, a linear classification method has $\hat{y} = \text{sign}\,(\lambda \cdot \mathbf{x} - b)$. This means that all points on one side of the hyperplane given by $\lambda \cdot \mathbf{x} - b = 0$ belong to one class, and all points on the other side belong to the other class; this hyperplane is therefore referred to as a separating hyperplane or a decision boundary. b is a bias term that prevents the decision boundary from being forced to go through the origin. (In two dimensions, the bias term corresponds to the y-intercept.) Given the linear classification formulation, the objective is to learn the weight vector such that the predicted output $\hat{y}$ matches the actual output y for as many new (unseen) instances as possible. SVMs are one of many possible methods for learning the weight vector.

SVMs are trained on a set of data (the **training set**) that provides many example

input/output pairs; this training set is disjoint from the **test set** that is used to evaluate the final model because most modern machine learning methods can achieve near-perfect performance on the training set that does not usually generalize well to unseen (test) data. The training data are used by SVMs to learn the weight vector. The actual method used by SVMs to learn the weights has a geometric intuition: the line (decision boundary) that is learned should be such that it separates the two classes with the largest possible margin, allowing for better classification performance on unseen data.

Figure 2.1 shows this intuition, compared to the decision boundary found by the simple perceptron algorithm (Rosenblatt, 1958). The two point classes have each coordinate sampled from normal distributions. Ultimately, we want to learn a classifier that can separate the means of the distributions while remaining resilient to the variance. So while the perceptron can separate the training data correctly, it is not likely to work well for new test data that are sampled from the same distributions.

To convert the SVM intuition into computable mathematics, assume that we have two parallel hyperplanes given by $\lambda \cdot \mathbf{x} - b = \pm 1$. We wish to maximize the distance between these two hyperplanes while ensuring that no data come between them, so each datum $\mathbf{x_i}$ has either $\lambda \cdot \mathbf{x_i} - b \geq 1$ or $\lambda \cdot \mathbf{x_i} - b \leq -1$. The distance between the hyperplanes is given by $\frac{2}{|\lambda|}$, so maximizing the margin is equivalent to minimizing $|\lambda|$. This is expressed as a quadratic programming optimization problem (note that i indexes our data, so $\mathbf{x_i}$ refers to an individual feature vector used to represent one point):

$$\begin{aligned} \min_{\lambda, b} \quad & \tfrac{1}{2} |\lambda|^2 \\ \text{s.t.} \quad & y_i \left(\lambda \cdot \mathbf{x_i} - b \right) \geq 1, \quad \forall i \end{aligned}$$

which can be solved using standard quadratic programming techniques (though in practice other faster algorithms such as coordinate descent are often used). Note that the $\frac{1}{2}$ term is a mathematical convenience, so that there is no coefficient after taking the derivative. This optimization problem is convex, so an optimal hyperplane will be found as long as the classes are linearly separable. The optimization returns the vectors that lie on the margin; these are all that are needed to construct the equation for the decision boundary (and hence, the weight vector) and as such are called the support vectors (hence the term Support Vector Machine).

Linear separability is an important caveat; real-life problems often present data that are not linearly separable. SVMs provide two ways of dealing with this: soft margins and kernels. Soft margins relax the constraint that all points must be outside the margins, allowing for some points to be mislabelled. This is accomplished by means of slack variables that allow for misclassified points up to a certain degree, controlled by a cost hyperparameter C. The modified optimization becomes:

$$\begin{aligned} \min_{\lambda, b, \xi} \quad & \tfrac{1}{2} |\lambda|^2 + C \sum \xi_i \\ \text{s.t.} \quad & y_i \left(\lambda \cdot \mathbf{x_i} - b \right) \geq 1 - \xi_i, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{aligned}$$

where C is the cost hyperparameter that controls how much the slack is penalized; as $C \to \infty$, the problem approaches the original hard-margin SVM problem. Figure 2.2 shows the SVM decision boundary for a problem that is not linearly separable. Note that, even if a given problem is linearly separable, the decision boundary that

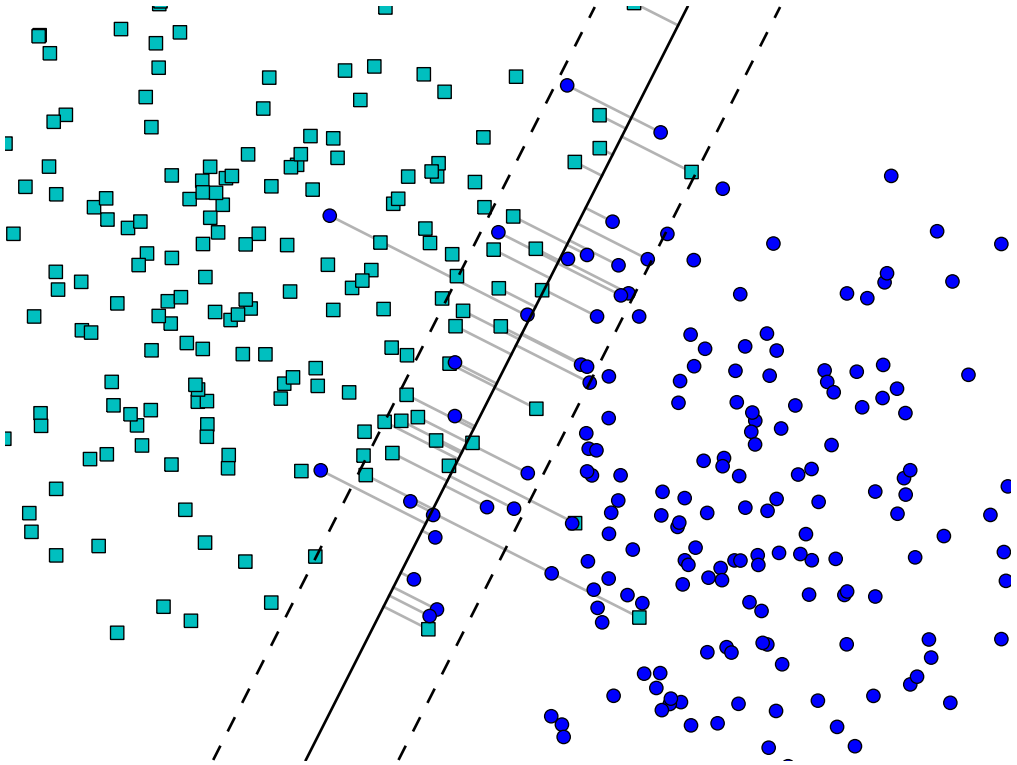

Figure 2.2: The decision boundary found using soft-margin SVMs for a non-linearly-separable problem. The perceptron algorithm would not converge for this problem while the SVM provides a good decision boundary even though there are clearly circles that penetrate square territory and vice versa. Points and lines are formatted as in Figure 2.1. Note that there are many more support vectors in this case, and there is no need for them to lie exactly on the margin thanks to the slack variables.

maximizes the margin may not be ideal if, for example, one or both of the classes has or have outliers. Allowing some misclassified points further helps in isolating the signal (the means of the two distributions) from the noise (the large variances of the distributions).

In practice, various values of C are tried: models trained using different values of C[1] on a common training set can be evaluated on a **development set**. The model yielding the best results provides an "optimal" value for C, which can then be used to train models for final system evaluation. The development set is used so that we do not unfairly advantage our systems by tuning our hyperparameters to be optimized for the testing data. Alternatively, cross-validation can also be used; in ten-fold cross-validation, for example, the training set is split into ten folds. One fold is designated for development, and the models are trained on the remaining data; this is repeated with each successive fold being used for development.[2]

Non-linear kernels allow SVMs to be applied to problems that are not linearly

[1] I usually use powers of two to try a large enough range of parameters; some use powers of ten.

[2] Held-out sets and cross-validation have important (theoretical) distinctions in what they optimize, but in my experience cross-validation usually works well for NLP tasks unless the increased computation time is an issue.

separable by mapping the input problem into a higher-dimensional feature space. Polynomial kernels allow n-degree mappings, while the radial basis function (RBF) kernel is equivalent to an infinite-dimensional mapping. These allow the learning of nonlinear decision boundaries. In my experience, using non-linear kernels has seldom provided significantly better results than the simple linear kernel used in the standard SVM formulation and are additionally cumbersome in that they require additional hyperparameter tuning (the RBF kernel, for example, has one additional parameter that must be tuned). Kernels also significantly slow down the entire process. For these reasons, I do not use any SVM kernels in this thesis.

There are several free SVM packages that are available. LIBSVM (Chang and Lin, 2001) is quite popular and available under the very permissive BSD license. LIBLINEAR (Fan et al., 2008) is very fast, able to solve problems with millions of features and thousands of data points in minutes, if not seconds. Like LIBSVM, it is available under the BSD license; unlike LIBSVM, it does not support any kernels. Since, as noted above, I do not use non-linear kernels, I employ LIBLINEAR to implement SVMs in this thesis.

2.2 Applying SVMs to re-ranking

Sometimes, we have an existing system for a task to which we would like to add information but cannot do so directly. This system provides multiple possible outputs for a given input, but does not always place the correct output at the top. This potentially applies to many NLP tasks: syntactic parsing, machine translation, etc. For a more intuitive example, consider search engines: imagine that we have an existing search engine that does not always place the best document in the top position, as indicated by user clicks on documents lower in the list. Because user clicks are not available during the design of the system (and may not be easy to incorporate), we can **re-rank** the search engine output using the user click information. We may also want to incorporate results from other search engines.

SVMs can be applied to re-ranking (Joachims, 2002), bringing with them their ability to apply many features to the problem. To understand this application, we must first formalize the problem; it is broadly defined as follows. We are given an input s; this is the overall task input, such as a sentence for parsing or a query for a search engine. For each input s we have m possible outputs $t_{i=1\cdots m}$; these are possible parse trees or documents for search engine retrieval. We define an integral rank y_i for each output t_i such that if $y_i > y_j$ then t_i is to be ranked higher than t_j. Given an input s and an output t_i, we have a featurizing function $\mathbf{F}(s, t_i)$ that returns a corresponding feature vector. I abbreviate $\mathbf{F}(s, t_i)$ as f_i where the input s is implied.

Ultimately, we are learning the coefficients (the weight vector) for a linear combination of the elements of the feature vector. Intuitively, we would like a weight vector λ that can reproduce our desired rankings; if $y_i > y_j$, we would like to have:

$$\lambda \cdot \mathbf{f_i} > \lambda \cdot \mathbf{f_j}$$

To this we can apply the SVM's maximum-margin intuition; we aim to learn a weight vector that ranks f_i higher than f_j by a certain margin. Adding this mar-

gin and the soft-margin slack gives:

$$\lambda \cdot \mathbf{f_i} \geq \lambda \cdot \mathbf{f_j} + 1 - \xi_{i,j}$$

We can rearrange this to get an inequality that is exactly like the SVM constraints:

$$\lambda \cdot \left(\mathbf{f_i} - \mathbf{f_j}\right) \geq 1 - \xi_{i,j}$$

The casting of re-ranking as an SVM problem is becoming clear. For convenience, we define $p(s)$ as the set of all output pairs (t_i, t_j) such that $y_i > y_j$. Where SVMs used for classification have one constraint for every input feature vector, SVMs used for re-ranking have several constraints (plus corresponding slack variable constraints): for a given input s, there is one constraint for every pair in $p(s)$. When we consider all of our data $s_{i=1\cdots n}$, we have the following quadratic programming optimization problem:

$$\begin{aligned} \min_{\lambda,\xi} \quad & \tfrac{1}{2}\left|\lambda\right|^2 + C\sum \xi_{i,j,k} \\ \text{s.t.} \quad & \\ \forall\,(t_j,t_k) \in p(s_i): \quad & \lambda \cdot \left(\mathbf{F}\left(s_i,t_j\right) - \mathbf{F}\left(s_i,t_k\right)\right) \geq 1 - \xi_{i,j,k}, \quad \forall i \\ & \xi_{i,j,k} \geq 0, \quad \forall i,j,k \end{aligned}$$

We can see that this is equivalent to the SVM classification problem: we need only make the simple substitution $\mathbf{x_i} = \mathbf{F}\left(s_i,t_j\right) - \mathbf{F}\left(s_i,t_k\right)$. Intuitively, we can interpret SVM re-ranking as a pairwise classification problem: given two outputs, should one be ranked higher than the other? The mechanics of the SVM allow us to do this without much added complexity: we need only compute the difference between two feature vectors. While there exist packages to do SVM ranking, such as SVM^{rank} (Joachims, 2006), I directly convert SVM ranking problems into SVM classification problems using a custom Python script, which allows me to use the very fast LIBLINEAR (Fan et al., 2008) while achieving the same overall performance.

2.3 Training SVM re-rankers

When training standard SVMs, the data (feature vector inputs with class label outputs) are supplied directly to the SVM. The algorithm does the optimization to find the optimal weights according to the SVM optimization equations above. When using SVM re-rankers, however, the training process is not as simple. The key is that we are *re*-ranking a base system's output, and in our ultimate use for an SVM re-ranking model, the re-ranker will be applied to a system's output on *unseen* test data. If we feed the problem training set into the base system, it will (usually) learn a model that works very well on the training set but less so on test data. If we then run the model on the training data to produce outputs for re-ranking, our re-ranker will not be able to learn how the re-ranking should be done in the general unseen case. The amount of information in training data and test data often differs as well: when counting n-grams, for example, features are constructed only for n-grams seen in the training set; new n-grams in the test set are not counted, and even if they were, the system would have no idea what to do with them since they never showed up in the training data.

The re-ranker, then, must be run on outputs that the system has not seen before. At the same time, we want to give the re-ranker as much training data as we can in order to obtain the best possible re-ranking model. My solution to this applies a process similar to cross-validation mentioned above. I split the training data into ten folds and train ten separate base system "sub-models", each one using a different set of nine folds. Each sub-model is then tested on the remaining fold of unseen data in the training set; the results from each of the sub-models are then concatenated together and the result is used as the training set for the SVM re-ranker. This gives me system outputs that are indicative of the base system's performance on unseen data while preserving the size of the training set.

Avoiding such concessions can be disastrous. If we train the re-ranker on system outputs that result from running the system on previously-seen (training) data, the re-ranker will quite likely learn that the base system is almost always right, thereby becoming effectively useless.

Finally, there are two important points that should be noted for training any kind of SVM and that apply to many other machine learning methods as well:

1. All features should be properly scaled. The usual way—as provided in the LIBSVM package—is to scale each feature linearly so that it falls between zero and one. If this is not done, performance generally suffers, as features with larger ranges tend to dominate the linear combination, which makes learning optimal weights more difficult. I have never seen a performance decrease from scaling—only an increase–so I always do feature scaling.

2. Certain feature types work better if the logarithm of the feature value is computed prior to scaling. This is particularly true if we are more interested in the *ratio* between two feature values than the *difference*. Ultimately, SVMs use linear combinations; this means that two data points are differentiated from each other according to the differences between their feature values (this is explicitly the case for SVM re-ranking). Probabilities are an example of one such feature type: we are usually interested in the ratio of one probability to another, not the difference. This applies to word or character counts as well (which can be interpreted as unnormalized probabilities).

Chapter 3

Related work

This chapter provides a brief look at previous work that is related to this thesis: I discuss grapheme-to-phoneme conversion, machine transliteration, and the combination of languages and systems for various tasks. Since the focus of this thesis is not the core tasks of G2P or machine transliteration, I do not survey the relevant methods in-depth and instead focus on the aspects that are especially relevant to this thesis. In particular, I explicitly describe the Festival, Sequitur, and DirecTL+ systems as I make use of them directly.

3.1 Grapheme-to-phoneme conversion

The ultimate use of G2P is the crucial role it plays in speech synthesis. From this perspective, we need to justify the development of general G2P methods. Is it really necessary to have complex methods that can convert an arbitrary string into its corresponding transcription? Would a simple pronunciation dictionary suffice?

In fact, pronunciation dictionaries form the basis for the popular Festival[1] end-to-end speech synthesis system: words have their transcriptions retrieved from a stored dictionary or **lexicon**. Unfortunately, word look-up turns out to be insufficient. The distribution of words is described by Zipf's law (Manning and Schütze, 1999, pp. 23–25): the frequency of a word is roughly inversely proportional to its rank order, or its position in a list of all words sorted by descending frequency. In layman's terms, this means that a few words occur very commonly (such as "the", "of", etc.), but most occur rarely. Black et al. (1998) found that in a corpus of news-style text, 4.6% of the words were words that did not occur in their lexicon; these are known as out-of-vocabulary or **OOV** words. Zipf's law makes it extremely difficult, if not impossible in practice, to construct a pronunciation dictionary that covers close to 100% of words.

One way of dealing with OOV words is to construct hand-written G2P rules. Intuitively, it seems simple: for a given word, we can look at it and come up with a rough collection of rules for why we pronounce each letter the way we do. For example, in the word "car", the 'a' is pronounced as /ɑ/, but in the word "care", it is pronounced as /e/. We attribute this to the 'e' at the end of the latter. Following this reasoning, we should be able to construct general transcription rules that should

[1] http://www.cstr.ed.ac.uk/projects/festival/

work for any unseen words.

Unfortunately, this intuition does not follow through, at least for English. It is not always possible to provide reasoning for a certain pronunciation, other than "that's just how it is". For example, the 'o' in "move" is pronounced as /u/ but as /ʌ/ in "love"; that this is due purely to the 'l' versus the 'm' is debatable. In general, manually constructing transcription rules is only feasible for languages where the orthography (writing script) has a high correspondence with the pronunciation; such languages are said to have more **transparent** orthographies. Spanish is one such language (Kominek and Black, 2006).

Even for a highly transparent script, it can be tedious to construct rules manually. Machine learning methods can help do so automatically, provided a sufficiently large training corpus. Once we have some sort of transcription model for OOV words (whether hand-built rules, rules learned via machine learning, or some other machine learning method), our general transcription process starts with looking up the word in the pronunciation lexicon; if it is not found, we then fall back to the transcription model. As a side note, if memory is an issue (as in embedded environments) we can reduce the size of the stored lexicon by removing the entries that the model can predict correctly. If, as is the case for many modern machine learning methods, the model can achieve near-perfect accuracy on the training data, this allows the removal of almost the entire lexicon.

3.1.1 Names

Names have often received special treatment in terms of G2P because they do not always follow the same rules as regular words. While names are generally written so as to roughly indicate pronunciation, they are often coloured by their etymological origins and can present groups of letters for which pronunciations are not readily available, or else may use unusual or rare pronunciations (in context) for certain letters. Pronunciation is further confounded by conventions used in writing words and names from one language in another's script (i.e., transliterating them). For example, modern Hindi speakers often drop word-final schwas, but this is not always reflected in the written form. My last name "Bhargava" is a good example of this: in Hindi, it is pronounced without the final 'a', as /bʰaːrgəʊ/ (Masica, 1991, pp. 107, 110). The convention is to include the final 'a' in the English spelling; this convention likely comes from Sanskrit romanization, where schwas are only deleted when marked explicitly in the source text. This process of schwa deletion occurs in a number of Indo-Aryan languages[2] (Masica, 1991, pp. 149–150), so knowing that a given name is of Indian origin may be helpful in pronouncing or transliterating it (Bhargava and Kondrak, 2010).

Names are a highly productive class: new, never-before-seen names are appearing all the time, especially in the news, which reduces the utility of simply storing names and their transcriptions (while the expense remains the same). Black et al. (1998) observed that of the words that were not in their lexicon, 76.6% were names. This tells us that, seeing as our G2P methods are used to supplement lexica for when OOV words are encountered, we must have G2P approaches that can handle names

[2] There has been some work in modelling schwa deletion computationally (Choudhury et al., 2004; Tyson and Nagar, 2009), which is especially relevant to G2P systems for these languages.

well. The utility of treating names specially, however, seems to vary per language. German names show a higher error rate than regular words (Kienappel and Kneser, 2001) and separate name-specific phoneme-to-phoneme converters have been applied to post-process the output from a G2P system for Dutch names (Yang et al., 2006; van den Heuvel et al., 2007). For English, however, Black et al. (1998) report similar accuracy on names as for other types of English words.

3.1.2 Festival

The popular Festival speech synthesis system follows the above transcription process: it stores lexica, which are consulted to obtain a transcription for a word. When an OOV word is encountered, Festival falls back on its general G2P model: a classification decision tree, also referred to as CARTs for Classification And Regression Trees.

CARTs apply machine learning to the concept of decision trees. The output for a given input is predicted by asking a series of questions about the input; the particular questions are learned automatically from training data. For example, in the "car"/"care" example, we might write a rule to pronounce 'a' as /ɑ/ unless two characters later there occurs a word-final 'e', in which case it is to be pronounced as /e/ (this is, of course, a gross oversimplification). CARTs would learn such a rule—in actuality, a series of rules—from the data.

The problem is set up as a classification problem: for each letter in the input word, the task is to predict the corresponding phoneme. In addition to the usual phonemes, a letter may also produce a **null phoneme** ϵ, in effect meaning that the letter is "deleted" from the pronunciation (for example, in the word "phoenix", we may say that the 'o' is deleted, so it would be mapped to the null phoneme). The features presented to the decision tree include the letter being considered, the four preceding characters, and the four following characters. The decision tree is trained on the featurized lexicon to learn the appropriate rules; this requires that individual letters in the words in the training data be aligned to the corresponding phonemes. This alignment is accomplished by means of an algorithm that requires a predefined list of allowable phonemes for each letter, which then picks the most likely phoneme for each letter using frequencies gathered from the entire training set. While Festival's CART-based classification approach makes sense at first glance, it suffers from the problem that the prediction of the phoneme(s) generated by a given letter is independent of the phoneme(s) predicted for the previous letter (Jiampojamarn, 2010).

3.1.3 Sequitur

Sequitur (Bisani and Ney, 2008) is a G2P system, not an end-to-end speech synthesis package like Festival. Sequitur implements the joint n-gram model, which combines graphemes and phonemes together into units dubbed graphones; multiple graphemes are allowed to be grouped with multiple phonemes. Probabilities of graphone sequences are computed using the n-gram approximation: the probability of an individual graphone in a sequence is given by how often it follows the previous $n-1$ graphones in the training data. The probabilities of each graphone in a given sequence are then multiplied together to yield the probability for the whole

sequence. To generate a transcription for a given input, the phoneme sequence that gives the highest graphone sequence probability is chosen. The segmentation parameters that split word-transcription pairs into graphone sequences are learned using the Expectation-Maximization (EM) algorithm.

Bisani and Ney (2008) report that SEQUITUR significantly outperforms methods based on decision trees, and in fact is a top-performing system. The joint n-gram approach works in a left-to-right manner, and the use of graphones as the operational unit allows SEQUITUR to construct better phoneme sequences, partially by making use of information regarding previous phoneme decisions.

3.1.4 DIRECTL+

SEQUITUR is a generative approach: it explicitly models the relationship between the inputs and the outputs, and the underlying machine learning algorithms aim to learn the parameters of that model. By contrast, DIRECTL+ (Jiampojamarn et al., 2010a) implements a discriminative method: the focus of the machine learning algorithm is to learn parameters that increase system performance. SVMs are also discriminative; they learn to discriminate between classes and the classes are modelled simply as linear combinations of features (which, in turn, are extracted from the inputs).

DIRECTL+'s features are based on groupings of graphemes and phonemes, necessitating alignment of the training data. This is done using M2M-ALIGNER (Jiampojamarn et al., 2007), which is an unsupervised alignment system, meaning that it is learns how to align sequence pairs using a corpus of *unaligned* sequence pairs (i.e., without seeing example alignments). M2M-ALIGNER is named as such because it allows the construction of many-to-many alignments rather than being forced to make one-to-one alignments that may not always make sense. For example, the 'ph' letters often map to the phoneme /f/ in English.[3] In a one-to-one or even one-to-many alignment, we would have to set either 'p' or 'h' as producing the null phoneme (i.e., being deleted; this is how FESTIVAL does it), when in fact the phenomenon at work is that the two *together* produce /f/. With many-to-many alignments, 'ph' can be aligned directly to /f/, allowing this two-to-one phenomenon to be expressed. M2M-ALIGNER's algorithm is an extension of an EM algorithm for learning a function to calculate the string edit distance (the number of edits between two strings) from a corpus of string pairs (Ristad and Yianilos, 1998).

Given the aligned training set, DIRECTL+ constructs feature vectors for each input-output pair and uses an online training algorithm—meaning that each training datum is processed one-by-one rather than processing all data together as a batch—to learn weights for the features. For each input, a number of possible outputs are generated, correct outputs identified, and then an SVM-like optimization is performed to increase the score of the correct outputs as compared to the others (this is very similar to the SVM re-ranking optimization constraints). To generate the possible outputs for a given input sequence, DIRECTL+ employs a phrasal decoder that uses the features and weights to successively build up candidate outputs.

It should be noted that DIRECTL+ is an extension of the original DIRECTL (Jiampojamarn et al., 2008). DIRECTL+ provides new features based on joint n-grams,

[3] Examples such as "aphelion" suggest that morphology may play a role in 'ph' producing /f/.

demonstrating an advantage of the discriminative (feature-based) approach: provided that they can be appropriately featurized, other methods can be incorporated into the discriminative model. Thanks in part to these extra features, DirecTL+ outperforms Sequitur, making it the state-of-the-art G2P system (Jiampojamarn et al., 2010a).

3.2 Machine transliteration

Conceptually, machine transliteration is very similar to G2P: an input string must be converted into a string in another script, with pronunciation being the key quality that is preserved. For G2P the output is a phonetic transcription; for transliteration, it is a representation in another language's script.

Linguistically, the process is more complex. When humans transliterate something, we consider the pronunciation of the source word in the source language, and then approximate that pronunciation in the target language.[4] The approximation is then used to write out the transliteration in the target orthography. Note that, because of the differences in the source- and target-language phonologies and orthographies, machine transliteration can be considered more complex than G2P. Knight and Graehl (1998) use a four-stage system that realizes the above conceptual process, and this includes a (specialized) G2P system.

The uses of transliterations are also different. Where G2P finds obvious application in speech synthesis, transliterations are vital to machine translation systems: named entities are transliterated (i.e., the *pronunciation* is roughly preserved), whereas ordinary words are translated (i.e., the *meaning* is roughly preserved). This has led some to refer to transliteration as phonetic translation (Li et al., 2009a).

Given this application, transliteration is usually applied only to named entities, although sometimes dictionary words are transliterated as well[5]. Given that names are difficult to handle just by storing them (see Chapter 3.1.1), the old-hat G2P method of storing transcriptions and reverting to a model when an OOV word is encountered will not work; machine transliteration needs a general approach. At the same time, transliterations are produced by humans more easily than are phonetic transcriptions: they do not require phonetic knowledge, and are a necessary byproduct of translations, which are appearing constantly (for international news stories, for example).

The 2009 and 2010 Named Entities Workshops had shared tasks in which teams submitted machine transliteration systems for evaluation on a common corpus (Li et al., 2009b, 2010). Both phoneme-based systems, where the conversion process is grapheme→phoneme→grapheme, as well as grapheme-based systems, for which the conversion process is grapheme→grapheme, were submitted; the ones that I

[4] If the objective is to *conserve* the *original pronunciation* as much as possible, which it usually is for transliteration, the transliterator should be able to produce the original pronunciation and therefore his or her proficiency should be in the less transparent language (or whichever language has the least transparent orthographic representation for the given name). This prevents opaque source names from being mis-transliterated and allows for correct resulting transliterations.

[5] For example, I would transliterate "University of Alberta" in its entirety, even though the only name is "Alberta"; the three words together refer to a single named entity and are hence all transliterated.

consider here are grapheme-based and are in fact G2P systems applied to transliteration. In particular, Finch and Sumita (2010) applied Sequitur along with a phrase-based machine translation system together to perform the transliteration task. Jiampojamarn et al. (2009, 2010b) applied DirecTL+ with some optional language-specific enhancements that I do not consider here. Both approaches yielded top results, though the rankings varied per-language.

Interestingly, we can see the increase in complexity of transliteration over G2P from the increase in performance seen using DirecTL+ with language-specific approaches. While DirecTL+ was applied directly to each transliteration task, various modifications were made on a per-language basis (converting Chinese to Pinyin, for example), and results were generally higher with these modifications than without. Furthermore, word accuracies for G2P are usually above 70%, depending on the specific corpus, while transliteration word accuracies fall much lower (such as around 45% for English-to-Hindi transliteration) for comparable corpus sizes.

3.3 Combining languages and systems

I focus in this thesis on G2P and machine transliteration, particularly on how data from the two tasks can be used to inform models for either. This is conceptually similar to approaches for model and system combination. In statistical machine translation (SMT), methods that incorporate translations from other languages (Utiyama and Isahara, 2007; Cohn and Lapata, 2007; Wu and Wang, 2009) have proven effective: when phrase translations are unavailable for a certain language, one can look at other languages where the translation *is* available and then translate from that language. A similar pivoting approach has also been applied to machine transliteration (Zhang et al., 2010); Khapra et al. (2010) refer to this as transliterating through "bridge" languages. Notably, many of these works have focused on cases in which there are less data available. Others have incorporated paraphrases (Callison-Burch et al., 2006), and applying multiple languages has also yielded success for part-of-speech tagging (Snyder et al., 2009).

Finch and Sumita (2010) combined two very different approaches to transliteration using simple linear interpolation: they merged Sequitur's output with that of a phrase-based SMT system, and then re-ranked the entire list using a linear combination of the scores from the two systems. The linear weights were hand-tuned. Similarly, Matusov et al. (2006) looked at the output of multiple machine translation systems, and used the various hypotheses to re-order each other, choosing a consensus hypothesis from a constructed confusion network.

Finally, Loots and Niesler (2009) considered the problem of adapting G2P corpora for one accent to another accent—the British Received Pronunciation (RP) to South African English (SAE), for example. They found that given a word for which there was no entry in the SAE corpus, it was better to find the same word in the RP corpus and convert its transcription than to try to guess the transcription in SAE directly from the word. Results were similar in the opposite direction and to and from American English corpora. However, their experiments were limited to decision trees, which as noted above perform rather poorly compared to newer methods (see also Appendix B), so it is not clear that this finding would extend to state-of-the-art G2P systems. Still, it represents a possible way of applying the information from

one G2P corpus to improve the performance of a G2P system on another corpus.

These approaches all combine various data of the same type or systems that perform the same task; my focus in this thesis is on additionally looking at *other* tasks for potentially useful information (transliterations for G2P, for example).

Chapter 4

Applying supplemental data to G2P and machine transliteration

In this chapter, I present my method for combining related transliteration and transcription data into base G2P and machine transliteration systems. Because the two base tasks are different, I first set the problem up in an abstract way before discussing the specific cases of G2P and machine transliteration. Once the problems have been described, I build up the case for applying SVM re-ranking and then present the experiments and discuss their results.

4.1 Abstract approach description

For the type of problems I seek to address, I assume, as is the case for machine translation, part-of-speech tagging, grapheme-to-phoneme conversion, machine transliteration, and others, that the ultimate goal is to learn a function that produces an output sequence t given an input sequence s. I further assume that there is available a system $T(s)$ that attempts this task and produces an n-best list of outputs $\widehat{t_1}, \widehat{t_2}, \ldots, \widehat{t_n}$ for the input s. T is imperfect; i.e., the correct output t may appear in a position in the list other than the first (or it may not appear at all, but in this case nothing can be done *post hoc*). It is reasonable to expect that such a system also provides a list of scores $k_1, k_2, \ldots, k_n$ corresponding to the outputs.

The ultimate objective is to achieve higher performance for the task in question. To this end, I turn to supplemental data sources $\mathcal{A}(s), \mathcal{B}(s), \ldots$ that, given s, provide corresponding information $\mathfrak{a}, \mathfrak{b}, \ldots$ respectively. Practically, $\mathfrak{a}, \mathfrak{b}, \ldots$ should be related to t in some way. For example, $\mathcal{A}, \mathcal{B}, \ldots$ could represent other systems that perform a similar or equivalent task to that of T. More concretely, if $T(s)$ is a machine transliteration system, one might use a completely different transliteration system $\mathcal{A}(s)$ to generate an alternative hypothesis for the same input s, as done by Finch and Sumita (2010). Given the supplemental data sources, the extra information $\mathfrak{a}, \mathfrak{b}, \ldots$ can be used to judge the quality of each output $\widehat{t_1}, \ldots, \widehat{t_n}$ and re-rank the output list appropriately. The specifics of how the re-ranking is done, or how the alternative information is applied, will vary per task; my proposal is that the same (or similar) type of information that is used in the main task can be re-applied across the supplemental data.

While it is generally true that incorporating information directly into an existing

system is better than injecting it *post hoc*, this is not always feasible. With generative approaches, we would have to find some way of modelling the relationship between the system inputs, outputs, and the supplemental data. Even discriminative approaches can present a challenge to injecting information that is supplemental to the task at hand; DirecTL+, for example, is discriminative, but its decoder needs to be able to generate features on-the-fly for partial grapheme-phoneme sequence pairs. Treating the system as an n-best list–generating black box bypasses this limitation, and further provides the advantage of an approach that can work on top of any system that can provide n-best output lists.

4.2 Re-ranking G2P and machine transliteration

My focus is on improving G2P and machine transliteration systems. Consider first the G2P case, in which we are trying to predict the transcription of a given input word. The key point that I would like to make here is that a transliteration of a given input can be thought of as providing some *alternative* pronunciation information about a given input; in transliteration, the aim is to write some named entity in a different script while preserving the pronunciation as much as possible. The pronunciation information is there implicitly, albeit in a different form than what we expect (i.e., phoneme sequences). In terms of actual use, this is a realistic scenario; transliterations can often be mined cheaply from the Web (see, for example, the results of the 2010 Named Entities Workshop (NEWS) Transliteration Mining Shared Task (Kumaran et al., 2010)) whereas transcriptions are scarce.

The presence of this alternative information should make clear the potential of casting G2P together with transliteration in the above framework: T would be some G2P system that provides n-best output lists and $\mathfrak{a}, \mathfrak{b}, \ldots$ would be transliterations of the input in various languages.

Similarly, when we are transliterating from one language to another—say, from English to Hindi—we can look at transliterations in other languages to provide some supplemental information. In this case, T would be a transliteration system and again $\mathfrak{a}, \mathfrak{b}, \ldots$ would be transliterations in other languages.

Using transliterations in this way suggests that perhaps G2P transcription data can be used as supplemental data for transliteration. This makes intuitive sense too: if we know the pronunciation (via a transcription) of a word, we should be able to transliterate it better. Lastly, applying G2P to transliterations suggests that G2P can be used to help G2P as well. At first glance, this seems to make little sense: if we have the transcription of a word, how can we help that with another transcription? The trick is to observe that there exist multiple pronunciation corpora, and because they often use different conventions or rules, merging them can have deleterious effects. Our task would then be to improve the performance of a G2P system trained on one corpus using data from another corpus.

To summarize, we have two base tasks—G2P and machine transliteration—that we would like to improve using two classes of supplemental data: transliterations and transcriptions. Table 4.1 shows the various task/supplemental data combinations with sample inputs, outputs, and supplemental data.

Task		Supplemental data					
		Transcriptions			Transliterations		
	G2P	Input	Outputs	Sup. data	Input	Outputs	Sup. data
		Sudan	sud@n sud{n ⋮ **sud#n**	sudAn	McGee	m@kJi **m@gi** ⋮ m@CJi	मगी マギー ⋮ Макги
	MTL	Input	Outputs	Sup. data	Input	Outputs	Sup. data
		Sudan	ズーダン **スーダン** ズダン ⋮ スユーダン	sud#n sudAn	DOS	डोस दोस **डॉस** ⋮ दॉनस	ドス ДОС ดอส ⋮

Table 4.1: G2P and machine transliteration (MTL) system inputs, outputs, and supplemental data. Correct outputs are shown **boxed.** The transcriptions are shown in the corpus's format rather than IPA to illustrate the differences between corpora.

4.2.1 Leveraging similarity directly

The question remains of how exactly to leverage the supplemental data. One possibility is to use some similarity function S that can provide the similarity between a candidate output $\widehat{t_i}$ and a supplemental datum a. If there is only one source of supplemental data—e.g., one transliteration language—a simple (and intuitive) approach would be to select from the n-best list the output that is most similar to the supplemental datum. Consider the case of applying transliterations to G2P in Table 4.1 (the top right square): the second output is closest perceptually to the Hindi transliteration of the input (as compared to the other outputs). Of course, this approach is overly simplistic in that it ignores the scores and relative ordering of the base system's output list.

A better idea, then, would be to use a linear combination of the base system's score k_i and the similarity $S\left(\widehat{t_i}, a\right)$ to provide a new score with which the base system's output list can be re-ranked. In other words, define:

$$r_i = \lambda S\left(\widehat{t_i}, a\right) + (1 - \lambda)\, k_i$$

where the linear combination weight $0 \leq \lambda \leq 1$ chooses how strongly the supplemental datum is used and is optimized on a held-out set. The output list can then be re-ranked using r_i rather than k_i. This approach is similar to the method used by Finch and Sumita (2010) to combine the scores of two different machine transliteration systems.

Both of these approaches depend on the existence of a similarity function S. Unfortunately, the system outputs in Table 4.1 are never in the same script as the supplemental data—the closest they get is when using other pronunciation corpora for G2P (the top left square). Sometimes it is possible to convert one to the other; in such cases, simple methods such as minimum edit (Levenshtein) distance (Jurafsky and Martin, 2009, pp. 73–75) or lowest common subsequence ratio (LCSR) can

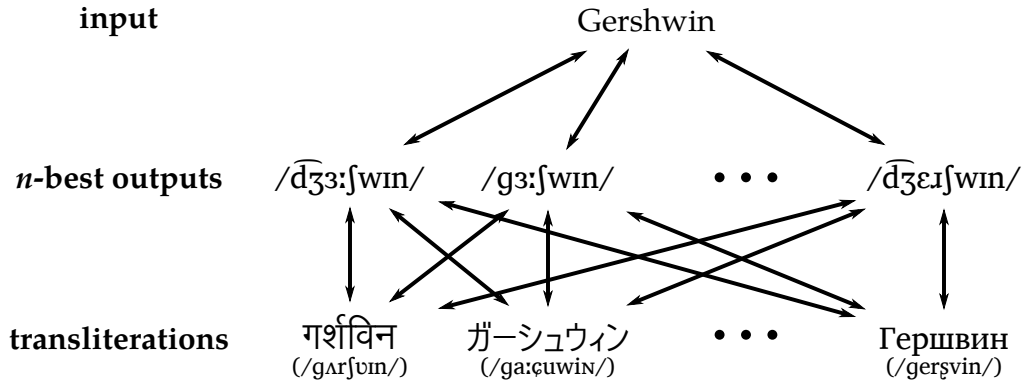

Figure 4.1: An example showing the pairs used for feature construction with transliterations being used to augment a G2P system. Each arrow links a pair used to generate features. One feature vector is constructed for each system output.

be applied. There are also more sophisticated methods such as ALINE (Kondrak, 2000), which specifically computes the distance between pairs of phonemes. Doing the actual conversion is usually difficult, however, requiring much training data or specific linguistic expertise.

A more general approach would compute the similarity between the output and the supplemental data directly. For example, the edit distance function can be learned from a sufficient set of training data using the EM algorithm (Ristad and Yianilos, 1998). In this thesis, I use M2M-Aligner (Jiampojamarn et al., 2007); as noted in Chapter 3.1.4, M2M-Aligner is an unsupervised aligner that uses an extension of the learned edit distance algorithm, allowing many-to-many alignments. For any alignment between two input strings that it computes, it provides an alignment probability that is representative of the similarity between the two strings. Since M2M-Aligner is unsupervised, it can be applied to compute alignments between any two scripts provided sufficient training data, allowing it to be used for all four cases in Table 4.1.

4.2.2 Applying SVM re-ranking

The methods described above in Chapter 4.2.1 are difficult to generalize to situations where there are multiple sources of supplemental data available, such as multiple transliteration languages. A re-ranker based on a linear combination is still possible, but having multiple data sources increases the number of linear weights, significantly complicating hand-tuning. Thankfully, SVM re-ranking offers a method for finding optimal weights for an arbitrary number of parameters. This additionally allows the incorporation of not just multiple data sources but multiple types of features: where before there were only similarities, there may now be features based on n-grams as well.

Applying SVM re-ranking means that a feature vector is constructed for every output. My process for this is (at a broad level) as follows:

1. For each input s, gather all supplemental data $\mathfrak{a}, \mathfrak{b}, \ldots$ that is available.

2. For an output $\widehat{t_i}$ generated by the base system $T(s)$, generate a pair with the

input $\left(\widehat{t_i}, s\right)$ and one pair with each supplemental datum $\left(\widehat{t_i}, \mathfrak{a}\right), \left(\widehat{t_i}, \mathfrak{b}\right), \ldots$

3. Generate features for each pair and add them to the feature vector f_i for the output $\widehat{t_i}$.

Figure 4.1 shows this visually for the case of transliterations being applied to G2P.

Broadly, my use of SVM re-ranking incorporates two feature classes: those based on scores and those based on n-grams. The score-based features are obtained either from the base system (for input-output pairs) or from M2M-Aligner (for output-supplemental pairs); in the latter case, I use the logarithm of M2M-Aligner's alignment probability. In addition to the scores, I also compute the differences between scores in the n-best list. For example, if there are two outputs, one with a system score of 10 and the other with a system score of 5, the feature vector for the first output includes 10 (the system score), 0 (the difference with the first output's score), and 5 (the difference with the second output's score) as features, while the second feature vector includes 5 (the system score), -5 (the difference with the first output's score), and 0 (the difference with the second output's score).

The set of binary (indicator) n-gram features includes those used in DirecTL+ (Jiampojamarn et al., 2010a). They can be divided into four types:

1. The **context features** bind output symbols with n-grams of varying sizes in a window of size c centred around a corresponding position on the input side.

2. The **transition features** are bigrams on the output side.

3. The **linear chain features** combine the context features with the bigram transition features.

4. The **joint n-gram features** are n-grams containing both input and output symbols.

I apply these features in a new way: instead of being constructed strictly from a given input-output set, I expand their use across all available supplemental data simultaneously. The n-gram features are applied across all output-supplemental pairs as well as the input-output pairs corresponding to the n-best lists.

Unlike a traditional G2P system, the re-ranking process has access to the entire outputs produced by the base system. By swapping the input and the output side, we can add *reverse* context and linear-chain features. For example, this allows the inclusion of a variety of n-grams in the supplemental strings with a single corresponding output character.

Table 4.2 shows the various n-gram feature types. Note that the construction of n-gram features presupposes a fixed alignment between the input and output sequences. If the base system does not provide any structure (alignments) in the output, I align them with M2M-Aligner. The transliteration-transcription pairs are aligned uniformly by M2M-Aligner, which at the same time produces the corresponding similarity scores. Unlike in machine translation, M2M-Aligner produces monotonic alignments—there is no reordering. This is the standard approach for G2P (Jiampojamarn and Kondrak, 2010) and machine transliteration, and makes sense: phonemes do not reorder themselves. If M2M-Aligner is unable to produce

context	$x_{i-c}y_i$	reverse context	x_iy_{i-c}
	$\vdots$		$\vdots$
	$x_{i+c}y_i$		x_iy_{i+c}
	$x_{i-c}x_{i-c+1}y_i$		$x_iy_{i-c}y_{i-c+1}$
	$\vdots$		$\vdots$
	$x_{i+c-1}x_{i+c}y_i$		$x_iy_{i+c-1}y_{i+c}$
	$\vdots$		$\vdots$
	$x_{i-c}\cdots x_{i+c}y_i$		$x_iy_{i-c}\cdots y_{i+c}$
transition	$y_{i-1}y_i$		
linear chain	$x_{i-c}y_{i-1}y_i$	reverse linear	$x_{i-1}x_iy_{i-c}$
	$\vdots$	chain	$\vdots$
	$x_{i+c}y_{i-1}y_i$		$x_{i-1}x_iy_{i+c}$
	$x_{i-c}x_{i-c+1}y_{i-1}y_i$		$x_{i-1}x_iy_{i-c}y_{i-c+1}$
	$\vdots$		$\vdots$
	$x_{i+c-1}x_{i+c}y_{i-1}y_i$		$x_{i-1}x_iy_{i+c-1}y_{i+c}$
	$\vdots$		$\vdots$
	$x_{i-c}\cdots x_{i+c}y_{i-1}y_i$		$y_{i-1}y_ix_{i-c}\cdots x_{i+c}$
joint n-gram	$x_{i+1-n}y_{i+1-n}x_iy_i$		
	$\vdots$		
	$x_{i-1}y_{i-1}x_iy_i$		
	$x_{i+1-n}y_{i+1-n}x_{i+2-n}y_{i+2-n}x_iy_i$		
	$\vdots$		
	$x_{i-2}y_{i-2}x_{i-1}y_{i-1}x_iy_i$		
	$\vdots$		
	$x_{i+1-n}y_{i+1-n}\cdots x_{i-1}y_{i-1}x_iy_i$		

Table 4.2: The different types of n-gram features used in the re-ranker. This table is an extension of one by Jiampojamarn et al. (2010a).

an alignment, I indicate this with a binary feature that is included with the n-gram features.

This re-ranking approach is similar to that of Song et al. (2010), who used an averaged perceptron re-ranker for English-to-Chinese machine transliteration. The main differences in my application of re-ranking is that I incorporate features from supplemental data, whereas their approach uses features constructed only from the inputs and system outputs and derivations thereof. I additionally use SVM re-ranking rather than perceptrons.

4.3 Experimental setup

4.3.1 Data

I use two corpora to provide transcriptions: Combilex (Richmond et al., 2009) and CELEX (Baayen et al., 1996). Combilex has just under 500 multi-word entries, which I discard, as it is not always simple to match the individual words with their transcriptions. I also discard all diacritics and duplicate entries, yielding 25 575 unique entries. CELEX requires no such preprocessing and provides 66 372 unique entries.

All experiments use 10% of the data set for testing, 10% for development, and the remaining 80% for training. The development set is used for initial tests and is merged with the training set for final testing (the base systems may then use a subset of this for their own development purposes during training).

The transliteration data come from the shared tasks of the 2009 and 2010 Named Entities Workshops (Li et al., 2009a, 2010). I combine the training and development sets from all of the 2010 English-source corpora as well as the English-to-Russian data from 2009, which makes nine languages in total. Note that these transliteration corpora are noisy: Jiampojamarn et al. (2009) found a significant increase in English-to-Hindi transliteration performance with a simple cleaning of the data. The number of entries in the transliteration corpora can be seen below in Table 4.3.

4.3.2 Base systems

Importantly, obtaining positive results with the SVM re-ranker on one base system only would not conclusively show that the overall method works, only that it works for the one system. Furthermore, the n-gram features I use are very similar to the DirecTL+ features, so applying them to any non-DirecTL+ system should provide an increase in performance simply by virtue of the different type of information being included (conceptually similar to how DirecTL+ incorporates the joint n-gram information used by Sequitur). Therefore, it is important that there be multiple base systems and that they include DirecTL+. Positive results with the latter would demonstrate that the supplemental data do provide useful information, and positive results over all or most base systems would demonstrate that incorporating that information via SVM re-ranking works in general.

I use the three G2P systems described in Chapter 3.1: Festival, Sequitur, and DirecTL+. I use these particular systems because they are freely available, Festival is very popular, and Sequitur and DirecTL+ represent state-of-the-art G2P approaches. All systems are capable of providing n-best output lists along with scores for each output, although for Festival they must be constructed manually from the list of output probabilities for each input character. As noted above, I use M2M-Aligner to provide alignments for n-gram feature generation; the training data are obtained by intersecting the supplemental data with the primary data for each language. I set a lower limit of -100 on the M2M-Aligner log-probabilities, and use the default of 2-2 alignments; deletions are enabled for the supplemental data side of the alignment.

Because of the similarity between G2P and machine transliteration, G2P systems can often be applied directly to machine transliteration; Finch and Sumita (2010) in part apply Sequitur and Jiampojamarn et al. (2009, 2010b) apply DirecTL+. I sim-

ilarly use Sequitur and DirecTL+ but exclude Festival for the machine transliteration experiments because it is unable to work with Unicode characters.

I run DirecTL+ with all of the features described by Jiampojamarn et al. (2010a): context features, transition features, linear chain features, and joint n-gram features. System parameters, such as maximum number of iterations, are determined during development. For Sequitur, I keep default options except for the enabling of the 10 best outputs. DirecTL+ and Sequitur are both modified to exclude any null outputs (i.e., those in which every input letter is deleted), and Sequitur is additionally modified to provide log-probabilities instead of regular probabilities. Festival requires a list of "allowable" alignments that generated manually. I use 10-best output lists for all systems.

Note that the three base systems differ slightly in terms of what structure information they provide in their outputs: Festival operates letter-by-letter, so I use the single-letter inputs with the corresponding outputs as the aligned units; DirecTL+ provides many-to-many alignments in its output; but Sequitur provides no information regarding the output structure. I use M2M-Aligner to provide alignments for Sequitur's outputs to ensure that inter-system comparisons remain fair.

The SVM re-ranker ultimately operates under the assumption of availability of supplemental data. For this reason, all reported scores are on a subset of the test set where the supplemental data are available. This means that the scores do not necessarily reflect the individual systems' performance on the full test set. See Appendix B for a full comparison of these G2P systems.

When comparing system performance, I use the McNemar test for statistical significance. In this thesis, all error reductions are statistically significant at the $p < 0.01$ level unless otherwise stated. I do not distinguish between the $p < 0.01$ and $p < 0.001$ levels here. See Appendix A for further discussion of the McNemar test and statistical significance.

In addition to comparing base system performance with the SVM re-ranked results, I also include the results achievable by an oracle re-ranker, i.e., a re-ranker that automatically selects the correct output from the n-best list. An oracle re-ranker is not a system that is implementable in a real-world scenario, as it requires prior knowledge of the correct output, but it allows us to see the upper bound for re-ranking and how close the use of supplemental data (via SVM re-ranking) can bring us to it.

4.4 Improving G2P with transliterations[1]

As described in Chapter 1.2, applying transliterations to G2P represents a realistic scenario: a G2P system may mis-predict an ambiguous character and transliterations can help disambiguate it. A transliteration cannot be followed blindly, however, lest the foreign phonology affect our pronunciation; following only one language can therefore be difficult. Furthermore, the transliterations are sometimes noisy or incorrect; for example, "Aretha" is transliterated as एरीथा, with the initial "A" as /e/ instead of /ə/ in the English-Hindi transliteration corpus. This is not a matter of adjusting for Hindi phonology: if I were to transliterate the name, I would

[1] A version of this chapter has been accepted for publication (Bhargava and Kondrak, 2011).

Language	Corpus size	Overlap
Bengali	12785	2484
Chinese	37753	5100
Hindi	12380	2852
Japanese	26206	8377
Kannada	10543	2440
Korean	6761	3561
Russian	6447	604
Tamil	10646	2499
Thai	27023	7068

Table 4.3: The number of unique single-word entries in the transliteration corpora for each Hindi language and the amount of common data (overlap) with the Combilex data.

not do it that way[2]. Similarly, the Russian transliteration of "McGee" shown in Table 4.1 provides a pronunciation closer to /məkgi/ than to the correct /məgi/ (even though both might be considered correct by humans, the pronunciation corpus does not indicate as such). Using multiple languages can help smooth out such noise in the data; the feature-based nature of SVM re-ranking allows us to do this easily.

To train the re-ranker, the transliteration data are intersected with the Combilex G2P data. I focus on Combilex because it provides information regarding whether an entry is a name or a core vocabulary word, allowing some comparisons between the two to be made. In total, there are 11 157 G2P entries for which there is at least one transliteration available. Table 4.3 shows the overlap between Combilex and the transliteration data for each language. As noted above, the base systems are trained on the full training set while the re-ranker is trained (as is necessary) on the subset of the training set for which at least one transliteration is available. In addition to this, the systems are evaluated in isolation on those names in the test set that appear in the names section of Combilex and similarly for the core vocabulary section; this allows us to see how well the re-ranking works in the two cases.

Table 4.4 shows the results on the test set of 1 243 words, with 724 of these words appearing in the names section and 527 appearing in the core vocabulary section (some words appear in both, which is why the numbers do not add up). The systems are evaluated on the full (intersected) Combilex test set; additionally, they are evaluated separately on the entries appearing in the names and core vocabulary sections. Festival shows the most improvement, but this is unsurprising given that it had the lowest base system performance. Sequitur also gives high error reductions while DirecTL+ sees more modest increases in performance.

Overall, we can see that the n-gram features are much more important than the score features, providing much higher error reductions. Furthermore, the transliteration-informed re-ranking does better for names than it does for core vocabulary. In practice, this means that transliterations are most useful for names; this makes sense considering that named entities are their primary *raison d'être*.

[2]Perhaps the pronunciation in India would be different from how I pronounce it as a native English speaker, but a transliteration is supposed to reflect the original pronunciation as closely as possible.

		Festival		Sequitur		DirecTL+	
Test set		Acc.	ERR	Acc.	ERR	Acc.	ERR
Full	Base	59.6		72.7		76.5	
	SVM-ngram	71.0	28.3	77.5	17.7	78.4	7.9
	SVM-score	63.7	10.2	73.9	*4.4*	77.2	*3.1*
	SVM-full	71.7	29.9	78.2	20.3	78.5	8.6
	Oracle-RR	83.5	59.2	93.7	77.1	93.0	70.2
Names	Base	56.6		72.5		75.7	
	SVM-full	70.9	32.8	78.6	22.1	78.7	12.5
	Oracle-RR	90.9	56.1	93.2	75.4	92.0	67.1
Core	Base	63.6		72.5		77.8	
	SVM-full	72.5	29.5	77.4	17.9	78.1	*1.7*
	Oracle-RR	86.9	64.1	94.5	80.0	94.5	75.2

Table 4.4: Word accuracies and error rate reduction (ERR) in percentages for Combilex G2P augmented by corresponding transliterations. Base is the base system while SVM-full represents the same system with its output re-ranked using transliterations. SVM-ngram and SVM-score are similar to SVM-full but use the n-gram and score features only, respectively. Oracle-RR shows the results of an oracle re-ranker. Statistically insignificant error reductions are shown *italicized*.

4.5 Improving G2P with an alternative pronunciation corpus

The next task is to use multiple G2P corpora (two, in my case) in tandem in an effective manner. A first instinct might be to merge distinct corpora, but this has two major problems: (*a*) the corpora, having been created by completely separate groups, use different conventions and are perhaps written for different accents (although here Combilex and CELEX both use the Received Pronunciation); and (*b*) the corpora likely use different phonemic representations. For the latter, it may seem sensible to convert the format of one corpus to that of another since the transcriptions are all supposed to be transparent representations of phonemes. Unfortunately, different corpora make varying levels of distinction between sounds; for example, Combilex differentiates between allophones of /l/ where CELEX does not. This makes the validity of simple conversion less clear even if it is still doable.

If we treat one corpus as our primary G2P corpus and the other as an alternative, we can apply SVM re-ranking to incorporate the alternative data. In order to demonstrate the utility of SVM re-ranking, it is important to show that it can provide better results than simple conversion and merging.

Practically, this means that I use CELEX as the primary G2P corpus since converting from Combilex to CELEX is possible but not vice versa; CELEX includes some phonemes where it is not clear how to map them to Combilex. (On the other hand, most would agree that mapping both allophones of /l/ in Combilex—one being the "regular l" /l/ and the other the "dark l" /ɫ/—to the one /l/ phoneme

in CELEX is acceptable.) As well, in order to make a proper comparison between corpus merging and re-ranking, the two approaches must be evaluated on the same corpus. The limiting factor is the latter, which must be evaluated on a subset of the two corpora that have common entries; re-ranking can only be done when there is an entry available in another corpus, just as with transliterations above in Chapter 4.4.

In addition to the above merging method, applying one corpus to another is similar to the notion of domain adaptation. Unfortunately, most methods for domain adaptation would work only for a single system, requiring modification of how it handles the data. Such methods are not applicable in a black-box manner. However, the work of Loots and Niesler (2009) is applicable (see Chapter 3.3); instead of using an external-corpus entry to re-rank a system's output list, a G2P system could be applied to convert the external-corpus transcription to a more appropriate format for the current corpus—a phoneme-to-phoneme (P2P) conversion, in other words. This provides an additional approach against which we can compare SVM re-ranking. To provide a use case, imagine being given a word and asked for its transcription in CELEX format. It is unavailable in the CELEX corpus, so instead we look in Combilex and find it there; we then convert the Combilex transcription to CELEX's format using a G2P system as noted above. Festival is not well-suited to this task, as there will often be extraneous phonemes in one transcription versus another; Festival can only handle such phonemes (that should be deleted) on one side, so I exclude Festival for this particular test.

The CELEX corpus, which has 66 372 unique entries, and the Combilex corpus, which has 25 575 unique entries, have 15 028 entries in common. Importantly, this confirms the viability of our usage scenario: CELEX has 51 344 entries that do not appear in Combilex, and Combilex has 10 547 entries that do not appear in CELEX. Each is therefore a potential resource to exploit in a real-world scenario.

The experimental procedure, then, is as follows:

1. Separate 10% of CELEX for testing.

2. Create a separate combined corpus wherein Combilex is converted to CELEX format and merged with CELEX; I do this separately for the training and test sets.

3. Train a G2P model on the CELEX-only training set, and another on the combined training set.

4. Intersect the CELEX training set with the Combilex corpus. Repeat for the CELEX test set.

5. Train the re-ranker on the intersected training set using the G2P system output from CELEX and alternative data from Combilex.

6. Train a P2P system (using any general G2P system) to convert the Combilex phonemes to the CELEX phonemes using the intersected training set.

7. Evaluate the base CELEX-only model, the combined model, the P2P converter, and the re-ranker on the intersected test set. Because the combined model was trained on two separate corpora, it is evaluated such that if *either* of the two

	Festival		Sequitur		DirecTL+	
	Acc.	ERR	Acc.	ERR	Acc.	ERR
Base	65.0		87.3		88.1	
CELEX+Combilex	62.1	—	74.2	—	71.6	—
P2P			85.7	—	87.0	—
SVM-ngram	84.2	54.8	93.0	45.0	92.7	40.0
SVM-score	80.8	45.2	92.0	37.2	91.8	30.9
SVM-full	84.1	54.6	92.7	42.9	92.0	32.6
Oracle-RR	88.7	67.8	97.6	81.2	96.7	72.5

Table 4.5: Word accuracies and error rate reductions (ERR) in percentages for CELEX G2P augmented by Combilex transcriptions. Base, SVM-ngram, SVM-score, SVM-full, and Oracle-RR are as in Table 4.4. P2P is the base system trained to convert Combilex transcriptions to CELEX transcriptions. CELEX+Combilex is the base system trained on a combined CELEX and Combilex training set.

corpora say an output is correct, it is considered as such. The other two models are evaluated using CELEX only.

Of course, system parameters are set using the separate development set.

Table 4.5 shows the results on the intersected test set of 1 498 words. We can see that merging the corpora provides a clear detriment in performance for data where an alternative transcription is available from another corpus. Even if we look at the full CELEX test set (as opposed to the intersected subset used in Table 4.5), DirecTL+ trained only on CELEX achieves 93.0% word accuracy on the CELEX test set where DirecTL+ trained on CELEX merged with Combilex achieves 87.3%. Evidently, the disparate conventions of the two corpora "confuse" the G2P models; the SVM re-ranker avoids this and learns the correspondences between the two corpora, re-ranking the output lists appropriately. We might further consider the P2P approach and suggest the use of the Combilex-to-CELEX converter in place of a G2P system to perform the phoneme-to-phoneme conversion (this would be entirely independent of any base system); this achieves 64.8% word accuracy, much lower than using a G2P system. Clearly a simple mapping script isn't enough to capture the differences between the corpora.

We see impressive error reductions for all three systems, though (as in Chapter 4.4) Festival benefits more than Sequitur which in turn benefits more than DirecTL+. Sequitur's re-ranked word accuracy is in fact slightly higher than that of DirecTL+, though not statistically significantly so. We also again see that the n-gram features are more important than the score features; in fact, even though the score features provide significant improvements when used on their own, they significantly ($p < 0.05$) *degrade* performance when used in conjunction with the n-gram features!

The differences between the two corpora account for the reason why the P2P approach does not work well, especially as compared to SVM re-ranking. For one, learning models for any string transduction task is difficult and complex, requiring much training data. On the other hand, when we have existing G2P outputs to

Language	Corpus size	Overlap
Bengali	12785	6349
Chinese	37753	2295
Japanese	26206	4132
Kannada	10543	6878
Korean	6761	1741
Russian	6447	307
Tamil	10646	7069
Thai	27023	3460

Table 4.6: The number of unique single-word entries in the transliteration corpora for each non-Hindi language and the amount of common data (overlap) with the Hindi data.

re-rank, the string generation aspect has been done and we need only learn how to re-rank the list. Models for these two tasks (P2P and re-ranking) are trained on the same amount of data, so it is expected that the much more difficult task would not perform as well. Furthermore, if the alternative corpus provides a radically different pronunciation to any of the ones we are trying to predict, it becomes nearly impossible for the P2P approach to convert it properly; with the re-ranking features that I use, the alignments would simply fail, falling back to the original G2P system's output.

4.6 Improving machine transliteration with transliterations from other languages

Applying other-language transliterations to a machine transliteration task in a given language is similar mechanically to applying transliterations to G2P as in Chapter 4.4. The situation is that we are tasked with transliterating some text and we have seen transliterations of the same text into other languages. Consider, for example, the Wikipedia page for John Petrucci. Versions of the article currently exist in a number of languages besides the primary[3] English, including Japanese, but there is none in Hindi. If we wanted to generate a stub article in Hindi, we would need to transliterate his name, and the numerous existing transliterations could be helpful.

The focus here is on one transliteration task; because of my linguistic familiarities, I choose English-to-Hindi transliteration as the primary task, with the other transliteration languages playing roles as supplemental data. The English-Hindi transliteration corpus contains a total of 12 380 unique single-word entries. Table 4.6 shows the amount of overlap between the English-Hindi corpus and the other English-source transliteration corpora.

The overall procedure is as in Chapter 4.4. 10% of the data is set aside for testing, and the rest is used to train the final English-Hindi transliteration model; the SVM re-ranking model is trained on an intersection of the English-Hindi translit-

[3] I say primary because John Petrucci is an American guitarist whose work is more known in North America than in most other parts of the world.

	Sequitur		DirecTL+	
	Acc.	ERR	Acc.	ERR
Base	44.5		45.2	
SVM-ngram	52.9	15.1	50.0	8.7
SVM-score	48.9	7.9	47.5	*4.2*
SVM-full	52.8	14.9	49.8	8.4
Oracle-RR	82.0	67.6	73.4	51.4

Table 4.7: Word accuracies and error rate reductions (ERR) in percentages for English-to-Hindi machine transliteration augmented by corresponding transliterations from other languages. Base, SVM-ngram, SVM-score, SVM-full, and Oracle-RR are as in Table 4.4. Changes that are only significant at the $p < 0.05$ level (rather than the usual $p < 0.01$) are *italicized*.

eration data and the transliteration data for the other languages (i.e., when other transliterations are available). In total, there are 10 077 words for which at least one transliteration from a non-Hindi language is available. The final test set is the intersection of the English-Hindi test set and the supplemental transliteration data.

Table 4.7 presents the results on the intersected test set of 1 002 words. As with G2P, we can see that the supplemental transliterations provide useful information to inform the main transliteration process via the SVM re-ranker. Notably, Sequitur's re-ranked performance significantly ($p < 0.05$) exceeds DirecTL+'s re-ranked performance, although as noted in Chapter 4.3.2 this is not necessarily indicative of the systems' performance on the *full* test set. This is attributable to the higher oracle re-ranker accuracy for Sequitur, which shows a much higher potential error rate reduction for re-ranking than is seen for DirecTL+ (on these particular data).

As in previous experiments, the n-gram features are again most important. Also as seen in Chapter 4.5, the score features seem to slightly degrade performance when used together with the n-gram features, though in this case this is not statistically significant.

4.7 Improving machine transliteration with transcriptions

Finally, we turn to applying transcription data to machine transliteration. In this case, we are asked to transliterate some text that happens to appear in a pronunciation dictionary. While transliteration is usually applied to names, sometimes parts of a name may also be dictionary words (see Chapter 3.2). In this case, referring to the pronunciation dictionary may be helpful.

Some preliminary experiments with Hindi showed some promising but statistically insignificant results due to the smaller size of the training set; I therefore use English-to-Japanese transliteration as Japanese had the largest overlap with the Combilex G2P corpus (see Table 4.3). The English-Japanese transliteration corpus has 5 788 entries for which there are entries in at least one of CELEX or Combilex.

As above, the SVM re-ranker is trained on an intersection of the transcription data and the English-Japanese transliteration data. Combilex and CELEX are re-

	Sequitur		DirecTL+	
	Acc.	ERR	Acc.	ERR
Base	60.9		64.0	
SVM-ngram	59.7	—	66.8	*8.0*
SVM-score	65.1	10.8	66.0	*5.6*
SVM-full	71.1	26.0	69.2	14.6
Oracle-RR	90.7	76.2	89.6	63.4

Table 4.8: Word accuracies and error rate reductions (ERR) in percentages for English-to-Japanese machine transliteration augmented by transcriptions from Combilex and CELEX. Base, SVM-ngram, SVM-score, SVM-full, and Oracle-RR are as in Table 4.4. Changes that are only significant at the $p < 0.05$ level (rather than the usual $p < 0.01$) are *italicized*.

garded as separate corpora, just as was done with the various transliteration languages in Chapter 4.4. The final test set is the intersection of the English-Japanese test set and the transcription data.

Table 4.8 presents the results on the intersected test set of 591 entries. Strangely, Sequitur sees a statistically insignificant *decrease* in performance with the n-gram features when used alone, even though they are helpful over the score features. I attribute this to an "unlucky" test set, as I did observe an increase in performance during development; by "unlucky" I mean that the particular test set is anomalously difficult to process using just the n-gram features and Sequitur as the base system. DirecTL+ behaves more as we have seen before: the n-gram features are more important than the score features. Both Sequitur's oracle and SVM re-ranking performance are higher than that of DirecTL+ as in Chapter 4.6, but here these differences are not statistically significant.

Finally, for comparison, I apply the P2P approach that was used in Chapter 4.5; in this case, that entails using the intersection of the transcription data with the transliteration data in order to train a phoneme-to-Japanese (P2J) converter. The overall idea behind this is that, when asked to transliterate a given text to Japanese, instead of doing so directly we first consult our pronunciation lexicon, retrieve the transcription for the text, and then convert the transcription to Japanese. Intuitively, one might expect the high quality of the transcription to be helpful. This approach does have the deficiency that it can only be applied using a single pronunciation corpus (at a time). I use Combilex, as it had by far the larger overlap of the two lexica, and compare this to the SVM re-ranker separately from the main test set since using only Combilex results in a slightly smaller intersected test set (than using both Combilex and CELEX).

Table 4.9 shows the results on the intersected test set of 532 entries. As in Chapter 4.5, the P2J approach decreases performance for both systems. This is again partially due to the smaller size of the intersected data: Sequitur and DirecTL+ need to learn full transduction models from the data while the re-ranker need only learn to score existing outputs. Under such circumstances, SVM re-ranking provides a clear performance advantage.

	Sequitur	DirecTL+
Base	62.4	65.4
P2J	47.9	53.6
SVM-full	72.2	70.7
Oracle-RR	91.9	88.2

Table 4.9: Word accuracies (%) for English-to-Japanese machine transliteration augmented by transcriptions from Combilex only. Base, SVM-ngram, SVM-score, SVM-full, and Oracle-RR are as in Table 4.4. P2J is the base system trained to convert Combilex transcriptions to Japanese.

4.8 Discussion

All experiments show significantly better overall performance over the base system using SVM re-ranking; my re-ranking approach also outperforms alternative approaches of incorporating the supplemental data, which consistently decrease system performance. This is true for all base systems, demonstrating that the SVM re-ranking method of incorporating supplemental data applies in general to the studied tasks; it is not tied to a certain base system. However, the magnitude of the performance increase *is* dependent on the base system. Festival has the lowest base system performance, providing a number of "easy" re-ranking targets ("low-hanging fruit"). Sequitur and DirecTL+, with their higher base system scores, see smaller increases than does Festival; the better the base system does, the harder it is to re-rank its output. As noted in Chapter 4.1, re-ranking treats the base system as a black box, but we can see that it is limited by how much room the base system leaves for improvement. Sequitur sometimes shows higher oracle re-ranker performance and in these cases the SVM re-ranker does commensurately better, significantly outperforming DirecTL+'s re-ranked performance in the G2P/transcription case (Chapter 4.5).

Moreover, the fact that both Festival and Sequitur show higher error reductions than DirecTL+ suggests that they were benefiting from the DirecTL+-style information present in the re-ranker; the n-gram features, after all, are styled after those used in DirecTL+. Since, despite this similarity, DirecTL+ still sees significant performance increases, we can surmise that the supplemental data do inherently provide some useful information. That the score features are usually able to provide significant performance increases when used on their own is further evidence of this, as there is no way for DirecTL+-style information to work its way into the re-ranking through the score features alone.

The Festival and Sequitur results should therefore be taken with a grain of salt. In particular, note the comparison of the re-ranking performance on core vocabulary versus names in Chapter 4.4. Using DirecTL+ as the base system allows us to consider the value of the supplemental data independently, and since the improvement for core vocabulary is insignificant, we can conclude that applying transliterations to G2P is not just *most* useful for names, but it is *only* useful for names (when incorporated via SVM re-ranking as presented in this thesis, that is). Of course, this is in the absence of more data; while insignificant, the small improvement in accuracy

observed for core vocabulary words suggests (but does not conclusively indicate) a positive effect for core vocabulary words as well.

The n-gram features are generally more helpful than the score features. While this may at first seem counterintuitive—the scores provide some measure of similarity, after all—consider that the M2M-Aligner scores themselves are built from the n-gram alignments. The score and the n-gram features thus *both* derive from these alignments. But the n-gram features are far more granular and allow weights to be put on specific features (n-gram alignments), tuning them specifically for reranking. The n-gram features also include a feature that indicates a failed alignment, a potentially important indicator of a bad candidate transcription. The SVM reranker can then learn that certain correspondences—such as those found between phonemes and foreign-language characters—are indicative (or not) of good (or bad) system outputs. We know, however, that the score features do provide additional information, since we do usually see some improvement when they are used on their own; this information comes from M2M-Aligner's training process, where it learns the relevant probabilities from the training data.

Chapter 5

Conclusion

In this thesis, I examined the issue of using supplemental data to improve the performance of a base system on a given task. In particular, I examined grapheme-to-phoneme conversion and machine transliteration, applying to them related supplemental transcriptions and transliterations. My approach appropriates SVM re-ranking to re-order output lists from existing G2P and machine transliteration systems.

Notably, this is (to the best of my knowledge) the first use of disparate tasks and data. While previous approaches have used intermediate languages for machine transliteration or incorporated multiple transliteration systems, my approach is general and can apply both transliterations to G2P and transcriptions to machine transliteration. The SVM re-ranker includes features based on alignment scores between candidate outputs and transliterations from a corpus as well as n-grams from said alignment. The n-gram features demonstrate that the *same features* used for the primary task can be helpful *post hoc* with supplemental data, suggesting similar possibilities for other NLP tasks.

The positive results achieved demonstrate not only that this approach works, but that the various supplemental data used do indeed provide useful information. With the DirecTL+ state-of-the-art base system, I achieved an error reduction of 8.6% for G2P using related transliterations and 32.6% using transcriptions from alternative corpora. Results were similar for machine transliteration: 8.4% error reduction for English-Hindi transliteration using corresponding transliterations from other languages and 14.6% using related transcriptions. These improvements all provide better performance than other approaches of incorporating the supplemental data, such as trying to convert directly from the supplemental data to the target data; such approaches all resulted in decreases in performance.

Finally, an analysis of the results showed that the n-gram features were generally more important than the score features. In the case of Festival and Sequitur, the n-gram features provided DirecTL+-style information, which of course was not present in the base systems. That DirecTL+ still sees significant re-ranking improvement and that the score features usually provide some improvement on their own is further evidence of the useful information present in the supplemental data. This also demonstrates that the information is usable in practice, although in some cases the information is only useful for certain classes: while examining the transliteration-based re-ranking of G2P, for example, I found that the transliterations were

mainly useful for names and not for core vocabulary.

5.1 Future work

While the work in this thesis uses supplemental *data* to re-rank outputs, we would ultimately like to be able to incorporate information from other *models*. For example, a transliteration model encodes all of the necessary information to generate a transliteration from a given text: this information may be more valuable than the simple transliteration on its own. If this information could be put to use (rather than approximated from the transliteration from a corpus), it would allow a system to benefit when there is no preexisting transliteration available.

Along similar lines, re-ranking approaches are necessarily *post hoc*, and cannot help when the output list does not include a correct output at all. This is demonstrated by the oracle re-ranking results presented here, which show that even with a perfect re-ranker there is still quite a gap to reaching perfect accuracy. Therefore, the next step in applying supplemental data is to incorporate them directly into the base system. Doing so in a system-independent manner may be difficult (or impossible!) but should yield higher performance: giving the system direct access to any extra information allows it to learn a better model from the beginning.

Finally, note that in addition to transliterations, the Web also provides many *ad hoc* transcriptions (e.g., trans-SKRIP-shuns). There is no common structure to these, but they are intended to help readers pronounce words based on approximately phonetic character groups that they presumably already know how to pronounce. As with any Web data, these would be very noisy, but I have shown that a noisy transliteration corpus can provide useful information; this may be the case for *ad hoc* transcriptions as well. An aspect of this includes mining these transcriptions from the Web automatically.

References

R. Harald Baayen, Richard Piepenbrock, and Leon Gulikers. 1996. The CELEX2 lexical database. LDC96L14.

Aditya Bhargava and Grzegorz Kondrak. 2010. Language identification of names with SVMs. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 693–696, Los Angeles, California, USA, June. Association for Computational Linguistics.

Aditya Bhargava and Grzegorz Kondrak. 2011. How do you pronounce your name? Improving G2P with transliterations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 399–408, Portland, Oregon, USA, June. Association for Computational Linguistics.

Maximilian Bisani and Hermann Ney. 2008. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):pages 434–451.

Alan W. Black, Kevin Lenzo, and Vincent Pagel. 1998. Issues in building general letter to sound rules. In *The Third ESCA/COCOSDA Workshop (ETRW) on Speech Synthesis*, Jenolan Caves House, Blue Mountains, New South Wales, Australia, November.

Chris Callison-Burch, Philipp Koehn, and Miles Osborne. 2006. Improved statistical machine translation using paraphrases. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 17–24, New York City, New York, USA, June. Association for Computational Linguistics.

Chih-Chung Chang and Chih-Jen Lin. 2001. *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Monojit Choudhury, Anupam Basu, and Sudeshna Sarkar. 2004. A diachronic approach for schwa deletion in Indo Aryan languages. In *Proceedings of the Seventh Meeting of the ACL Special Interest Group in Computational Phonology*, pages 20–26, Barcelona, Spain, July. Association for Computational Linguistics.

Trevor Cohn and Mirella Lapata. 2007. Machine translation by triangulation: Making effective use of multi-parallel corpora. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 728–735, Prague, Czech Republic, June. Association for Computational Linguistics.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning*, 20:pages 273–297.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:pages 1871–1874.

Andrew Finch and Eiichiro Sumita. 2010. Transliteration using a phrase-based statistical machine translation system to re-score the output of a joint multigram model. In *Proceedings of the 2010 Named Entities Workshop (NEWS 2010)*, pages 48–52, Uppsala, Sweden, July. Association for Computational Linguistics.

Sittichai Jiampojamarn. 2010. Grapheme-to-phoneme conversion and its application to transliteration. Ph.D. thesis, University of Alberta.

Sittichai Jiampojamarn, **Aditya Bhargava**, Qing Dou, Kenneth Dwyer, and Grzegorz Kondrak. 2009. DirecTL: a language independent approach to transliteration. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (NEWS 2009)*, pages 28–31, Suntec, Singapore, August. Association for Computational Linguistics.

Sittichai Jiampojamarn, Colin Cherry, and Grzegorz Kondrak. 2008. Joint processing and discriminative training for letter-to-phoneme conversion. In *Proceedings of ACL-08: HLT*, pages 905–913, Columbus, Ohio, USA, June. Association for Computational Linguistics.

Sittichai Jiampojamarn, Colin Cherry, and Grzegorz Kondrak. 2010a. Integrating joint n-gram features into a discriminative training framework. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 697–700, Los Angeles, California, USA, June. Association for Computational Linguistics.

Sittichai Jiampojamarn, Kenneth Dwyer, Shane Bergsma, **Aditya Bhargava**, Qing Dou, Mi-Young Kim, and Grzegorz Kondrak. 2010b. Transliteration generation and mining with limited training resources. In *Proceedings of the 2010 Named Entities Workshop (NEWS 2010)*, pages 39–47, Uppsala, Sweden, July. Association for Computational Linguistics.

Sittichai Jiampojamarn and Grzegorz Kondrak. 2010. Letter-phoneme alignment: An exploration. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 780–788, Uppsala, Sweden, July. Association for Computational Linguistics.

Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. 2007. Applying many-to-many alignments and hidden Markov models to letter-to-phoneme conversion. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 372–379, Rochester, New York, USA, April. Association for Computational Linguistics.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, Edmonton, Alberta, Canada. Association for Computing Machinery.

Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226, Philadelphia, Pennsylvania, USA. Association for Computing Machinery.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson Education, Inc., Upper Saddle River, New Jersey, USA, 2nd edition.

Mitesh M. Khapra, A Kumaran, and Pushpak Bhattacharyya. 2010. Everybody loves a rich cousin: An empirical study of transliteration through bridge languages. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 420–428, Los Angeles, California, June. Association for Computational Linguistics.

Anne K. Kienappel and Reinhard Kneser. 2001. Designing very compact decision trees for grapheme-to-phoneme transcription. In *EUROSPEECH-2001*, pages 1911–1914, Aalborg, Denmark, September.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4):pages 599–612.

John Kominek and Alan W. Black. 2006. Learning pronunciation dictionaries: Language complexity and word selection strategies. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 232–239, New York City, New York, USA, June. Association for Computational Linguistics.

Grzegorz Kondrak. 2000. A new algorithm for the alignment of phonetic sequences. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 288–295, Seattle, Washington, USA, April.

A. Kumaran, Mitesh M. Khapra, and Haizhou Li. 2010. Report of NEWS 2010 transliteration mining shared task. In *Proceedings of the 2010 Named Entities Workshop (NEWS 2010)*, pages 21–28, Uppsala, Sweden, July. Association for Computational Linguistics.

Haizhou Li, A Kumaran, Vladimir Pervouchine, and Min Zhang. 2009a. Report of NEWS 2009 machine transliteration shared task. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (NEWS 2009)*, pages 1–18, Suntec, Singapore, August. Association for Computational Linguistics.

Haizhou Li, A Kumaran, Min Zhang, and Vladimir Pervouchine. 2009b. Whitepaper of NEWS 2009 machine transliteration shared task. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (NEWS 2009)*, pages 19–26, Suntec, Singapore, August. Association for Computational Linguistics.

Haizhou Li, A Kumaran, Min Zhang, and Vladimir Pervouchine. 2010. Report of NEWS 2010 transliteration generation shared task. In *Proceedings of the 2010 Named Entities Workshop (NEWS 2010)*, pages 1–11, Uppsala, Sweden, July. Association for Computational Linguistics.

Linsen Loots and Thomas R. Niesler. 2009. Data-driven phonetic comparison and conversion between south african, british and american english pronunciations. In *Proceedings of Interspeech*, Brighton, UK, September.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, USA.

Colin P. Masica. 1991. *The Indo-Aryan languages*. Cambridge University Press, Cambridge, UK.

Evgeny Matusov, Nicola Ueffing, and Hermann Ney. 2006. Computing consensus translation from multiple machine translation systems using enhanced hypotheses alignment. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2006)*, pages 33–40, Trento, Italy, April. Association for Computational Linguistics.

Korin Richmond, Robert Clark, and Sue Fitt. 2009. Robust LTS rules with the Combilex speech technology lexicon. In *Proceedings of Interspeech*, pages 1295–1298, Brighton, UK, September.

Eric Sven Ristad and Peter N. Yianilos. 1998. Learning string edit distance. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 20(5):pages 522–532.

Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):pages 386–408.

Benjamin Snyder, Tahira Naseem, Jacob Eisenstein, and Regina Barzilay. 2009. Adding more languages improves unsupervised multilingual part-of-speech tagging: a bayesian non-parametric approach. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 83–91, Boulder, Colorado, USA, June. Association for Computational Linguistics.

Yan Song, Chunyu Kit, and Hai Zhao. 2010. Reranking with multiple features for better transliteration. In *Proceedings of the 2010 Named Entities Workshop (NEWS 2010)*, pages 62–65, Uppsala, Sweden, July. Association for Computational Linguistics.

Na'im R. Tyson and Ila Nagar. 2009. Prosodic rules for schwa-deletion in hindi text-to-speech synthesis. *International Journal of Speech Technology*, 12:pages 15–25.

Masao Utiyama and Hitoshi Isahara. 2007. A comparison of pivot methods for phrase-based statistical machine translation. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 484–491, Rochester, New York, USA, April. Association for Computational Linguistics.

Henk van den Heuvel, Jean-Pierre Martens, and Nanneke Konings. 2007. G2P conversion of names. what can we do (better)? In *Proceedings of Interspeech*, pages 1773–1776, Antwerp, Belgium, August.

Hua Wu and Haifeng Wang. 2009. Revisiting pivot language approach for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 154–162, Suntec, Singapore, August. Association for Computational Linguistics.

Qian Yang, Jean-Pierre Martens, Nanneke Konings, and Henk van den Heuvel. 2006. Development of a phoneme-to-phoneme (p2p) converter to improve the grapheme-to-phoneme (g2p) conversion of names. In *Proceedings of the 2006 International Conference on Language Resources and Evaluation*, pages 2570–2573, Genoa, Italy, May.

Min Zhang, Xiangyu Duan, Vladimir Pervouchine, and Haizhou Li. 2010. Machine transliteration: Leveraging on third languages. In *Coling 2010: Posters*, pages 1444–1452, Beijing, China, August. Coling 2010 Organizing Committee.

Appendix A

The McNemar test

Statistical significance is an important aspect to consider when comparing the performance of two systems, especially when the difference between the two is small or test data are scarce. In principle, the question being asked is this: what is the probability that the difference we're seeing is by chance? The task is to demonstrate that the null hypothesis—that there is, in actuality, no real difference between the performance of the two systems—is false. Generally, if the null hypothesis is found to have a probability of less than 5% of being correct (i.e., $p < 0.05$), it is said to be false, and the observed result is said to be **statistically significant**. The lower the probability of the null hypothesis, the more certain we can be of the result (and the more statistically significant it is). One way to interpret the probability is as the expected number of times the observed result would be seen if the experiment were repeated 100 times and the null hypothesis were true.

The McNemar test is one test to determine if a difference between two systems is statistically significant. The McNemar test looks specifically at the number of cases in which one system gets the answer right and the other system does not (and vice versa). In Table A.1, these are b and c. The McNemar test statistic is:

$$\chi^2 = \frac{(b-c)^2}{b+c}$$

and has a chi-squared distribution with one degree of freedom. The value of χ^2 is used to determine the significance level of the difference between the two systems in question, with the various levels ($p < 0.05$, $p < 0.01$, etc.) obtainable from a chi-squared distribution CDF table.

The formula for the McNemar test statistic has the important property that a stable error reduction is more significant than an unstable error reduction of equal magnitude. By stable, I mean the degree to which the error reduction is due only to improvements. For example, consider the following two cases:

1. System 2 gets five items correct that System 1 gets incorrect ($c = 5$). There are no items that System 1 handled correctly that System 2 handles incorrectly ($b = 0$).

2. System 2 gets 10 items correct that System 1 gets incorrect ($c = 10$). There are five items that System 1 handled correctly that System 2 handles incorrectly ($b = 5$).

		System 1	
		Incorrect	Correct
System 2	Incorrect	a	b
	Correct	c	d

Table A.1: Values relevant for the McNemar test, which uses only the b and c values.

The error reduction in terms of the number of items is the same in both cases: 5, making the numerator in the test statistic 25 in both cases. Because of the additional incorrect items in the second case, however, the denominator in the test statistic is different between the two; in case 1 it is 5 and in case 2 it is 15. Case 1 will then be found to be statistically significant, while case 2 will not.

Appendix B

G2P system comparison

Here I present a brief comparison of the G2P systems used in Chapter 4. The purpose is to provide an independent comparison of the three systems (Festival, Sequitur, and DirecTL+) on a common data set while considering not just their word accuracies but other aspects such as oracle re-ranker accuracies, runtime, model size, etc.

B.1 Experimental setup

As in Chapter 4, I use 10% of the data for testing and 10% for development; CELEX contains 66 372 unique entries while Combilex has 25 575. Since here I am not evaluating with any extra data, the full sets can be used rather than a subset. This allows us to see the performance of each system on the *full* data as well as consider the *overall* re-ranking potential using an oracle re-ranker. As in Chapter 4, an oracle re-ranker is one that simply selects the correct output in the n-best output list if it exists, which is what a perfect re-ranker would do.

As before I use 10-best output lists. System parameters such as the joint n-gram order for Sequitur or the maximum number of iterations for DirecTL+ are determined during development. Other than enabling the 10-best outputs, I use default settings for Sequitur and DirecTL+ and enable all features for the latter. As in Chapter 4 I construct Festival's output list manually from the list of possibilities provided for each character in the input word. All experiments were performed with an Intel Core i3-350M 2.26 GHz CPU with 4GB of RAM running Ubuntu 10.10.

B.2 Results

For both corpora, the patterns are the same: in terms of accuracy, Festival is far behind Sequitur, which in turn is significantly bested by DirecTL+, although the somewhat higher oracle re-ranking accuracy for Sequitur (also observed variously throughout Chapter 4) suggests higher re-ranking potential for Sequitur than for DirecTL+. Sequitur and DirecTL+'s increased word accuracies come at the cost of significantly higher training times, while DirecTL+ also has models that are one to two orders of magnitude larger than either Sequitur's or Festival's.

With no limiting factors, DirecTL+ provides the best performance, but if space

		WA	PA	OR	t_{trn}	t_{tst}	MS
CELEX	Festival	70.8	94.1	91.7	112	86	6.8
	Sequitur	91.9	98.1	98.7	1603	933	20.0
	DirecTL+	93.0	98.3	98.2	1576	718	678
Combilex	Festival	57.8	89.9	83.9	27	27	3.2
	Sequitur	70.8	92.4	93.8	185	13	8.6
	DirecTL+	75.3	93.6	92.4	222	186	400

Table B.1: G2P system comparison on CELEX (6 637 unique entries) and Combilex (2 802 unique entries) test sets. WA is word accuracy (%); PA is phoneme accuracy (%); OR is oracle re-ranker accuracy (%); t_{trn} is training time (minutes); t_{tst} is testing time (s); and MS is model size (MiB).

is an issue then Sequitur may be the better choice. DirecTL+ stores its models in plain text, which suggests room for improvement via storage in a binary format (and/or with compression), but there is only so much that the size can be reduced given the large number of string-based features and their weights. It may be possible to reduce the DirecTL+ model sizes by (for example) removing the features that have weights close to zero, but this has not been explored and may have negative effects on the system's accuracy. Of course, Festival is more than just a G2P system and provides end-to-end speech synthesis, but if it is being used as part of another system, Festival's use for G2P cannot be recommended unless training time is extremely limited (which is seldom the case; testing time may be a factor but once a model is trained it can be used repeatedly).