CH 1

Summary

In this chapter, we have provided an overview of the history of corpus and computational
linguistics, and the general methodology for creating an annotated corpus. Specifically,
we have covered the following points:

• Natural language annotation is an important step in the process of training com puters to understand human speech for tasks such as Question Answering, Machine
Translation, and summarization.

• All of the layers of linguistic research, from phonetics to semantics to discourse
analysis, are used in different combinations for different ML tasks.

• In order for annotation to provide statistically useful results, it must be done on a
sufficiently large dataset, called a *corpus*. The study of language using corpora is
corpus linguistics.

Summary

• Corpus linguistics began in the 1940s, but did not become a feasible way to study
language until decades later, when the technology caught up to the demands of the
theory.

• A corpus is a collection of machine-readable texts that are representative of natural
human language. Good corpora are *representative* and *balanced* with respect to the
genre or language that they seek to represent.

• The uses of computers with corpora have developed over the years from simple keyword-in-context (KWIC) indexes and concordances that allowed full-text docu
ments to be searched easily, to modern, statistically based ML techniques.

• Annotation is the process of augmenting a corpus with higher-level information,
such as part-of-speech tagging, syntactic bracketing, anaphora resolution, and word
senses. Adding this information to a corpus allows the computer to find features
that can make a defined task easier and more accurate.

• Once a corpus is annotated, the data can be used in conjunction with ML algorithms
that perform classification, clustering, and pattern induction tasks.

• Having a good annotation scheme and accurate annotations is critical for machine
learning that relies on data outside of the text itself. The process of developing the
annotated corpus is often cyclical, with changes made to the tagsets and tasks as the
data is studied further.

• Here we refer to the annotation development cycle as the MATTER cycle—Model,
Annotate, Train, Test, Evaluate, Revise.

• Often before reaching the Test step of the process, the annotation scheme has already
gone through several revisions of the Model and Annotate stages.

• This book will show you how to create an accurate and effective annotation scheme
for a task of your choosing, apply the scheme to your corpus, and then use ML
techniques to train a computer to perform the task you designed.

C 2

Summary

In this chapter we discussed what you need to create a good definition of your goal, and
how your goal can influence your dataset. In particular, we looked at the following
points:

• Having a clear definition of your annotation task will help you stay on track when
you start creating task definitions and writing annotation guidelines.

• There is often a trade-off in annotation tasks between informativity and accuracy
—be careful that you aren't sacrificing too much of one in favor of another.
• Clearly defining the scope of your task will make it much easier to decide on the
sources of your corpus—and later, the annotation tags and guidelines.

C3

Summary

In this chapter we introduced you to the tools you need to analyze the linguistic content
of a corpus as well as the kinds of techniques and tools you will need to perform a variety
of statistical analytics. In particular, we discussed the following:
• Corpus analytics comprises statistical and probabilistic tools that provide data
analysis over your corpus and information for performing inferential statistics. This
will be necessary information when you take your annotated corpus and train an
ML algorithm on it.
• It is necessary to distinguish between the occurrence of a word in a corpus (the
token) and the word itself (the type).
• The total number of tokens in a corpus gives us the corpus size.
• The total number of types in a corpus gives us the vocabulary size.
• The rank/frequency profile of the words in a corpus assigns a ranking to the words,
according to how many tokens there are of that word.

• The frequency spectrum of the word gives the number of word types that have a
given frequency.
• Zipf's law is a power law stating that the frequency of any word is inversely proportional to its rank.
• Constructing n-grams over the tokens in a corpus is the first step in building language models for many
NLP applications.
• Pointwise mutual information is a measure of how dependent one word is on another in a text. This can
be used to identify bigrams that are true collocations in a
corpus.
• Language models for predicting sequence behavior can be simplified by making the Markov assumption,
namely, when predicting a word, only pay attention to the word before it.

C4

Summary

In this chapter we defined what models and specifications are, and looked at some of
the factors that should be taken into account when creating a model and spec for your
own annotation task. Specifically, we discussed the following:
• The model of your annotation project is the abstract representation of your goal,
and the specification is the concrete representation of it.
• XML DTDs are a handy way to represent a specification; they can be applied directly
to an annotation task.
• Most models and specifications can be represented using three types of tags:
document-level labels, extent tags, and link tags.
• When creating your specification, you will need to consider the trade-off between
generality and specificity. Going too far in either direction can make your task
confusing and unmanageable.
• Searching existing datasets, annotation guidelines, and related publications and
conferences is a good way to find existing models and specifications for your task.

• Even if no existing task is a perfect fit for your goal, modifying an existing specification can be a good way to keep your project grounded in linguistic theories.
• Interoperability and standardization are concerns if you want to be able to share your projects with other people. In particular, text encoding and annotation format can have a big impact on how easily other people can use your corpus and annot ations.
• Both ISO standards and community-driven standards are useful bases for creating your model and specification.
• Regional differences in standards of writing, text representation, and other natural language conventions can have an effect on your task, and may need to be repre sented in your specification.

C5

Summary

In this chapter we discussed some of the different methods for representing annotations for a corpus annotation task. In particular, we noted the following:
• Annotations can be stored in many different ways, but it's important to choose a format that will be flexible and easy to change later if you need to. We recommend stand-off, XML-based formats.
• In some cases, such as single-label document classification tasks, there are many ways to store annotation data, but these techniques are essentially isomorphic. In such cases, choose which method to use by considering how you plan to use the data, and what methods work best for your annotators.
• For most annotation tasks, such as those requiring multiple labels on a document, and especially those requiring extent annotation and linking, it will be useful to have annotation software for your annotators to use. See Appendix B for a list of available software.
• Extent annotation can take many forms, but character-based stand-off annotation is the format that will make it easier to make any necessary changes to the annotat ions later, and also make it easier to merge with other annotations.
• If you do choose to use character-based stand-off annotation, be careful about what encodings you use for your data, especially when you create the corpus in the first place. Different programming languages and operating systems have different de fault settings for character encoding, and it's vital that you use a format that will work for all your annotators (or at least be willing to dictate what resources your annotators use).
• Using industry standards such as XML and annotation standards such as LAF for your annotations will make it much easier for you to interact with other annotated corpora, and make it easier for you to share your own work.

C6

Summary

In this chapter we discussed how to apply your model and spec to your corpus through the process of annotation, and how to create a gold standard corpus that can be used to train and test your ML algorithms. Some of the important points are summarized here:
• The "A" in the MATTER cycle is composed of a lot of different parts, including creating annotation guidelines, finding annotators, choosing an annotation tool,

training annotators, checking for IAA scores, revising guidelines, and finally, ad judicating. Don't be put off by the number of steps outlined in this chapter; just take
your time and take good notes.
• Guidelines and specifications are related, but they are not the same thing. The guidelines will determine how the model is applied to the text—even if you are using the same model, if the guidelines are different, they can result in a very different set of annotations.

• Creating a good set of annotation guidelines and an accurate and useful annotation won't happen on the first try. You will need to revise your guidelines and retrain your annotators, probably more than once. That's fine; just remember to allow yourself time when planning your annotation task. Within the MATTER cycle is the MAMA cycle, and no task is perfect straight off the bat.
• One of the things you will have to consider about the annotation process is what information you want to present to your annotators—giving them preprocessed data could ease the annotation process, but it could also bias your annotators, so consider what information you want to present to your annotators.
• Because you'll need to go through the MAMA cycle multiple times, it's a good idea to set aside a portion of your corpus on which to test your annotation guidelines while you work out the kinks. This set can be used in your gold standard later, but shouldn't be given to your annotators right away once the guidelines are finalized.
• When you're writing your annotation guidelines, there are a few questions that you'll find it necessary to answer for your annotators in the document. But the most important thing is to keep your guidelines clear and to the point, and provide plenty of examples for your annotators to follow.
• When finding annotators for a task, you need to consider what type of knowledge they will need to complete your annotation task accurately (and, if possible, quick ly), what language they should speak natively, and how much time you have to
annotate. The last consideration may play a role in how many annotators you need to hire to complete your task on schedule.
• The annotation software that you give to your annotators to create the annotations will have an effect on how easily and accurately the annotations are created, so keep that in mind when choosing what tool you will use. Using more than one piece of software for the same task could cause confusion and irregularities in the annotat ion, so it's better to pick one and stick with it.
• Once your annotators have annotated your sample set of texts, it's time to evaluate the IAA scores. While there are many different ways to determine agreement, the two most common in computational linguistics are Cohen's Kappa and Fleiss's Kappa. Cohen's Kappa is used if you have only two annotators annotating a docu ment, while Fleiss's Kappa is used for situations where more than two annotators are used on a dataset.
• Based on how good your agreement scores are, you can decide whether or not your task is ready to go past the test set and on to the full corpus. You will probably need to revise your task at least once, so don't be discouraged by low IAA scores.
• Interpreting IAA scores isn't an exact science—a number of factors can influence whether a score indicates that an annotation task is well defined, including the number of tags, the subjectivity of the annotation task, and the number of annotators.

• Additionally, the items being annotated can have an effect on how you calculate IAA. While it's easy to calculate agreement when applying a single label to an entire document, there is some debate about how IAA scores should be calculated when applying tags to text extents. Regardless of what method you decide to apply for calculating IAA scores, keep track of the decisions you make so that other people can understand how you came up with your numbers.

• Having high IAA scores mean your task is likely to be reproducible, which is helpful
when creating a sufficiently large corpus. However, just because a task is reproducible doesn't necessarily mean it will be suitable for feeding to ML algorithms. Similarly, just because a task doesn't have great agreement scores doesn't mean it will not be good for ML tasks. However, a reproducible task will be easier to create a large corpus for, and the bigger your corpus, the more likely you are to get good ML results, so putting some effort into creating your annotation guidelines will pay off in the end.
• Once you've reached acceptable IAA scores on your annotation test set, you can set
your annotators loose on the full corpus. When you have the full corpus, it's time
to adjudicate the annotations and create the gold standard corpus that you will use
to train your ML algorithms.
• Adjudication is best performed by people who helped create the annotation guide
lines. Bringing in new people to perform the adjudication can cause more confusion
and noise in your dataset.
• Calculating IAA agreement scores between adjudicators can be a good way to ensure
that your adjudicated corpus is consistent. The more consistent your corpus is, the
more accurate your ML results will be.

C7

Summary

In this chapter we looked at how the model and annotation you have been developing
are able to feed into the ML algorithm that you will use for approximating the target
function you are interested in learning. We discussed the differences between the different feature types: n-gram features, structure-dependent features, and annotation dependent features. We reviewed how these features are deployed in several important learning algorithms, focusing on decision tree learning and Naive Bayes learning. Here is a summary of what you learned:
• ML algorithms are programs that get better as they are exposed to more data. ML
algorithms have been used in a variety of computational linguistic tasks, from POS
tagging to discourse structure recognition.
• There are three main types of ML algorithms: supervised, unsupervised, and semisupervised.

• Supervised learning uses annotated data to train an algorithm to identify features
in the data that are relevant to the intended function of the algorithm.
• N-gram features allow algorithms to take information about the words that are in
a document and examine aspects of the data such as term frequency to create as
sociations with different types of classifications.
• Structure-dependent features are defined by the properties of the data, such as
strings of characters, HTML or other types of markup tags, or other ways a docu ment can be organized.
• Annotation-dependent features are associated with the annotation and reflect the
model of the annotation task.
• A learning task is defined in five steps: choose the corpus that will be trained on,
identify the target function of the algorithm, choose how the target function will
be represented (the features), select an ML algorithm to train, and evaluate the
results.
• Another way to use an annotated corpus in a software system is to design a rulebased
system: a program or set of programs that does not rely on an ML algorithm
being trained to do a task, but rather has a set of rules that encode the features that
an algorithm could be trained to identify.
• Rule-based systems are a good way to identify features that may be useful in a
document without having to take the time to train an algorithm. For some tasks
(e.g., temporal expression recognition), rule-based systems outperform ML algorithms.

• Classification algorithms are used to apply the most likely label (or classification) to a collection. They can be applied at a document, sentence, phrase, word, or any other level of language that is appropriate for your task.
• Using n-gram features is the simplest way to start with a classification system, but structure-dependent features and annotation-dependent features will help with more complex tasks such as event recognition or sentiment analysis.
• Decision trees are a type of ML algorithm that essentially ask “20 questions” of a corpus to determine what label should be applied to each item. The hierarchy of the tree determines the order in which the classifications are applied.
• The “questions” asked at each branch of a decision tree can be structure-dependent, annotation-dependent, or any other type of feature that can be discovered about the data.
• A Naive Bayes learning algorithm tries to assign the probability that an item will be assigned a particular classification, based on the association between what features are associated with each item, and how strongly those features are associated with a particular label as determined by the Bayes Theorem.

• The K-nearest neighbor algorithm uses much simpler rules than the Naive Bayes methods, with simple associations between features and classifications.
• Unsupervised learning is the process of “teaching” an algorithm without giving it any starting information about the classifications that exist in the dataset.
• Classic examples of unsupervised learning are clustering algorithms, which find natural groupings in the data in order to create sets of similar items. Essentially, clustering algorithms find their own features for the datasets.
• Because clustering algorithms don’t take features from the users but instead discover their own, the groups that a classifier creates may not be the ones that you have in mind for your task.
• Semi-supervised learning techniques allow you to use a small amount of labeled data to generate labels for larger sets of data, and are an effective way to deal with very large datasets.
• If you aren’t sure where to start with your algorithm training, try to find a task or annotation similar to your own and see what algorithms are commonly used for those projects. If there aren’t any projects just like yours, try to find ones that have similar features or feature types.

Ch8

• Corpus linguistics began in the 1940s, but did not become a feasible way to study language until decades later, when the technology caught up to the demands of the theory.
• A corpus is a collection of machine-readable texts that are representative of natural human language. Good corpora are *representative* and *balanced* with respect to the genre or language that they seek to represent.
• The uses of computers with corpora have developed over the years from simple keyword-in-context (KWIC) indexes and concordances that allowed full-text docu ments to be searched easily, to modern, statistically based ML techniques.
• Annotation is the process of augmenting a corpus with higher-level information, such as part-of-speech tagging, syntactic bracketing, anaphora resolution, and word senses. Adding this information to a corpus allows the computer to find features that can make a defined task easier and more accurate.
• Once a corpus is annotated, the data can be used in conjunction with ML algorithms that perform classification, clustering, and pattern induction tasks.
• Having a good annotation scheme and accurate annotations is critical for machine learning that relies on data outside of the text itself. The process of developing the annotated corpus is often cyclical, with changes made to the tagsets and tasks as the data is studied further.

• Here we refer to the annotation development cycle as the MATTER cycle—Model, Annotate, Train, Test, Evaluate, Revise.
• Often before reaching the Test step of the process, the annotation scheme has already
gone through several revisions of the Model and Annotate stages.
• This book will show you how to create an accurate and effective annotation scheme for a task of your choosing, apply the scheme to your corpus, and then use ML techniques to train a computer to perform the task you designed.Summary

In this chapter we discussed how to test and evaluate your ML algorithm's performance
on your dev-test and final-test data. Some of the key points in this process are summarized here:
• Algorithms are tested multiple times over the development-training and development-testing sets of your corpus. After each run of the algorithm on the dev-test data, evaluation of the results is performed and the algorithm/features are adjusted as necessary.
• Creating a confusion matrix of your results is an excellent way to identify problems
with your algorithm's performance, and also makes calculating accuracy scores much easier.
• Calculating the percentage accuracy for your algorithm provides a baseline level
of evaluation, but it doesn't tell the whole story about how your algorithm is performing.
• Precision and recall are metrics used to compare the number of true positives, false
positives, and false negatives that your algorithm is producing. A high precision
score indicates that your algorithm is not returning many false positives; a high
recall score indicates that it is not producing a lot of false negatives.
• Depending on the purpose of your algorithm, you may want to favor high recall
over high precision, or vice versa.
• An F-measure is the harmonic mean of the precision and recall scores, and provides
a more robust overall metric of your results.
• Other evaluation metrics such as ROC, chi-squared, t-tests, and ANOVA can be
used to identify discrepancies and variables in your algorithm as well.
• The interpretation of evaluation scores is highly dependent on the current state of
the art in research areas similar to what you are working on—you'll need to see how
other, similar algorithms are performing to determine how yours stacks up.
• If your dataset is too small to divide into training and testing sets, you can use crossvalidation
to evaluate your algorithm.
• Beware of overfitting your algorithm to your training set!
Run your algorithm over your final testing set as few times as possible (ideally, once),
and make sure to report the evaluation metrics for that run, as well as any changes
that you made between runs.
Summary

CH9

Summary

In this chapter we discussed some of the important considerations that should be taken
into account when revising your annotation task, or reporting the results to other re
searchers. In particular, we suggested the following:
• Once you've reached the Revision stage of the MATTER cycle, it's time to look back
and consider what you could do differently, and how your annotations and algorithm can be used in the future.
• Consider whether there are any changes that you would make to the corpus, or
whether the domain of the corpus could be expanded.
• If there were aspects of your model or specifications that annotators found difficult
to implement or that weren't useful for machine learning, how would you change
them? Or would you remove them entirely?

• Similarly, are there any tags or classes that you would add to your model or specification?

• Many different factors can affect annotation tasks, including annotation software, the guidelines, and the annotators themselves: is there anything about those aspects of your task that you would change?
• If you focused on annotation-based features during your ML training and testing, are there any structure-based features that you could look at adding next time? Or vice versa?
• As for reporting about your work, the more details that you make available to other researchers, the more useful they will find your corpus and annotations. If an an notation task is difficult to understand, it's unlikely to be used in other research.
• You don't have to publish a journal article or conference paper to make your corpus available—making your data available on a website is good too!
• When you're writing up the details of your annotation and ML task, consider all the different types of people who might be interested in your work: people looking to create their own annotation based on yours, people who want to do theoretical research on your corpus, people who want to train their own ML algorithms, and so on. Try to make available the relevant information that each of those groups will want to know about.
• Let people know how you created your corpus and what criteria you used to include (or exclude) certain texts.
• When explaining your model and specifications, don't just publish the DTD—make sure you explain where those tags and attributes came from and what theories they are grounded in.
• Because so many factors can affect an annotation task, it's important to report on as many details as you can, including information about the annotators, the soft ware, how the IAA scores were calculated, and so on. That will allow readers to better compare your work to other corpora and annotations.
• Similarly, let people know how you trained your ML algorithm and which features were useful for you (and which ones weren't), as well as how you calculated the accuracy scores.
• If you have ideas about how you would revise your annotation or algorithm in the future, let people know! Ideas for expanding your corpus or annotation are always welcome, but it's also important to acknowledge places where you might have made a mistake that can be fixed later.
• Most important, the more information you share about your task, the more useful your task will be to other people, even if they don't always agree with the decisions you made.

CH10

Summary

In this chapter we provided an overview of the MAMA cycle for TimeML, including a discussion of the changes in goal, model, and specification. In particular, we looked at the following:
• TimeML (formerly TenseML) was developed to provide a way to annotate events and their temporal anchorings in text, specifically news articles. The bulk of the TimeML specification development was done at a workshop in early 2002.
• One of the workshop's goals was to create a gold standard annotated corpus, called TimeBank. While initially TimeBank was going to contain 300–500 articles, the density of the TimeML annotation eventually left that to be cut down to 183. Howe ver, those articles are available for use as a resource from *http://timeml.org/*.
• TimeML didn't spring from the minds of the workshop's working group fully

formed; it was based on some existing temporal and event annotations, and a lot of
existing theories about linguistics and temporal reasoning.
Summary

• A primary source for TimeML was the dissertation work of Andrea Setzer, which
resulted in the Sheffield Temporal Annotation Guidelines (STAG). The TIMEX2 tag
was also an influencing factor, though eventually the working group created their
own temporal annotation tag, TIMEX3.
• The specification for TimeML went through multiple iterations of the MAMA cycle,
both during the course of the working group and afterward, as more refinements
were made to the tags and attributes.
• During the workshop, periodic “annotation fests” were held, which allowed members of the working group to try out the current specifications on a small set of
documents, which proved to be a very effective way of finding places where the spec
wasn’t suited to the reality of the texts.
• The final pre-ISO TimeML specification (version 1.2.1) consisted of tags: TIMEX3,
EVENT, MAKEINSTANCE, SIGNAL, TLINK, ALINK, SLINK, and CONFIDENCE.
• Each of the TimeML tags has a set of attributes that is used to capture the relevant
information about the text, such as the verb tenses, modality, relation type of the
links, and value of the temporal expression (such as a calendar date or date relative
to another event).
• The current version of TimeBank (version 1.2) is annotated with version 1.2.1 of
TimeML. Because the specifications changed over the years, most of the annotations
in the current TimeBank are modifications of the previous annotation efforts, which
was more efficient than reannotating the entire corpus from scratch.
• ISO-TimeML is an expanded version of TimeML that allows for the annotation of
temporal information in languages other than English, and provides more robust
handling of complex temporal expressions. It is also compliant with other ISO
standards, and provides more interoperability options for different annotation
frameworks.
• Even though TimeML is currently an ISO standard, it is still undergoing changes
as more research is done into temporal information and events. Some of these
changes include adding new types of temporal objects, called Narrative Containers;
adapting the TimeML specification and guidelines to new domains, such as clinical
texts; and adding information about event classes and structures to TimeMLcompliant
resources.

CH11

Summary

In this chapter we discussed the process used to automate the creation of TimeMLannotated
texts through the building of the TARSQI Toolkit. In particular, we discussed
the following:
• The TARSQI Toolkit (TTK) provides an example of a long-term production cycle
for a system built to reproduce human annotation. While this chapter didn’t go into
details of the Training–Evaluation cycles of each individual component, it did pro
vide a “big picture” look at a complex system.
• Each component in the TTK is designed to recreate a tag in the TimeML specification—and later versions of the TTK use multiple components for some of the
tags.
• Using a divide-and-conquer approach may be the best way to approach your own
annotation, especially if you have a task where some parts of the annotation rely on

the existence of others (e.g., TLINKs can be made without TIMEX3 and EVENT tags
already in the document).
• Rule-based systems can be an excellent way to try out ideas about how to automat
ically create your annotations, and in some cases will still outperform ML algo
rithms.
• The question of whether rule-based or statistics-based systems are superior for
complicated tasks such as temporal processing has not yet been answered, as both
systems tend to perform at nearly the same accuracy levels.
• There is always more that can be done to improve an automatic annotation system,
whether it's simply improving accuracy, or expanding the genres or domains that
the system can be used to annotate.

CH12

Conclusion:

And Finally...

In this chapter our goal was to show you the role that annotation is playing in cuttingedge
developments in computational linguistics and machine learning. We pointed out
how the different components of the MATTER development cycle are being experi mented with and
improved, including new ways to collect annotations, better methods
for training algorithms, ways of leveraging cloud computing and distributed data, and
stronger communities for resource sharing and collaboration.
Because of a lack of understanding of the role that annotation plays in the development
of computational linguistics systems, there is always some discussion that the role of
annotation is outdated; that with enough data, accurate clusters can be found without
the need for human-vetted categories or other labels. But in fact, what we hope to have
shown here is that the exact opposite is true. Namely, that only with a better under
standing of how annotation reflects one's understanding of linguistic phenomena will
a better set of features for the model and algorithm be created, whether it is rule-based
or statistical.

248