Creating a Document

Categorization Corpus

[Terminology](#_3h7w6qm0oniv)

[Introduction](#_he7596blmwo)

[The MATTER Cycle](#_j3q3y17i7dgw)

[Modeling the Problem](#_odnorkajjvwc)

[How many categories?](#_b9raqh9daxtf)

[Are there any relationships between categories?](#_v7tx1l2ey2vg)

[How separable are your categories?](#_grsw10kxu9ze)

[Data Collection](#_d7g80uvh6k8b)

[Data Desiderata](#_7pmnc4n4mux3)

[Balance](#_cbvae6fndxet)

[Representativeness](#_hothlg8s2pjo)

[Size](#_iu8v8zyodutd)

[Intellectual Property](#_jsvf5spjhewg)

[Annotation](#_7oyvkxdtrxfb)

[Annotation Guidelines](#_2e6690wruzke)

[Annotation Tools](#_r1ytow444p9h)

[Measuring Annotator Reliability](#_4cafomnlixse)

[Training and Evaluating your Custom Model](#_k4ypb2awxora)

[Experimental Setup](#_by05o8g4shem)

[Common Pitfalls](#_kn9014s47ohx)

[Insufficient Data](#_k4l91va244h9)

[Imbalanced Data](#_bgn29jnwrgq6)

[Duplicate Data](#_vhnthol40qm5)

[Unclean Data](#_wddl5o7gfx1i)

[Badly encoded data](#_4hpi3x2d1rh)

[Markup](#_jz3c4p8ky9ys)

[Metadata](#_a6ta6vmfcig3)

# Terminology

While these specific terms may have broader definitions in other contexts, within the scope of this document, we will use the following definitions:

* Data: written prose (we are only concerned with natural language data)
* Document: a discrete, cohesive unit of data (e.g., a tweet, a letter, a news article, etc.—this is intentionally loosely defined)
* Category: a class or division of documents
* Label: the name of a particular category
* Model: a set of categories or labels (and possibly relationships between categories)
* Taxonomy: a model exhibiting a hierarchy or tree-like structure branching from most general to most specific
* Corpus: a collection (literally “body”) of documents
* NLP: Natural Language Processing
* Annotation: the process of assigning labels to documents
* Manual annotation: the process of annotating by hand (by humans)
* Classifier: a system that assigns labels to documents
* Classification: the process of automatically annotating documents using a classifier (by machine)

# Introduction

This document is intended for an audience who is interested in building a corpus with the intent to develop a customized, automated, document categorizer to classify natural language documents. This is a particular type of problem within the domain of Natural Language Processing (NLP). If this is your first exposure to developing an NLP solution, we highly recommend reading *Natural Language Annotation for Machine Learning: A guide to corpus-building for applications*[[1]](#footnote-1) (NLAML). We will assume a development cycle based on the MATTER cycle proposed by Pustejovsky and Stubbs:

## The MATTER Cycle

1. **M**odel: Determine the set of categories of interest.
2. **A**nnotate: Manually assign category labels to a set of documents
3. **T**rain: train a classifier using a training set of the manually labeled documents.
4. **T**est: Test the classifier on a held-out development set of the manually labeled documents that does not overlap with the training set and perform error analysis.
5. **E**valuate: You can evaluate the classifier on yet another held out subset of your manually labeled documents. (To preserve the experimental utility of your evaluation set, do ***not*** perform error analysis on your evaluation set).
6. **R**evise: Revise your model or consider augmenting your corpus by annotating additional documents.

Generally, you will not attain excellent results with a single cycle. Don’t get discouraged if your initial results are unsatisfactory. Attaining highly accurate models typically requires multiple iterations. It can be a lot of work between collecting raw data, cleaning your data, manually annotating your data, and performing error analysis.

# Modeling the Problem

The first step in the MATTER cycle is to define the set of categories that are of interest to you. Scoping the problem is important. At least for a single iteration of the MATTER cycle, you need to concretely define the set of categories of interest—obviously you can revise your model later, but you need to choose a concrete model as a target. There are several considerations here:

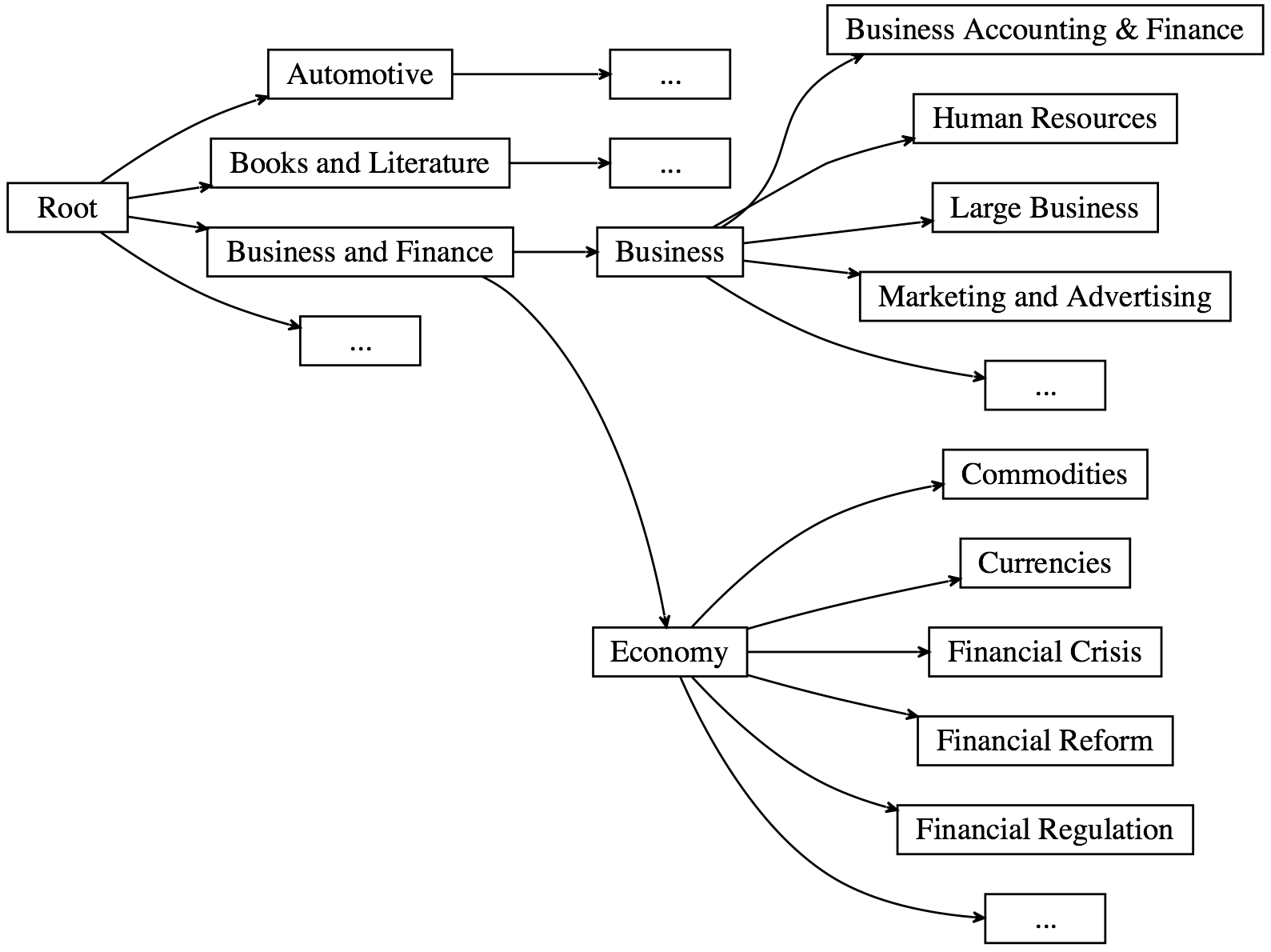
1. How many categories?
2. Are there any relationships between categories?
3. How separable or your categories?

## How many categories?

Note that the number of categories you are interested in will determine the scope of your effort later on. For instance, if you decide to include thousands of categories in your model, when it comes time to manually annotate documents, or perform error analysis, it will be cumbersome to sort through all the possible labels for such a complex model. We recommend keeping your model as small as possible while still capturing the distinctions you wish to capture within your problem domain. If you can find categories that are very similar to one another, it may be appropriate to combine them. If you find your initial categories are too broad, you may need to split them up. You should also consider that, within your problem domain, it is best to try to cover all possibilities with your model. That is to say, while it may seem convenient, it is not recommended to include a ‘catch-all’ or ‘other’ category within your model.

## Are there any relationships between categories?

For some domains, a classification problem may lend itself to a taxonomic model with a hierarchy of categories. It is important to consider such hierarchical relationships when specifying your model. If there are hierarchical relationships between your categories, do you care about the upper levels of the hierarchy, or only the more specific categories? For instance, one standard taxonomy developed by the Interactive Advertising Bureau is the [IAB Tech Lab Taxonomy](https://www.iab.com/guidelines/iab-quality-assurance-guidelines-qag-taxonomy/). The following is a small subset of the IAB taxonomy:



Depending on the depth of your taxonomy, you need to determine at what level of granularity to cut off your model. At a certain point, the distinctions between categories may be too subtle even for humans to reliably determine. For instance, considering the subset of the IAB taxonomy above, if you are interested in the domain of Business and Finance, would it be sufficient to distinguish Business from Economy? Or, is it important to go to the more specific subcategories? For a human readable layout of a hierarchical model, it can be extremely helpful to lay out a taxonomy as a graph (such as above), or in a spreadsheet or similar tabular layout (this is how the [IAB Tech Lab Taxonomy is distributed](https://www.iab.com/wp-content/uploads/2017/11/IAB_Tech_Lab_Content_Taxonomy_V2_Final_2017-11.xlsx)). For a machine readable format, a tabular format may be suitable, or you may consider a data format such as XML or JSON (the IAB Tech Lab distributes [a machine readable JSON version](https://github.com/InteractiveAdvertisingBureau/taxonomy) of their taxonomy as well).

## How separable are your categories?

At the extremes of being overly specific, or overly general, your classification task will become more difficult. At first, it is recommended that you experiment with more general categories before attempting to make fine-grained distinctions. It is also important to consider the level of mutual exclusivity in your model. Are there categories that may overlap with one-another? If at all possible, try to keep your categories mutually exclusive.

If you find yourself in a situation where you have overlapping categories, consider adding an additional category to cover the overlap. For example, imagine you are building a model for classifying cooking recipes, and you are considering a model with Baking and Desserts. You may find that there is a lot of overlap between these two categories, so it may be appropriate to consider splitting Deserts into Baked Desserts and Non-Baked Desserts. Alternatively, it may be preferable to make Baked Desserts a subcategory of Baking. While intuitions can be helpful, it is important to motivate these sorts of decisions based on the data itself rather than what you *think* the data should look like. Sometimes the data will surprise you, and a model based on preconceptions that are not based on real data will suffer accordingly. Note that in the case of overlapping labels you may be interested in multi-label classification, where a document may be assigned to multiple categories.

Another consideration is whether the categories in your model are mutually exclusive with one another. It is important to note these constraints and relationships as they will help to guide your manual annotation process as well as your overall design. In the case of mutually exclusive labels, or those labels that are totally independent of one another, you may be interested in creating multiple classifiers and combining their outputs. For instance, you may want one model to classify document topics, but another model to classify emotions expressed in documents. These are essentially independent because any given emotion can be expressed along with any given topic. In this case it may be appropriate to separate these concerns by creating separate models. It still may be worth experimenting to see if a single, multi-label model can be trained to classify documents across these independent dimensions.

# Data Collection

Data collection is an important part of the process. The sources you consider for data collection should be reflective of the real-world data you intend to classify. There are several characteristics to consider when collecting your data.

## Data Desiderata

A statistical classifier makes predictions based on what it learned from training data. Therefore, your training data must satisfy certain requirements. These characteristics are:

### Balance

The dataset should be balanced so as to avoid bias on the part of the model. For each category, you want to have a roughly equal number of examples. It may also be important to scrutinize available metadata about your dataset. You should ensure that different metadata values are distributed evenly across the distribution of values. For example, if the dataset is constructed from multiple sources, ideally there shouldn’t be disproportionately more data from one source than from another. If the data collected belongs to different time periods, try to ensure that the documents form a balanced distribution across time. When sampling data from various sources, you may need to downsample your data in order to maintain balance. Depending on the variables and the task, it may be beneficial to downsample your data in order to maintain balance even if it reduces the size of your dataset. That is to say, it’s OK to remove documents from your data set in order to maintain balance as long as the overall size of your data set is sufficient (the overall size requirements may differ from task to task, see below).

### Representativeness

The dataset must represent all the categories in your model. Additionally, the kind of data you are interested in should be reflected in your corpus. If you are interested in classifying tweets, then you should collect tweets for your corpus (rather than, say, Wikipedia articles, instruction manuals, or sports commentary transcripts). Similarly, if you are interested in classifying news articles by topic, your corpus should consist of news articles (not tweets or novels). Language, domain, genre, register, and document size are all aspects of the content that you should consider when assessing representativeness. Also, corroborate that the dataset is free of noise. Documents that are not relevant for the task or do not contain sufficient lexical content should be removed. E.g., a document whose content consists of a URL (and nothing else) is not a useful sample (unless you are analyzing URLs).

Note that you can use Rosette’s language identification functionality to select only those documents that are written in the languages of interest to you. If you are interested in multiple languages, you will need to create a separate corpus for each language—document classification models trained on one language are not expected to function across different languages.

### Size

Generally, the more the data, the better. This means that the model has more data to train on and will be able to make more confident decisions when making predictions. However, be careful not to pad your corpus with non-representative data only to increase the size of the corpus. Balance *and* representativeness are both important. One strategy that can be used to assess if you have enough data is to plot a [learning curve](https://en.wikipedia.org/wiki/Learning_curve). Since you cannot plot a learning curve until you get to the Test part of the MATTER cycle, when you are first collecting your dataset, it is helpful to collect more data than you think you may need. You can always throw away data later. It can be difficult to restart a data collection effort later on (or impossible if a data source becomes unavailable).

### Intellectual Property

If you are using your data for a personal or academic project, there may be intellectual property concerns, but they are fewer than if you plan to use a data set for commercial purposes. It is important to investigate who is the owner of the data, and what licensing restrictions exist on the data (if any). If you own the data, you may consider putting the data in the public domain, or creating an explicit license to allow or prevent others from deriving value from your data as you see fit. You can find some licenses that are friendly for commercial purposes [here](https://opensource.org/licenses/). If you have concerns about intellectual property, always consult a lawyer.

# 

# Annotation

The first thing to consider when annotating your data is the annotation task itself and any tools that your annotators will use to perform the task. To achieve high quality manual annotations there are several considerations:

1. Annotation guidelines
2. Annotation tools
3. Measuring annotator reliability

## Annotation Guidelines

In the case of document categorization, annotators must assign labels to documents, so it is important to keep this in mind when writing guidelines and selecting an annotation tool. Many annotation tools you will find are intended for specific NLP tasks that require assigning labels to linguistic units (words, sentences, etc.) within documents rather than documents themselves. This will have a major impact when deciding on an annotation environment, since some interfaces are more suitable than others for certain levels of annotation. To keep track of a large collection of documents, if your documents don’t already have an inherent schema for identifying them, we recommend assigning each document a [Universally Unique Identifier (UUID)](https://en.wikipedia.org/wiki/Universally_unique_identifier). This will help you (and your annotators) manage the annotation process and provides a consistent, convenient way to refer to specific documents.

It is often helpful to run a pilot annotation effort when synthesizing annotation guidelines. This could be a small subset of annotators who will work on the broader annotation effort, or it could be the author of the guidelines. It is highly recommended that the author do some annotations him or herself—it’s very difficult to guide others through a task you have not performed yourself. During the pilot annotation effort, you will encounter examples or edge cases that you hadn’t considered, and these will be useful to provide guidance about how to handle those cases (or to revise your model to account for them).

When composing annotation guidelines, it is important to describe each category and what it is intended to capture. While descriptions are useful, examples are essential. For each category, include positive examples—cases where the label in question applies. Sometimes negative examples are equally instructive—examples where the label in question does not apply. For examples that are not straightforward (such as those that annotators disagreed on), it is important to discuss those examples and explain the reasoning behind the final decision that was made.

## Annotation Tools

An annotation tool like [brat](http://brat.nlplab.org/) is designed for tasks that require assigning labels to individual words, phrases, or other linguistic units within documents. For document categorization, however, it may be simpler to use a spreadsheet application, such as Google Sheets. Whatever tool you decide to use, keep in mind that it will be most convenient for your annotators if they can easily access the full text of the documents and choose a label from your model to assign to each document. If there are inherent constraints on your category labels, it is often best to enforce those constraints in the annotation tool itself rather than rely on your annotators. For example, if you are using Google Sheets, set up data validation so that annotators’ inputs will be rejected if they fall outside your model’s defined set of categories. Any extra cognitive load or room for error that you add to the task has the potential to reduce the quality of the annotations (not to mention, it will make your annotators unhappy). It is also important to consider the size of the corpus. Some annotation environments may not be able to support a corpus past a certain size. Finally, factor in the resources needed to set up the annotation environment for the annotators, i.e. time to train the annotators and provide them with the necessary software if need be.

The following are some of the available annotation tools for NLP applications:

|  |  |  |  |
| --- | --- | --- | --- |
| **Software** | **Use** | **Language** | **URL** |
| *brat* | Manual Annotation | Language-independent | *http://brat.nlplab.org/* |
| *BAT (Brandeis Annotation Tool)* | Corpus annotation | Language-independent | *http://timeml.org/site/bat/* |
| *Djangology: A lightweight web-based tool for distributed collaborative text annotation* | Distributed collaborative text annotation | Language-independent | *http://sourceforge.net/projects/djangology/* |
| *Ellogon* | Manual annotation, machine annotation, project management | Language-independent | *http://www.ellogon.org/* |
| *Knowtator* | Manual annotation | Language-independent | *http://knowtator.sourceforge.net/index.shtml* |
| *MAE (Multipurpose Annotation Environment)* | Manual annotation | Language-independent | *http://code.google.com/p/mae-annotation/* |
| *SAPIENT: Semantic Annotation of Papers Interface and Enrichment Tool* | Manual semantic annotation of scientific concepts | Document Classification Text categorization English | *http://www.aber.ac.uk/en/cs/research/cb/projects/sapienta/software/* |

## 

## Measuring Annotator Reliability

It is essential to ascertain the level of reliability of your annotators when performing manual annotations. If your annotators are unreliable, their annotations will lead to a less accurate classifier. To ensure reliable annotations, it is important to select some subset of your data to be annotated in parallel by multiple human annotators. You can use this subset to measure inter-annotator reliability. We recommend using [Krippendorff’s alpha](https://en.wikipedia.org/wiki/Krippendorff%27s_alpha), a statistical inter-rater reliability metric, to measure reliability. Krippendorff’s alpha scores range from -1.0 to 1.0 with 1.0 indicating perfect agreement between annotators. A score of 0.0 indicates agreement no better than random chance (you can imagine this is the equivalent of your annotators picking their labels randomly out of a hat). A reliability score of 0.80 or greater is generally considered sufficient (though, the higher the better). Lower scores may potentially indicate issues with your data, your annotation guidelines, or your annotators’ understanding of the task. A low level of inter-annotator agreement will ultimately lead to a less accurate classifier, so it is recommended to repeatedly measure the reliability of your annotators until they achieve a satisfactory level of agreement. In cases where annotators disagree, these are usually good examples to include in your annotation guidelines, and it can be useful to have a discussion with your annotators as a group to reach a consensus.

# Training and Evaluating your Custom Model

We document the process of training and evaluating in the Categorizer Field Training Kit (FTK) user guide. Here we will provide some guidance about setting up your corpus in order to experiment with your data.

## Experimental Setup

The standard practice for supervised learning (where a statistical model learns from gold-standard, human annotated training data) is to split up, i.e., partition, your annotated data into parts:

1. Training set
2. Testing set
   1. Development set
   2. Evaluation set

The experimental motivation for splitting the data between training and testing sets is that if you evaluate a model on documents that it has seen during training, you are not measuring the expected performance on new, unseen data that the model will be expected to encounter in the real world. It can be a helpful sanity check to evaluate a model against the data it was trained on, but, you should expect the results to be near perfect, and highly inflated compared to expected real-world performance.

Ideally, each of the training and testing partitions maintains a balance amongst the different categories. So, assuming your corpus was balanced to begin with, you could maintain balance by dividing the data labeled with each category into 3 and then recombining these.

For example, we may have a small corpus with 300 documents annotated with the following food-related categories:

|  |  |
| --- | --- |
| **Category** | **Documents** |
| DESSERTS-AND-BAKING | 100 |
| DINING-OUT | 100 |
| FOOD-MOVEMENTS | 100 |

You might partition your corpus as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Category** | **Total** | **Training** | **Development** | **Evaluation** |
| DESSERTS-AND-BAKING | 100 | 60 | 20 | 20 |
| DINING-OUT | 100 | 60 | 20 | 20 |
| FOOD-MOVEMENTS | 100 | 60 | 20 | 20 |

While you are developing your classifier, you would experiment by training a classifier using the training set and testing the classifier against the development set. You might experiment with different features, different model parameters, or different learning strategies to determine what works best. You also might inspect the documents in the development set where your model is making incorrect predictions (this is called error analysis). The process of trying different strategies to optimize your model is called tuning—and specifically, you are tuning your model to perform well on the *development set*. Note, however, that you do *not* want to tune your classifier to the *evaluation set*.

When you have experimented thoroughly and are satisfied with your model’s results on the development set, you can then test against the evaluation set. Note that, experimentally, it is poor practice to look at the documents in the evaluation set once you’ve begun tuning. If you look at the evaluation set documents, you may end up deciding to make changes to your model that would improve its performance on the evaluation set. However, this is a potential pitfall, because if you tune your model specifically to the evaluation set, then, if you decide to run additional tuning experiments, you will have invalidated your evaluation set.

The purpose of the evaluation set is to be used as held-out data in order to validate that your model will generalize to new, unseen data. If you tune your classifier to the evaluation set, you will end up overestimating your classifier’s real-world performance. If you manually inspect the evaluation set, then, experimentally, the evaluation data is invalid. If you invalidate your evaluation data, you will need to come up with new, manually annotated, gold-standard evaluation data to hold-out. So, to avoid going back to annotate additional documents (which is expensive and time-consuming), it is important to keep your evaluation set separate and to avoid tuning your models specifically to that set of data. This will ensure that you have an experimentally valid measure of your model’s accuracy on unseen documents.

# Common Pitfalls

## Insufficient Data

When you are developing your corpus, it is not possible to know, a priori, how much data will be sufficient. Sometimes you will select a corpus and perform careful manual annotation and train a system only to discover that your classifier does not perform well. One way to assess if your classifier is hampered by a lack of data is to plot a so-called learning curve. To plot a learning curve, divide your training data set into parts. Then train your model on the first part and plot its performance with respect to size (number of documents) trained on, evaluating against a held out test set. Then, add the next part of the training set and plot the performance again, evaluating against the same held out test set. Generally, the trend of this plot should show an increase in performance as more data is added. If this is not the case, it usually indicates a problem with the annotations. If the trend of this plot increases, but levels off after a certain amount of training data has been added, then adding additional data is unlikely to help. If, however, the trend of the plot continues to increase without leveling off, it is recommended to add additional data to your training set until you reach a point where adding more data achieves diminishing returns.

## Imbalanced Data

If your training set is imbalanced, the categories that have a small number of examples relative to other classes may perform poorly. In this case, there are potentially two remedies:

1. Add more data to the categories until they match the more plentiful categories.
2. Downsample the more plentiful categories to match the category which has the smallest number of training documents.

Option 2 is only recommended in case you have ruled out that your overall corpus size is insufficiently large (see [the previous section](#_k4l91va244h9)).

## Duplicate Data

One thing to avoid in your corpus is the inclusion of duplicate documents or near-duplicate documents. Duplicate data can cause issues because the frequencies of different vocabulary terms may be skewed. Also a potential issue is if duplicates or near duplicates occur across your train-test partition. This is a form of “data leakage”—data in your testing set should never overlap with your training set. There are various tools for finding duplicate documents in a corpus. We recommend:

1. Rosette TF-IDF deduplicator (part of the Rosette Entity Extractor Field Training Kit)
2. [Onion](http://corpus.tools/wiki/Onion)

## Unclean Data

An ideal corpus includes only documents whose contents are clean, plain-text. We list some undesirable qualities of your data that can have an impact on your classifier here.

### Badly encoded data

Most plain-text documents you will find are encoded as UTF-8, however, sometimes you will find data encoded with other encodings. We recommend converting all your data to UTF-8. Some recommended tools for detecting and converting character encodings:

1. [chardet](https://pypi.org/project/chardet/)
2. [file](http://pubs.opengroup.org/onlinepubs/9699919799/utilities/file.html)
3. [iconv](https://www.gnu.org/savannah-checkouts/gnu/libiconv/documentation/libiconv-1.15/iconv.1.html)

### Markup

If your documents’ text are embedded within markup (such as HTML, XML, Markdown, etc.), then you will need to extract the plain-text. Some suggested tools that attempt to automatically extracting text from marked up documents are:

1. [Boilerpipe](https://github.com/kohlschutter/boilerpipe) (there is a web-app wrapper [here](https://boilerpipe-web.appspot.com/))
2. [Apache Tika](https://tika.apache.org/)

If you have documents marked up with a particularly cumbersome markup language, you may consider converting it to a friendlier format. One particularly useful tool for markup conversion is [Pandoc](http://pandoc.org/).

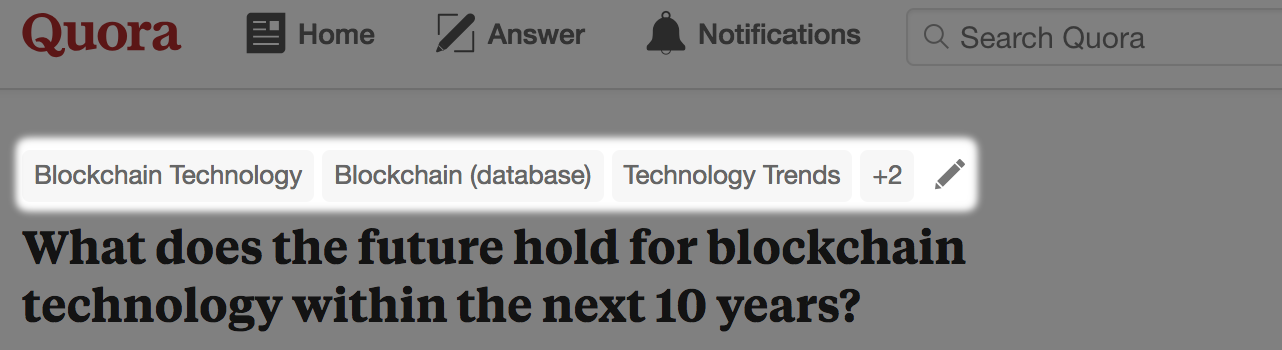
Sometimes you will have to write your own software to extract text. You should research what tools your go-to programming language offers—most programming languages will have libraries for parsing markup of various kinds. For example, Python’s standard library offers [several tools](https://docs.python.org/3.6/library/markup.html) for working with HTML, XML, and other similar markup languages. Some useful non-standard Python tools for working with various markup formats include:

1. [Html2text](https://pypi.org/project/html2text/)
2. [Beautiful Soup](https://www.crummy.com/software/BeautifulSoup/)

### Metadata

Some documents, especially those collected from structured websites, may include text that is not part of the document body, such as titles, headers, footers, site map information, or other metadata. At best, such metadata can just be noise, essentially artificially over-representing the frequencies of certain terms in your corpus. At worst, the metadata can cause your classifier to learn to classify documents based on the metadata itself, rather than the document content. For example, if you collect a set of documents from a website where the category label of the document is present in the metadata, then your classifier will learn from this signal since it will be a perfect predictor of the category.

For instance, on the website [Quora](https://www.quora.com/), documents are tagged with categories. If you collected a corpus from Quora, you could use these tags in the metadata as if they are category annotations (which might save you from having to annotate the documents yourself). However, you must ensure that you don’t include these category tags in the document text! For example, here’s a Quora page about Blockchain technology that has been tagged:



If every document in your corpus that belongs to the ‘Blockchain Technology’ category has the phrase ‘Blockchain Technology’ in it, then your classifier is simply going to learn that any document with those terms belongs to the ‘Blockchain Technology’ category. This is called ‘overfitting’—your classifier ends up fitting exactly to the training data and it won’t generalize to real world data. If a document body happens to mention ‘block chain’ (two words), or ‘cryptographically hashed, distributed transaction ledgers’, these terms are relevant to the category for ‘Blockchain Technology’. But, if your classifier learns explicitly from the category tags, and not from the document body, these signals would be ignored. If your classifier is ever asked to classify a document that hasn’t been pre-tagged with Quora tags, it will not have that signal to rely on, so it will be at a major disadvantage. Additionally, if the documents in your test set have these tags, then the difficulty of classifying those documents will be artificially reduced to the point of triviality—all the classifier will need to do is emit the labels for the tags in the test documents. This is not a simulation of a real-world classification problem (most documents in the real world will not have been annotated with category information like Quora), so your evaluation scores will be artificially high, overestimating the performance of your classifier. All this is to say, make sure that you take care to clean your documents and carefully consider what is actually part of the document body text—all document metadata should be separated as a preprocess.

1. Pustejovsky, J., & Stubbs, A. (2012). Natural Language Annotation for Machine Learning: A guide to corpus-building for applications. "O'Reilly Media, Inc.". [↑](#footnote-ref-1)