Chapter 4

Linguistic applications of classification

Having covered several techniques for classification, this chapter shifts the focus from mathematics to linguistic applications. Later in the chapter, we will consider the design decisions involved in text classification, as well as best practices for evaluation.

4.1 Sentiment and opinion analysis

A popular application of text classification is to automatically determine the **sentiment** or **opinion polarity** of documents such as product reviews and social media posts. For example, marketers are interested to know how people respond to advertisements, services, and products (Hu and Liu, 2004); social scientists are interested in how emotions are affected by phenomena such as the weather (Hannak et al., 2012), and how both opinions and emotions spread over social networks (Coviello et al., 2014; Miller et al., 2011). In the field of **digital humanities**, literary scholars track plot structures through the flow of sentiment across a novel (Jockers, 2015).[1]

Sentiment analysis can be framed as a direct application of document classification, assuming reliable labels can be obtained. In the simplest case, sentiment analysis is a two or three-class problem, with sentiments of POSITIVE, NEGATIVE, and possibly NEUTRAL. Such annotations could be annotated by hand, or obtained automatically through a variety of means:

- Tweets containing happy emoticons can be marked as positive, sad emoticons as negative (Read, 2005; Pak and Paroubek, 2010).

[1] Comprehensive surveys on sentiment analysis and related problems are offered by Pang and Lee (2008) and Liu (2015).

- Reviews with four or more stars can be marked as positive, three or fewer stars as negative (Pang et al., 2002).
- Statements from politicians who are voting for a given bill are marked as positive (towards that bill); statements from politicians voting against the bill are marked as negative (Thomas et al., 2006).

The bag-of-words model is a good fit for sentiment analysis at the document level: if the document is long enough, we would expect the words associated with its true sentiment to overwhelm the others. Indeed, **lexicon-based sentiment analysis** avoids machine learning altogether, and classifies documents by counting words against positive and negative sentiment word lists (Taboada et al., 2011).

Lexicon-based classification is less effective for short documents, such as single-sentence reviews or social media posts. In these documents, linguistic issues like **negation** and **irrealis** (Polanyi and Zaenen, 2006) — events that are hypothetical or otherwise non-factual — can make bag-of-words classification ineffective. Consider the following examples:

(4.1) a. That's not bad for the first day.
b. This is not the worst thing that can happen.
c. It would be nice if you acted like you understood.
d. There is no reason at all to believe that the polluters are suddenly going to become reasonable. (Wilson et al., 2005)
e. This film should be brilliant. The actors are first grade. Stallone plays a happy, wonderful man. His sweet wife is beautiful and adores him. He has a fascinating gift for living life fully. It sounds like a great plot, **however**, the film is a failure. (Pang et al., 2002)

A minimal solution is to move from a bag-of-words model to a bag-of-**bigrams** model, where each base feature is a pair of adjacent words, e.g.,

$$(\textit{that's}, \textit{not}), (\textit{not}, \textit{bad}), (\textit{bad}, \textit{for}), \ldots \quad [4.1]$$

Bigrams can handle relatively straightforward cases, such as when an adjective is immediately negated; trigrams would be required to extend to larger contexts (e.g., *not the worst*). But this approach will not scale to more complex examples like (4.1d) and (4.1e). More sophisticated solutions try to account for the syntactic structure of the sentence (Wilson et al., 2005; Socher et al., 2013), or apply more complex classifiers such as convolutional neural networks (Kim, 2014), which are described in chapter 3.

4.1.1 Related problems

Subjectivity Closely related to sentiment analysis is **subjectivity detection**, which requires identifying the parts of a text that express subjective opinions, as well as other non-

factual content such as speculation and hypotheticals (Riloff and Wiebe, 2003). This can be done by treating each sentence as a separate document, and then applying a bag-of-words classifier: indeed, Pang and Lee (2004) do exactly this, using a training set consisting of (mostly) subjective sentences gathered from movie reviews, and (mostly) objective sentences gathered from plot descriptions. They augment this bag-of-words model with a graph-based algorithm that encourages nearby sentences to have the same subjectivity label.

Stance classification In debates, each participant takes a side: for example, advocating for or against proposals like adopting a vegetarian lifestyle or mandating free college education. The problem of stance classification is to identify the author's position from the text of the argument. In some cases, there is training data available for each position, so that standard document classification techniques can be employed. In other cases, it suffices to classify each document as whether it is in support or opposition of the argument advanced by a previous document (Anand et al., 2011). In the most challenging case, there is no labeled data for any of the stances, so the only possibility is group documents that advocate the same position (Somasundaran and Wiebe, 2009). This is a form of **unsupervised learning**, discussed in chapter 5.

Targeted sentiment analysis The expression of sentiment is often more nuanced than a simple binary label. Consider the following examples:

(4.2) a. The vodka was good, but the meat was rotten.
b. Go to Heaven for the climate, Hell for the company. –*Mark Twain*

These statements display a mixed overall sentiment: positive towards some entities (e.g., *the vodka*), negative towards others (e.g., *the meat*). **Targeted sentiment analysis** seeks to identify the writer's sentiment towards specific entities (Jiang et al., 2011). This requires identifying the entities in the text and linking them to specific sentiment words — much more than we can do with the classification-based approaches discussed thus far. For example, Kim and Hovy (2006) analyze sentence-internal structure to determine the topic of each sentiment expression.

Aspect-based opinion mining seeks to identify the sentiment of the author of a review towards predefined aspects such as PRICE and SERVICE, or, in the case of (4.2b), CLIMATE and COMPANY (Hu and Liu, 2004). If the aspects are not defined in advance, it may again be necessary to employ unsupervised learning methods to identify them (e.g., Branavan et al., 2009).

Emotion classification While sentiment analysis is framed in terms of positive and negative categories, psychologists generally regard **emotion** as more multifaceted. For example, Ekman (1992) argues that there are six basic emotions — happiness, surprise, fear,

sadness, anger, and contempt — and that they are universal across human cultures. Alm et al. (2005) build a linear classifier for recognizing the emotions expressed in children's stories. The ultimate goal of this work was to improve text-to-speech synthesis, so that stories could be read with intonation that reflected the emotional content. They used bag-of-words features, as well as features capturing the story type (e.g., jokes, folktales), and structural features that reflect the position of each sentence in the story. The task is difficult: even human annotators frequently disagreed with each other, and the best classifiers achieved accuracy between 60-70%.

4.1.2 Alternative approaches to sentiment analysis

Regression A more challenging version of sentiment analysis is to determine not just the class of a document, but its rating on a numerical scale (Pang and Lee, 2005). If the scale is continuous, it is most natural to apply **regression**, identifying a set of weights $\boldsymbol{\theta}$ that minimize the squared error of a predictor $\hat{y} = \boldsymbol{\theta} \cdot \boldsymbol{x} + b$, where b is an offset. This approach is called **linear regression**, and sometimes **least squares**, because the regression coefficients $\boldsymbol{\theta}$ are determined by minimizing the squared error, $(y - \hat{y})^2$. If the weights are regularized using a penalty $\lambda||\boldsymbol{\theta}||_2^2$, then it is **ridge regression**. Unlike logistic regression, both linear regression and ridge regression can be solved in closed form as a system of linear equations.

Ordinal ranking In many problems, the labels are ordered but discrete: for example, product reviews are often integers on a scale of $1 - 5$, and grades are on a scale of $A - F$. Such problems can be solved by discretizing the score $\boldsymbol{\theta} \cdot \boldsymbol{x}$ into "ranks",

$$\hat{y} = \underset{r:\, \boldsymbol{\theta}\cdot\boldsymbol{x} \geq b_r}{\operatorname{argmax}}\, r, \qquad [4.2]$$

where $\boldsymbol{b} = [b_1 = -\infty, b_2, b_3, \ldots, b_K]$ is a vector of boundaries. It is possible to learn the weights and boundaries simultaneously, using a perceptron-like algorithm (Crammer and Singer, 2001).

Lexicon-based classification Sentiment analysis is one of the only NLP tasks where hand-crafted feature weights are still widely employed. In **lexicon-based classification** (Taboada et al., 2011), the user creates a list of words for each label, and then classifies each document based on how many of the words from each list are present. In our linear classification framework, this is equivalent to choosing the following weights:

$$\theta_{y,j} = \begin{cases} 1, & j \in \mathcal{L}_y \\ 0, & \text{otherwise}, \end{cases} \qquad [4.3]$$

where $\mathcal{L}_y$ is the lexicon for label y. Compared to the machine learning classifiers discussed in the previous chapters, lexicon-based classification may seem primitive. However, supervised machine learning relies on large annotated datasets, which are time-consuming and expensive to produce. If the goal is to distinguish two or more categories in a new domain, it may be simpler to start by writing down a list of words for each category.

An early lexicon was the *General Inquirer* (Stone, 1966). Today, popular sentiment lexicons include SENTIWORDNET (Esuli and Sebastiani, 2006) and an evolving set of lexicons from Liu (2015). For emotions and more fine-grained analysis, *Linguistic Inquiry and Word Count* (LIWC) provides a set of lexicons (Tausczik and Pennebaker, 2010). The MPQA lexicon indicates the polarity (positive or negative) of 8221 terms, as well as whether they are strongly or weakly subjective (Wiebe et al., 2005). A comprehensive comparison of sentiment lexicons is offered by Ribeiro et al. (2016). Given an initial **seed lexicon**, it is possible to automatically expand the lexicon by looking for words that frequently co-occur with words in the seed set (Hatzivassiloglou and McKeown, 1997; Qiu et al., 2011).

4.2 Word sense disambiguation

Consider the the following headlines:

(4.3) a. Iraqi head seeks arms
b. Prostitutes appeal to Pope
c. Drunk gets nine years in violin case[2]

These headlines are ambiguous because they contain words that have multiple meanings, or **senses**. Word sense disambiguation is the problem of identifying the intended sense of each word token in a document. Word sense disambiguation is part of a larger field of research called **lexical semantics**, which is concerned with meanings of the words.

At a basic level, the problem of word sense disambiguation is to identify the correct sense for each word token in a document. Part-of-speech ambiguity (e.g., noun versus verb) is usually considered to be a different problem, to be solved at an earlier stage. From a linguistic perspective, senses are not properties of words, but of **lemmas**, which are canonical forms that stand in for a set of inflected words. For example, *arm*/N is a lemma that includes the inflected form *arms*/N — the /N indicates that it we are referring to the noun, and not its **homonym** *arm*/V, which is another lemma that includes the inflected verbs (*arm*/V, *arms*/V, *armed*/V, *arming*/V). Therefore, word sense disambiguation requires first identifying the correct part-of-speech and lemma for each token,

[2]These examples, and many more, can be found at `http://www.ling.upenn.edu/~beatrice/humor/headlines.html`

and then choosing the correct sense from the inventory associated with the corresponding lemma.[3] (Part-of-speech tagging is discussed in § 8.1.)

4.2.1 How many word senses?

Words sometimes have many more than two senses, as exemplified by the word *serve*:

- [FUNCTION]: *The tree stump served as a table*
- [CONTRIBUTE TO]: *His evasive replies only served to heighten suspicion*
- [PROVIDE]: *We serve only the rawest fish*
- [ENLIST]: *She served in an elite combat unit*
- [JAIL]: *He served six years for a crime he didn't commit*
- [LEGAL]: *They were served with subpoenas*[4]

These sense distinctions are annotated in **WORDNET** (`http://wordnet.princeton.edu`), a lexical semantic database for English. WORDNET consists of roughly 100,000 **synsets**, which are groups of lemmas (or phrases) that are synonymous. An example synset is $\{chump^1, fool^2, sucker^1, mark^9\}$, where the superscripts index the sense of each lemma that is included in the synset: for example, there are at least eight other senses of *mark* that have different meanings, and are not part of this synset. A lemma is **polysemous** if it participates in multiple synsets.

WORDNET defines the scope of the word sense disambiguation problem, and, more generally, formalizes lexical semantic knowledge of English. (WordNets have been created for a few dozen other languages, at varying levels of detail.) Some have argued that WordNet's sense granularity is too fine (Ide and Wilks, 2006); more fundamentally, the premise that word senses can be differentiated in a task-neutral way has been criticized as linguistically naïve (Kilgarriff, 1997). One way of testing this question is to ask whether people tend to agree on the appropriate sense for example sentences: according to Mihalcea et al. (2004), people agree on roughly 70% of examples using WordNet senses; far better than chance, but less than agreement on other tasks, such as sentiment annotation (Wilson et al., 2005).

***Other lexical semantic relations** Besides **synonymy**, WordNet also describes many other lexical semantic relationships, including:

- **antonymy**: x means the opposite of y, e.g. FRIEND-ENEMY;

[3] Navigli (2009) provides a survey of approaches for word-sense disambiguation.

[4] Several of the examples are adapted from WORDNET (Fellbaum, 2010).

- **hyponymy**: x is a special case of y, e.g. RED-COLOR; the inverse relationship is **hypernymy**;
- **meronymy**: x is a part of y, e.g., WHEEL-BICYCLE; the inverse relationship is **holonymy**.

Classification of these relations can be performed by searching for characteristic patterns between pairs of words, e.g., *X, such as Y*, which signals hyponymy (Hearst, 1992), or *X but Y*, which signals antonymy (Hatzivassiloglou and McKeown, 1997). Another approach is to analyze each term's **distributional statistics** (the frequency of its neighboring words). Such approaches are described in detail in chapter 14.

4.2.2 Word sense disambiguation as classification

How can we tell living *plants* from manufacturing *plants*? The context is often critical:

(4.4) a. Town officials are hoping to attract new manufacturing plants through weakened environmental regulations.
b. The endangered plants play an important role in the local ecosystem.

It is possible to build a feature vector using the bag-of-words representation, by treating each context as a pseudo-document. The feature function is then,

$$f((\textit{plant}, \textit{The endangered plants play an} \ldots), y) = \\ \{(\textit{the}, y) : 1, (\textit{endangered}, y) : 1, (\textit{play}, y) : 1, (\textit{an}, y) : 1, \ldots\}$$

As in document classification, many of these features are irrelevant, but a few are very strong predictors. In this example, the context word *endangered* is a strong signal that the intended sense is biology rather than manufacturing. We would therefore expect a learning algorithm to assign high weight to $(\textit{endangered}, \text{BIOLOGY})$, and low weight to $(\textit{endangered}, \text{MANUFACTURING})$.[5]

It may also be helpful to go beyond the bag-of-words: for example, one might encode the position of each context word with respect to the target, e.g.,

$$f((\textit{bank}, \textit{I went to the bank to deposit my paycheck}), y) = \\ \{(i-3, \textit{went}, y) : 1, (i+2, \textit{deposit}, y) : 1, (i+4, \textit{paycheck}, y) : 1\}$$

These are called **collocation features**, and they give more information about the specific role played by each context word. This idea can be taken further by incorporating additional syntactic information about the grammatical role played by each context feature, such as the **dependency path** (see chapter 11).

[5]The context bag-of-words can be also used be used to perform word-sense disambiguation without machine learning: the Lesk (1986) algorithm selects the word sense whose dictionary definition best overlaps the local context.

Using such features, a classifier can be trained from labeled data. A **semantic concordance** is a corpus in which each open-class word (nouns, verbs, adjectives, and adverbs) is tagged with its word sense from the target dictionary or thesaurus. SemCor is a semantic concordance built from 234K tokens of the Brown corpus (Francis and Kucera, 1982), annotated as part of the WORDNET project (Fellbaum, 2010). SemCor annotations look like this:

(4.5) As of Sunday$^{1}_{N}$ night$^{1}_{N}$ there was$^{4}_{V}$ no word$^{2}_{N}$ ...,

with the superscripts indicating the annotated sense of each polysemous word, and the subscripts indicating the part-of-speech.

As always, supervised classification is only possible if enough labeled examples can be accumulated. This is difficult in word sense disambiguation, because each polysemous lemma requires its own training set: having a good classifier for the senses of *serve* is no help towards disambiguating *plant*. For this reason, unsupervised and **semi-supervised** methods are particularly important for word sense disambiguation (e.g., Yarowsky, 1995). These methods will be discussed in chapter 5. Unsupervised methods typically lean on the heuristic of "one sense per discourse", which means that a lemma will usually have a single, consistent sense throughout any given document (Gale et al., 1992). Based on this heuristic, we can propagate information from high-confidence instances to lower-confidence instances in the same document (Yarowsky, 1995). Semi-supervised methods combine labeled and unlabeled data, and are discussed in more detail in chapter 5.

4.3 Design decisions for text classification

Text classification involves a number of design decisions. In some cases, the design decision is clear from the mathematics: if you are using regularization, then a regularization weight λ must be chosen. Other decisions are more subtle, arising only in the low level "plumbing" code that ingests and processes the raw data. Such decision can be surprisingly consequential for classification accuracy.

4.3.1 What is a word?

The bag-of-words representation presupposes that extracting a vector of word counts from text is unambiguous. But text documents are generally represented as a sequences of characters (in an encoding such as ascii or unicode), and the conversion to bag-of-words presupposes a definition of the "words" that are to be counted.

Whitespace	Isn't	Ahab,	Ahab?	;)				
Treebank	Is	n't	Ahab	,	Ahab	?	;	)
Tweet	Isn't	Ahab	,	Ahab	?	;)		
TokTok (Dehdari, 2014)	Isn	'	t	Ahab	,	Ahab	?	;)

Figure 4.1: The output of four NLTK tokenizers, applied to the string *Isn't Ahab, Ahab? ;)*

Tokenization

The first subtask for constructing a bag-of-words vector is **tokenization**: converting the text from a sequence of characters to a sequence of **word!tokens**. A simple approach is to define a subset of characters as whitespace, and then split the text on these tokens. However, whitespace-based tokenization is not ideal: we may want to split conjunctions like *isn't* and hyphenated phrases like *prize-winning* and *half-asleep*, and we likely want to separate words from commas and periods that immediately follow them. At the same time, it would be better not to split abbreviations like *U.S.* and *Ph.D.* In languages with Roman scripts, tokenization is typically performed using regular expressions, with modules designed to handle each of these cases. For example, the NLTK package includes a number of tokenizers (Loper and Bird, 2002); the outputs of four of the better-known tokenizers are shown in Figure 4.1. Social media researchers have found that emoticons and other forms of orthographic variation pose new challenges for tokenization, leading to the development of special purpose tokenizers to handle these phenomena (O'Connor et al., 2010).

Tokenization is a language-specific problem, and each language poses unique challenges. For example, Chinese does not include spaces between words, nor any other consistent orthographic markers of word boundaries. A "greedy" approach is to scan the input for character substrings that are in a predefined lexicon. However, Xue et al. (2003) notes that this can be ambiguous, since many character sequences could be segmented in multiple ways. Instead, he trains a classifier to determine whether each Chinese character, or **hanzi**, is a word boundary. More advanced sequence labeling methods for word segmentation are discussed in § 8.4. Similar problems can occur in languages with alphabetic scripts, such as German, which does not include whitespace in compound nouns, yielding examples such as *Freundschaftsbezeigungen* (demonstration of friendship) and *Dilettantenaufdringlichkeiten* (the importunities of dilettantes). As Twain (1997) argues, "*These things are not words, they are alphabetic processions.*" Social media raises similar problems for English and other languages, with hashtags such as *#TrueLoveInFourWords* requiring decomposition for analysis (Brun and Roux, 2014).

Original	The	Williams	sisters	are	leaving	this	tennis	centre
Porter stemmer	the	william	sister	are	leav	thi	tenni	centr
Lancaster stemmer	the	william	sist	ar	leav	thi	ten	cent
WordNet lemmatizer	The	Williams	**sister**	are	leaving	this	tennis	centre

Figure 4.2: Sample outputs of the Porter (1980) and Lancaster (Paice, 1990) stemmers, and the WORDNET lemmatizer

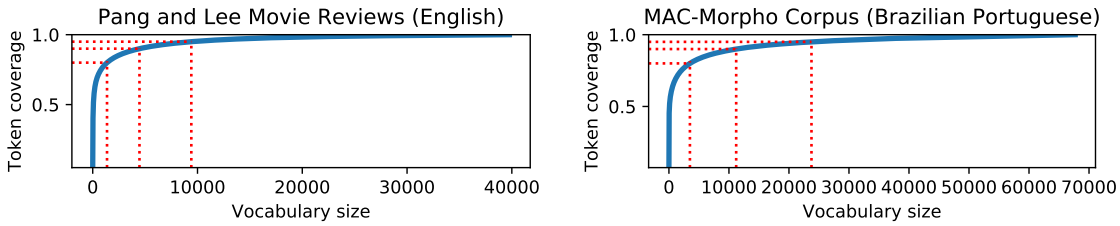
Text normalization

After splitting the text into tokens, the next question is which tokens are really distinct. Is it necessary to distinguish *great*, *Great*, and *GREAT*? Sentence-initial capitalization may be irrelevant to the classification task. Going further, the complete elimination of case distinctions will result in a smaller vocabulary, and thus smaller feature vectors. However, case distinctions might be relevant in some situations: for example, *apple* is a delicious pie filling, while *Apple* is a company that specializes in proprietary dongles and power adapters.

For Roman script, case conversion can be performed using unicode string libraries. Many scripts do not have case distinctions (e.g., the Devanagari script used for South Asian languages, the Thai alphabet, and Japanese kana), and case conversion for all scripts may not be available in every programming environment. (Unicode support is an important distinction between Python's versions 2 and 3, and is a good reason for migrating to Python 3 if you have not already done so. Compare the output of the code `"\à l\'hôtel".upper()` in the two language versions.)

Case conversion is a type of **text normalization**, which refers to string transformations that remove distinctions that are irrelevant to downstream applications (Sproat et al., 2001). Other forms of normalization include the standardization of numbers (e.g., *1,000* to *1000*) and dates (e.g., *August 11, 2015* to *2015/11/08*). Depending on the application, it may even be worthwhile to convert all numbers and dates to special tokens, `!NUM` and `!DATE`. In social media, there are additional orthographic phenomena that may be normalized, such as expressive lengthening, e.g., *cooooool* (Aw et al., 2006; Yang and Eisenstein, 2013). Similarly, historical texts feature spelling variations that may need to be normalized to a contemporary standard form (Baron and Rayson, 2008).

A more extreme form of normalization is to eliminate **inflectional affixes**, such as the *-ed* and *-s* suffixes in English. On this view, *whale*, *whales*, and *whaling* all refer to the same underlying concept, so they should be grouped into a single feature. A **stemmer** is a program for eliminating affixes, usually by applying a series of regular expression substitutions. Character-based stemming algorithms are necessarily approximate, as shown in Figure 4.2: the Lancaster stemmer incorrectly identifies *-ers* as an inflectional suffix of



(a) Movie review data in English

(b) News articles in Brazilian Portuguese

Figure 4.3: Tradeoff between token coverage (y-axis) and vocabulary size, on the NLTK movie review dataset, after sorting the vocabulary by decreasing frequency. The red dashed lines indicate 80%, 90%, and 95% coverage.

sisters (by analogy to *fix/fixers*), and both stemmers incorrectly identify *-s* as a suffix of *this* and *Williams*. Fortunately, even inaccurate stemming can improve bag-of-words classification models, by merging related strings and thereby reducing the vocabulary size.

Accurately handling irregular orthography requires word-specific rules. **Lemmatizers** are systems that identify the underlying lemma of a given wordform. They must avoid the over-generalization errors of the stemmers in Figure 4.2, and also handle more complex transformations, such as *geese→goose*. The output of the WordNet lemmatizer is shown in the final line of Figure 4.2. Both stemming and lemmatization are language-specific: an English stemmer or lemmatizer is of little use on a text written in another language. The discipline of **morphology** relates to the study of word-internal structure, and is described in more detail in § 9.1.2.

The value of normalization depends on the data and the task. Normalization reduces the size of the feature space, which can help in generalization. However, there is always the risk of merging away linguistically meaningful distinctions. In supervised machine learning, regularization and smoothing can play a similar role to normalization — preventing the learner from overfitting to rare features — while avoiding the language-specific engineering required for accurate normalization. In unsupervised scenarios, such as content-based information retrieval (Manning et al., 2008) and topic modeling (Blei et al., 2003), normalization is more critical.

4.3.2 How many words?

Limiting the size of the feature vector reduces the memory footprint of the resulting models, and increases the speed of prediction. Normalization can help to play this role, but a more direct approach is simply to limit the vocabulary to the N most frequent words in the dataset. For example, in the MOVIE-REVIEWS dataset provided with NLTK (originally from Pang et al., 2002), there are 39,768 word types, and 1.58M tokens. As shown

in Figure 4.3a, the most frequent 4000 word types cover 90% of all tokens, offering an order-of-magnitude reduction in the model size. Such ratios are language-specific: in for example, in the Brazilian Portuguese Mac-Morpho corpus (Aluísio et al., 2003), attaining 90% coverage requires more than 10000 word types (Figure 4.3b). This reflects the morphological complexity of Portuguese, which includes many more inflectional suffixes than English.

Eliminating rare words is not always advantageous for classification performance: for example, names, which are typically rare, play a large role in distinguishing topics of news articles. Another way to reduce the size of the feature space is to eliminate **stopwords** such as *the*, *to*, and *and*, which may seem to play little role in expressing the topic, sentiment, or stance. This is typically done by creating a **stoplist** (e.g., NLTK.CORPUS.STOPWORDS), and then ignoring all terms that match the list. However, corpus linguists and social psychologists have shown that seemingly inconsequential words can offer surprising insights about the author or nature of the text (Biber, 1991; Chung and Pennebaker, 2007). Furthermore, high-frequency words are unlikely to cause overfitting in discriminative classifiers. As with normalization, stopword filtering is more important for unsupervised problems, such as term-based document retrieval.

Another alternative for controlling model size is **feature hashing** (Weinberger et al., 2009). Each feature is assigned an index using a hash function. If a hash function that permits collisions is chosen (typically by taking the hash output modulo some integer), then the model can be made arbitrarily small, as multiple features share a single weight. Because most features are rare, accuracy is surprisingly robust to such collisions (Ganchev and Dredze, 2008).

4.3.3 Count or binary?

Finally, we may consider whether we want our feature vector to include the *count* of each word, or its *presence*. This gets at a subtle limitation of linear classification: it's worse to have two *failure*s than one, but is it really twice as bad? Motivated by this intuition, Pang et al. (2002) use binary indicators of presence or absence in the feature vector: $f_j(\boldsymbol{x}, y) \in \{0, 1\}$. They find that classifiers trained on these binary vectors tend to outperform feature vectors based on word counts. One explanation is that words tend to appear in clumps: if a word has appeared once in a document, it is likely to appear again (Church, 2000). These subsequent appearances can be attributed to this tendency towards repetition, and thus provide little additional information about the class label of the document.

4.4 Evaluating classifiers

In any supervised machine learning application, it is critical to reserve a held-out test set. This data should be used for only one purpose: to evaluate the overall accuracy of a single

classifier. Using this data more than once would cause the estimated accuracy to be overly optimistic, because the classifier would be customized to this data, and would not perform as well as on unseen data in the future. It is usually necessary to set hyperparameters or perform feature selection, so you may need to construct a **tuning** or **development set** for this purpose, as discussed in § 2.2.5.

There are a number of ways to evaluate classifier performance. The simplest is **accuracy**: the number of correct predictions, divided by the total number of instances,

$$\text{acc}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N} \sum_{i}^{N} \delta(y^{(i)} = \hat{y}). \quad [4.4]$$

Exams are usually graded by accuracy. Why are other metrics necessary? The main reason is **class imbalance**. Suppose you are building a classifier to detect whether an electronic health record (EHR) describes symptoms of a rare disease, which appears in only 1% of all documents in the dataset. A classifier that reports $\hat{y} = \text{NEGATIVE}$ for all documents would achieve 99% accuracy, but would be practically useless. We need metrics that are capable of detecting the classifier's ability to discriminate between classes, even when the distribution is skewed.

One solution is to build a **balanced test set**, in which each possible label is equally represented. But in the EHR example, this would mean throwing away 98% of the original dataset! Furthermore, the detection threshold itself might be a design consideration: in health-related applications, we might prefer a very sensitive classifier, which returned a positive prediction if there is even a small chance that $y^{(i)} = \text{POSITIVE}$. In other applications, a positive result might trigger a costly action, so we would prefer a classifier that only makes positive predictions when absolutely certain. We need additional metrics to capture these characteristics.

4.4.1 Precision, recall, and F-MEASURE

For any label (e.g., positive for presence of symptoms of a disease), there are two possible errors:

- **False positive**: the system incorrectly predicts the label.
- **False negative**: the system incorrectly fails to predict the label.

Similarly, for any label, there are two ways to be correct:

- **True positive**: the system correctly predicts the label.
- **True negative**: the system correctly predicts that the label does not apply to this instance.

Classifiers that make a lot of false positives have low **precision**: they predict the label even when it isn't there. Classifiers that make a lot of false negatives have low **recall**: they fail to predict the label, even when it is there. These metrics distinguish these two sources of error, and are defined formally as:

$$\text{RECALL}(\boldsymbol{y}, \hat{\boldsymbol{y}}, k) = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad [4.5]$$

$$\text{PRECISION}(\boldsymbol{y}, \hat{\boldsymbol{y}}, k) = \frac{\text{TP}}{\text{TP} + \text{FP}}. \quad [4.6]$$

Recall and precision are both conditional likelihoods of a correct prediction, which is why their numerators are the same. Recall is conditioned on k being the correct label, $y^{(i)} = k$, so the denominator sums over true positive and false negatives. Precision is conditioned on k being the prediction, so the denominator sums over true positives and false positives. Note that true negatives are not considered in either statistic. The classifier that labels every document as "negative" would achieve zero recall; precision would be $\frac{0}{0}$.

Recall and precision are complementary. A high-recall classifier is preferred when false positives are cheaper than false negatives: for example, in a preliminary screening for symptoms of a disease, the cost of a false positive might be an additional test, while a false negative would result in the disease going untreated. Conversely, a high-precision classifier is preferred when false positives are more expensive: for example, in spam detection, a false negative is a relatively minor inconvenience, while a false positive might mean that an important message goes unread.

The F**-MEASURE** combines recall and precision into a single metric, using the harmonic mean:

$$F\text{-MEASURE}(\boldsymbol{y}, \hat{\boldsymbol{y}}, k) = \frac{2rp}{r + p}, \quad [4.7]$$

where r is recall and p is precision.[6]

Evaluating multi-class classification Recall, precision, and F-MEASURE are defined with respect to a specific label k. When there are multiple labels of interest (e.g., in word sense disambiguation or emotion classification), it is necessary to combine the F-MEASURE across each class. **Macro** F**-MEASURE** is the average F-MEASURE across several classes,

$$\text{Macro-}F(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} F\text{-MEASURE}(\boldsymbol{y}, \hat{\boldsymbol{y}}, k) \quad [4.8]$$

[6] F-MEASURE is sometimes called F_1, and generalizes to $F_\beta = \frac{(1+\beta^2)rp}{\beta^2 p + r}$. The β parameter can be tuned to emphasize recall or precision.

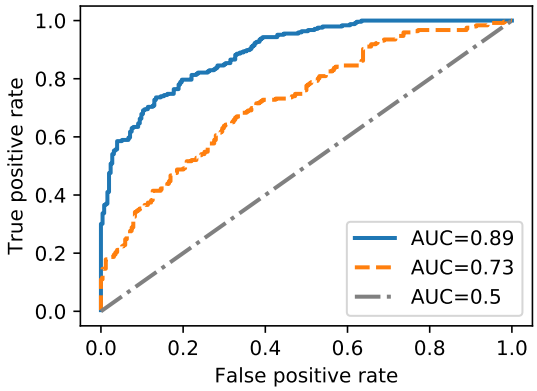


Figure 4.4: ROC curves for three classifiers of varying discriminative power, measured by AUC (area under the curve)

In multi-class problems with unbalanced class distributions, the macro F-MEASURE is a balanced measure of how well the classifier recognizes each class. In **micro F-MEASURE**, we compute true positives, false positives, and false negatives for each class, and then add them up to compute a single recall, precision, and F-MEASURE. This metric is balanced across instances rather than classes, so it weights each class in proportion to its frequency — unlike macro F-MEASURE, which weights each class equally.

4.4.2 Threshold-free metrics

In binary classification problems, it is possible to trade off between recall and precision by adding a constant "threshold" to the output of the scoring function. This makes it possible to trace out a curve, where each point indicates the performance at a single threshold. In the **receiver operating characteristic (ROC)** curve,[7] the x-axis indicates the **false positive rate**, $\frac{\text{FP}}{\text{FP+TN}}$, and the y-axis indicates the recall, or **true positive rate**. A perfect classifier attains perfect recall without any false positives, tracing a "curve" from the origin (0,0) to the upper left corner (0,1), and then to (1,1). In expectation, a non-discriminative classifier traces a diagonal line from the origin (0,0) to the upper right corner (1,1). Real classifiers tend to fall between these two extremes. Examples are shown in Figure 4.4.

The ROC curve can be summarized in a single number by taking its integral, the **area under the curve (AUC)**. The AUC can be interpreted as the probability that a randomly-selected positive example will be assigned a higher score by the classifier than a randomly-

[7] The name "receiver operator characteristic" comes from the metric's origin in signal processing applications (Peterson et al., 1954). Other threshold-free metrics include **precision-recall curves**, **precision-at-k**, and **balanced F-MEASURE**; see Manning et al. (2008) for more details.

selected negative example. A perfect classifier has $\text{AUC} = 1$ (all positive examples score higher than all negative examples); a non-discriminative classifier has $\text{AUC} = 0.5$ (given a randomly selected positive and negative example, either could score higher with equal probability); a perfectly wrong classifier would have $\text{AUC} = 0$ (all negative examples score higher than all positive examples). One advantage of AUC in comparison to F-MEASURE is that the baseline rate of 0.5 does not depend on the label distribution.

4.4.3 Classifier comparison and statistical significance

Natural language processing research and engineering often involves comparing different classification techniques. In some cases, the comparison is between algorithms, such as logistic regression versus averaged perceptron, or L_2 regularization versus L_1. In other cases, the comparison is between feature sets, such as the bag-of-words versus positional bag-of-words (see § 4.2.2). **Ablation testing** involves systematically removing (ablating) various aspects of the classifier, such as feature groups, and testing the **null hypothesis** that the ablated classifier is as good as the full model.

A full treatment of hypothesis testing is beyond the scope of this text, but this section contains a brief summary of the techniques necessary to compare classifiers. The main aim of hypothesis testing is to determine whether the difference between two statistics — for example, the accuracies of two classifiers — is likely to arise by chance. We will be concerned with chance fluctuations that arise due to the finite size of the test set.[8] An improvement of 10% on a test set with ten instances may reflect a random fluctuation that makes the test set more favorable to classifier c_1 than c_2; on another test set with a different ten instances, we might find that c_2 does better than c_1. But if we observe the same 10% improvement on a test set with 1000 instances, this is highly unlikely to be explained by chance. Such a finding is said to be **statistically significant** at a level p, which is the probability of observing an effect of equal or greater magnitude when the null hypothesis is true. The notation $p < .05$ indicates that the likelihood of an equal or greater effect is less than 5%, assuming the null hypothesis is true.[9]

The binomial test

The statistical significance of a difference in accuracy can be evaluated using classical tests, such as the **binomial test**.[10] Suppose that classifiers c_1 and c_2 disagree on N instances in a

[8] Other sources of variance include the initialization of non-convex classifiers such as neural networks, and the ordering of instances in online learning such as stochastic gradient descent and perceptron.

[9] Statistical hypothesis testing is useful only to the extent that the existing test set is representative of the instances that will be encountered in the future. If, for example, the test set is constructed from news documents, no hypothesis test can predict which classifier will perform best on documents from another domain, such as electronic health records.

[10] A well-known alternative to the binomial test is **McNemar's test**, which computes a **test statistic** based on the number of examples that are correctly classified by one system and incorrectly classified by the other.

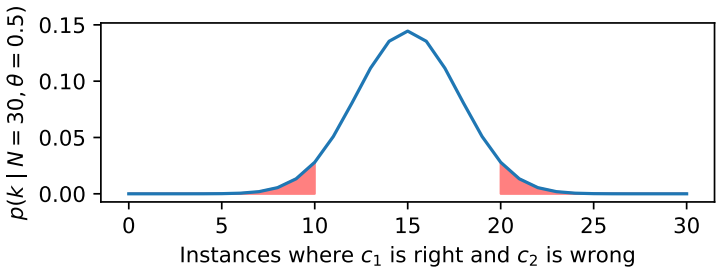


Figure 4.5: Probability mass function for the binomial distribution. The pink highlighted areas represent the cumulative probability for a significance test on an observation of $k = 10$ and $N = 30$.

test set with binary labels, and that c_1 is correct on k of those instances. Under the null hypothesis that the classifiers are equally accurate, we would expect k/N to be roughly equal to $1/2$, and as N increases, k/N should be increasingly close to this expected value. These properties are captured by the **binomial distribution**, which is a probability over counts of binary random variables. We write $k \sim \text{Binom}(\theta, N)$ to indicate that k is drawn from a binomial distribution, with parameter N indicating the number of random "draws", and θ indicating the probability of "success" on each draw. Each draw is an example on which the two classifiers disagree, and a "success" is a case in which c_1 is right and c_2 is wrong. (The label space is assumed to be binary, so if the classifiers disagree, exactly one of them is correct. The test can be generalized to multi-class classification by focusing on the examples in which exactly one classifier is correct.)

The **probability mass function** (PMF) of the binomial distribution is,

$$\text{p}_{\text{Binom}}(k; N, \theta) = \binom{N}{k} \theta^k (1-\theta)^{N-k}, \quad [4.9]$$

with θ^k representing the probability of the k successes, $(1-\theta)^{N-k}$ representing the probability of the $N-k$ unsuccessful draws. The expression $\binom{N}{k} = \frac{N!}{k!(N-k)!}$ is a binomial coefficient, representing the number of possible orderings of events; this ensures that the distribution sums to one over all $k \in \{0, 1, 2, \ldots, N\}$.

Under the null hypothesis, when the classifiers disagree, each classifier is equally likely to be right, so $\theta = \frac{1}{2}$. Now suppose that among N disagreements, c_1 is correct $k < \frac{N}{2}$ times. The probability of c_1 being correct k or fewer times is the **one-tailed p-value**,

The null hypothesis distribution for this test statistic is known to be drawn from a chi-squared distribution with a single degree of freedom, so a p-value can be computed from the cumulative density function of this distribution (Dietterich, 1998). Both tests give similar results in most circumstances, but the binomial test is easier to understand from first principles.

because it is computed from the area under the binomial probability mass function from 0 to k, as shown in the left tail of Figure 4.5. This **cumulative probability** is computed as a sum over all values $i \leq k$,

$$\Pr_{\text{Binom}}\left(\text{count}(\hat{y}_2^{(i)} = y^{(i)} \neq \hat{y}_1^{(i)}) \leq k; N, \theta = \frac{1}{2}\right) = \sum_{i=0}^{k} \text{p}_{\text{Binom}}\left(i; N, \theta = \frac{1}{2}\right). \quad [4.10]$$

The one-tailed p-value applies only to the asymmetric null hypothesis that c_1 is at least as accurate as c_2. To test the **two-tailed** null hypothesis that c_1 and c_2 are equally accurate, we would take the sum of one-tailed p-values, where the second term is computed from the right tail of Figure 4.5. The binomial distribution is symmetric, so this can be computed by simply doubling the one-tailed p-value.

Two-tailed tests are more stringent, but they are necessary in cases in which there is no prior intuition about whether c_1 or c_2 is better. For example, in comparing logistic regression versus averaged perceptron, a two-tailed test is appropriate. In an ablation test, c_2 may contain a superset of the features available to c_1. If the additional features are thought to be likely to improve performance, then a one-tailed test would be appropriate, if chosen in advance. However, such a test can only prove that c_2 is more accurate than c_1, and not the reverse.

***Randomized testing**

The binomial test is appropriate for accuracy, but not for more complex metrics such as F-MEASURE. To compute statistical significance for arbitrary metrics, we can apply randomization. Specifically, draw a set of M **bootstrap samples** (Efron and Tibshirani, 1993), by resampling instances from the original test set with replacement. Each bootstrap sample is itself a test set of size N. Some instances from the original test set will not appear in any given bootstrap sample, while others will appear multiple times; but overall, the sample will be drawn from the same distribution as the original test set. We can then compute any desired evaluation on each bootstrap sample, which gives a distribution over the value of the metric. Algorithm 7 shows how to perform this computation.

To compare the F-MEASURE of two classifiers c_1 and c_2, we set the function $\delta(\cdot)$ to compute the difference in F-MEASURE on the bootstrap sample. If the difference is less than or equal to zero in at least 5% of the samples, then we cannot reject the one-tailed null hypothesis that c_2 is at least as good as c_1 (Berg-Kirkpatrick et al., 2012). We may also be interested in the 95% **confidence interval** around a metric of interest, such as the F-MEASURE of a single classifier. This can be computed by sorting the output of Algorithm 7, and then setting the top and bottom of the 95% confidence interval to the values at the 2.5% and 97.5% percentiles of the sorted outputs. Alternatively, you can fit a normal distribution to the set of differences across bootstrap samples, and compute a Gaussian confidence interval from the mean and variance.

Algorithm 7 Bootstrap sampling for classifier evaluation. The original test set is $\{\boldsymbol{x}^{(1:N)}, \boldsymbol{y}^{(1:N)}\}$, the metric is $\delta(\cdot)$, and the number of samples is M.

procedure BOOTSTRAP-SAMPLE($\boldsymbol{x}^{(1:N)}, \boldsymbol{y}^{(1:N)}, \delta(\cdot), M$)
  for $t \in \{1, 2, \ldots, M\}$ **do**
    for $i \in \{1, 2, \ldots, N\}$ **do**
      $j \sim \text{UniformInteger}(1, N)$
      $\tilde{\boldsymbol{x}}^{(i)} \leftarrow \boldsymbol{x}^{(j)}$
      $\tilde{\boldsymbol{y}}^{(i)} \leftarrow \boldsymbol{y}^{(j)}$
    $d^{(t)} \leftarrow \delta(\tilde{\boldsymbol{x}}^{(1:N)}, \tilde{\boldsymbol{y}}^{(1:N)})$
  return $\{d^{(t)}\}_{t=1}^{M}$

As the number of bootstrap samples goes to infinity, $M \rightarrow \infty$, the bootstrap estimate is increasingly accurate. A typical choice for M is 10^4 or 10^5; larger numbers of samples are necessary for smaller p-values. One way to validate your choice of M is to run the test multiple times, and ensure that the p-values are similar; if not, increase M by an order of magnitude. This is a heuristic measure of the **variance** of the test, which can decreases with the square root $\sqrt{M}$ (Robert and Casella, 2013).

4.4.4 *Multiple comparisons

Sometimes it is necessary to perform multiple hypothesis tests, such as when comparing the performance of several classifiers on multiple datasets. Suppose you have five datasets, and you compare four versions of your classifier against a baseline system, for a total of 20 comparisons. Even if none of your classifiers is better than the baseline, there will be some chance variation in the results, and in expectation you will get one statistically significant improvement at $p = 0.05 = \frac{1}{20}$. It is therefore necessary to adjust the p-values when reporting the results of multiple comparisons.

One approach is to require a threshold of $\frac{\alpha}{m}$ to report a p value of $p < \alpha$ when performing m tests. This is known as the **Bonferroni correction**, and it limits the overall probability of incorrectly rejecting the null hypothesis at α. Another approach is to bound the **false discovery rate** (FDR), which is the fraction of null hypothesis rejections that are incorrect. Benjamini and Hochberg (1995) propose a p-value correction that bounds the fraction of false discoveries at α: sort the p-values of each individual test in ascending order, and set the significance threshold equal to largest k such that $p_k \leq \frac{k}{m}\alpha$. If $k > 1$, the FDR adjustment is more permissive than the Bonferroni correction.

4.5 Building datasets

Sometimes, if you want to build a classifier, you must first build a dataset of your own. This includes selecting a set of documents or instances to annotate, and then performing the annotations. The scope of the dataset may be determined by the application: if you want to build a system to classify electronic health records, then you must work with a corpus of records of the type that your classifier will encounter when deployed. In other cases, the goal is to build a system that will work across a broad range of documents. In this case, it is best to have a *balanced* corpus, with contributions from many styles and genres. For example, the Brown corpus draws from texts ranging from government documents to romance novels (Francis, 1964), and the Google Web Treebank includes annotations for five "domains" of web documents: question answers, emails, newsgroups, reviews, and blogs (Petrov and McDonald, 2012).

4.5.1 Metadata as labels

Annotation is difficult and time-consuming, and most people would rather avoid it. It is sometimes possible to exploit existing metadata to obtain labels for training a classifier. For example, reviews are often accompanied by a numerical rating, which can be converted into a classification label (see § 4.1). Similarly, the nationalities of social media users can be estimated from their profiles (Dredze et al., 2013) or even the time zones of their posts (Gouws et al., 2011). More ambitiously, we may try to classify the political affiliations of social media profiles based on their social network connections to politicians and major political parties (Rao et al., 2010).

The convenience of quickly constructing large labeled datasets without manual annotation is appealing. However this approach relies on the assumption that unlabeled instances — for which metadata is unavailable — will be similar to labeled instances. Consider the example of labeling the political affiliation of social media users based on their network ties to politicians. If a classifier attains high accuracy on such a test set, is it safe to assume that it accurately predicts the political affiliation of all social media users? Probably not. Social media users who establish social network ties to politicians may be more likely to mention politics in the text of their messages, as compared to the average user, for whom no political metadata is available. If so, the accuracy on a test set constructed from social network metadata would give an overly optimistic picture of the method's true performance on unlabeled data.

4.5.2 Labeling data

In many cases, there is no way to get ground truth labels other than manual annotation. An annotation protocol should satisfy several criteria: the annotations should be *expressive* enough to capture the phenomenon of interest; they should be *replicable*, meaning that

another annotator or team of annotators would produce very similar annotations if given the same data; and they should be *scalable*, so that they can be produced relatively quickly. Hovy and Lavid (2010) propose a structured procedure for obtaining annotations that meet these criteria, which is summarized below.

1. **Determine what to annotate**. This is usually based on some theory of the underlying phenomenon: for example, if the goal is to produce annotations about the emotional state of a document's author, one should start with a theoretical account of the types or dimensions of emotion (e.g., Mohammad and Turney, 2013). At this stage, the tradeoff between expressiveness and scalability should be considered: a full instantiation of the underlying theory might be too costly to annotate at scale, so reasonable approximations should be considered.

2. Optionally, one may **design or select a software tool to support the annotation effort**. Existing general-purpose annotation tools include BRAT (Stenetorp et al., 2012) and MMAX2 (Müller and Strube, 2006).

3. **Formalize the instructions for the annotation task**. To the extent that the instructions are not explicit, the resulting annotations will depend on the intuitions of the annotators. These intuitions may not be shared by other annotators, or by the users of the annotated data. Therefore explicit instructions are critical to ensuring the annotations are replicable and usable by other researchers.

4. **Perform a pilot annotation** of a small subset of data, with multiple annotators for each instance. This will give a preliminary assessment of both the replicability and scalability of the current annotation instructions. Metrics for computing the rate of agreement are described below. Manual analysis of specific disagreements should help to clarify the instructions, and may lead to modifications of the annotation task itself. For example, if two labels are commonly conflated by annotators, it may be best to merge them.

5. **Annotate the data**. After finalizing the annotation protocol and instructions, the main annotation effort can begin. Some, if not all, of the instances should receive multiple annotations, so that inter-annotator agreement can be computed. In some annotation projects, instances receive many annotations, which are then aggregated into a "consensus" label (e.g., Danescu-Niculescu-Mizil et al., 2013). However, if the annotations are time-consuming or require significant expertise, it may be preferable to maximize scalability by obtaining multiple annotations for only a small subset of examples.

6. **Compute and report inter-annotator agreement, and release the data**. In some cases, the raw text data cannot be released, due to concerns related to copyright or

privacy. In these cases, one solution is to publicly release **stand-off annotations**, which contain links to document identifiers. The documents themselves can be released under the terms of a licensing agreement, which can impose conditions on how the data is used. It is important to think through the potential consequences of releasing data: people may make personal data publicly available without realizing that it could be redistributed in a dataset and publicized far beyond their expectations (boyd and Crawford, 2012).

Measuring inter-annotator agreement

To measure the replicability of annotations, a standard practice is to compute the extent to which annotators agree with each other. If the annotators frequently disagree, this casts doubt on either their reliability or on the annotation system itself. For classification, one can compute the frequency with which the annotators agree; for rating scales, one can compute the average distance between ratings. These raw agreement statistics must then be compared with the rate of agreement by chance — the expected level of agreement that would be obtained between two annotators who ignored the data.

Cohen's Kappa is widely used for quantifying the agreement on discrete labeling tasks (Cohen, 1960; Carletta, 1996),[11]

$$\kappa = \frac{\text{agreement} - E[\text{agreement}]}{1 - E[\text{agreement}]}. \quad [4.11]$$

The numerator is the difference between the observed agreement and the chance agreement, and the denominator is the difference between perfect agreement and chance agreement. Thus, $\kappa = 1$ when the annotators agree in every case, and $\kappa = 0$ when the annotators agree only as often as would happen by chance. Various heuristic scales have been proposed for determining when κ indicates "moderate", "good", or "substantial" agreement; for reference, Lee and Narayanan (2005) report $\kappa \approx 0.45 - 0.47$ for annotations of emotions in spoken dialogues, which they describe as "moderate agreement"; Stolcke et al. (2000) report $\kappa = 0.8$ for annotations of **dialogue acts**, which are labels for the purpose of each turn in a conversation.

When there are two annotators, the expected chance agreement is computed as,

$$E[\text{agreement}] = \sum_k \hat{\Pr}(Y = k)^2, \quad [4.12]$$

where k is a sum over labels, and $\hat{\Pr}(Y = k)$ is the empirical probability of label k across all annotations. The formula is derived from the expected number of agreements if the annotations were randomly shuffled. Thus, in a binary labeling task, if one label is applied to 90% of instances, chance agreement is $.9^2 + .1^2 = .82$.

[11] For other types of annotations, Krippendorf's alpha is a popular choice (Hayes and Krippendorff, 2007; Artstein and Poesio, 2008).

Crowdsourcing

Crowdsourcing is often used to rapidly obtain annotations for classification problems. For example, **Amazon Mechanical Turk** makes it possible to define "human intelligence tasks (hits)", such as labeling data. The researcher sets a price for each set of annotations and a list of minimal qualifications for annotators, such as their native language and their satisfaction rate on previous tasks. The use of relatively untrained "crowdworkers" contrasts with earlier annotation efforts, which relied on professional linguists (Marcus et al., 1993). However, crowdsourcing has been found to produce reliable annotations for many language-related tasks (Snow et al., 2008). Crowdsourcing is part of the broader field of **human computation** (Law and Ahn, 2011).For a critical examination of ethical issues related to crowdsourcing, see Fort et al. (2011).

Additional resources

Many of the preprocessing issues discussed in this chapter also arise in information retrieval. See Manning et al. (2008) for discussion of tokenization and related algorithms. For more on hypothesis testing in particular and replicability in general, see (Dror et al., 2017, 2018).

Exercises

1. As noted in § 4.3.3, words tend to appear in clumps, with subsequent occurrences of a word being more probable. More concretely, if word j has probability $\phi_{y,j}$ of appearing in a document with label y, then the probability of two appearances $(x_j^{(i)} = 2)$ is greater than $\phi_{y,j}^2$.

   Suppose you are applying Naïve Bayes to a binary classification. Focus on a word j which is more probable under label $y = 1$, so that,

   $$\Pr(w = j \mid y = 1) > \Pr(w = j \mid y = 0). \quad [4.13]$$

   Now suppose that $x_j^{(i)} > 1$. All else equal, will the classifier overestimate or underestimate the posterior $\Pr(y = 1 \mid \boldsymbol{x})$?

2. Prove that F-measure is never greater than the arithmetic mean of recall and precision, $\frac{r+p}{2}$. Your solution should also show that F-measure is equal to $\frac{r+p}{2}$ iff $r = p$.

3. Given a binary classification problem in which the probability of the "positive" label is equal to α, what is the expected F-MEASURE of a random classifier which ignores the data, and selects $\hat{y} = +1$ with probability $\frac{1}{2}$? (Assume that $\mathrm{p}(\hat{y}) \perp \mathrm{p}(y)$.) What is the expected F-MEASURE of a classifier that selects $\hat{y} = +1$ with probability α (also independent of $y^{(i)}$)? Depending on α, which random classifier will score better?

4. Suppose that binary classifiers c_1 and c_2 disagree on $N = 30$ cases, and that c_1 is correct in $k = 10$ of those cases.

   - Write a program that uses primitive functions such as $\exp$ and factorial to compute the **two-tailed** p-value — you may use an implementation of the "choose" function if one is avaiable. Verify your code against the output of a library for computing the binomial test or the binomial CDF, such as SCIPY.STATS.BINOM in Python.
   - Then use a randomized test to try to obtain the same p-value. In each sample, draw from a binomial distribution with $N = 30$ and $\theta = \frac{1}{2}$. Count the fraction of samples in which $k \leq 10$. This is the one-tailed p-value; double this to compute the two-tailed p-value.
   - Try this with varying numbers of bootstrap samples: $M \in \{100, 1000, 5000, 10000\}$. For $M = 100$ and $M = 1000$, run the test 10 times, and plot the resulting p-values.
   - Finally, perform the same tests for $N = 70$ and $k = 25$.

5. SemCor 3.0 is a labeled dataset for word sense disambiguation. You can download it,[12] or access it in NLTK.CORPORA.SEMCOR.

   Choose a word that appears at least ten times in SemCor (*find*), and annotate its WordNet senses across ten randomly-selected examples, without looking at the ground truth. Use online WordNet to understand the definition of each of the senses.[13] Have a partner do the same annotations, and compute the raw rate of agreement, expected chance rate of agreement, and Cohen's kappa.

6. Download the Pang and Lee movie review data, currently available from `http://www.cs.cornell.edu/people/pabo/movie-review-data/`. Hold out a randomly-selected 400 reviews as a test set.

   Download a sentiment lexicon, such as the one currently available from Bing Liu, `https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html`. Tokenize the data, and classify each document as positive iff it has more positive sentiment words than negative sentiment words. Compute the accuracy and F-MEASURE on detecting positive reviews on the test set, using this lexicon-based classifier.

   Then train a discriminative classifier (averaged perceptron or logistic regression) on the training set, and compute its accuracy and F-MEASURE on the test set.

   Determine whether the differences are statistically significant, using two-tailed hypothesis tests: Binomial for the difference in accuracy, and bootstrap for the difference in macro-F-MEASURE.

[12] e.g., `https://github.com/google-research-datasets/word_sense_disambigation_corpora` or `http://globalwordnet.org/wordnet-annotated-corpora/`

[13] http://wordnetweb.princeton.edu/perl/webwn

The remaining problems will require you to build a classifier and test its properties. Pick a multi-class text classification dataset that is not already tokenized. One example is a dataset of New York Times headlines and topics (Boydstun, 2013).[14] Divide your data into training (60%), development (20%), and test sets (20%), if no such division already exists. If your dataset is very large, you may want to focus on a few thousand instances at first.

7. Compare various vocabulary sizes of $10^2, 10^3, 10^4, 10^5$, using the most frequent words in each case (you may use any reasonable tokenizer). Train logistic regression classifiers for each vocabulary size, and apply them to the development set. Plot the accuracy and Macro-F-MEASURE with the increasing vocabulary size. For each vocabulary size, tune the regularizer to maximize accuracy on a subset of data that is held out from the training set.

8. Compare the following tokenization algorithms:

   - Whitespace, using a regular expression;
   - The Penn Treebank tokenizer from NLTK;
   - Splitting the input into non-overlapping five-character units, regardless of whitespace or punctuation.

   Compute the token/type ratio for each tokenizer on the training data, and explain what you find. Train your classifier on each tokenized dataset, tuning the regularizer on a subset of data that is held out from the training data. Tokenize the development set, and report accuracy and Macro-F-MEASURE.

9. Apply the Porter and Lancaster stemmers to the training set, using any reasonable tokenizer, and compute the token/type ratios. Train your classifier on the stemmed data, and compute the accuracy and Macro-F-MEASURE on stemmed development data, again using a held-out portion of the training data to tune the regularizer.

10. Identify the best combination of vocabulary filtering, tokenization, and stemming from the previous three problems. Apply this preprocessing to the test set, and compute the test set accuracy and Macro-F-MEASURE. Compare against a baseline system that applies no vocabulary filtering, whitespace tokenization, and no stemming.

    Use the binomial test to determine whether your best-performing system is significantly more accurate than the baseline.

[14] Available as a CSV file at `http://www.amber-boydstun.com/supplementary-information-for-making-the-news.html`. Use the field TOPIC_2DIGIT for this problem.

Use the bootstrap test with $M = 10^4$ to determine whether your best-performing system achieves significantly higher macro-F-MEASURE.