CHAPTER

7 Logistic Regression

Numquam ponenda est pluralitas sine necessitate
'Plurality should never be proposed unless needed'
William of Occam

We turn now to a second algorithm for classification called **multinomial logistic regression**, sometimes referred to within language processing as **maximum entropy** modeling, **MaxEnt** for short. Logistic regression belongs to the family of classifiers known as the **exponential** or **log-linear** classifiers. Like naive Bayes, it works by extracting some set of weighted features from the input, taking logs, and combining them linearly (meaning that each feature is multiplied by a weight and then added up). Technically, **logistic regression** refers to a classifier that classifies an observation into one of two classes, and **multinomial logistic** regression is used when classifying into more than two classes, although informally and in this chapter we sometimes use the shorthand **logistic regression** even when we are talking about multiple classes.

MaxEnt

log-linear classifier

The most important difference between naive Bayes and logistic regression is that logistic regression is a **discriminative** classifier while naive Bayes is a **generative** classifier. To see what this means, recall that the job of a probabilistic classifier is to choose which output label y to assign an input x, choosing the y that maximizes $P(y|x)$. In the naive Bayes classifier, we used Bayes rule to estimate this best y indirectly from the likelihood $P(x|y)$ (and the prior $P(y)$):

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y|x) = \underset{y}{\operatorname{argmax}} P(x|y)P(y) \tag{7.1}$$

generative model

Because of this indirection, naive Bayes is a **generative model**: a model that is trained to **generate** the data x from the class y. The likelihood term $P(x|y)$ expresses that we are given the class y and are trying to predict which features we expect to see in the input x. Then we use Bayes rule to compute the probability we really want: $P(y|x)$.

discriminative model

But why not instead just directly compute $P(y|x)$? A **discriminative model** takes this direct approach, computing $P(y|x)$ by discriminating among the different possible values of the class y rather than first computing a likelihood:

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y|x) \tag{7.2}$$

While logistic regression thus differs in the way it estimates probabilities, it is still like naive Bayes in being a linear classifier. Logistic regression estimates $P(y|x)$ by extracting some set of features from the input, combining them linearly (multiplying each feature by a weight and adding them up), and then applying a function to this combination.

We can't, however, just compute $P(y|x)$ directly from features and weights as follows:

$$P(y|x) \;?=\; \sum_{i=1}^{N} w_i f_i \tag{7.3}$$

$$?=\; w \cdot f \tag{7.4}$$

Stop for a moment to figure out why this doesn't produce a legal probability. The problem is that the expression $\sum_{i=1}^{N} w_i f_i$ produces values from $-\infty$ to ∞; nothing in the equation above forces the output to be a legal probability, that is, to lie between 0 and 1. In fact, since weights are real-valued, the output might even be negative!

We'll solve this in two ways. First, we'll wrap the exp function around the weight-feature dot-product $w \cdot f$, which will make the values positive, and we'll create the proper denominator to make everything a legal probability and sum to 1. While we're at it, let's assume now that the target y is a variable that ranges over different classes; we want to know the probability that it takes on the particular value of the class c:

$$p(y=c|x) = p(c|x) \;=\; \frac{1}{Z}\exp\sum_{i} w_i f_i \tag{7.5}$$

indicator function

So far we've been assuming that the features f_i are real-valued, but it is more common in language processing to use binary-valued features. A feature that takes on only the values 0 and 1 is called an **indicator function**. Furthermore, the features are not just a property of the observation x, but are instead a property of both the observation x and the candidate output class c. Thus, in MaxEnt, instead of the notation f_i or $f_i(x)$, we use the notation $f_i(c,x)$, meaning feature i for a particular class c for a given observation x:

$$p(c|x) \;=\; \frac{1}{Z}\exp\left(\sum_{i} w_i f_i(c,x)\right) \tag{7.6}$$

Fleshing out the normalization factor Z , and specifying the number of features as N gives us the final equation for computing the probability of y being of class c given x in MaxEnt:

$$p(c|x) \;=\; \frac{\exp\left(\sum_{i=1}^{N} w_i f_i(c,x)\right)}{\sum_{c' \in C}\exp\left(\sum_{i=1}^{N} w_i f_i(c',x)\right)} \tag{7.7}$$

7.1 Features in Multinomial Logistic Regression

Let's look at some sample features for a few NLP tasks to help understand this perhaps unintuitive use of features that are functions of both the observation x and the class c,

Suppose we are doing text classification, and we would like to know whether to assign the sentiment class $+$, $-$, or 0 (neutral) to a document. Here are five potential features, representing that the document x contains the word *great* and the class is

$+$ (f_1), contains the word *second-rate* and the class is $-$ (f_2), and contains the word *no* and the class is $-$ (f_3).

$$\begin{aligned}
f_1(c,x) &= \begin{cases} 1 & \text{if "great"} \in x \;\&\; c = + \\ 0 & \text{otherwise} \end{cases} \\
f_2(c,x) &= \begin{cases} 1 & \text{if "second-rate"} \in x \;\&\; c = - \\ 0 & \text{otherwise} \end{cases} \\
f_3(c,x) &= \begin{cases} 1 & \text{if "no"} \in x \;\&\; c = - \\ 0 & \text{otherwise} \end{cases} \\
f_4(c,x) &= \begin{cases} 1 & \text{if "enjoy"} \in x \;\&\; c = - \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

Each of these features has a corresponding weight, which can be positive or negative. Weight $w_1(x)$ indicates the strength of *great* as a cue for class $+$, $w_2(x)$ and $w_3(x)$ the strength of *second-rate* and *no* for the class $-$. These weights would likely be positive—logically negative words like *no* or *nothing* turn out to be more likely to occur in documents with negative sentiment (Potts, 2011). Weight $w_4(x)$, the strength of *enjoy* for $-$, would likely have a negative weight. We'll discuss in the following section how these weights are learned.

Since each feature is dependent on both a property of the observation and the class being labeled, we would have additional features for the links between *great* and the negative class $-$, or *no* and the neutral class 0, and so on.

Similar features could be designed for other language processing classification tasks. For period disambiguation (deciding if a period is the end of a sentence or part of a word), we might have the two classes EOS (end-of-sentence) and not-EOS and features like f_1 below expressing that the current word is lower case and the class is EOS (perhaps with a positive weight), or that the current word is in our abbreviations dictionary ("Prof.") and the class is EOS (perhaps with a negative weight). A feature can also express a quite complex combination of properties. For example a period following a upper cased word is a likely to be an EOS, but if the word itself is *St.* and the previous word is capitalized, then the period is likely part of a shortening of the word *street*.

$$\begin{aligned}
f_1(c,x) &= \begin{cases} 1 & \text{if "}Case(w_i) = \text{Lower" } \;\&\; c = \text{EOS} \\ 0 & \text{otherwise} \end{cases} \\
f_2(c,x) &= \begin{cases} 1 & \text{if "}w_i \in \text{AcronymDict" } \;\&\; c = \text{EOS} \\ 0 & \text{otherwise} \end{cases} \\
f_3(c,x) &= \begin{cases} 1 & \text{if "}w_i = \text{St." } \;\&\; \text{"}Case(w_{i-1}) = \text{Upper" } \;\&\; c = \text{EOS} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

In Chapter 10 we'll see features for the task of part-of-speech tagging. It's even possible to do discriminative language modeling as a classification task. In this case the set C of classes is the vocabulary of the language, and the task is to predict the next word using features of the previous words (traditional N-gram contexts). In that case, the features might look like the following, with a unigram feature for the word *the* (f_1) or *breakfast* (f_2), or a bigram feature for the context word *American* predicting *breakfast* (f_3). We can even create features that are very difficult to create in a traditional generative language model like predicting the word *breakfast* if the previous word ends in the letters *-an* like *Italian*, *American*, or *Malaysian* (f_4).

$$
\begin{aligned}
f_1(c,x) &= \begin{cases} 1 & \text{if ``}c = \text{the''} \\ 0 & \text{otherwise} \end{cases} \\
f_2(c,x) &= \begin{cases} 1 & \text{if ``}c = \text{breakfast''} \\ 0 & \text{otherwise} \end{cases} \\
f_3(c,x) &= \begin{cases} 1 & \text{if ``}w_{i-1} = \text{American; \& } \; c = \text{breakfast''} \\ 0 & \text{otherwise} \end{cases} \\
f_4(c,x) &= \begin{cases} 1 & \text{if ``}w_{i-1}\text{ends in -an; \& } \; c = \text{breakfast''} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

feature templates

The features for the task of discriminative language models make it clear that we'll often need large numbers of features. Often these are created automatically via **feature templates**, abstract specifications of features. For example a trigram template might create a feature for every predicted word and pair of previous words in the training data. Thus the feature space is sparse, since we only have to create a feature if that n-gram exists in the training set.

The feature is generally created as a hash from the string descriptions. A user description of a feature as, "bigram(American breakfast)" is hashed into a unique integer i that becomes the feature number f_i.

7.2 Classification in Multinomial Logistic Regression

In logistic regression we choose a class by using Eq. 7.7 to compute the probability for each class and then choose the class with the maximum probability.

Fig. 7.1 shows an excerpt from a sample movie review in which the four feature defined in Eq. 7.8 for the two-class sentiment classification task are all 1, with the weights set as $w_1 = 1.9$, $w_2 = .9$, $w_3 = .7$, $w_4 = -.8$.

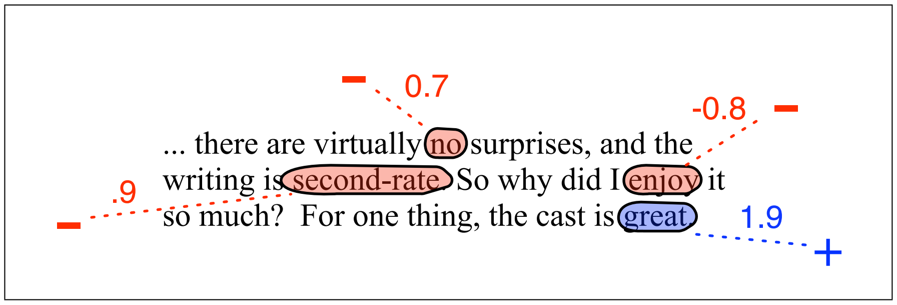
– 0.7
-0.8 –
... there are virtually no surprises, and the
writing is second-rate. So why did I enjoy it
so much? For one thing, the cast is great.
.9
–
1.9
+

Figure 7.1 Some features and their weights for the positive and negative classes. Note the *negative* weight for *enjoy* meaning that it is evidence against the class negative −.

Given these 4 features and the input review x, $P(+|x)$ and $P(-|x)$ can be computed with Eq. 7.7:

$$P(+|x) = \frac{e^{1.9}}{e^{1.9} + e^{.9+.7-.8}} = .82 \tag{7.8}$$

$$P(-|x) = \frac{e^{.9+.7-.8}}{e^{1.9} + e^{.9+.7-.8}} = .18 \tag{7.9}$$

If the goal is just classification, we can even ignore the denominator and the exp and just choose the class with the highest dot product between the weights and features:

$$\begin{aligned}
\hat{c} &= \operatorname*{argmax}_{c \in C} P(c|x) \\
&= \operatorname*{argmax}_{c \in C} \frac{\exp\left(\sum_{i=1}^{N} w_i f_i(c,x)\right)}{\sum_{c' \in C} \exp\left(\sum_{i=1}^{N} w_i f_i(c',x)\right)} \\
&= \operatorname*{argmax}_{c \in C} \exp \sum_{i=1}^{N} w_i f_i(c,x) \\
&= \operatorname*{argmax}_{c \in C} \sum_{i=1}^{N} w_i f_i(c,x) \qquad (7.10)
\end{aligned}$$

Computing the actual probability rather than just choosing the best class, however, is useful when the classifier is embedded in a larger system, as in a sequence classification domain like part-of-speech tagging (Section **??**).

Note that while the index in the inner sum of features in Eq. 7.10 ranges over the entire list of N features, in practice in classification it's not necessary to look at every feature, only the non-zero features. For text classification, for example, we don't have to consider features of words that don't occur in the test document.

7.3 Learning Logistic Regression

conditional maximum likelihood estimation

How are the parameters of the model, the weights w, learned? The intuition is to choose weights that make the classes of the training examples more likely. Indeed, logistic regression is trained with **conditional maximum likelihood estimation**. This means we choose the parameters w that maximize the (log) probability of the y labels in the training data given the observations x.

For an individual training observation $x^{(j)}$ in our training set (we'll use superscripts to refer to individual observations in the training set—this would be each individual document for text classification) the optimal weights are:

$$\hat{w} = \operatorname*{argmax}_{w} \log P(y^{(j)}|x^{(j)}) \qquad (7.11)$$

For the entire set of observations in the training set, the optimal weights would then be:

$$\hat{w} = \operatorname*{argmax}_{w} \sum_{j} \log P(y^{(j)}|x^{(j)}) \qquad (7.12)$$

The objective function L that we are maximizing is thus

$$L(w) = \sum_{j} \log P(y^{(j)}|x^{(j)})$$

$$= \sum_j \log \frac{\exp\left(\sum_{i=1}^N w_i f_i(y^{(j)}, x^{(j)})\right)}{\sum_{y' \in Y} \exp\left(\sum_{i=1}^N w_i f_i(y'^{(j)}, x^{(j)})\right)}$$

$$= \sum_j \log \exp\left(\sum_{i=1}^N w_i f_i(y^{(j)}, x^{(j)})\right) - \sum_j \log \sum_{y' \in Y} \exp\left(\sum_{i=1}^N w_i f_i(y'^{(j)}, x^{(j)})\right)$$

Finding the weights that maximize this objective turns out to be a convex optimization problem, so we use hill-climbing methods like stochastic gradient ascent, L-BFGS (Nocedal 1980, Byrd et al. 1995), or conjugate gradient. Such gradient ascent methods start with a zero weight vector and move in the direction of the *gradient*, $L'(w)$, the partial derivative of the objective function $L(w)$ with respect to the weights. For a given feature dimension k, this derivative can be shown to be the difference between the following two counts:

$$L'(w) = \sum_j f_k(y^{(j)}, x^{(j)}) - \sum_j \sum_{y' \in Y} P(y'|x^{(j)}) f_k(y'^{(j)}, x^{(j)}) \tag{7.13}$$

These two counts turns out to have a very neat interpretation. The first is just the count of feature f_k in the data (the number of times f_k is equal to 1). The second is the expected count of f_k, under the probabilities assigned by the current model:

$$L'(w) = \sum_j \text{Observed count}(f_k) - \text{Expected count}(f_k) \tag{7.14}$$

Thus in optimal weights for the model the model's expected feature values match the actual counts in the data.

7.4 Regularization

There is a problem with learning weights that make the model perfectly match the training data. If a feature is perfectly predictive of the outcome because it happens to only occur in one class, it will be assigned a very high weight. The weights for features will attempt to perfectly fit details of the training set, in fact too perfectly, modeling noisy factors that just accidentally correlate with the class. This problem is called **overfitting**.

overfitting

regularization

To avoid overfitting a **regularization** term is added to the objective function in Eq. 7.13. Instead of the optimization in Eq. 7.12, we optimize the following:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha R(w) \tag{7.15}$$

where $R(w)$, the regularization term, is used to penalize large weights. Thus a setting of the weights that matches the training data perfectly, but uses lots of weights with high values to do so, will be penalized more than than a setting that matches the data a little less well, but does so using smaller weights.

L2 regularization

There are two common regularization terms $R(w)$. **L2 regularization** is a quadratic function of the weight values, named because is uses the (square of the) L2 norm of the weight values. The L2 norm, $||W||_2$, is the same as the **Euclidean distance**:

$$R(W) = ||W||_2^2 = \sum_{j=1}^{N} w_j^2 \tag{7.16}$$

The L2 regularized objective function becomes:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{i=1}^{N} w_i^2 \tag{7.17}$$

L1 regularization

L1 regularization is a linear function of the weight values, named after the L1 norm $||W||_1$, the sum of the absolute values of the weights, or **Manhattan distance** (the Manhattan distance is the distance you'd have to walk between two points in a city with a street grid like New York):

$$R(W) = ||W||_1 = \sum_{i=1}^{N} |w_i| \tag{7.18}$$

The L1 regularized objective function becomes:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{i=1}^{N} |w_i| \tag{7.19}$$

These kinds of regularization come from statistics, where L1 regularization is called **'the lasso'** or **lasso regression** (Tibshirani, 1996) and L2 regression is called **ridge regression**, and both are commonly used in language processing. L2 regularization is easier to optimize because of its simple derivative (the derivative of w^2 is just $2w$), while L1 regularization is more complex (the derivative of $|w|$ is non-continuous at zero). But where L2 prefers weight vectors with many small weights, L1 prefers sparse solutions with some larger weights but many more weights set to zero. Thus L1 regularization leads to much sparser weight vectors, that is, far fewer features.

Both L1 and L2 regularization have Bayesian interpretations as constraints on the prior of how weights should look. L1 regularization can be viewed as a Laplace prior on the weights. L2 regularization corresponds to assuming that weights are distributed according to a gaussian distribution with mean $\mu = 0$. In a gaussian or normal distribution, the further away a value is from the mean, the lower its probability (scaled by the variance σ). By using a gaussian prior on the weights, we are saying that weights prefer to have the value 0. A gaussian for a weight w_j is

$$\frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(w_j - \mu_j)^2}{2\sigma_j^2}\right) \tag{7.20}$$

If we multiply each weight by a gaussian prior on the weight, we are thus maximizing the following constraint:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \prod_j^{M} P(y^{(j)}|x^{(j)}) \times \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(w_i - \mu_j)^2}{2\sigma_j^2}\right) \tag{7.21}$$

which in log space, with $\mu = 0$, and assuming $2\sigma^2 = 1$, corresponds to

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_j \log P(y^{(j)}|x^{(j)}) - \alpha \sum_{i=1}^{N} w_i^2 \tag{7.22}$$

which is in the same form as Eq. 7.17.

7.5 Feature Selection

feature selection

The regularization technique introduced in the previous section is useful for avoiding overfitting by removing or downweighting features that are unlikely to generalize well. Many kinds of classifiers, however, including naive Bayes, do not have regularization, and so instead **feature selection** is used to choose the important features to keep and remove the rest. The basis of feature selection is to assign some metric of goodness to each feature, rank the features, and keep the best ones. The number of features to keep is a meta-parameter that can be optimized on a dev set.

information gain

Features are generally ranked by how informative they are about the classification decision. A very common metric is **information gain**. Information gain tells us how many bits of information the presence of the word gives us for guessing the class, and can be computed as follows (where c_i is the ith class and $\bar{w}$ means that a document does not contain the word w):

$$\begin{aligned} G(w) = \; & -\sum_{i=1}^{C} P(c_i) \log P(c_i) \\ & + P(w) \sum_{i=1}^{C} P(c_i|w) \log P(c_i|w) \\ & + P(\bar{w}) \sum_{i=1}^{C} P(c_i|\bar{w}) \log P(c_i|\bar{w}) \end{aligned} \tag{7.23}$$

Other metrics for feature selection include χ^2, pointwise mutual information, and GINI index; see Yang and Pedersen (1997) for a comparison and Guyon and Elisseeff (2003) for a broad introduction survey of feature selection.

While feature selection is important for unregularized classifiers, it is sometimes also used in regularized classifiers in applications where speed is critical, since it is often possible to get equivalent performance with orders of magnitude fewer features.

7.6 Choosing a classifier and features

Logistic regression has a number of advantages over naive Bayes. The overly strong conditional independence assumptions of Naive Bayes mean that if two features are in fact correlated naive Bayes will multiply them both in as if they were independent, overestimating the evidence. Logistic regression is much more robust to correlated features; if two features f_1 and f_2 are perfectly correlated, regression will simply assign half the weight to w_1 and half to w_2.

Thus when there are many correlated features, logistic regression will assign a more accurate probability than naive Bayes. Despite the less accurate probabilities, naive Bayes still often makes the correct classification decision. Furthermore, naive Bayes works extremely well (even better than logistic regression or SVMs) on small datasets (Ng and Jordan, 2002) or short documents (Wang and Manning, 2012). Furthermore, naive Bayes is easy to implement and very fast to train. Nonetheless, algorithms like logistic regression and SVMs generally work better on larger documents or datasets.

bias-variance tradeoff
bias
variance

Classifier choice is also influenced by the **bias-variance tradeoff**. The **bias** of a classifier indicates how accurate it is at modeling different training sets. The **variance** of a classifier indicates how much its decisions are affected by small changes in training sets. Models with low bias (like SVMs with polynomial or RBF kernels) are very accurate at modeling the training data. Models with low variance (like naive Bayes) are likely to come to the same classification decision even from slightly different training data. But low-bias models tend to be so accurate at fitting the training data that they overfit, and do not generalize well to very different test sets. And low-variance models tend to generalize so well that they may not have sufficient accuracy. Thus any given model trades off bias and variance. Adding more features decreases bias by making it possible to more accurately model the training data, but increases variance because of overfitting. Regularization and feature selection are ways to improve (lower) the variance of classifier by downweighting or removing features that are likely to overfit.

In addition to the choice of a classifier, the key to successful classification is the design of appropriate features. Features are generally designed by examining the training set with an eye to linguistic intuitions and the linguistic literature on the domain. A careful error analysis on the training or dev set. of an early version of a system often provides insights into features.

For some tasks it is especially helpful to build complex features that are combinations of more primitive features. We saw such a feature for period disambiguation above, where a period on the word *St.* was less likely to be the end of sentence if the previous word was capitalized. For logistic regression and naive Bayes these combination features or **feature interactions** have to be designed by hand.

feature interactions

Some other machine learning models can automatically model the interactions between features. For tasks where these combinations of features are important (especially when combination of categorical features and real-valued features might be helpful), the most useful classifiers may be such classifiers,including Support Vector Machines (**SVMs**) with polynomial or RBF kernels, and **random forests**. See the pointers at the end of the chapter.

SVMs
random forests

7.7 Summary

This chapter introduced multinomial **logistic regression** (**MaxEnt**) models for **classification**.

- Multinomial logistic regression (also called MaxEnt or the Maximum Entropy classifier in language processing) is a discriminative model that assigns a class to an observation by computing a probability from an exponential function of a weighted set of features of the observation.

- **Regularization** is important in MaxEnt models for avoiding overfitting.
- **Feature selection** can be helpful in removing useless features to speed up training, and is also important in unregularized models for avoiding overfitting.

Bibliographical and Historical Notes

Maximum entropy modeling, including the use of regularization, was first applied to natural language processing (specifically machine translation) in the early 1990s at IBM (Berger et al. 1996, Della Pietra et al. 1997), and was soon applied to other NLP tasks like part-of-speech tagging and parsing (Ratnaparkhi 1996, Ratnaparkhi 1997) and text classification Nigam et al. (1999). See Chen and Rosenfeld (2000), Goodman (2004), and Dudík et al. (2007) on regularization for maximum entropy models.

More on classification can be found in machine learning textbooks (Hastie et al. 2001, Witten and Frank 2005, Bishop 2006, Murphy 2012).

Exercises

Berger, A., Della Pietra, S. A., and Della Pietra, V. J. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, *22*(1), 39–71.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Byrd, R. H., Lu, P., and Nocedal, J. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing*, *16*, 1190–1208.

Chen, S. F. and Rosenfeld, R. (2000). A survey of smoothing techniques for ME models. *IEEE Transactions on Speech and Audio Processing*, *8*(1), 37–50.

Della Pietra, S. A., Della Pietra, V. J., and Lafferty, J. D. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*(4), 380–393.

Dudík, M., Phillips, S. J., and Schapire, R. E. (2007). Maximum entropy density estimation with generalized regularization and an application to species distribution modeling.. *Journal of Machine Learning Research*, *8*(6).

Goodman, J. (2004). Exponential priors for maximum entropy models. In *ACL-04*.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *The Journal of Machine Learning Research*, *3*, 1157–1182.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2001). *The Elements of Statistical Learning*. Springer.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT press.

Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS 14*, pp. 841–848.

Nigam, K., Lafferty, J., and McCallum, A. (1999). Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, pp. 61–67.

Nocedal, J. (1980). Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, *35*, 773–782.

Potts, C. (2011). On the negativity of negation. In Li, N. and Lutz, D. (Eds.), *Proceedings of Semantics and Linguistic Theory 20*, pp. 636–659. CLC Publications, Ithaca, NY.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *EMNLP 1996*, Philadelphia, PA, pp. 133–142.

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *EMNLP 1997*, Providence, RI, pp. 1–10.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, *58*(1), 267–288.

Wang, S. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL 2012*, pp. 90–94.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques* (2nd Ed.). Morgan Kaufmann.

Yang, Y. and Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *ICML*, pp. 412–420.