This lecture is about **text categorization** (or **classification**). We discussed various applications of text categorization:

- labeling news articles with topics
- detecting sentiment in a movie or product or restaurant review
- deciding whether email is spam or not, or whether it is something you will think is urgent
- determining the reading level of text and automated essay grading
- author identification
- genre (e.g., news report, editorial, advertisement) classification
- language identification

We present two approaches to classification. The first is to think *probabilistically*, building on the ideas we covered in the lectures on language modeling. Probabilistic classification can often be understood as a **noisy channel model**. The source model picks a class according to the class prior distribution, $p(y)$. Then the channel generates the document, conditioned on the selected class. The source model is usually just the empirical distribution over classes seen in the training data, perhaps smoothed. One possibility for the channel is to use a class-specific language model (e.g., a language model for the language of each class, trained only on examples from that class). Another possibility is to represent each possible input as a vector of features and use **naïve Bayes classifier**. Imagine mapping your input (a document) into a d-dimensional space using a vector function $\mathbf{\Phi} : \mathcal{X} \rightarrow \mathbb{R}^d$. It is helpful to think of $\mathbf{\Phi}$ as consisting of d real-valued **feature functions** $\phi_i : \mathcal{X} \rightarrow \mathbb{R}$ that each look at an input and return a single real value. Stacking a d feature functions outputs into a vector gives you a point $\mathbf{\Phi}(x)$ in a d-dimensional space that corresponds to your document. The naïve Bayes classifier then assumes a "naïve" model that generates each feature value in the vector *independently*, given the class. The decoding problem is then to pick the most probable class, given the feature values $\mathbf{\Phi}(x)$,

$$C(x) = \arg\max_y \left[p(y) \times \prod_{j=1}^{d} p(\phi_j(x) \mid y) \right].$$

An alternative is to think *geometrically*. Rather than using a probabilistic model to relate inputs and outputs, a geometric classifier is a function C that first applies $\mathbf{\Phi}$ then maps regions of the "feature space" to classes, $\mathbb{R}^d \rightarrow \mathcal{Y}$.

A **linear classifier** is defined by a hyperplane boundary (points $\mathbf{u}$ such that $\mathbf{w}^\top \mathbf{u} = 0$). For a given document x, if $\mathbf{w}^\top \mathbf{\Phi}(x) < 0$, we choose one class, else the other.

We discussed one way to generalize linear classifiers to handle more than two classes: let $\mathbf{\Phi}$ depend on both x and y, and at test time choose the class y that maximizes $\mathbf{w}^\top \mathbf{\Phi}(x, y)$.

The perceptron is an algorithm for finding a hyperplane ($\mathbf{w}$) that separates training examples from two classes, if one exists. It's very simple and easy to implement; note that if the training data are not separable, the perceptron won't converge. Finding the right features (i.e., a set that separate, or nearly separate the training data) is very important!