An Introduction To Natural Language Processing

Chapter 2: Linear Text Classification

Jacob Eisenstein

Roadmap for this chapter

- Basic representations of text data for classification
- Four linear classifiers
  - Naïve Bayes
  - Perceptron
  - Large-margin (support vector machine)
  - Logistic regression

The text classification problem

Given a text $\boldsymbol{w} = (w_1, w_2, \ldots, w_T) \in \mathcal{V}^*$
choose a label $y \in \mathcal{Y}$.

The text classification problem

Given a text $\boldsymbol{w} = (w_1, w_2, \ldots, w_T) \in \mathcal{V}^*$
choose a label $y \in \mathcal{Y}$.

- Some direct applications:

Sentiment analysis	$\mathcal{Y} = \{\text{positive}, \text{negative}, \text{neutral}\}$
Spam filtering	$\mathcal{Y} = \{\text{spam}, \text{not-spam}\}$
Language identification	$\mathcal{Y} = \{\text{Mandarin}, \text{English}, \text{Spanish}, \ldots\}$

- Indirectly, methods from text classification apply to a huge range of settings in natural language processing, and will appear again and again throughout the course.

The bag-of-words

- One challenge is that the sequential representation $(w_1, w_2, \ldots, w_T)$ may have a different length T for every document.
- The bag-of-words is a fixed-length representation, which consists of a vector of word counts:

$$\boldsymbol{w} = \text{It was the best of times, it was the worst of times}$$
$$\boldsymbol{x} = [\underbrace{\text{aardvark}}_{0}, \ldots, \underbrace{\text{best}}_{1}, \ldots, \underbrace{\text{it}}_{2}, \ldots, \underbrace{\text{of}}_{2}, \ldots, \underbrace{\text{zyther}}_{0}]$$

  The length of $\boldsymbol{x}$ is equal to the size of the vocabulary, V.
- For each $\boldsymbol{x}$, there may be many possible $\boldsymbol{w}$, depending on word order.

Linear classification on the bag-of-words

Let $\psi(\boldsymbol{x}, y)$ score the compatibility of bag-of-words $\boldsymbol{x}$ and label y. Then,

$$\hat{y} = \underset{y}{\text{argmax}}\, \psi(\boldsymbol{x}, y).$$

In a **linear classifier**, this scoring function has a simple form,

$$\psi(\boldsymbol{x}, y) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \sum_{j=1} \theta_j \times f_j(\boldsymbol{x}, y),$$

where $\boldsymbol{\theta}$ is a vector of weights, and $\boldsymbol{f}$ is a **feature function**.

Linear classification in python

```
def compute_score(x,y,weights):
    total = 0
    for feature,count in feature_function(x,y).items():
        total += weights[feature] * count
    return total
```

Feature functions

In classification, the feature function is usually a simple combination of $\boldsymbol{x}$ and y, such as,

$$f_j(\boldsymbol{x}, y) = \begin{cases} x_{\text{whale}}, & \text{if } y = \textsc{Fiction}, \\ 0, & \text{otherwise.} \end{cases}$$

Feature functions

In classification, the feature function is usually a simple combination of $\boldsymbol{x}$ and y, such as,

$$f_j(\boldsymbol{x}, y) = \begin{cases} x_{\text{whale}}, & \text{if } y = \textsc{Fiction}, \\ 0, & \text{otherwise.} \end{cases}$$

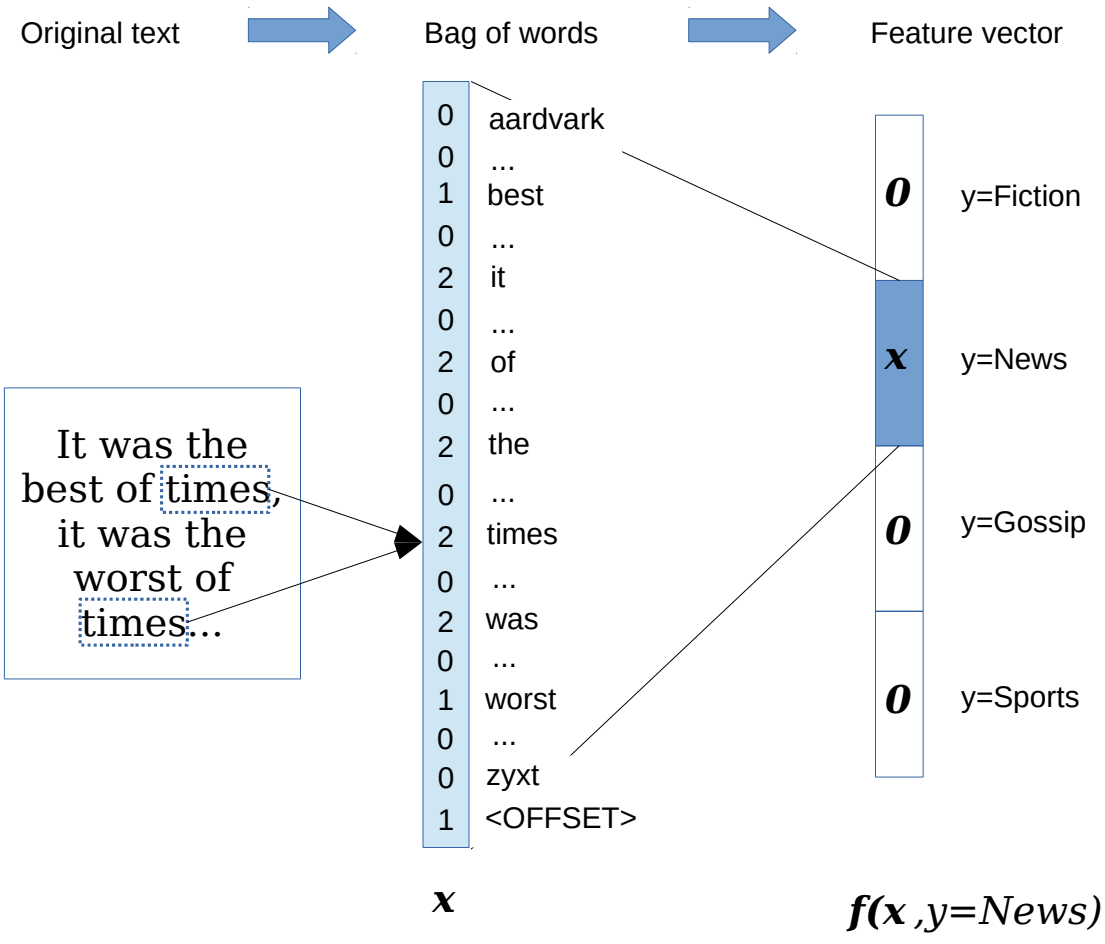
This corresponds to the following formalization as a column vector,

$$\boldsymbol{f}(\boldsymbol{x}, y = 1) = [\boldsymbol{x}; \underbrace{0; 0; \ldots; 0}_{(K-1)\times V}]$$

$$\boldsymbol{f}(\boldsymbol{x}, y = 2) = [\underbrace{0; 0; \ldots; 0}_{V}; \boldsymbol{x}; \underbrace{0; 0; \ldots; 0}_{(K-2)\times V}]$$

$$\boldsymbol{f}(\boldsymbol{x}, y = K) = [\underbrace{0; 0; \ldots; 0}_{(K-1)\times V}; \boldsymbol{x}],$$

Another view of the bag-of-words and feature function



Summary and next steps

- To summarize, our classification function is,

$$\hat{y} = \underset{y}{\text{argmax}}\, \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y),$$

  where $\boldsymbol{f}$ is a feature function and $\boldsymbol{x}$ is the bag-of-words representation.
- The **learning** problem is to find the right weights $\boldsymbol{\theta}$. The remainder of this chapter describes four learning algorithms:
  - Naïve Bayes
  - Perceptron
  - Large-margin (support vector machine)
  - Logistic regression

  All these methods assume a labeled dataset $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.

Probabilistic classification

Naïve Bayes is a probabilistic classifier. It takes the following strategy:

- Define a **probability model**, $\mathrm{p}(\boldsymbol{x}, y)$.
- Estimate the parameters of the probability model by **maximum likelihood** — that is, by maximizing the likelihood of the dataset.
- Set the scoring function equal to the log-probability,

$$\psi(\boldsymbol{x}, y) = \log \mathrm{p}(\boldsymbol{x}, y) = \log \mathrm{p}(y \mid \boldsymbol{x}) + C,$$

where C is constant in y. This ensures that,

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, \mathrm{p}(y \mid \boldsymbol{x}).$$

A probability model for text classification

- First, assume each instance is independent of the others,

$$\mathrm{p}(\boldsymbol{x}^{(1:N)}, \boldsymbol{y}^{(1:N)}) = \prod_{i=1}^{N} \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)}).$$

- Apply the chain rule of probability,

$$\mathrm{p}(\boldsymbol{x}, y) = \mathrm{p}(\boldsymbol{x} \mid y) \times \mathrm{p}(y).$$

- Define the parametric form of each probability,

$$\mathrm{p}(y) = \text{Categorical}(\boldsymbol{\mu}) \qquad \mathrm{p}(\boldsymbol{x} \mid y) = \text{Multinomial}(\boldsymbol{\phi}, T).$$

  - The **multinomial** is a distribution over vectors of counts.
  - The **parameters** $\boldsymbol{\mu}$ and $\boldsymbol{\phi}$ are vectors of probabilities.

The multinomial distribution

- Suppose the word $whale$ has probability ϕ_j.
  What is the probability that this word appears 3 times?
- Each word's probability is exponentiated by its count,

$$\text{Multinomial}(\boldsymbol{x}; \boldsymbol{\phi}, T) = \prod_{j=1}^{V} \phi_j^{x_j}.$$

The multinomial distribution

- Suppose the word $whale$ has probability ϕ_j.
  What is the probability that this word appears 3 times?
- Each word's probability is exponentiated by its count,

$$\text{Multinomial}(\boldsymbol{x}; \boldsymbol{\phi}, T) = \frac{\left(\sum_{j=1}^{V} x_j\right)!}{\prod_{j=1}^{V}(x_j!)} \prod_{j=1}^{V} \phi_j^{x_j}.$$

- The coefficient is the count of the number of possible orderings of $\boldsymbol{x}$. Crucially, it does not depend on the frequency parameter $\boldsymbol{\phi}$.

Naïve Bayes text classification

Naïve Bayes can be reformulated within our linear classification framework by setting $\boldsymbol{\theta}$ equal to the log parameters,

$$\begin{aligned}
\boldsymbol{\theta} &= [\boldsymbol{\theta}^{(1)}; \boldsymbol{\theta}^{(2)}; \ldots; \boldsymbol{\theta}^{(K)}] \\
\boldsymbol{\theta}^{(y)} &= [\log \phi_{y,1}; \log \phi_{y,2}; \ldots; \log \phi_{y,V}; \log \mu_y] \\
\psi(\boldsymbol{x}, y) &= \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y) = \log \mathrm{p}(\boldsymbol{x} \mid y) + \log \mathrm{p}(y),
\end{aligned}$$

where $\boldsymbol{f}(\boldsymbol{x}, y)$ is extended to include an "offset" 1 for each possible label after the word counts.

Estimating Naïve Bayes

In relative frequency estimation, the parameters are set to empirical frequencies:

$$\hat{\phi}_{y,j} = \frac{\text{count}(y, j)}{\sum_{j'=1}^{V} \text{count}(y, j')} = \frac{\sum_{i:y^{(i)}=y} x_j^{(i)}}{\sum_{j'=1}^{V} \sum_{i:y^{(i)}=y} x_{j'}^{(i)}}$$

$$\hat{\mu}_y = \frac{\text{count}(y)}{\sum_{y'} \text{count}(y')}.$$

As shown in the book, this turns out to be identical to the **maximum likelihood estimate**,

$$\hat{\boldsymbol{\phi}}, \hat{\boldsymbol{\mu}} = \underset{\boldsymbol{\phi}, \boldsymbol{\mu}}{\text{argmax}} \prod_{i=1}^{N} \text{p}(\boldsymbol{x}^{(i)}, y^{(i)}) = \underset{\boldsymbol{\phi}, \boldsymbol{\mu}}{\text{argmax}} \sum_{i=1}^{N} \log \text{p}(\boldsymbol{x}^{(i)}, y^{(i)}). \quad (1)$$

Smoothing, bias, variance

To deal with low counts, it can be helpful to smooth probabilities,

$$\hat{\phi}_{y,j} = \frac{\alpha + \text{count}(y, j)}{V\alpha + \sum_{j'=1}^{V} \text{count}(y, j')}.$$

- Smoothing introduces **bias**, moving the parameters away from their maximum-likelihood estimates.
- But it corrects **variance**, which is the extent to which the parameters depend on idiosyncrasies of a finite dataset.
- The smoothing term α is a **hyperparameter**, which must be tuned on a **development set**.

Too naïve?

Naïve Bayes is so called because:

- Bayes rule is used to convert the observation probability $\mathrm{p}(\boldsymbol{x} \mid y)$ into the label probability $\mathrm{p}(y \mid \boldsymbol{x})$.
- The multinomial distribution naïvely ignores dependencies between words, and treats every word as equally informative.
- Suppose $na\ddot{i}ve$ and $Bayes$ always occur together. Should we really count them both for classification?
- **Discriminative** classifiers avoid this problem by not attempting to model the "generative" probability $\mathrm{p}(\boldsymbol{x})$.

The perceptron classifier

A simple learning rule:

- Run the current classifier on an instance in the training data, obtaining $\hat{y} = \text{argmax}_y \, \psi(\boldsymbol{x}^{(i)}, y)$.
- If the prediction is incorrect:
  - Increase the weights for the features of the true label $y^{(i)}$.
  - Decrease the weights for the features of the predicted label $\hat{y}$.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}).$$

- Repeat until all training instances are correctly classified, or you run out of time.

The perceptron classifier

A simple learning rule:

- Run the current classifier on an instance in the training data, obtaining $\hat{y} = \text{argmax}_y \, \psi(\boldsymbol{x}^{(i)}, y)$.
- If the prediction is incorrect:
  - Increase the weights for the features of the true label $y^{(i)}$.
  - Decrease the weights for the features of the predicted label $\hat{y}$.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}).$$

- Repeat until all training instances are correctly classified, or you run out of time.

If the dataset is **linearly separable** — if there is some $\boldsymbol{\theta}$ that correctly labels all the training instances — then this method is guaranteed to find it.

Loss functions

Many classifiers can be viewed as minimizing a **loss function** on the weights. Such a function should have two properties:

- It should be a good proxy for the accuracy of the classifier.
- It should be easy to optimize.

Do you see why 1 – accuracy is not a good loss function?

Perceptron as gradient descent

The perceptron can be viewed as optimizing the loss function:

$$\ell_{\text{perceptron}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')$$

Perceptron as gradient descent

The perceptron can be viewed as optimizing the loss function:

$$\ell_{\text{perceptron}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')$$

The gradient of the perceptron loss is part of the perceptron update:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ell_{\text{perceptron}} = -\boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y})$$

$$\begin{aligned} \boldsymbol{\theta}^{(t+1)} \leftarrow & \boldsymbol{\theta}^{(t)} - \frac{\partial}{\partial \boldsymbol{\theta}} \ell_{\text{perceptron}} && \text{(gradient descent!)} \\ = & \boldsymbol{\theta}^{(t)} + \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}). \end{aligned}$$

Large margin learning

For better generalization, the correct label should outscore all other labels by a large **margin**:

$$\gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y') \quad (2)$$

Large margin learning

For better generalization, the correct label should outscore all other labels by a large **margin**:

$$\gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y') \quad (2)$$

The margin can be incorporated into a **margin loss**:

$$\begin{aligned}\ell_{\text{margin}} &= \max\left(0, 1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)})\right) \\ &= -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \max_{\hat{y} \in \mathcal{Y}} c(y^{(i)}, \hat{y}) + \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}).\end{aligned}$$

Minimizing the margin loss yields an update that is very similar to the perceptron.

Large margin learning

For better generalization, the correct label should outscore all other labels by a large **margin**:

$$\gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y') \quad (2)$$

The margin can be incorporated into a **margin loss**:

$$\begin{aligned}\ell_{\text{margin}} &= \max\left(0, 1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)})\right) \\ &= -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \max_{\hat{y} \in \mathcal{Y}} c(y^{(i)}, \hat{y}) + \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}).\end{aligned}$$

Minimizing the margin loss yields an update that is very similar to the perceptron.

Large margin learning

For better generalization, the correct label should outscore all other labels by a large **margin**:

$$\gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) - \max_{y' \neq y^{(i)}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y') \quad (2)$$

The margin can be incorporated into a **margin loss**:

$$\begin{aligned}\ell_{\text{margin}} &= \max\left(0, 1 - \gamma(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)})\right) \\ &= -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \max_{\hat{y} \in \mathcal{Y}} c(y^{(i)}, \hat{y}) + \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, \hat{y}).\end{aligned}$$

Minimizing the margin loss yields an update that is very similar to the perceptron.

Logistic regression

- Perceptron and large margin classification are **discriminative**: they learn to discriminate correct and incorrect labels.
- Naïve Bayes is **probabilistic**: it assigns calibrated confidence scores to its predictions.
- Logistic regression is both discriminative and probabilistic. It directly computes the conditional probability of the label:

$$\mathrm{p}(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y'))}$$

  - Exponentiation ensures that the probabilities are non-negative.
  - Normalization ensures that the probabilities sum to one.

Logistic regression

- Perceptron and large margin classification are **discriminative**: they learn to discriminate correct and incorrect labels.
- Naïve Bayes is **probabilistic**: it assigns calibrated confidence scores to its predictions.
- Logistic regression is both discriminative and probabilistic. It directly computes the conditional probability of the label:

$$\mathrm{p}(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y'))}$$

  - Exponentiation ensures that the probabilities are non-negative.
  - Normalization ensures that the probabilities sum to one.

Logistic regression

- Perceptron and large margin classification are **discriminative**: they learn to discriminate correct and incorrect labels.
- Naïve Bayes is **probabilistic**: it assigns calibrated confidence scores to its predictions.
- Logistic regression is both discriminative and probabilistic. It directly computes the conditional probability of the label:

$$\mathrm{p}(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}, y'))}$$

  - Exponentiation ensures that the probabilities are non-negative.
  - Normalization ensures that the probabilities sum to one.

Learning logistic regression

Two equivalent views of logistic regression learning:

- **Maximization** of the conditional log-likelihood,

$$\log \mathrm{p}(\boldsymbol{y}^{(1:N)} \mid \boldsymbol{x}^{(1:N)}; \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \mathrm{p}(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

- **Minimization** of the logistic loss,

$$\ell_{\textsc{LogReg}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) = -\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y^{(i)}) + \log \sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}^{(i)}, y')).$$

(Compare with the perceptron loss!)

High-dimensional classification

Possible problems:

- What if the number of features in $\boldsymbol{f}(\boldsymbol{x}, y)$ is larger than the number of training instances?
- What happens in logistic regression if a feature appears only with one label? What will its weight be?

These problems relate to the **variance** of a classifier — its sensitivity to idiosyncratic features of the training data.

Regularization

Learning can often be made more robust by **regularization**: penalizing large weights.

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell_{\text{LogReg}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) + \lambda ||\boldsymbol{\theta}||_2^2, \tag{3}$$

where

- $||\boldsymbol{\theta}||_2^2 = \sum_j \theta_j^2$.
- The scalar λ controls the strength of regularization.
- The **support vector machine** classifier combines regularization with the margin loss.

Regularization

Learning can often be made more robust by **regularization**: penalizing large weights.

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \ell_{\text{LogReg}}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}, y^{(i)}) + \lambda ||\boldsymbol{\theta}||_2^2, \tag{3}$$

where

- $||\boldsymbol{\theta}||_2^2 = \sum_j \theta_j^2$.
- The scalar λ controls the strength of regularization.
- The **support vector machine** classifier combines regularization with the margin loss.

Gradient descent

Logistic regression, perceptron, and large-margin classification all learn by minimizing a **loss function**. A general strategy for minimization is **gradient descent**:

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i=1}^{N} \ell(\boldsymbol{\theta}^{(t)}; \boldsymbol{x}^{(i)}, y^{(i)}), \tag{4}$$

where $\eta \in \mathbb{R}_+$ is the **learning rate**.

Stochastic gradient descent

- Computing the gradient over all instances is expensive.
- **Stochastic** gradient descent approximates the gradient by its value on a single instance:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i=1}^{N} \ell(\boldsymbol{\theta}^{(t)}; \boldsymbol{x}^{(i)}, y^{(i)}) \approx C \times \frac{\partial}{\partial \boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{(t)}; \boldsymbol{x}^{(i)}, y^{(i)}), \quad (5)$$

  where $(\boldsymbol{x}^{(i)}, y^{(i)})$ is sampled at random from the training set.
- **Minibatch** gradient descent approximates the gradient by its value on small number of instances. This is suited to GPU architectures, and is commonly used in deep learning.

Summary of linear classification

	Pros	**Cons**
Naïve Bayes	simple, probabilistic, fast	not very accurate
Perceptron	simple, accurate	not probabilistic, may overfit
Large margin	error-driven learning, can be regularized	not probabilistic
Logistic regression	error-driven learning, regularized	more difficult to implement