Chapter 4

Discriminative Linear Classifiers

4.1

Log-Linear Models

Log-linear Models are a type of *discriminative linear models* that give probabilistively interpretable scores to outputs.

The form of Discriminative Linear Classifiers:

$$score(x, y) = \vec{\theta} \cdot \vec{\phi}(x, y)$$

where $\vec{\theta} \cdot \vec{\phi}(x, y)$ is a feature vector representation of (x, y) and $\vec{\theta}$ is the model(parameter vector).

Different instances of discriminative linear models:

- SVMs
- Perceptrons
- Log-linear models

Define a probabilistic linear discriminative model

Inspiration from an observation of the Naive Bayes classifier:

P (c | d) $\propto \prod_{i=1}^{n} P\left(w_i|c\right) P(c)$

The log form of P(c|d) is a linear model:

log P (c | d) $\propto \sum_{i=1}^{n} \log P\left(w_i|c\right) + \log P(c)$

which is similar to a discriminative linear model.

Log-linear model for multi-class classification

Make $P(y|x) \propto e^{\vec{\theta}\cdot\vec{\phi}(x,y)}$ so that $log P(y|x) \propto \vec{\theta} \cdot \vec{\phi}(x,y)$

Derive $P(y|x)$ of log-linear models by normalisation over C(the set of all possible outputs):

$$P(y|x) = \frac{e^{\vec{\theta}\cdot\vec{\phi}(x,y)}}{\sum_{y' \in C} e^{\vec{\theta}\cdot\vec{\phi}(x,y')}}$$

Which can also be described as:

$$P(y|x) = \ softmax_C(\vec{\theta} \cdot \vec{\phi}(x,y))$$

Softmax function(an exponential function) maps an input in [-∞,∞] to [0,1].

Log-linear model form for binary classification

Logit function or **sigmoid function** is an exponential function that maps a number in [-∞,∞] to [0,1].

$Logit(x) = \frac{e^x}{1+e^x}$

Using the logit function, a binary classifier $score(y = +1) = \vec{\theta} \cdot \vec{\phi}(x) \in [-\infty, +\infty]$ can be mapped into a probabilistic classifier

$$P(y = +1|x) = logit(\vec{\theta} \cdot \vec{\phi}(x)) = \frac{e^{\vec{\theta} \cdot \vec{\phi}(x)}}{1+e^{\vec{\theta} \cdot \vec{\phi}(x)}}$$
$$P(y = -1|x) = logit(\vec{\theta} \cdot \vec{\phi}(x)) = \frac{e^{\vec{\theta} \cdot \vec{\phi}(x)}}{1+e^{\vec{\theta} \cdot \vec{\phi}(x)}}$$

Training log-linear models

We want to train the parameters $\vec{\theta}$ so that the scores $P(\cdot)$ truly represent probabilities.

Given a set of training examples $D = \{(x_i, y_i)\}|_{i=1}^{N}$,

we use maximum likelihood estimation (MLE) to train a log-linear model. The training objective is

$$P(Y|X) = \prod_i P\left(y_i|x_i\right)$$

which is maximising the conditional likelihood of training data.

4.1.1

Training Binary log-linear models

Given $P(y=+1|x) = \frac{e^{\vec{\theta}\cdot\vec{\phi}(x)}}{1+e^{\vec{\theta}\cdot\vec{\phi}(x)}}$, our MLE training objective is

$$P(Y|X) = \prod_i P\left(y_i|x_i\right)$$
$$= \prod_{i^+} P(y=+1|x_i) \prod_{i^-} (1 - P(y=+1|x_i))$$

Maximising $P(Y|X)$ can be achieved by maximising

$$logP(X|Y)$$
$$= \sum_i logP\left(y_i|x_i\right)$$
$$= \sum_{i^+} \log P(y=+1|x_i) + \sum_{i^-} \log(1 - P(y=+1|x_i))$$
$$= \sum_{i^+} \log \frac{e^{\vec{\theta}\cdot\vec{\phi}\left(x_i\right)}}{1+e^{\vec{\theta}\cdot\vec{\phi}\left(x_i\right)}} + \sum_{i^-} \log \frac{1}{1+e^{\vec{\theta}\cdot\vec{\phi}\left(x_i\right)}}$$
$$= \sum_{i^+} \left(\vec{\theta}\cdot\vec{\phi}\left(x_i^+\right) - \log\left(1 + e^{\vec{\theta}\cdot\vec{\phi}\left(x_i^+\right)}\right)\right) - \sum_{i^-} \log \Bigg(1 + e^{\vec{\theta}\cdot\vec{\phi}\left(x_i^-\right)}$$

For linear model, the grandient of the objective is :

$$\vec{g} = \frac{\partial \log P(Y|X)}{\vec{\theta}}$$
$$= \textstyle\sum_{i^+} \left(\vec{\phi}\,(x_i) - \frac{e^{\vec{\theta}\cdot\vec{\phi}(x_i)}}{1+e^{\vec{\theta}\cdot\vec{\phi}(x_i)}}\vec{\phi}\,(x_i)\right) - \textstyle\sum_{i^-} \left(\frac{e^{\vec{\theta}\cdot\vec{\phi}(x_i)}}{1+e^{\vec{\theta}\cdot\vec{\phi}(x_i)}}\vec{\phi}\,(x_i)\right)$$
$$= \textstyle\sum_{i^+} \left(1 - \frac{e^{\vec{\theta}\cdot\vec{\phi}(x_i)}}{1+e^{\vec{\theta}\cdot\vec{\phi}(x_i)}}\right)\vec{\phi}\,(x_i) - \textstyle\sum_{i^-} \left(\frac{e^{\vec{\theta}\cdot\vec{\phi}(x_i)}}{1+e^{\vec{\theta}\cdot\vec{\phi}(x_i)}}\right)\vec{\phi}\,(x_i)$$
$$= \textstyle\sum_{i^+} (1 - P(y=+1|x_i))\vec{\phi}\,(x_i) - \textstyle\sum_{i^-} P(y=+1|x_i)\vec{\phi}\,(x_i)$$

Gradient descent

A simple numerical solution to the minimization of convex functions.

```
*Gradient Descent*
**Inputs**: An objective function F;
**Initialization**:$\vec{\theta}_0$=random(), t=0;
**repeat**:
    |$\vec{g}_t$=$\nabla\vec{\theta}_t F(\vec{\theta}_t)$;
    |$\vec{\theta}_{t+1}$=$\vec{\theta}_t - \alpha\vec{g}_t$;
    |$t$=$t+1$;
**until** $||\vec{\theta}_t - \vec{\theta}_{t-1}|| < \epsilon$
**outputs**:$\vec{\theta}_t$;
```

Here α is the **learning rate**(hyper-parameter), the value of α influences both the accuracy and the efficiency of gradient descent.

Stochastic gradient descent

(batch) Gradient descent can be used to minimize the negative log-likelihood of log-linear models.
But finding $\vec{g}$ at each iteration can be computationally inefficient.

A common solution is a *online* learning algorithm called **stochastic gradient descent (SGD)**
SGD updates model parameters more frequently, converge much faster, while does not always converge to the same otimal point as gradient descent.
The algorithm of SGD training is highly similar in structure to the perceptron algorithm but more fine-grained and results in the probabilistic interpretation of the model.

Mini-batch SGD

A compromise between gradient descent and SGD training.

Split the set of training examples D into several equal-sized subsets $D_1, D_2, ..., D_M$, each containing N/M training examples.

The mini-batch size N/M controls the tradeoff between efficiency and accuracy of approximation.

Make a **random shuffle** to the training set before each training iteration, which can improve the accuracies for some NLP tasks and datasets.

4.1.2

Training multi-class log-linear models

For training pairs $\vec{\phi}(x_i, y_i)$, where $y_i \in C, |C| >= 2.$

The probability of $y_i = \mathbf{c}, \mathbf{c} \in C$ is:

$$P(y_i = c|x_i) = \frac{e^{\vec{\theta}\cdot\vec{\phi}(x_i,\mathbf{c})}}{\sum_{\mathbf{c}'\in C} e^{\vec{\theta}\cdot\vec{\phi}(x_i,\mathbf{c}')}}$$

The log-likelihood of D is

$$\log P(Y|X) = \sum_i \log P\left(y_i|x_i\right)$$
$$= \sum_i \left(\vec{\theta}\cdot\vec{\phi}\left(x_i, y_i\right) - \log\left(\sum_{\mathbf{c}\in C} e^{\vec{\theta}\cdot\vec{\phi}(x_i,\mathbf{c})}\right)\right)$$

For each training example,

$$logP(y_i|x_i) = \vec{\theta}\cdot\vec{\phi}\left(x_i, y_i\right) - \log\left(\sum_{\mathbf{c}\in C} e^{\vec{\theta}\cdot\vec{\phi}(x_i,\mathbf{c})}\right)$$

The local gradient is:

$$\vec{g} = \frac{\partial \log P(y_i|x_i)}{\partial\vec{\theta}}$$
$$= \sum_{\mathbf{c}\in C}\left(\vec{\phi}\left(x_i, y_i\right) - \vec{\phi}\left(x_i, \mathbf{c}\right)\right) P(y = \mathbf{c}|x_i)$$

4.2

SGD training of SVMs

4.2.1 Binary classification

The training objective of binary classification SVM can be modified into minimising $\frac{1}{2}||\vec{\theta}||^2 + C\sum_i \max\left(0, 1 - y_i\left(\vec{\theta} \cdot \vec{\phi}(x_i)\right)\right)$ given $D = \{(x_i, y_i)\}|_{i=1}^N$
which is equivalent to minimizing
$\sum_i \max\left(0, 1 - y_i\left(\vec{\theta} \cdot \vec{\phi}(x_i)\right)\right) + \frac{1}{2}\lambda||\vec{\theta}||^2$
where λ is a hyper-parameter of the model,just as the role of C.

This equation can be optimised using sub-gradient descent.
For each training example, the derivative of the local training objective is

$$\begin{cases} \lambda\vec{\theta} & \text{if } 1 - y_i\vec{\theta} \cdot \vec{\phi}(x_i) \leq 0 \\ \lambda\vec{\theta} - y_i\vec{\phi}(x_i) & \text{otherwise} \end{cases}$$

Multi-class SVM.

The training objective of multi-class SVM is to minimise
$\frac{1}{2}\ |\vec{\theta}|\Big|^2 + C\sum_i \max\left(0, 1 - \vec{\theta}\cdot\vec{\phi}\,(x_i, y_i) + \max_{\mathbf{c}\neq y_i}\vec{\theta}\cdot\vec{\phi}\,(x_i, \mathbf{c})\right)$
which is equivalent to minimising
$\sum_i \max\left(0, 1 - \vec{\theta}\cdot\vec{\phi}\,(x_i, y_i) + \max_{\mathbf{c}\neq y_i}\vec{\theta}\cdot\vec{\phi}\,(x_i, \mathbf{c})\right) + \frac{1}{2}\lambda\|\vec{\theta}\|^2$
where $(x_i, y_i) \in D, \lambda = \frac{1}{C}$.

Using SGD, the derivative for each training example is:
$\begin{cases} \lambda\vec{\theta} & \text{if } 1 - \vec{\theta}\cdot\vec{\phi}\,(x_i, y_i) + \vec{\theta}\cdot\vec{\phi}\,(x_i, \\ \lambda\vec{\theta} - \left(\vec{\phi}\,(x_i, y_i) - \vec{\phi}\,(x_i, z_i)\right) & \text{otherwise} \end{cases}$
where $z_i = argmax_{\mathbf{c}\neq y_i}\vec{\theta}\cdot\vec{\phi}(x_i, \mathbf{c})$.

4.2.2 A perceptron training objective function

Perceptron updates can also be viewed as SGD training of a certain objective function.

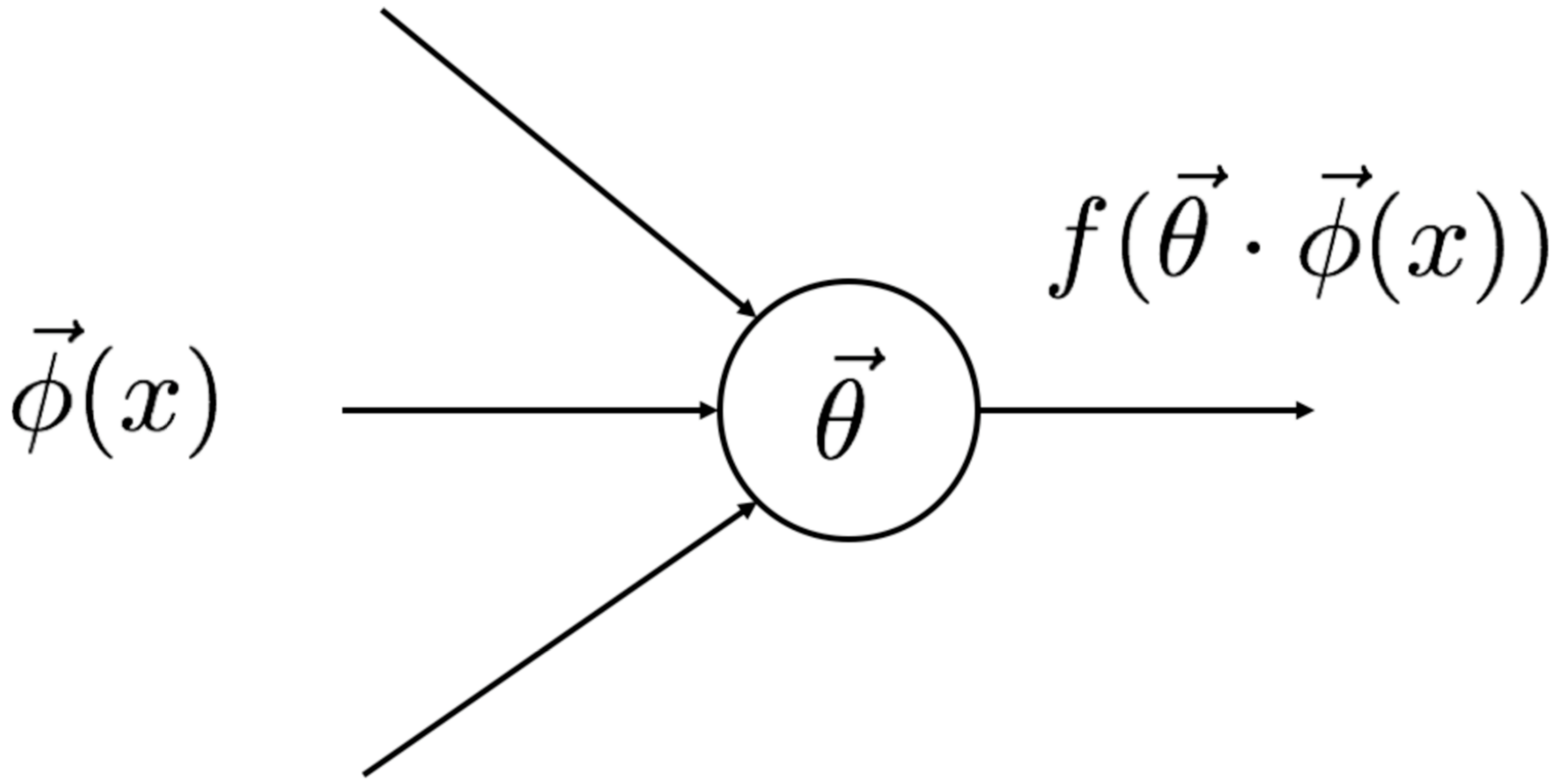
The training objective is to minimize

$$\sum_{i=1}^{N} \max\left(0, -\vec{\theta} \cdot \vec{\phi}\,(x_i, y_i) + \vec{\theta} \cdot \vec{\phi}\left(x_i, \arg\max_{\mathbf{c}} \vec{\theta} \cdot \vec{\phi}\,(x_i, \mathbf{c})\right)\right)$$

4.3

A Generalised Linear Model for classification

Generalised linear classification model



- parameter vector $\vec{\theta}$
- feature vector $\vec{\phi}$
- output class label y using the dot product $\vec{\theta} \cdot \vec{\phi}$

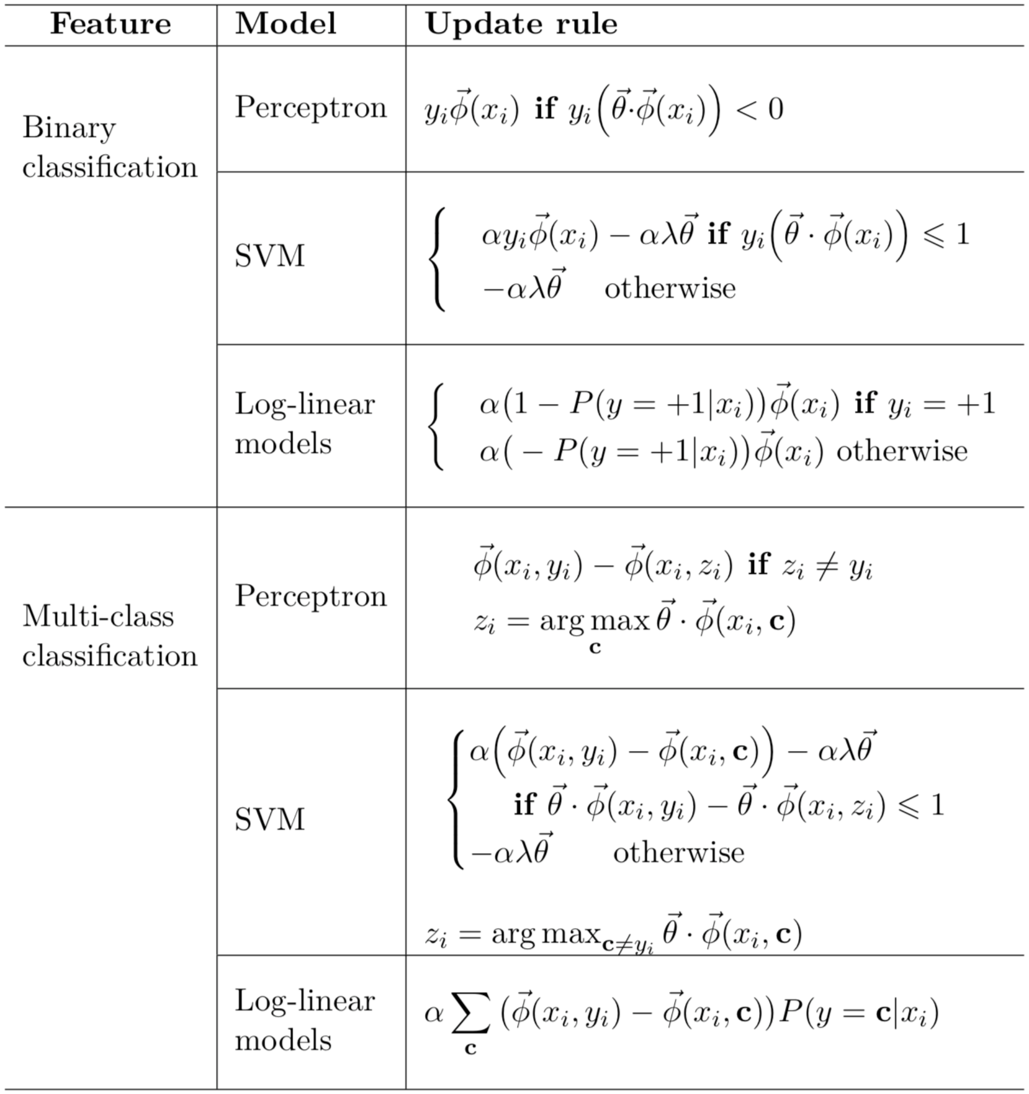
4.3.1 Unified Online Training

Algorithm 8: Online learning for generalized linear model

```
Inputs: $D = \{(x_i, y_i)\}|_{i=1}^{N}$;
Initialization: $\vec{\theta} \leftarrow \vec{0}$, $t \leftarrow 0$;
repeat
    for $i \in [1 \ldots N]$ do
        ParameterUpdate($x_i, y_i$);
    $t \leftarrow t + 1$;
until $t = T$;
Outputs: $\vec{\theta}$;
```

Given a set of training data $D = \{(x_i, y_i)\}|_{i=1}^{N}$, the algorithm goes over D for T iterations, processing each training example (x_i, y_i),and update model parameters when neccessary.

$ParameterUpdate(x_i, y_i)$ for perceptrons,SVMs and log-linear models



Feature	Model	Update rule
Binary classification	Perceptron	$y_i\vec{\phi}(x_i)$ **if** $y_i\left(\vec{\theta}\cdot\vec{\phi}(x_i)\right) < 0$
	SVM	$\begin{cases} \alpha y_i\vec{\phi}(x_i) - \alpha\lambda\vec{\theta} \textbf{ if } y_i\left(\vec{\theta}\cdot\vec{\phi}(x_i)\right) \leqslant 1 \\ -\alpha\lambda\vec{\theta} \quad \text{otherwise} \end{cases}$
	Log-linear models	$\begin{cases} \alpha(1 - P(y = +1 \mid x_i))\vec{\phi}(x_i) \textbf{ if } y_i = +1 \\ \alpha(-P(y = +1 \mid x_i))\vec{\phi}(x_i) \text{ otherwise} \end{cases}$
Multi-class classification	Perceptron	$\vec{\phi}(x_i, y_i) - \vec{\phi}(x_i, z_i)$ **if** $z_i \neq y_i$ $z_i = \arg\max_{\mathbf{c}} \vec{\theta}\cdot\vec{\phi}(x_i, \mathbf{c})$
	SVM	$\begin{cases} \alpha\left(\vec{\phi}(x_i, y_i) - \vec{\phi}(x_i, \mathbf{c})\right) - \alpha\lambda\vec{\theta} \\ \quad \textbf{if } \vec{\theta}\cdot\vec{\phi}(x_i, y_i) - \vec{\theta}\cdot\vec{\phi}(x_i, z_i) \leqslant 1 \\ -\alpha\lambda\vec{\theta} \quad \text{otherwise} \end{cases}$ $z_i = \arg\max_{\mathbf{c}\neq y_i} \vec{\theta}\cdot\vec{\phi}(x_i, \mathbf{c})$
	Log-linear models	$\alpha\sum_{\mathbf{c}}(\vec{\phi}(x_i, y_i) - \vec{\phi}(x_i, \mathbf{c}))P(y = \mathbf{c} \mid x_i)$

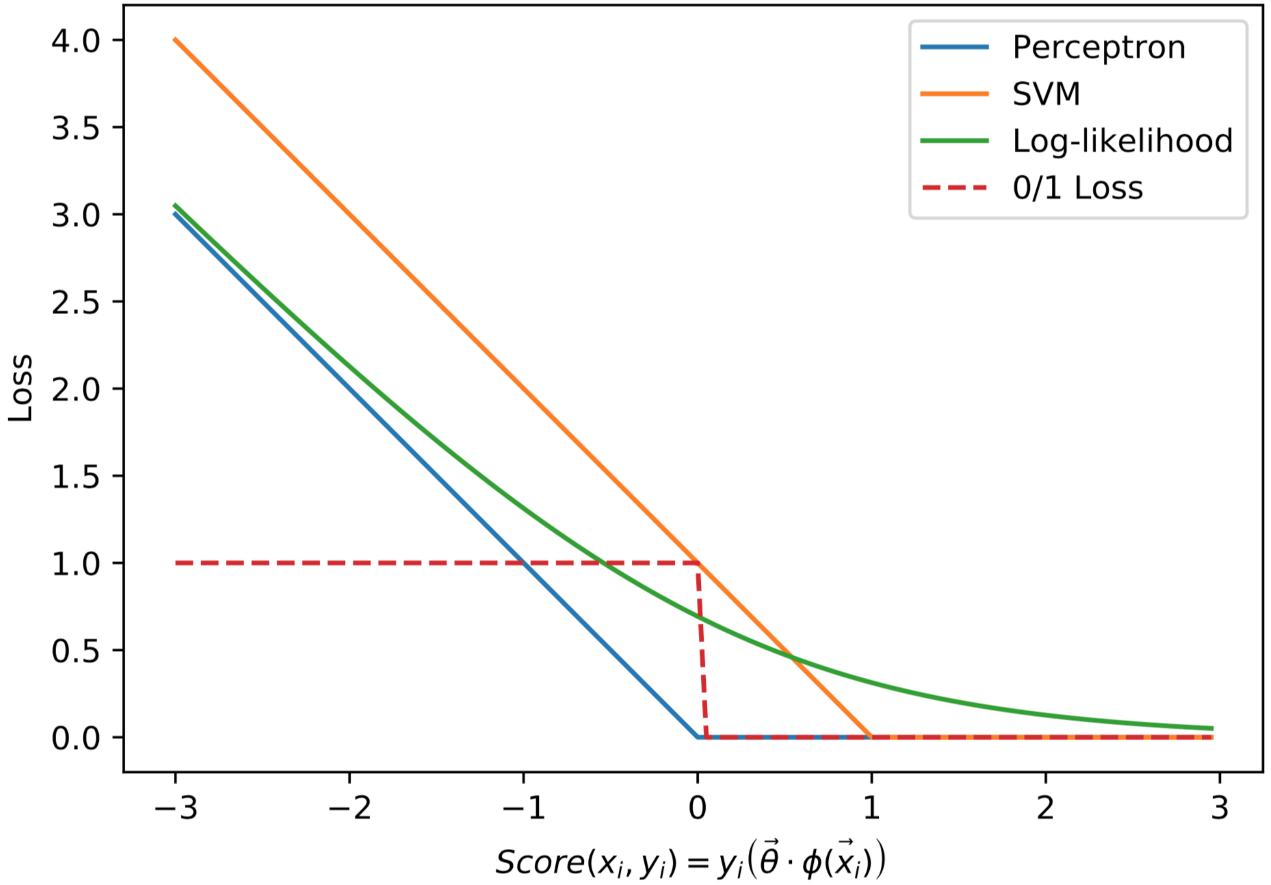
4.3.2 Loss Functions

The training objectives for linear models can be regarded as to minimise different **loss functions** of a model over a training set.

Feature	**Model**	**Loss function**
Binary classification	Perceptron	$\sum_{i=1}^{N} \max\left(0, -y_i\vec{\theta}\cdot\vec{\phi}(x_i)\right)$
	SVM	$\sum_{i=1}^{N} \max\left(0, 1 - y_i\vec{\theta}\cdot\vec{\phi}(x_i)\right) + \frac{1}{2}\lambda\lVert\vec{\theta}\rVert^2$
	Log-linear models	$\sum_{i=1}^{N} \log(1 + e^{-y_i\vec{\theta}\cdot\vec{\phi}(x_i)})$
Multi-class classification	Perceptron	$\sum_{i=1}^{N} \max\Big(0, -\vec{\theta}\cdot\vec{\phi}(x_i, y_i) + \vec{\theta}\cdot\vec{\phi}(x_i, \arg\max_{\mathbf{c}} \vec{\theta}\cdot\vec{\phi}(x_i, \mathbf{c}))\Big)$
	SVM	$\sum_{i=1}^{N} \max\Big(0, 1 - \vec{\theta}\cdot\vec{\phi}(x_i, y_i) + \max_{\mathbf{c}\neq y_i} \vec{\theta}\cdot\vec{\phi}(x_i, \mathbf{c})\Big) + \frac{1}{2}\lambda\lVert\vec{\theta}\rVert^2$
	Log-linear models	$\sum_{i=1}^{N} \Big(\log\left(\sum_{\mathbf{c}} e^{\vec{\theta}\cdot\vec{\phi}(x_i,\mathbf{c})}\right) - \vec{\theta}\cdot\vec{\phi}(x_i, y_i)\Big)$

Different types of loss functions

- **Hinge loss**: the loss functions of SVMs and perceptrons
- **Log-likelihood loss**: the loss functions for log-linear models
- $0/1$ **loss**:loss is 1 for an incorrect output and 0 for a correct output.



Risks

The true **expected risk** of a linear model with parameter can be formulated as

$$risk(\vec{\theta}) = \textstyle\sum_{x,y} loss(\vec{\theta} \cdot \phi(x,y))P(x,y),$$

which cannot be calculated, we use **empirical risk** as a proxy

$$r\tilde{i}sk(\vec{\theta}) = \tfrac{1}{N} \textstyle\sum_{i=1}^{N} loss\left(\vec{\theta} \cdot \phi\left(x_i, y_i\right)\right), \left(x_i, y_i\right) \in D$$

4.3.3 Regularization

A large element in the parameter vector $\vec{\theta}$ implies higher reliance of the model to its corresponding feature, sometimes unnecessarily much.

L2 regularisation and **L1 regularisation** directly minimise a polynomial of $\vec{\theta}$ in loss functions, help reduce over-fitting of models on given training data.

4.4

Working with Multiple Models

4.4.1 Comparing model performances

- different training objectives (large margin or log-likelihood)
- different feature definitions
- different hyperparameters (number of training iterations, learning rate)

A useful way to make choice between alternative models is to make *Empirical comparisons*

We can calculate the probability of obtaining the observed test results, we take small values of such a probability as significance levels, using the significance levels the degree of generalizability.

4.4.2 Ensemble models

Ensemble approach: a combination of multiple models for better accuracies.

Voting: a simple method to ensemble different models.

Given a set of models $M = (m_1, m_2, ...m_{|M|})$ and output classes $C = (c_1, c_2, ..., c_{|C|})$, the output class y for a given input x can be decided by counting the vote (*hard* $0/1$ *votes*):

$v_i = \sum_{j=1}^{|M|} \mathbf{1}\left(y\left(m_j\right), c_i\right)$
majority voting chooses the class label that receives more than half the total votes,
plurality voting chooses the class with the most votes.

More fine-grained voting methods are soft voting and weighted voting.

Stacking:
use the outputs of one model as features to inform another model.

Training for stacking
the stacking method trains A after B is trained.

We use **K-fold jackknifing** to make model B output accuracies on the training data as close to the test scenario as possible.

Bagging use different subsets of D to obtain different models and then emsemble them. Voting is then performed between models given a test case. Bagging can outperform a single model for many tasks.

4.4.3 Tri-training and co-training

Semi-supervised learning use different models trained on D to predict the labels on a set of unlabelled data U, augmenting D with the outputs that most models agree on.

- Tri-training
- Co-training
- Self-training

the more accurate the baseline models are on U, the more likely that the new data form U can be correct and useful.

4.5 Summary

- Log-linear models for binary and multi-class classification
- Stochastic Gradient Descent (SGD) training of log-linear models and SVMs
- A generallised linear discriminative model for text classification
- The correlation between SVMs, perceptrons and log-linear models in terms of training of training objective (loss) functions and regularisation terms
- Significance testing
- Ensemble methods for integrating different models