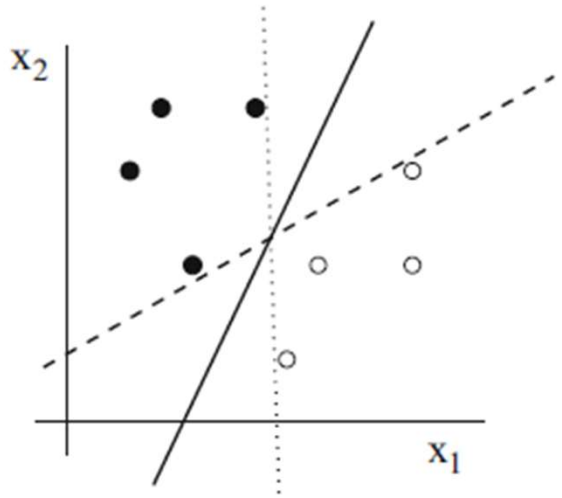
Unit-2

Shallow Neural Networks

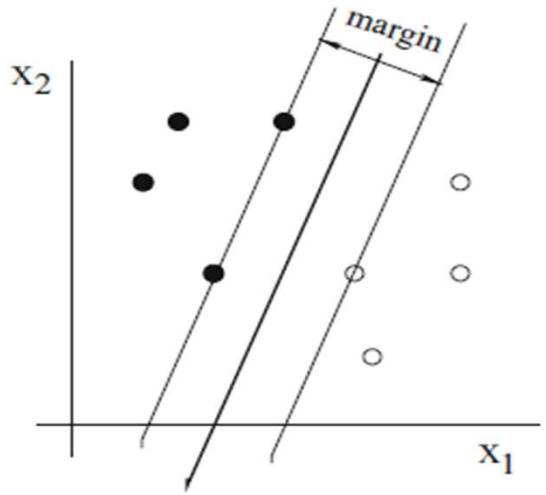
Linear Support Vector Machines

Fig. 4.8 Linearly separable classes can be separated in infinitely many different ways. Question is, which of the classifiers that are perfect on the training set will do best on future data



Linear Support Vector Machines

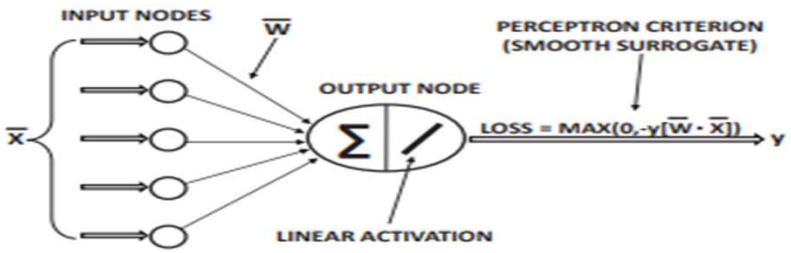
Fig. 4.9 The technique of the *support vector machine* looks for a separating hyperplane that has the maximum *margin*



Linear Support Vector Machines

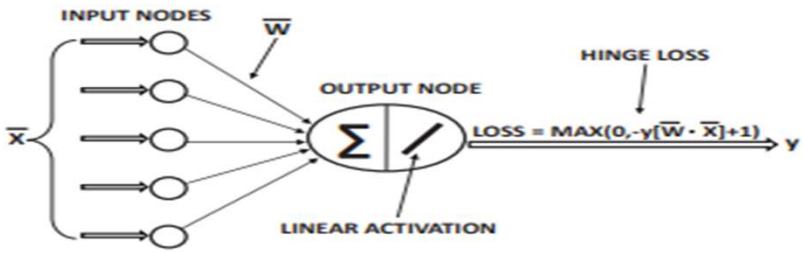
- The technique of the support vector machines involves identifying the best classifier and support vectors.
- The solid line is the best classifier.
- The graph shows also two thinner lines, parallel to the classifier, each at the same distance. They pass through the examples nearest to the classifier.
  - These examples are called support vectors because each example is a vector of attributes.
- The task for machine learning is to identify the support vectors that maximize the margin.
- The simplest technique would simply try all possible n-tuples of examples, and measure the margin implied by each such choice.

Recap: Perceptron versus Linear Support Vector Machine



(a) Perceptron

Loss $= \max\{0, -y(\overline{W} \cdot \overline{X})\}$



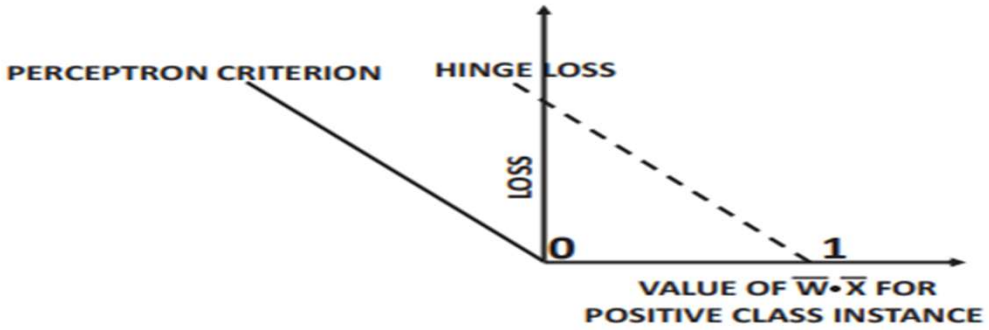
(b) SVM

Loss $= \max\{0, 1 - y(\overline{W} \cdot \overline{X})\}$

- The Perceptron criterion is a minor variation of hinge loss with identical update of $\overline{W} \Leftarrow \overline{W} + \alpha y \overline{X}$ in both cases.

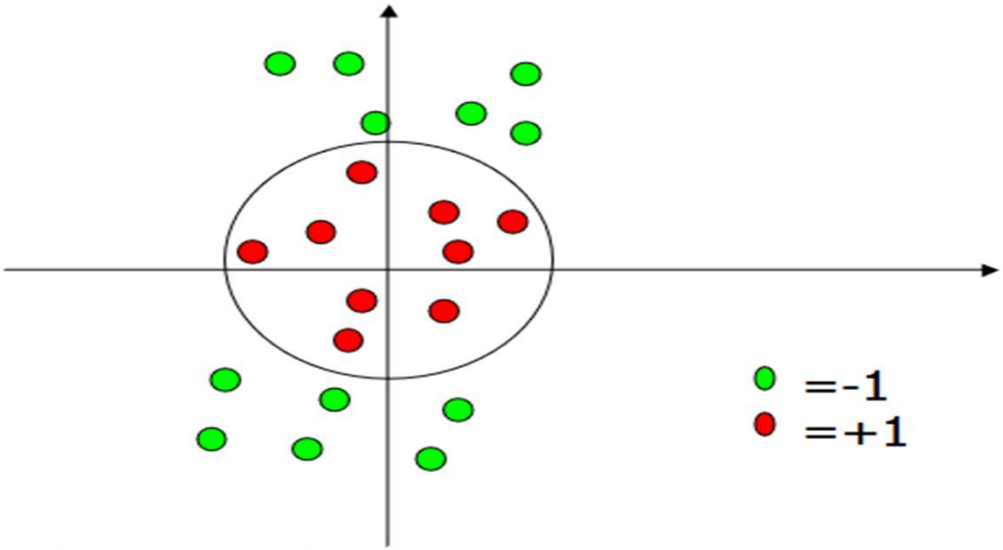
- We update only for misclassified instances in perceptron, but update *also* for "marginally correct" instances in SVM.

Perceptron Criterion versus Hinge Loss



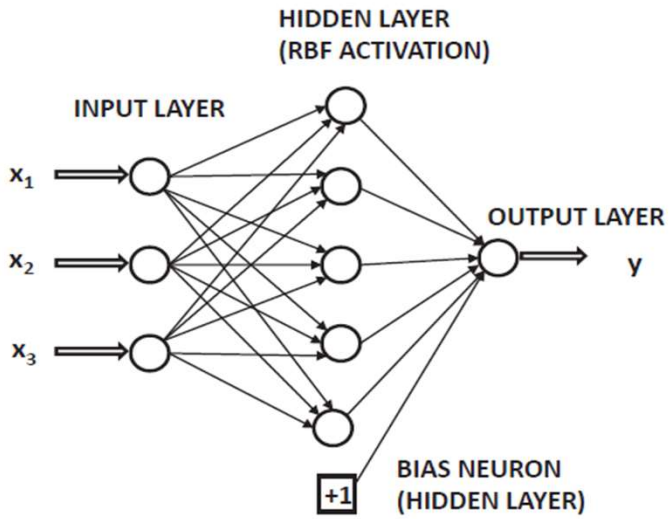
- Loss for positive class training instance at varying values of $\overline{W} \cdot \overline{X}$.

Problems with linear SVM



What if the decision function is not linear? What transform would separate these?

What About the Kernel SVM?



- RBF Network for unsupervised feature engineering.
  - Unsupervised feature engineering is good for noisy data.
  - Supervised feature engineering (with deep learning) is good for learning rich structure.

Much of Machine Learning is a Shallow Neural Model

- By minor changes to the architecture of perceptron we can get:
  - Linear regression, Fisher discriminant, and Widrow-Hoff learning ⇒ Linear activation in output node
  - Logistic regression ⇒ Sigmoid activation in output node with log loss
- Multinomial logistic regression ⇒ Softmax Activation in Final Layer
- Singular value decomposition ⇒ Linear autoencoder
- Incomplete matrix factorization for Recommender Systems ⇒ Autoencoder-like architecture with single hidden layer (also used in *word2vec*)

Log Loss Function/Binary Cross Entropy

- Binary cross entropy, also referred to as logarithmic loss or log loss, is a metric used to evaluate models by measuring the extent of incorrect labeling of data classes.
- **Binary Cross Entropy is the negative average of the log of corrected predicted probabilities.**
- It penalizes the model for deviations in probability that result in misclassification of the labels.
- When the log loss value is low, it indicates a high level of accuracy in the model's predictions.

Log Loss Function/Binary Cross Entropy

- Binary cross entropy, also referred to as logarithmic loss or log loss, is given by following formula where y_{ij} denotes the target value and p_{ij} denotes the predicted value:

$$logloss = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij}\log(p_{ij})$$

- N is the number of rows
- M is the number of classes

Why do We Care about Connections?

- Connections tell us about the cases that it makes sense to use conventional machine learning:
  - If you have less data with noise, you want to use conventional machine learning.
  - If you have a lot of data with rich structure, you want to use neural networks.
  - Structure is often learned by using deep neural architectures.
- Architectures like convolutional neural networks can use domain-specific insights.

Neural Models for Linear Regression, Classification, and the Fisher Discriminant [Connections with Widrow-Hoff Learning]

Widrow-Hoff Rule: The Neural Avatar of Linear Regression

- The perceptron (1958) was historically followed by Widrow-Hoff Learning (1960).

- Identical to linear regression when applied to numerical targets.

  - Originally proposed by Widrow and Hoff for binary targets (not natural for regression).

Linear Regression: An Introduction

- In linear regression, we have training pairs $(\overline{X_i}, y_i)$ for $i \in \{1 \ldots n\}$, so that $\overline{X_i}$ contains d-dimensional features and y_i contains a numerical target.

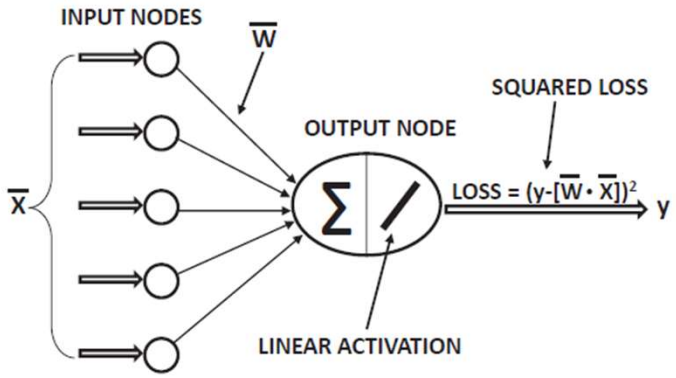
- We use a linear parameterized function to predict $\hat{y}_i = \overline{W} \cdot \overline{X_i}$.

- Goal is to learn $\overline{W}$, so that the sum-of-squared differences between observed y_i and predicted $\hat{y}_i$ is minimized over the entire training data.

- Solution exists in closed form, but requires the inversion of a potentially large matrix.

- Gradient-descent is typically used anyway.

Linear Regression with Numerical Targets:Neural Model



- Predicted output is $\hat{y}_i = \overline{W} \cdot \overline{X}_i$ and loss is $L_i = (y_i - \hat{y}_i)^2$.

- Gradient-descent update is $\overline{W} \Leftarrow \overline{W} - \alpha \frac{\partial L_i}{\partial \overline{W}} = \overline{W} + \alpha(y_i - \hat{y}_i)\overline{X}_i$.

Widrow-Hoff: Linear Regression with Binary Targets

- For $y_i \in \{-1, +1\}$, we use same loss of $(y_i - \hat{y}_i)^2$, and update of $\overline{W} \Leftarrow \overline{W} + \alpha \underbrace{(y_i - \hat{y}_i)}_{\text{delta}} \overline{X}_i$.
  - When applied to binary targets, it is referred to as delta rule.
  - Perceptron uses the same update with $\hat{y}_i = \text{sign}\{\overline{W} \cdot \overline{X}_i\}$, whereas Widrow-Hoff uses $\hat{y}_i = \overline{W} \cdot \overline{X}_i$.
- **Potential drawback:** Retrogressive treatment of well-separated points caused by the pretension that binary targets are real-valued.
  - If $y_i = +1$, and $\overline{W} \cdot \overline{X}_i = 10^6$, the point will be heavily penalized for strongly correct classification!
  - Does not happen in perceptron.

Comparison of Widrow-Hoff with Perceptron and SVM

- Convert the binary loss functions and updates to a form more easily comparable to perceptron using $y_i^2 = 1$:

- Loss of $(\overline{X}_i, y_i)$ is $(y_i - \overline{W} \cdot \overline{X}_i)^2 = (1 - y_i[\overline{W} \cdot \overline{X}_i])^2$
  Update: $\overline{W} \Leftarrow \overline{W} + \alpha y_i(1 - y_i[\overline{W} \cdot \overline{X}_i])\overline{X}_i$

	Perceptron	L_1-Loss SVM
Loss	$\max\{-y_i(\overline{W} \cdot \overline{X}_i), 0\}$	$\max\{1 - y_i(\overline{W} \cdot \overline{X}_i), 0\}$
Update	$\overline{W} \Leftarrow \overline{W} + \alpha y_i I(-y_i[\overline{W} \cdot \overline{X}_i] > 0)\overline{X}_i$	$\overline{W} \Leftarrow \overline{W} + \alpha y_i I(1 - y_i[\overline{W} \cdot \overline{X}_i] > 0)\overline{X}_i$

	Widrow-Hoff	Hinton's L_2-Loss SVM
Loss	$(1 - y_i(\overline{W} \cdot \overline{X}_i))^2$	$\max\{1 - y_i(\overline{W} \cdot \overline{X}_i), 0\}^2$
Update	$\overline{W} \Leftarrow \overline{W} + \alpha y_i(1 - y_i[\overline{W} \cdot \overline{X}_i])\overline{X}_i$	$\overline{W} \Leftarrow \overline{W} + \alpha y_i \max\{(1 - y_i[\overline{W} \cdot \overline{X}_i]), 0\}\overline{X}_i$

Fisher Discriminant

- The Fisher discriminant is formally defined as a direction $\overline{W}$ along which the ratio of inter-class variance to the intra-class variance is maximized in the projected data.
- By choosing a scalar b in order to define the hyperplane $\overline{W} \cdot \overline{X} = b$, it is possible to model the separation between the two classes.
- This hyperplane is used for classification.
- Both the data and the targets need to be mean-centered, which allows the bias variable *b* to be set to 0.

Connections with Fisher Discriminant

- Consider a binary classification problem with training instances $(\overline{X}_i, y_i)$ and $y_i \in \{-1, +1\}$.
  - Mean-center each feature vector as $\overline{X}_i - \overline{\mu}$.
  - Mean-center the binary class by subtracting $\sum_{i=1}^{n} y_i/n$ from each y_i.
- Use the delta rule $\overline{W} \Leftarrow \overline{W} + \alpha \underbrace{(y_i - \hat{y}_i)}_{\text{delta}} \overline{X}_i$ for learning.
- Learned vector is the Fisher discriminant!

https://towardsdatascience.com/fishers-linear-discriminant-intuitively-explained-52a1ba79e1bb

Logistic Regression

- Logistic regression is a probabilistic model that classifies the instances in terms of probabilities.
- Because the classification is probabilistic, a natural approach for optimizing the parameters is to ensure that the predicted probability of the observed class for each training instance is as large as possible.

Logistic Regression

- This goal is achieved by using the notion of ***maximum likelihood estimation*** in order to learn the parameters of the model.
  - The likelihood of the training data is defined as the product of the probabilities of the observed labels of each training instance.
- By using the negative logarithm of this value, one obtains a loss function in minimization form.
  - The output node uses the negative log-likelihood as a loss function.

Logistic Regression

- **Data:** $\{x_i, y_i\}_{i=1}^n$
- **Model:** Our approximation of the relation between $\mathbf{x}$ and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w^T x})}}$$
$$or \quad \hat{y} = \mathbf{w^T x}$$
$$or \quad \hat{y} = \mathbf{x^T W x}$$

  or just about any function

- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data
- **Learning algorithm:** An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)
- **Objective/Loss/Error function:** To guide the learning algorithm - the learning algorithm should aim to minimize the loss function

Logistic Regression

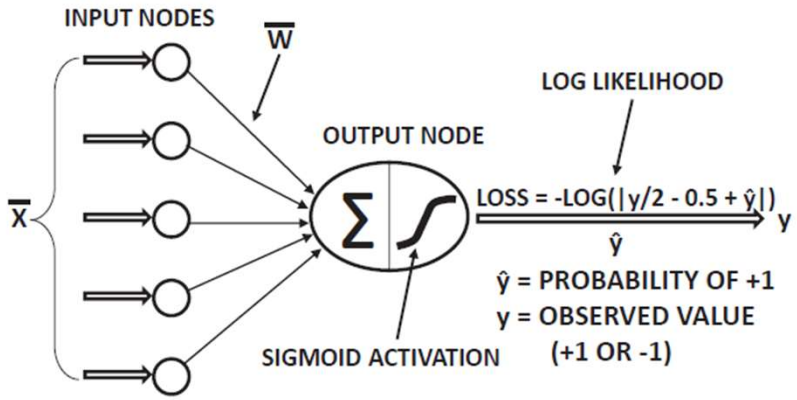
The output layer can be formulated with the sigmoid activation function

- In logistic regression, the sigmoid function is applied to $\overline{W} \cdot \overline{X}_i$, which predicts the probability that y_i is $+1$.

$$\hat{y}_i = P(y_i = 1) = \frac{1}{1 + \exp(-\overline{W} \cdot \overline{X}_i)}$$

- We want to *maximize* $\hat{y}_i$ for positive class instances and $1 - \hat{y}_i$ for negative class instances.
  - Same as *minimizing* $-\log(\hat{y}_i)$ for positive class instances and $-\log(1 - \hat{y}_i)$ for negative instances.
  - Same as minimizing loss $L_i = -\log(|y_i/2 - 0.5 + \hat{y}_i|)$.
  - Alternative form of loss $L_i = \log(1 + \exp[-y_i(\overline{W} \cdot \overline{X}_i)])$

Logistic Regression: Neural Model



- Predicted output is $\widehat{y}_i = 1/(1 + \exp(-\overline{W} \cdot \overline{X}_i))$ and loss is $L_i = -\log(|y_i/2 - 0.5 + \widehat{y}_i|) = \log(1 + \exp[-y_i(\overline{W} \cdot \overline{X}_i)])$.
  - Gradient-descent update is $\overline{W} \Leftarrow \overline{W} - \alpha \frac{\partial L_i}{\partial \overline{W}}$.

$$\overline{W} \Leftarrow \overline{W} + \alpha \frac{y_i \overline{X_i}}{1 + \exp[y_i(\overline{W} \cdot \overline{X_i})]}$$

Interpreting the Logistic Update

- An important multiplicative factor in the update increment is $1/(1 + \exp[y_i(\overline{W} \cdot \overline{X}_i)])$.

- This factor is $1 - \hat{y}_i$ for positive instances and $\hat{y}_i$ for negative instances $\Rightarrow$ Probability of mistake!

- Interpret as: $\overline{W} \Leftarrow \overline{W} + \alpha \left[\text{Probability of mistake on } (\overline{X_i}, y_i)\right] (y_i \overline{X_i})$

Comparing Updates of Different Models

- The unregularized updates of the perceptron, SVM, Widrow-Hoff, and logistic regression can all be written in the following form:

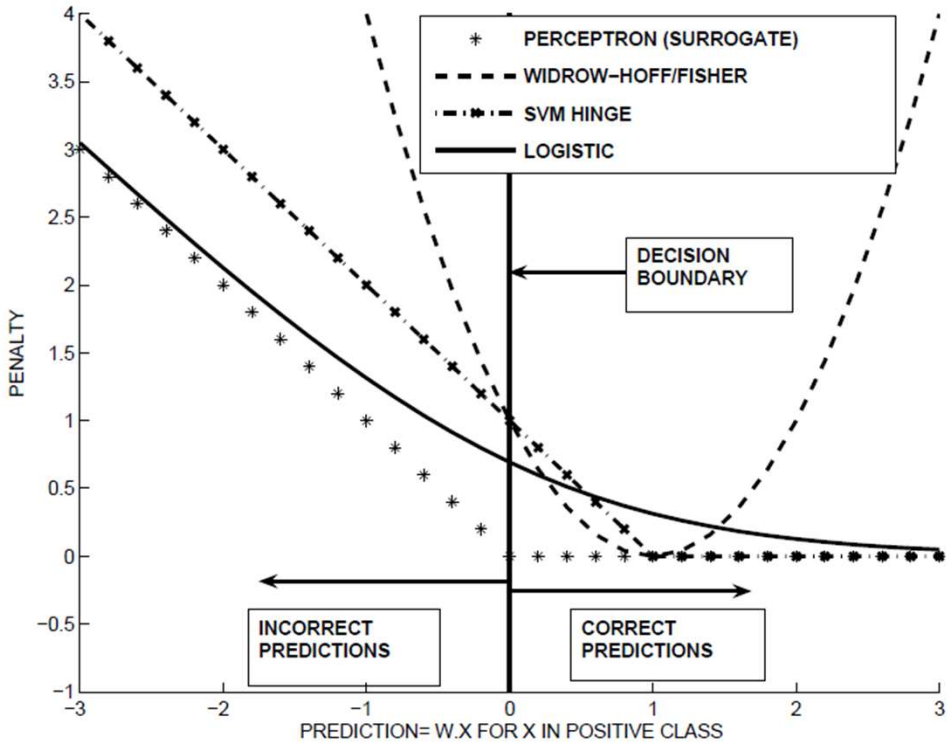
$$\overline{W} \Leftarrow \overline{W} + \alpha y_i \delta(\overline{X}_i, y_i)\overline{X}_i$$

- The quantity $\delta(\overline{X}_i, y_i)$ is a *mistake function*, which is:
  - Raw mistake value $(1 - y_i(\overline{W} \cdot \overline{X}_i))$ for Widrow-Hoff
  - Mistake indicator whether $(0 - y_i(\overline{W} \cdot \overline{X}_i)) > 0$ for perceptron.
  - Margin/mistake indicator whether $(1 - y_i(\overline{W} \cdot \overline{X}_i)) > 0$ for SVM.
  - Probability of mistake on $(\overline{X}_i, y_i)$ for logistic regression.

Comparing Updates of Different Models

- All the derived updates typically correspond to stochastic gradient-descent updates that are encountered both in traditional machine learning and in neural networks.
- The updates are the same whether or not we use a neural architecture to represent the models for these algorithms.
- The key point is that with greater availability of data one can incorporate additional nodes and depth to increase the model's capacity, explaining the superior behavior of neural networks with larger data sets

Comparing Loss Functions of Different Models



Other Comments on Logistic Regression

- Many classical neural models use repeated computational units with logistic and tanh activation functions in hidden layers.

- One can view these methods as feature engineering models that stack multiple logistic regression models.

- The stacking of multiple models creates inherently more powerful models than their individual components.

The Softmax Activation Function and Multinomial Logistic Regression

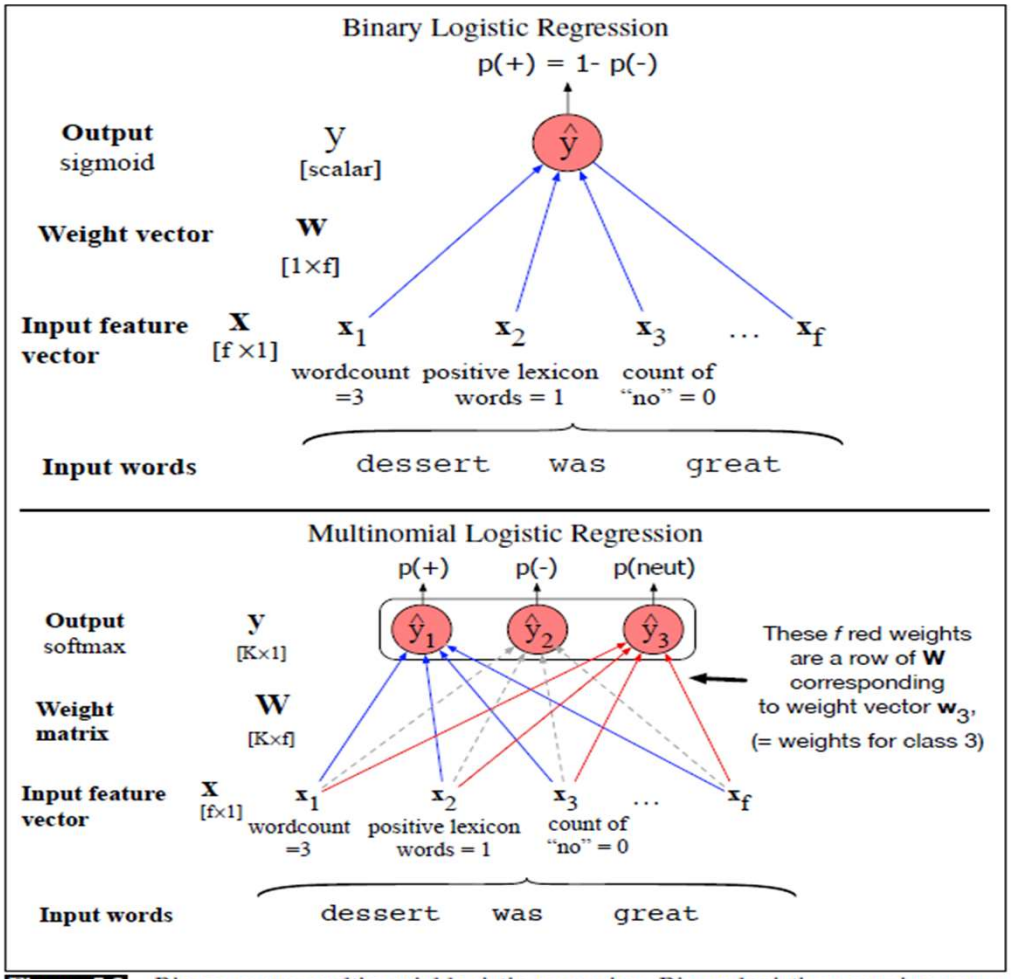


Figure 5.3 Binary versus multinomial logistic regression. Binary logistic regression uses a single weight vector $\mathbf{w}$, and has a scalar output $\hat{y}$. In multinomial logistic regression we have K separate weight vectors corresponding to the K classes, all packed into a single weight matrix $\mathbf{W}$, and a vector output $\hat{\mathbf{y}}$.

Binary Classes versus Multiple Classes

- All the models discussed so far discuss only the binary class setting in which the class label is drawn from $\{-1, +1\}$.

- Many natural applications contain multiple classes without a natural ordering among them:
  - Predicting the category of an image (e.g., *truck*, *carrot*).
  - *Language models:* Predict the next word in a sentence.

- Models like logistic regression are naturally designed to predict two classes.

Generalizing Logistic Regression

- Logistic regression produces probabilities of the two outcomes of a binary class.

- *Multinomial* logistic regression produces probabilities of multiple outcomes.

  - In order to produce probabilities of multiple classes, we need an activation function with a vector output of probabilities.

  - The *softmax activation function* is a vector-based generalization of the sigmoid activation used in logistic regression.

- Multinomial logistic regression is also referred to as softmax classifier.

The Softmax Activation Function

- The softmax activation function is a natural vector-centric generalization of the scalar-to-scalar sigmoid activation $\Rightarrow$ vector-to-vector function.

- Logistic sigmoid activation: $\Phi(v) = 1/(1 + \exp(-v))$.

- Softmax activation: $\Phi(v_1 \ldots v_k) = \frac{1}{\sum_{i=1}^{k} \exp(v_i)} [\exp(v_1) \ldots \exp(v_k)]$

  - The k outputs (probabilities) sum to 1.

- Binary case of using sigmoid(v) is identical to using 2-element softmax activation with arguments $(v, 0)$.

  - Multinomial logistic regression with 2-element softmax is equivalent to binary logistic regression.

Loss Functions for Softmax

- Recall that we use the negative logarithm of the probability of observed class in binary logistic regression.
  - Natural generalization to multiple classes.
  - Cross-entropy loss: Negative logarithm of the probability of correct class.
  - Probability distribution among incorrect classes has no effect.
- Softmax activation is used almost exclusively in output layer and (almost) always paired with cross-entropy loss.

Cross-Entropy Loss of Softmax

- Like the binary logistic case, the loss L is a negative log probability.

$$\text{Softmax Probability Vector} \Rightarrow [\hat{y}_1, \hat{y}_2, \ldots \hat{y}_k]$$

$$[\hat{y}_1 \ldots \hat{y}_k] = \frac{1}{\sum_{i=1}^{k} \exp(v_i)} [\exp(v_1) \ldots \exp(v_k)]$$

- The loss is $-\log(\hat{y}_c)$, where $c \in \{1 \ldots k\}$ is the correct class of that training instance.

Cross-Entropy Loss of Softmax

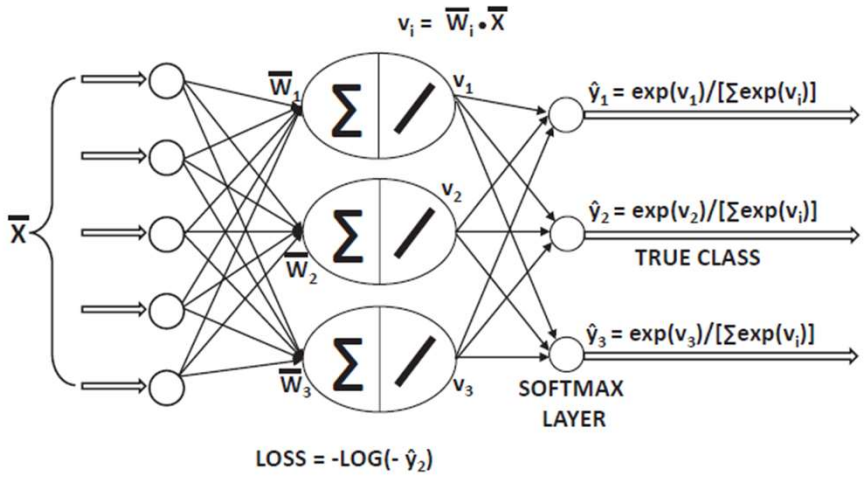
The cross-entropy loss may be expressed in terms of either the input features or in terms of the softmax pre-activation values $v_r = \overline{W_r} \cdot \overline{X_i}$ as follows:

$$L_i = -\log[P(c(i)|\overline{X_i})] \tag{2.30}$$

$$= -\overline{W}_{c(i)} \cdot \overline{X_i} + \log[\sum_{j=1}^{k} \exp(\overline{W_j} \cdot \overline{X_i})] \tag{2.31}$$

$$= -v_{c(i)} + \log[\sum_{j=1}^{k} \exp(v_j)] \tag{2.32}$$

Multinomial Logistic Regression



- The ith training instance is $(\overline{X}_i, c(i))$, where $c(i) \in \{1 \ldots k\}$ is class index $\Rightarrow$ Learn k parameter vectors $\overline{W}_1 \ldots \overline{W}_k$.
  - Define real-valued score $v_r = \overline{W}_r \cdot \overline{X}_i$ for rth class.
  - Convert scores to probabilities $\hat{y}_1 \ldots \hat{y}_k$ with softmax activation on $v_1 \ldots v_k \Rightarrow$ Hard or soft prediction

Computing the Derivative of the Loss

- The cross-entropy loss for the ith training instance is $L_i = -\log(\widehat{y}_{c(i)})$.

- For gradient-descent, we need to compute $\frac{\partial L_i}{\partial \overline{W}_r}$.

- Using chain rule of differential calculus, we get:

$$\frac{\partial L_i}{\partial \overline{W}_r} = \sum_j \left(\frac{\partial L_i}{\partial v_j}\right)\left(\frac{\partial v_j}{\partial \overline{W}_r}\right) = \frac{\partial L_i}{\partial v_r}\underbrace{\frac{\partial v_r}{\partial \overline{W}_r}}_{\overline{X}_i} + \text{Zero-terms}$$

$$= \begin{cases} -\overline{X}_i(1-\widehat{y}_r) & \text{if } r = c(i) \\ \overline{X}_i\,\widehat{y}_r & \text{if } r \neq c(i) \end{cases}$$

Gradient Descent Update

- Each separator $\overline{W_r}$ is updated using the gradient:

$$\overline{W_r} \Leftarrow \overline{W_r} - \alpha \frac{\partial L_i}{\partial \overline{W_r}}$$

- Substituting the gradient from the previous slide, we obtain:

$$\overline{W_r} \Leftarrow \overline{W_r} + \alpha \begin{cases} \overline{X_i} \cdot (1 - \hat{y}_r) & \text{if } r = c(i) \\ -\overline{X_i} \cdot \hat{y}_r & \text{if } r \neq c(i) \end{cases}$$

Multiclass Classification

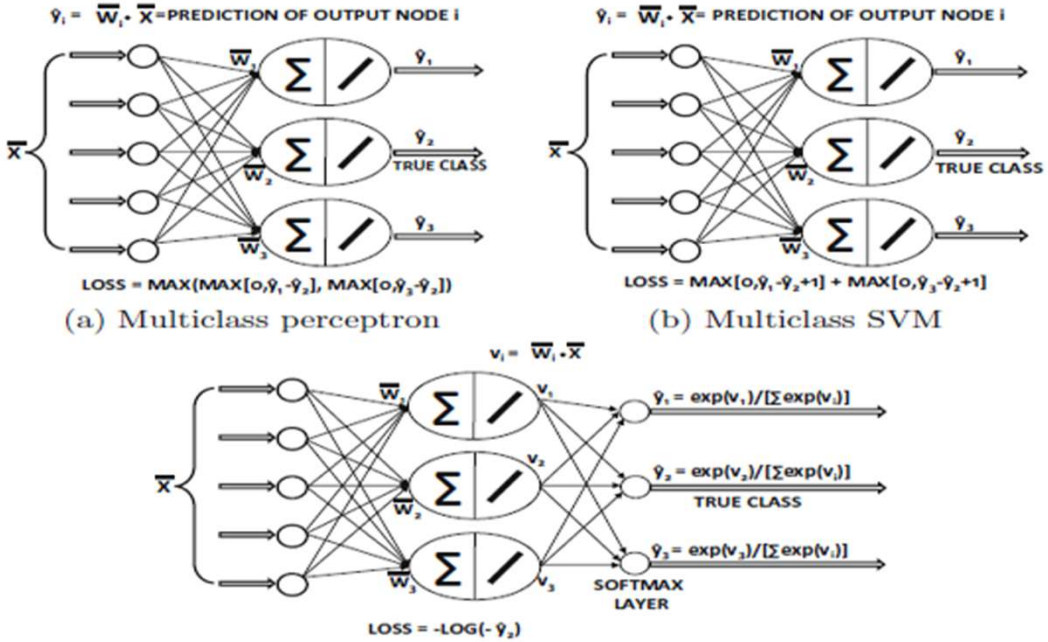


Figure 2.5: Multiclass models: In each case, class 2 is assumed to be the ground-truth class.

Multiclass Perceptron

- If the correct class $r = c(i)$ receives the largest of output $\overline{W_r} \cdot \overline{X_i}$, then no update needs to be executed.
- Otherwise, below update is made to each separator $\overline{W_r}$

$$\overline{W_r} \Leftarrow \overline{W_r} + \begin{cases} \alpha \overline{X_i} & \text{if } r = c(i) \\ -\alpha \overline{X_i} & \text{if } r \neq c(i) \text{ is most misclassified prediction} \\ 0 & \text{otherwise} \end{cases} \qquad (2.24)$$

Multiclass Perceptron Example

- Suppose you have image data scenario with just three classes: A, B, and C.
- Each image is a 1D vector of length 5. Let's assume that the feature vectors are normalized.
  - Initial weights for Class A: [0, 0, 0, 0, 0]
  - Initial weights for Class B: [0, 0, 0, 0, 0]
  - Initial weights for Class C: [0, 0, 0, 0, 0]
- Imagine you're training on an image belonging to Class A.
- After prediction, the scores might look like this:
  - Predicted scores: A: 0.7, B: 0.2, C: -0.1
- Since the true class is A, no updates are needed in this case.
- However, if the prediction was incorrect, the weights would be adjusted accordingly.

Multiclass SVM:
Weston-Watkins SVM

- In multiclass classification using Support Vector Machines (SVM), there are several strategies to extend binary SVMs to handle three or more classes.
- One common approach is the "One-vs-One" and "One-vs-All" methods.
- In the One-vs-One strategy, you create a binary classifier for each pair of classes.
- In the One-vs-All strategy, you create a binary classifier for each class against the rest.

Multiclass SVM:
Weston-Watkins SVM

- In the One-vs-All strategy, you create a binary classifier for each class against the rest. In your case, you would create three classifiers during training:

1. Apple vs. (Orange + Banana)
2. Orange vs. (Apple + Banana)
3. Banana vs. (Apple + Orange)

- Classification:
- When classifying a new fruit, you apply all three SVMs and select the class for which the SVM assigns the highest confidence score.

Multiclass SVM:
Weston-Watkins SVM

- The Weston-Watkins SVM varies on the multiclass perceptron in two ways:

1. The multiclass perceptron only updates the linear separator of a class that is predicted most incorrectly along with the linear separator of the true class.
    - Whereas, the Weston-Watkins SVM updates the separator of any class that is predicted more favorably than the true class.
    - In both cases, the separator of the observed class is updated by the same aggregate amount as the incorrect classes (but in the opposite direction).
2. Not only does the Weston-Watkins SVM update the separator in the case of misclassification, **it updates the separators in cases where an incorrect class gets a prediction that is "uncomfortably close" to the true class.** This is based on the notion of margin.

Multiclass SVM: Weston-Watkins SVM

The loss function L_i for the ith training instance $(\overline{X_i}, c(i))$ in the Weston-Watkins SVM is as follows:

$$L_i = \sum_{r:r \neq c(i)} \max(\overline{W_r} \cdot \overline{X_i} - \overline{W_{c(i)}} \cdot \overline{X_i} + 1, 0) \qquad (2.25)$$

- If the loss function L_i is 0, the gradient of the loss function is 0 as well.
  - Therefore, no update is required when the training instance is classified correctly with sufficient margin with respect to the second-best class.

  If the loss function is non-zero, we have either a misclassified or a "barely correct" prediction in which the second-best and best class prediction are not sufficiently separated.
  - In such cases, the gradient of the loss is non-zero.

Multiclass SVM:
Weston-Watkins SVM

Let $\delta(r, X_i)$ be a 0/1 indicator function, which is 1 when the rth class separator contributes positively to the loss function. The partial derivative of loss is given by:

$$\frac{\partial L_i}{\partial \overline{W_r}} = \begin{cases} -\overline{X_i}[\sum_{j \neq r} \delta(j, \overline{X_i})] & \text{if } r = c(i) \\ \overline{X_i}[\delta(r, \overline{X_i})] & \text{if } r \neq c(i) \end{cases} \tag{2.26}$$

This results in the following stochastic gradient-descent step for the rth separator $\overline{W_r}$ at earning rate α:

$$\overline{W_r} \Leftarrow \overline{W_r}(1 - \alpha\lambda) + \alpha \begin{cases} \overline{X_i}[\sum_{j \neq r} \delta(j, \overline{X_i})] & \text{if } r = c(i) \\ -\overline{X_i}[\delta(r, \overline{X_i})] & \text{if } r \neq c(i) \end{cases} \tag{2.27}$$