CHAPTER 1
The Machine Learning Landscape

Equation 1-1. A simple linear model

$$\text{life\_satisfaction} = \theta_0 + \theta_1 \times \text{GDP\_per\_capita}$$

CHAPTER 2

End-to-End Machine Learning Project

Equation 2-1. Root Mean Square Error (RMSE)

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)} \right)^2}$$

The following equations are located in the "Notations" sidebar, on page 40, in the "Select a performance measure" section:

$$\mathbf{x}^{(1)} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1{,}416 \\ 38{,}372 \end{pmatrix}$$

and:

$$y^{(1)} = 156{,}400$$

$$\mathbf{X} = \begin{pmatrix} \left(\mathbf{x}^{(1)}\right)^\intercal \\ \left(\mathbf{x}^{(2)}\right)^\intercal \\ \vdots \\ \left(\mathbf{x}^{(1999)}\right)^\intercal \\ \left(\mathbf{x}^{(2000)}\right)^\intercal \end{pmatrix} = \begin{pmatrix} -118.29 & 33.91 & 1{,}416 & 38{,}372 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Equation 2-2. Mean absolute error (MAE)

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^{m} \left| h\left(\mathbf{x}^{(i)}\right) - y^{(i)} \right|$$

CHAPTER 3
Classification

Equation 3-1. Precision

$$\text{precision} = \frac{TP}{TP + FP}$$

Equation 3-2. Recall

$$\text{recall} = \frac{TP}{TP + FN}$$

Equation 3-3. F_1 score

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

CHAPTER 4

Training Models

Equation 4-1. Linear Regression model prediction

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Equation 4-2. Linear Regression model prediction (vectorized form)

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

The following note is located in the "Linear Regression" section, on page 113:

In Machine Learning, vectors are often represented as *column vectors*, which are 2D arrays with a single column. If $\boldsymbol{\theta}$ and $\mathbf{x}$ are column vectors, then the prediction is $\hat{y} = \boldsymbol{\theta}^\intercal \mathbf{x}$, where $\boldsymbol{\theta}^\intercal$ is the *transpose* of $\boldsymbol{\theta}$ (a row vector instead of a column vector) and $\boldsymbol{\theta}^\intercal \mathbf{x}$ is the matrix multiplication of $\boldsymbol{\theta}^\intercal$ and $\mathbf{x}$. It is of course the same prediction, except that it is now represented as a single-cell matrix rather than a scalar value. In this book I will use this notation to avoid switching between dot products and matrix multiplications.

Equation 4-3. MSE cost function for a Linear Regression model

$$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^\intercal \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

Equation 4-4. Normal Equation

$$\widehat{\boldsymbol{\theta}} = \left(\mathbf{X}^\intercal \mathbf{X}\right)^{-1} \quad \mathbf{X}^\intercal \quad \mathbf{y}$$

The following paragraph is located in the "Normal Equation" section, on page 116:

This function computes $\widehat{\boldsymbol{\theta}} = \mathbf{X}^{+}\mathbf{y}$, where $\mathbf{X}^{+}$ is the *pseudoinverse* of $\mathbf{X}$ (specifically, the Moore-Penrose inverse).

The following paragraph is located in the "Normal Equation" section, on page 117:

The pseudoinverse itself is computed using a standard matrix factorization technique called *Singular Value Decomposition* (SVD) that can decompose the training set matrix $\mathbf{X}$ into the matrix multiplication of three matrices $\mathbf{U}\ \mathbf{\Sigma}\ \mathbf{V}^\intercal$ (see `numpy.linalg.svd()`). The pseudoinverse is computed as $\mathbf{X}^{+} = \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^\intercal$. To compute the matrix $\mathbf{\Sigma}^{+}$, the algorithm takes $\mathbf{\Sigma}$ and sets to zero all values smaller than a tiny threshold value, then it replaces all the nonzero values with their inverse, and finally it transposes the resulting matrix. This approach is more efficient than computing the Normal Equation, plus it handles edge cases nicely: indeed, the Normal Equation may not work if the matrix $\mathbf{X}^\intercal\mathbf{X}$ is not invertible (i.e., singular), such as if $m < n$ or if some features are redundant, but the pseudoinverse is always defined.

Equation 4-5. Partial derivatives of the cost function

$$\frac{\partial}{\partial \theta_j}\text{MSE}(\boldsymbol{\theta}) = \frac{2}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}^\intercal \mathbf{x}^{(i)} - y^{(i)}\right) x_j^{(i)}$$

Equation 4-6. Gradient vector of the cost function

$$\nabla_{\boldsymbol{\theta}}\,\mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial\theta_0}\mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial\theta_1}\mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial\theta_n}\mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m}\mathbf{X}^\intercal(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Equation 4-7. Gradient Descent step

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta\,\nabla_{\boldsymbol{\theta}}\,\mathrm{MSE}\left(\boldsymbol{\theta}\right)$$

The following paragraph is located in the "Ridge Regression" section, on page 135:

Ridge Regression (also called *Tikhonov regularization*) is a regularized version of Linear Regression: a *regularization term* equal to $\alpha\sum_{i=1}^{n}\theta_i^2$ is added to the cost function.

Equation 4-8. Ridge Regression cost function

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha\frac{1}{2}\sum_{i=1}^{n}\theta_i^2$$

Equation 4-9. Ridge Regression closed-form solution

$$\widehat{\boldsymbol{\theta}} = \left(\mathbf{X}^\intercal\mathbf{X} + \alpha\mathbf{A}\right)^{-1}\ \ \mathbf{X}^\intercal\ \ \mathbf{y}$$

Equation 4-10. Lasso Regression cost function

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha\sum_{i=1}^{n}\left|\theta_i\right|$$

Equation 4-11. Lasso Regression subgradient vector

$$g(\boldsymbol{\theta}, J) = \nabla_{\boldsymbol{\theta}}\,\mathrm{MSE}(\boldsymbol{\theta}) + \alpha \begin{pmatrix} \mathrm{sign}\left(\theta_1\right) \\ \mathrm{sign}\left(\theta_2\right) \\ \vdots \\ \mathrm{sign}\left(\theta_n\right) \end{pmatrix} \quad \text{where } \mathrm{sign}\left(\theta_i\right) = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{cases}$$

Equation 4-12. Elastic Net cost function

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + r\alpha\Sigma_{i=1}^{n}\left|\theta_i\right| + \frac{1-r}{2}\alpha\Sigma_{i=1}^{n}\theta_i^2$$

Equation 4-13. Logistic Regression model estimated probability (vectorized form)

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^{\mathsf{T}}\boldsymbol{\theta})$$

Equation 4-14. Logistic function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

The following paragraph is located in the "Estimating Probabilities" section, on page 143:

Once the Logistic Regression model has estimated the probability $\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x})$ that an instance $\mathbf{x}$ belongs to the positive class, it can make its prediction $\hat{y}$ easily (see Equation 4-15).

Equation 4-15. Logistic Regression model prediction

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$

Equation 4-16. Cost function of a single training instance

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$

Equation 4-17. Logistic Regression cost function (log loss)

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\Sigma_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

Equation 4-18. Logistic cost function partial derivatives

$$\frac{\partial}{\partial\theta_j}\mathrm{J}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\boldsymbol{\theta}^{\top}\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Equation 4-19. Softmax score for class k

$$s_k(\mathbf{x}) = \mathbf{x}^{\top}\boldsymbol{\theta}^{(k)}$$

Equation 4-20. Softmax function

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\Sigma_{j=1}^{K}\exp\left(s_j(\mathbf{x})\right)}$$

Equation 4-21. Softmax Regression classifier prediction

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\boldsymbol{\theta}^{(k)}\right)^{\top}\mathbf{x}\right)$$

Equation 4-22. Cross entropy cost function

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\Sigma_{i=1}^{m}\Sigma_{k=1}^{K}y_k^{(i)}\log\left(\hat{p}_k^{(i)}\right)$$

Equation 4-23. Cross entropy gradient vector for class k

$$\nabla_{\boldsymbol{\theta}^{(k)}}J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$$

CHAPTER 5

Support Vector Machines

Equation 5-1. Gaussian RBF

$$\phi_{\gamma}(\mathbf{x}, \ell) = \exp\left(-\gamma \| \mathbf{x} - \ell \|^2\right)$$

Equation 5-2. Linear SVM classifier prediction

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^\intercal \mathbf{x} + b < 0, \\ 1 & \text{if } \mathbf{w}^\intercal \mathbf{x} + b \geq 0 \end{cases}$$

Equation 5-3. Hard margin linear SVM classifier objective

$$\begin{aligned} &\underset{\mathbf{w}, b}{\text{minimize}} && \frac{1}{2}\mathbf{w}^\intercal \mathbf{w} \\ &\text{subject to} && t^{(i)}\left(\mathbf{w}^\intercal \mathbf{x}^{(i)} + b\right) \geq 1 \quad \text{for } i = 1, 2, \cdots, m \end{aligned}$$

Equation 5-4. Soft margin linear SVM classifier objective

$$\begin{aligned} &\underset{\mathbf{w}, b, \zeta}{\text{minimize}} && \frac{1}{2}\mathbf{w}^\intercal \mathbf{w} + C \sum_{i=1}^{m} \zeta^{(i)} \\ &\text{subject to} && t^{(i)}\left(\mathbf{w}^\intercal \mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and} \quad \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m \end{aligned}$$

Equation 5-5. Quadratic Programming problem

$$\begin{aligned}
&\underset{\mathbf{p}}{\text{Minimize}} && \frac{1}{2}\mathbf{p}^\intercal\mathbf{H}\mathbf{p} \quad + \quad \mathbf{f}^\intercal\mathbf{p}\\
&\text{subject to} && \mathbf{A}\mathbf{p} \le \mathbf{b}\\
&\text{where} && \begin{cases}
\mathbf{p} & \text{is an } n_p\text{-dimensional vector } (n_p = \text{number of parameters}),\\
\mathbf{H} & \text{is an } n_p \times n_p \text{ matrix},\\
\mathbf{f} & \text{is an } n_p\text{-dimensional vector},\\
\mathbf{A} & \text{is an } n_c \times n_p \text{ matrix } (n_c = \text{number of constraints}),\\
\mathbf{b} & \text{is an } n_c\text{-dimensional vector}.
\end{cases}
\end{aligned}$$

Note that the expression $\mathbf{A}\ \mathbf{p} \le \mathbf{b}$ defines n_c constraints: $\mathbf{p}^\intercal\ \mathbf{a}^{(i)} \le b^{(i)}$ for $i = 1, 2, \cdots, n_c$, where $\mathbf{a}^{(i)}$ is the vector containing the elements of the i^{th} row of $\mathbf{A}$ and $b^{(i)}$ is the i^{th} element of $\mathbf{b}$.

You can easily verify that if you set the QP parameters in the following way, you get the hard margin linear SVM classifier objective:

- $n_p = n + 1$, where n is the number of features (the +1 is for the bias term).
- $n_c = m$, where m is the number of training instances.
- $\mathbf{H}$ is the $n_p \times n_p$ identity matrix, except with a zero in the top-left cell (to ignore the bias term).
- $\mathbf{f} = 0$, an n_p-dimensional vector full of 0s.
- $\mathbf{b} = -1$, an n_c-dimensional vector full of –1s.
- $\mathbf{a}^{(i)} = -t^{(i)}\ \dot{\mathbf{x}}^{(i)}$, where $\dot{\mathbf{x}}^{(i)}$ is equal to $\mathbf{x}^{(i)}$ with an extra bias feature $\dot{\mathbf{x}}_0 = 1$.

Equation 5-6. Dual form of the linear SVM objective

$$\begin{aligned}
&\underset{\alpha}{\text{minimize}}\ \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}\mathbf{x}^{(i)\intercal}\mathbf{x}^{(j)} \quad - \quad \sum_{i=1}^{m}\alpha^{(i)}\\
&\qquad\qquad \text{subject to} \quad \alpha^{(i)} \ge 0 \quad \text{for } i = 1, 2, \cdots, m
\end{aligned}$$

Once you find the vector $\hat{\boldsymbol{\alpha}}$ that minimizes this equation (using a QP solver), use Equation 5-7 to compute $\hat{\mathbf{w}}$ and $\hat{b}$ that minimize the primal problem.

Equation 5-7. From the dual solution to the primal solution

$$\hat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \mathbf{x}^{(i)}$$

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \hat{\mathbf{w}}^\intercal \mathbf{x}^{(i)}\right)$$

Equation 5-8. Second-degree polynomial mapping

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} {x_1}^2 \\ \sqrt{2}\, x_1 x_2 \\ {x_2}^2 \end{pmatrix}$$

Equation 5-9. Kernel trick for a second-degree polynomial mapping

$$\begin{aligned} \phi(\mathbf{a})^\intercal \phi(\mathbf{b}) \quad &= \begin{pmatrix} {a_1}^2 \\ \sqrt{2}\, a_1 a_2 \\ {a_2}^2 \end{pmatrix}^\intercal \begin{pmatrix} {b_1}^2 \\ \sqrt{2}\, b_1 b_2 \\ {b_2}^2 \end{pmatrix} = {a_1}^2 {b_1}^2 + 2 a_1 b_1 a_2 b_2 + {a_2}^2 {b_2}^2 \\ &= \left(a_1 b_1 + a_2 b_2\right)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^\intercal \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = \left(\mathbf{a}^\intercal \mathbf{b}\right)^2 \end{aligned}$$

This paragraph is located in the "Kernelized SVMs" section, on page 170:

Here is the key insight: if you apply the transformation ϕ to all training instances, then the dual problem (see Equation 5-6) will contain the dot product $\phi(\mathbf{x}^{(i)})^\intercal\ \phi(\mathbf{x}^{(j)})$. But if ϕ is the second-degree polynomial transformation defined in Equation 5-8, then you can replace this dot product of transformed vectors simply by $\left(\mathbf{x}^{(i)\intercal}\mathbf{x}^{(j)}\right)^2$. So, you don't need to transform the training instances at all; just replace the dot product by its square in Equation 5-6. The result will be strictly the same as if you had gone through the trouble of transforming the training set then fitting a linear SVM algorithm, but this trick makes the whole process much more computationally efficient.

Equation 5-10. Common kernels

$$
\begin{aligned}
\text{Linear:} \quad & K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^\intercal \mathbf{b} \\
\text{Polynomial:} \quad & K(\mathbf{a}, \mathbf{b}) = \left(\gamma \mathbf{a}^\intercal \mathbf{b} + r\right)^d \\
\text{Gaussian RBF:} \quad & K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right) \\
\text{Sigmoid:} \quad & K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^\intercal \mathbf{b} + r\right)
\end{aligned}
$$

Equation 5-11. Making predictions with a kernelized SVM

$$
\begin{aligned}
h_{\widehat{\mathbf{w}}, \hat{b}}\left(\phi\left(\mathbf{x}^{(n)}\right)\right) &= \widehat{\mathbf{w}}^\intercal \phi\left(\mathbf{x}^{(n)}\right) + \hat{b} = \left(\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \phi\left(\mathbf{x}^{(i)}\right)\right)^\intercal \phi\left(\mathbf{x}^{(n)}\right) + \hat{b} \\
&= \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \left(\phi\left(\mathbf{x}^{(i)}\right)^\intercal \phi\left(\mathbf{x}^{(n)}\right)\right) + \hat{b} \\
&= \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \hat{\alpha}^{(i)} t^{(i)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(n)}\right) + \hat{b}
\end{aligned}
$$

Equation 5-12. Using the kernel trick to compute the bias term

$$
\begin{aligned}
\hat{b} &= \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \widehat{\mathbf{w}}^\intercal \phi\left(\mathbf{x}^{(i)}\right)\right) = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \left(\sum_{j=1}^{m} \hat{\alpha}^{(j)} t^{(j)} \phi\left(\mathbf{x}^{(j)}\right)\right)^\intercal \phi\left(\mathbf{x}^{(i)}\right)\right) \\
&= \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \sum_{\substack{j=1 \\ \hat{\alpha}^{(j)} > 0}}^{m} \hat{\alpha}^{(j)} t^{(j)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)\right)
\end{aligned}
$$

Equation 5-13. Linear SVM classifier cost function

$$
J(\mathbf{w}, b) = \frac{1}{2} \mathbf{w}^\intercal \mathbf{w} \quad + \quad C \sum_{i=1}^{m} max\left(0, 1 - t^{(i)}\left(\mathbf{w}^\intercal \mathbf{x}^{(i)} + b\right)\right)
$$

CHAPTER 6
Decision Trees

Equation 6-1. Gini impurity

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

Equation 6-2. CART cost function for classification

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

$$\text{where } \begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$$

Equation 6-3. Entropy

$$H_i = - \sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2 \left(p_{i,k}\right)$$

Equation 6-4. CART cost function for regression

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

CHAPTER 7

Ensemble Learning and Random Forests

Equation 7-1. Weighted error rate of the j^{th} predictor

$$r_j = \frac{\displaystyle\sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\displaystyle\sum_{i=1}^{m} w^{(i)}} \quad \text{where } \hat{y}_j^{(i)} \text{ is the } j^{\text{th}} \text{ predictor's prediction for the } i^{\text{th}} \text{ instance.}$$

Equation 7-2. Predictor weight

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

Equation 7-3. Weight update rule

$$\begin{aligned} &\text{for } i = 1, 2, \cdots, m \\ &w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \widehat{y_j}^{(i)} = y^{(i)} \\ w^{(i)} \exp\left(\alpha_j\right) & \text{if } \widehat{y_j}^{(i)} \neq y^{(i)} \end{cases} \end{aligned}$$

Then all the instance weights are normalized (i.e., divided by $\sum_{i=1}^{m} w^{(i)}$).

Equation 7-4. AdaBoost predictions

$$\hat{y}(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x}) = k}}^{N} \alpha_j \quad \text{where } N \text{ is the number of predictors.}$$

CHAPTER 8

Dimensionality Reduction

Equation 8-1. Principal components matrix

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

Equation 8-2. Projecting the training set down to d dimensions

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$$

Equation 8-3. PCA inverse transformation, back to the original number of dimensions

$$\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}}\mathbf{W}_d{}^{\mathsf{T}}$$

The following paragraph is located in the "LLE" section, on page 231:

Here's how LLE works: for each training instance $\mathbf{x}^{(i)}$, the algorithm identifies its k closest neighbors (in the preceding code $k = 10$), then tries to reconstruct $\mathbf{x}^{(i)}$ as a linear function of these neighbors. More specifically, it finds the weights $w_{i,j}$ such that the squared distance between $\mathbf{x}^{(i)}$ and $\sum_{j=1}^{m} w_{i,j}\mathbf{x}^{(j)}$ is as small as possible, assuming $w_{i,j} = 0$ if $\mathbf{x}^{(j)}$ is not one of the k closest neighbors of $\mathbf{x}^{(i)}$. Thus the first step of LLE is the constrained optimization problem described in Equation 8-4, where $\mathbf{W}$ is the weight matrix containing all the weights $w_{i,j}$. The second constraint simply normalizes the weights for each training instance $\mathbf{x}^{(i)}$.

Equation 8-4. LLE step one: linearly modeling local relationships

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\mathbf{x}^{(i)} - \sum_{j=1}^{m} w_{i,j} \mathbf{x}^{(j)} \right)^2$$

$$\text{subject to } \begin{cases} w_{i,j} = 0 & \text{if } \mathbf{x}^{(j)} \text{ is not one of the } k \text{ c.n. of } \mathbf{x}^{(i)} \\ \sum_{j=1}^{m} w_{i,j} = 1 & \text{for } i = 1, 2, \cdots, m \end{cases}$$

After this step, the weight matrix $\widehat{\mathbf{W}}$ (containing the weights $\widehat{w}_{i,j}$) encodes the local linear relationships between the training instances. The second step is to map the training instances into a d-dimensional space (where $d < n$) while preserving these local relationships as much as possible. If $\mathbf{z}^{(i)}$ is the image of $\mathbf{x}^{(i)}$ in this d-dimensional space, then we want the squared distance between $\mathbf{z}^{(i)}$ and $\sum_{j=1}^{m} \widehat{w}_{i,j} \mathbf{z}^{(j)}$ to be as small as possible. This idea leads to the unconstrained optimization problem described in Equation 8-5. It looks very similar to the first step, but instead of keeping the instances fixed and finding the optimal weights, we are doing the reverse: keeping the weights fixed and finding the optimal position of the instances' images in the low-dimensional space. Note that $\mathbf{Z}$ is the matrix containing all $\mathbf{z}^{(i)}$.

Equation 8-5. LLE step two: reducing dimensionality while preserving relationships

$$\widehat{\mathbf{Z}} = \underset{\mathbf{Z}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\mathbf{z}^{(i)} - \sum_{j=1}^{m} \widehat{w}_{i,j} \mathbf{z}^{(j)} \right)^2$$

CHAPTER 9

Unsupervised Learning Techniques

The following bullet point is located in the "Centroid initialization methods", on page 244:

- Take a new centroid $\mathbf{c}^{(i)}$, choosing an instance $\mathbf{x}^{(i)}$ with probability $D\left(\mathbf{x}^{(i)}\right)^2$ / $\sum_{j=1}^{m} D\left(\mathbf{x}^{(j)}\right)^2$, where D($\mathbf{x}^{(i)}$) is the distance between the instance $\mathbf{x}^{(i)}$ and the closest centroid that was already chosen. This probability distribution ensures that instances farther away from already chosen centroids are much more likely be selected as centroids.

The following bullet point is located in the "Gaussian Mixtures" section, on page 260:

- If $z^{(i)}=j$, meaning the i^{th} instance has been assigned to the j^{th} cluster, the location $\mathbf{x}^{(i)}$ of this instance is sampled randomly from the Gaussian distribution with mean $\boldsymbol{\mu}^{(j)}$ and covariance matrix $\boldsymbol{\Sigma}^{(j)}$. This is noted $\mathbf{x}^{(i)} \sim \mathcal{N}\left(\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)}\right)$.

Equation 9-1. Bayesian information criterion (BIC) and Akaike information criterion (AIC)

$$\begin{aligned} BIC &= \log(m)p - 2\log\left(\hat{L}\right) \\ AIC &= 2p - 2\log\left(\hat{L}\right) \end{aligned}$$

Equation 9-2. Bayes' theorem

$$p(\mathbf{z}|\mathbf{X}) = \text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} = \frac{p(\mathbf{X}|\mathbf{z})\, p(\mathbf{z})}{p(\mathbf{X})}$$

Equation 9-3. The evidence p(X) is often intractable

$$p\left(\mathbf{X}\right) = \int p(\mathbf{X}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

Equation 9-4. KL divergence from q(z) to p(z|X)

$$\begin{aligned}
D_{KL}(q \parallel p) &= \mathbb{E}_q\left[\log \frac{q(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{X})}\right] \\
&= \mathbb{E}_q[\log q(\mathbf{z}) - \log p(\mathbf{z} \mid \mathbf{X})] \\
&= \mathbb{E}_q\left[\log q(\mathbf{z}) - \log \frac{p(\mathbf{z}, \mathbf{X})}{p(\mathbf{X})}\right] \\
&= \mathbb{E}_q[\log q(\mathbf{z}) - \log p(\mathbf{z}, \mathbf{X}) + \log p(\mathbf{X})] \\
&= \mathbb{E}_q[\log q(\mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{z}, \mathbf{X})] + \mathbb{E}_q[\log p(\mathbf{X})] \\
&= \mathbb{E}_q[\log p(\mathbf{X})] - \left(\mathbb{E}_q[\log p(\mathbf{z}, \mathbf{X})] - \mathbb{E}_q[\log q(\mathbf{z})]\right) \\
&= \log p(\mathbf{X}) - \text{ELBO} \\
&\quad \text{where ELBO} = \mathbb{E}_q[\log p(\mathbf{z}, \mathbf{X})] - \mathbb{E}_q[\log q(\mathbf{z})]
\end{aligned}$$

CHAPTER 10

Introduction to Artificial Neural Networks with Keras

Equation 10-1. Common step functions used in Perceptrons (assuming threshold = 0)

$$\text{heaviside}\ (z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \ge 0 \end{cases} \qquad \text{sgn}\ (z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Equation 10-2. Computing the outputs of a fully connected layer

$$h_{\mathbf{W},\,\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$$

Equation 10-3. Perceptron learning rule (weight update)

$$w_{i,\,j}{}^{(\text{next step})} = w_{i,\,j} + \eta\left(y_j - \hat{y}_j\right)x_i$$

CHAPTER 11

Training Deep Neural Networks

Equation 11-1. Glorot initialization (when using the logistic activation function)

Normal distribution with mean 0 and variance $\sigma^2 = \frac{1}{fan_{\text{avg}}}$

Or a uniform distribution between $-r$ and $+r$, with $r = \sqrt{\frac{3}{fan_{\text{avg}}}}$

Equation 11-2. ELU activation function

$$\text{ELU}_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

Equation 11-3. Batch Normalization algorithm

$$\begin{aligned}
&1. \quad \boldsymbol{\mu}_B = \frac{1}{m_B} \sum_{i=1}^{m_B} \mathbf{x}^{(i)} \\
&2. \quad {\boldsymbol{\sigma}_B}^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} \left(\mathbf{x}^{(i)} - \boldsymbol{\mu}_B\right)^2 \\
&3. \quad \widehat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \boldsymbol{\mu}_B}{\sqrt{{\boldsymbol{\sigma}_B}^2 + \varepsilon}} \\
&4. \quad \mathbf{z}^{(i)} = \boldsymbol{\gamma} \otimes \widehat{\mathbf{x}}^{(i)} + \boldsymbol{\beta}
\end{aligned}$$

The following equation is located in the "Batch Normalization" section, on page 343:

$\hat{\mathbf{v}} \leftarrow \hat{\mathbf{v}} \times \text{momentum} + \mathbf{v} \times (1 - \text{momentum})$

Equation 11-4. Momentum algorithm

1. $\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$

Equation 11-5. Nesterov Accelerated Gradient algorithm

1. $\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta} + \beta\mathbf{m})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{m}$

Equation 11-6. AdaGrad algorithm

1. $\mathbf{s} \leftarrow \mathbf{s} + \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\, \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \oslash \sqrt{\mathbf{s} + \varepsilon}$

The following paragraph is located in the "AdaGrad" section, on pages 354 and 355:

The second step is almost identical to Gradient Descent, but with one big difference: the gradient vector is scaled down by a factor of $\sqrt{\mathbf{s} + \varepsilon}$ (the $\oslash$ symbol represents the element-wise division, and ε is a smoothing term to avoid division by zero, typically set to 10^{-10}). This vectorized form is equivalent to simultaneously computing $\theta_i \leftarrow \theta_i - \eta\, \partial J(\boldsymbol{\theta}) / \partial\theta_i / \sqrt{s_i + \varepsilon}$ for all parameters θ_i.

Equation 11-7. RMSProp algorithm

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1 - \beta)\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$
2. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\, \nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta}) \oslash \sqrt{\mathbf{s} + \varepsilon}$

Equation 11-8. Adam algorithm

$$\begin{array}{ll}
1. & \mathbf{m} \leftarrow \beta_1 \mathbf{m} - \left(1 - \beta_1\right) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\
2. & \mathbf{s} \leftarrow \beta_2 \mathbf{s} + \left(1 - \beta_2\right) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\
3. & \widehat{\mathbf{m}} \leftarrow \dfrac{\mathbf{m}}{1 - {\beta_1}^t} \\
4. & \widehat{\mathbf{s}} \leftarrow \dfrac{\mathbf{s}}{1 - {\beta_2}^t} \\
5. & \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\, \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}} + \varepsilon}
\end{array}$$

CHAPTER 12

Custom Models and Training with TensorFlow

There are no equations in this chapter.

CHAPTER 13

Loading and Preprocessing Data with TensorFlow

There are no equations in this chapter.

CHAPTER 14

Deep Computer Vision Using Convolutional Neural Networks

Equation 14-1. Computing the output of a neuron in a convolutional layer

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \times w_{u,v,k',k} \quad \text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

Equation 14-2. Local response normalization (LRN)

$$b_i = a_i \left(k + \alpha \sum_{j=j_\text{low}}^{j_\text{high}} {a_j}^2 \right)^{-\beta} \quad \text{with} \begin{cases} j_\text{high} = \min\left(i + \frac{r}{2}, f_n - 1\right) \\ j_\text{low} = \max\left(0, i - \frac{r}{2}\right) \end{cases}$$

CHAPTER 15

Processing Sequences Using RNNs and CNNs

Equation 15-1. Output of a recurrent layer for a single instance

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x{}^{\mathsf{T}}\mathbf{x}_{(t)} + \mathbf{W}_y{}^{\mathsf{T}}\mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

Equation 15-2. Outputs of a layer of recurrent neurons for all instances in a mini-batch

$$\begin{aligned}\mathbf{Y}_{(t)} &= \phi\left(\mathbf{X}_{(t)}\mathbf{W}_x + \mathbf{Y}_{(t-1)}\mathbf{W}_y + \mathbf{b}\right)\\ &= \phi\left(\left[\mathbf{X}_{(t)} \quad \mathbf{Y}_{(t-1)}\right]\mathbf{W} + \mathbf{b}\right) \text{ with } \mathbf{W} = \begin{bmatrix}\mathbf{W}_x\\ \mathbf{W}_y\end{bmatrix}\end{aligned}$$

Equation 15-3. LSTM computations

$$\begin{aligned}\mathbf{i}_{(t)} &= \sigma\left(\mathbf{W}_{xi}{}^{\mathsf{T}}\mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{\mathsf{T}}\mathbf{h}_{(t-1)} + \mathbf{b}_i\right)\\ \mathbf{f}_{(t)} &= \sigma\left(\mathbf{W}_{xf}{}^{\mathsf{T}}\mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{\mathsf{T}}\mathbf{h}_{(t-1)} + \mathbf{b}_f\right)\\ \mathbf{o}_{(t)} &= \sigma\left(\mathbf{W}_{xo}{}^{\mathsf{T}}\mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{\mathsf{T}}\mathbf{h}_{(t-1)} + \mathbf{b}_o\right)\\ \mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{\mathsf{T}}\mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{\mathsf{T}}\mathbf{h}_{(t-1)} + \mathbf{b}_g\right)\\ \mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}\\ \mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right)\end{aligned}$$

Equation 15-4. GRU computations

$$
\begin{aligned}
\mathbf{z}_{(t)} &= \sigma\left(\mathbf{W}_{xz}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_{z}\right)\\
\mathbf{r}_{(t)} &= \sigma\left(\mathbf{W}_{xr}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{\top}\mathbf{h}_{(t-1)} + \mathbf{b}_{r}\right)\\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{\top}\mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{\top}\left(\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}\right) + \mathbf{b}_{g}\right)\\
\mathbf{h}_{(t)} &= \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + \left(1 - \mathbf{z}_{(t)}\right) \otimes \mathbf{g}_{(t)}
\end{aligned}
$$

CHAPTER 16

Natural Language Processing with RNNs and Attention

Equation 16-1. Attention mechanisms

$$\begin{aligned}
\widetilde{\mathbf{h}}_{(t)} &= \sum_i \alpha_{(t,i)} \mathbf{y}_{(i)} \\
\text{with } \alpha_{(t,i)} &= \frac{\exp\left(e_{(t,i)}\right)}{\sum_{i'} \exp\left(e_{(t,i')}\right)} \\
\text{and } e_{(t,i)} &= \begin{cases} {\mathbf{h}_{(t)}}^\top \mathbf{y}_{(i)} & \textit{dot} \\ {\mathbf{h}_{(t)}}^\top \mathbf{W}\, \mathbf{y}_{(i)} & \textit{general} \\ \mathbf{v}^\top \tanh\left(\mathbf{W}\left[\mathbf{h}_{(t)}; \mathbf{y}_{(i)}\right]\right) & \textit{concat} \end{cases}
\end{aligned}$$

Equation 16-2. Sine/cosine positional encodings

$$\begin{aligned}
P_{p,2i} &= \sin\left(p/10000^{2i/d}\right) \\
P_{p,2i+1} &= \cos\left(p/10000^{2i/d}\right)
\end{aligned}$$

Equation 16-3. Scaled Dot-Product Attention

$$\text{Attention}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_{keys}}}\right)\mathbf{V}$$

CHAPTER 17

Representation Learning and Generative Learning Using Autoencoders and GANs

Equation 17-1. Kullback–Leibler divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

Equation 17-2. KL divergence between the target sparsity p and the actual sparsity q

$$D_{\mathrm{KL}}(p \parallel q) = p \ \log \frac{p}{q} + (1 - p) \log \frac{1 - p}{1 - q}$$

Equation 17-3. Variational autoencoder's latent loss

$$\mathscr{L} = -\frac{1}{2} \sum_{i=1}^{n} \left[1 + \log\left(\sigma_i^{\,2}\right) - \sigma_i^{\,2} - \mu_i^{\,2}\right]$$

Equation 17-4. Variational autoencoder's latent loss, rewritten using $\boldsymbol{\gamma} = \log(\boldsymbol{\sigma}^2)$

$$\mathscr{L} = -\frac{1}{2} \sum_{i=1}^{n} \left[1 + \gamma_i - \exp\left(\gamma_i\right) - \mu_i^{\,2}\right]$$

The following list item is located in the "Progressive Growing of GANs", on page 603:

Equalized learning rate

Initializes all weights using a simple Gaussian distribution with mean 0 and standard deviation 1 rather than using He initialization. However, the weights are scaled down at runtime (i.e., every time the layer is executed) by the same factor as in He initialization: they are divided by $\sqrt{2/n_{\text{inputs}}}$, where n_{inputs} is the number of inputs to the layer. \[…\]

CHAPTER 18
Reinforcement Learning

Equation 18-1. Bellman Optimality Equation

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma \cdot V^*(s')\right] \quad \text{for all } s$$

Equation 18-2. Value Iteration algorithm

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma \cdot V_k(s')\right] \quad \text{for all } s$$

Equation 18-3. Q-Value Iteration algorithm

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma \cdot \max_{a'} Q_k(s', a')\right] \quad \text{for all } (s, a)$$

Once you have the optimal Q-Values, defining the optimal policy, noted $\pi^*(s)$, is trivial: when the agent is in state s, it should choose the action with the highest Q-Value for that state: $\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s, a)$.

Equation 18-4. TD Learning algorithm

$$V_{k+1}(s) \leftarrow (1-\alpha)V_k(s) + \alpha\left(r + \gamma \cdot V_k(s')\right)$$

or, equivalently:

$$V_{k+1}(s) \leftarrow V_k(s) + \alpha \cdot \delta_k(s, r, s')$$

with $\delta_k(s, r, s') = r + \gamma \cdot V_k(s') - V_k(s)$

The following paragraph is located in the "Temporal Difference Learning" on page 630:

A more concise way of writing the first form of this equation is to use the notation $a \underset{\alpha}{\leftarrow} b$, which means $a_{k+1} \leftarrow (1 - \alpha) \cdot a_k + \alpha \cdot b_k$. So, the first line of Equation 18-4 can be rewritten like this: $V(s) \underset{\alpha}{\leftarrow} r + \gamma \cdot V(s')$.

Equation 18-5. Q-Learning algorithm

$$Q(s, a) \underset{\alpha}{\leftarrow} r + \gamma \cdot \max_{a'} \; Q(s', a')$$

Equation 18-6. Q-Learning using an exploration function

$$Q(s, a) \underset{\alpha}{\leftarrow} r + \gamma \cdot \max_{a'} \; f(Q(s', a'), N(s', a'))$$

Equation 18-7. Target Q-Value

$$Q_{\text{target}}(s, a) = r + \gamma \cdot \max_{a'} \; Q_{\boldsymbol{\theta}}(s', a')$$

CHAPTER 19
Training and Deploying TensorFlow Models at Scale

There are no equations in this chapter.

APPENDIX A
Exercise Solutions

The following list item is the solution to exercise 7 from chapter 5, and is located on page 724:

- Let's call the QP parameters for the hard margin problem $\mathbf{H}'$, $\mathbf{f}'$, $\mathbf{A}'$, and $\mathbf{b}'$ (see "Quadratic Programming" on page 167). The QP parameters for the soft margin problem have m additional parameters ($n_p = n + 1 + m$) and m additional constraints ($n_c = 2m$). They can be defined like so:
  - $\mathbf{H}$ is equal to $\mathbf{H}'$, plus m columns of 0s on the right and m rows of 0s at the bottom: $\mathbf{H} = \begin{pmatrix} \mathbf{H}' & 0 & \cdots \\ 0 & 0 & \\ \vdots & & \ddots \end{pmatrix}$
  - $\mathbf{f}$ is equal to $\mathbf{f}'$ with m additional elements, all equal to the value of the hyperparameter C.
  - $\mathbf{b}$ is equal to $\mathbf{b}'$ with m additional elements, all equal to 0.
  - $\mathbf{A}$ is equal to $\mathbf{A}'$, with an extra $m \times m$ identity matrix $\mathbf{I}_m$ appended to the right, $-\mathbf{I}_m$ just below it, and the rest filled with 0s: $\mathbf{A} = \begin{pmatrix} \mathbf{A}' & \mathbf{I}_m \\ \mathbf{0} & -\mathbf{I}_m \end{pmatrix}$

APPENDIX B

Machine Learning Project Checklist

There are no equations in this appendix.

APPENDIX C

SVM Dual Problem

The following paragraph is located on page 761:

In this example the partial derivatives are: $\begin{cases} \frac{\partial}{\partial x} g(x, y, \alpha) = 2x - 3\alpha \\ \frac{\partial}{\partial y} g(x, y, \alpha) = 2 - 2\alpha \\ \frac{\partial}{\partial \alpha} g(x, y, \alpha) = -3x - 2y - 1 \end{cases}$

Equation C-1. Generalized Lagrangian for the hard margin problem

$$\mathscr{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^\intercal \mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)} \left(t^{(i)} \left(\mathbf{w}^\intercal \mathbf{x}^{(i)} + b \right) - 1 \right)$$

$$\text{with} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

The following paragraph is located on page 762:

Just like with the Lagrange multipliers method, you can compute the partial derivatives and locate the stationary points. If there is a solution, it will necessarily be among the stationary points $\left(\widehat{\mathbf{w}}, \hat{b}, \hat{\alpha}\right)$ that respect the *KKT conditions*:

- Respect the problem's constraints: $t^{(i)}\left(\widehat{\mathbf{w}}^\intercal \mathbf{x}^{(i)} + \hat{b}\right) \geq 1 \quad \text{for } i = 1, 2, \ldots, m.$
- Verify $\hat{\alpha}^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m.$
- Either $\hat{\alpha}^{(i)} = 0$ or the i^{th} constraint must be an *active constraint*, meaning it must hold by equality: $t^{(i)}\left(\widehat{\mathbf{w}}^\intercal \mathbf{x}^{(i)} + \hat{b}\right) = 1$. This condition is called the *complementary*

slackness condition. It implies that either $\hat{\alpha}^{(i)} = 0$ or the i^{th} instance lies on the boundary (it is a support vector).

Equation C-2. Partial derivatives of the generalized Lagrangian

$$\begin{aligned}\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}, b, \alpha) = \mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)}t^{(i)}\mathbf{x}^{(i)}\\ \frac{\partial}{\partial b}\mathcal{L}(\mathbf{w}, b, \alpha) = -\sum_{i=1}^{m} \alpha^{(i)}t^{(i)}\end{aligned}$$

Equation C-3. Properties of the stationary points

$$\begin{aligned}\widehat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}^{(i)}t^{(i)}\mathbf{x}^{(i)}\\ \sum_{i=1}^{m} \hat{\alpha}^{(i)}t^{(i)} = 0\end{aligned}$$

Equation C-4. Dual form of the SVM problem

$$\mathcal{L}\left(\widehat{\mathbf{w}}, \hat{b}, \alpha\right) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}\mathbf{x}^{(i)\intercal}\mathbf{x}^{(j)} \quad - \quad \sum_{i=1}^{m} \alpha^{(i)}$$
$$\text{with} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

Equation C-5. Bias term estimation using the dual form

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1\\ \hat{\alpha}^{(i)} > 0}}^{m} \left[t^{(i)} - \widehat{\mathbf{w}}^{\intercal}\mathbf{x}^{(i)}\right]$$

APPENDIX D
Autodiff

Equation D-1. Partial derivatives of f(x, y)

$$\frac{\partial f}{\partial x} = \frac{\partial\left(x^2 y\right)}{\partial x} + \frac{\partial y}{\partial x} + \frac{\partial 2}{\partial x} = y\frac{\partial\left(x^2\right)}{\partial x} + 0 + 0 = 2xy$$

$$\frac{\partial f}{\partial y} = \frac{\partial\left(x^2 y\right)}{\partial y} + \frac{\partial y}{\partial y} + \frac{\partial 2}{\partial y} = x^2 + 1 + 0 = x^2 + 1$$

Equation D-2. Definition of the derivative of a function h(x) at point x_0

$$\begin{aligned} h'\left(x_0\right) &= \lim_{x \to x_0} \frac{h(x) - h\left(x_0\right)}{x - x_0} \\ &= \lim_{\varepsilon \to 0} \frac{h\left(x_0 + \varepsilon\right) - h\left(x_0\right)}{\varepsilon} \end{aligned}$$

Equation D-3. A few operations with dual numbers

$$\lambda(a + b\varepsilon) = \lambda a + \lambda b\varepsilon$$

$$(a + b\varepsilon) + (c + d\varepsilon) = (a + c) + (b + d)\varepsilon$$

$$(a + b\varepsilon) \times (c + d\varepsilon) = ac + (ad + bc)\varepsilon + (bd)\varepsilon^2 = ac + (ad + bc)\varepsilon$$

Equation D-4. Chain rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial n_i} \times \frac{\partial n_i}{\partial x}$$

APPENDIX E

Other Popular ANN Architectures

Equation E-1. Probability that the ith neuron will output 1

$$p\left(s_i^{(\text{next step})} = 1\right) = \sigma\left(\frac{\sum_{j=1}^{N} w_{i,j} s_j + b_i}{T}\right)$$

Equation E-2. Contrastive divergence weight update

$$w_{i,j} \leftarrow w_{i,j} + \eta\left(\mathbf{x}\mathbf{h}^\intercal - \mathbf{x}'\mathbf{h}'^\intercal\right)$$

APPENDIX F

Special Data Structures

There are no equations in this appendix.

APPENDIX G

TensorFlow Graphs

There are no equations in this appendix.