Machine Learning

Personal Formula Collection

Patrick Bucher

December 31, 2020

Contents

1 Linear Regression

1.1 One Variable

Prediction:

$$y = h(x) = \theta_0 + \theta_1 x = \theta^T x$$

Cost Function (Squared Error):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient Descent:

$$\theta = \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

1.2 Multiple Variables

Prediction:

$$y = h(x) = \theta_0 + \theta_1 x_1 = + \cdots + \theta_n x_n = \theta^T x$$

Cost Function:

$$J(\theta_j) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_j^{(i)}) - y_j^{(i)})^2$$

Gradient Descent:

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}, j := 0 \ldots n$$

An additional feature $x_0 = 1$ is introduced, so that the vector x becomes $n + 1$ dimensional, which simplifies the matrix calculations.

Normal Equation:

$$\theta = (X^T X)^{-1} X^T y$$

Octave (Complexity with n features: $O(n^3)$):

```
theta = pinv(X'*X)*X'*y
```

1.3 Normalization

$$x_i = \frac{x_i - \mu_i}{s_i}$$

Octave:

```
X = (X - mean(X)) ./ std(X)
```

2 Classification

Binary Classification: $y \in \{0, 1\}$, where 0 signifies negative or absent, and 1 signifies positive or present.

2.1 Logistic Regression

$$0 \leq h_{\theta(x)} \leq 1$$

Sigmoid Activation Function g (with asymptotes at y 0 and 1, to be interpreted as probabilities):

$$h_\theta = g(\theta^T x), g(z) = \frac{1}{1 + e^{-z}}$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Cost Function: $-log(h_\theta(x))$ for $y = 1$ and $-log(1 - h_\theta(x))$ for $y = 0$, combined:

$$C(h_\theta(x), y) = -y \cdot log(h_\theta(x)) - (1 - y) \cdot log(1 - h_\theta(x))$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} C(h_\theta(x^{(i)}), y^{(i)})$$

With maximum likelihood estimation:

$$J(\theta) = \frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \cdot log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot log(1 - h_\theta(x^{(i)})) \right]$$

Prediction:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Gradient Descent (for each j in θ):

$$\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Vectorized:

$$\theta := \theta - \frac{\alpha}{m}\sum_{i=1}^{m}\left[(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}\right]$$

$$\theta := \theta - \frac{\alpha}{m}X^T(g(X\theta) - \vec{y})$$

2.1.1 Regularization (Gradient Descent)

Regularization mitigates the problem of overfitting for higher-order polynomials.
Regularization term (only regularize θ_j for $j \geq 1$, but not θ_0):

$$\lambda\sum_{j=1}^{m}\theta_j^2$$

Regularized Cost Function:

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Regularized Gradient Descent (for θ_j with $j \geq 1$):

$$\theta_0 := \theta_0 - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right]$$

$$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left((h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right]$$

2.1.2 Regularization (Normal Equation)

To regularize using the normal equation, $(n+1)(n+1)$ matrix L with i rows and j columns and the values 1 (for $i = j \wedge i \geq 1 \wedge j \geq 1$) and 0 (all the other indices), respectively, has to be created. (This is an identity matrix of size $n+1$ with the value at $(0,0)$ set to 0.)

$$\theta = (X^TX + \lambda L)^{-1}X^Ty$$

With regularization, the matrix is always inversible.

3 Neural Networks

Definitions:

- x_0: bias unit
- $a_i^{(j)}$: activation unit i of layer j
- $\Theta^{(j)}$: weight matrix between layer j and $j+1$

Given layer j with s_j units, and layer $j+1$ with s_{j+1} units, the matrix $\Theta^{(j)}$ has the dimensions $s_{j+1} \times (s_j + 1)$.

3.1 Activation

Neural network with three units in the one hidden layer:

$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \dots) \\
a_2^{(2)} &= g(\Theta_{20}^{(2)} x_0 + \Theta_{21}^{(2)} x_1 + \dots) \\
a_3^{(2)} &= g(\Theta_{30}^{(3)} x_0 + \Theta_{31}^{(3)} x_1 + \dots) \\
h_\Theta(x) &= a_1^3 = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \dots)
\end{aligned}
$$

3.1.1 Vectorization

With (forward propagation):

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

And:

$$z_1^{(2)} = \Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3$$

Follows:

$$a_1^{(2)} = g(z_1^{(2)})$$

So that:

$$z^{(2)} = \Theta^{(1)} x = \Theta^{(1)} a^{(1)}$$

Output layer:

$$h_\Theta = a^{(3)} = g(z^{(3)})$$

3.2 Cost Function

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) log(1-(h_\Theta(x^{(i)}))_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

With $(h_\Theta(x))_i$ being the i^{th} output. Note that regularization is *not* added to the bias unit, i.e. only for $j \geq 1$.

3.3 Forward Propagation

With a single training example (x, y). The first activation is the input (a bias unit $a_0^{(1)} = 1$ must be added before):

$$a^{(1)} = x$$

The second activation is computed using Θ and the sigmoid function $g(z)$:

$$z^{(2)} = \Theta^{(2)} a^{(2)}$$
$$a^{(2)} = g(z^{(2)})$$

The bias unit $a_0^{(2)} = 1$ must be added again, then the further activations (l) are computed:

$$z^{(l)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l)} = g(z^{(l)})$$

Finally, the output (layer L) is computed:

$$z^{(L)} = \Theta^{(L)} a^{(L)}$$
$$a^{(L)} = g(z^{(L)}) = h_\Theta(x)$$

3.4 Backpropagation

The δ for the rightmost layer L is computed as:

$$\delta^L = a^{(L)} - y$$

The further δ values are computed from right to left, down to $l = 2$ (no δ for the input layer):

$$\delta^{(l)} = \delta^{(l+1)} \Theta^{(l)} g'(z^{(l)})$$

With (bias unit included in $a^{(l)}$):

$$g'(z^{(l)}) = a^{(l)}(1 - a^{(l)})$$

The Δ values are computed as ($a^{(l)}$ *without* bias unit):

$$\Delta^{(l)} = (\delta^{(l+1)})^T a^{(l)}$$

Finally, the gradients D for $j \geq 1$ are computed as follows:

$$D_{ij}^{(l)} = \frac{1}{m}(\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)})$$

And without regularization for $j = 0$, respectively:

$$D_{ij}^{(l)} = \frac{1}{m}(\Delta_{ij}^{(l)}$$

Which is the partial derivative of the cost function:

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)}$$

3.5 Gradient Checking

Estimate the derivative of $J(\Theta)$ with $\varepsilon \approx 10^{-4}$ (two-sided difference):

$$\frac{d}{d\Theta}J(\Theta) \approx \frac{J(\Theta + \varepsilon) - J(\Theta - \varepsilon)}{2\varepsilon}$$

The result should only deviate from the D values by a rounding margin.

3.6 Random initialization

When working with neural networks, Θ must be initialized to a random value symmetrically around 0. A (10×11) matrix is initialized as follows (Octave):

```
init_epsilon = 0.1;
Theta = rand(10,11) * (2 * init_epsilon) - init_epsilon;
```

4 Error Metrics

Confusion Matrix:

		actual	
		1	0
prediction	1	true positive	false positive
	0	false negative	true negative

Precision ($0 \leq P \leq 1$):

$$P = \frac{tp}{tp + fp}$$

Recall ($0 \leq R \leq 1$):

$$R = \frac{tp}{tp + fn}$$

F_1 Score ($0 \leq F_1 \leq 1$):

$$F_1 = 2\frac{PR}{P + R}$$

Some rules of thumb:

- A higher classification threshold leads to a higher precision and a lower recall.
- A lower classification threshold leads to a lower precision and a higher recall.
- Many features can help to lower the bias.
- Many training examples can help to lower the variance.
- If a human expert can predict y based on x, more training data can help.

5 Support Vector Machines

The prediction yields 0 and 1 rather than probabilities. Cost Functions with Safety Margins (*Large Margin Classifier*):

$$\begin{aligned} &\text{cost}_0(\theta^T x^{(i)}) : 1 \quad \text{if} \quad \theta^T x \leq -1, \quad \text{else} \quad 0 \\ &\text{cost}_1(\theta^T x^{(i)}) : 1 \quad \text{if} \quad \theta^T x \geq +1, \quad \text{else} \quad 0 \end{aligned}$$

Minimize θ ($C = \frac{1}{\lambda}$):

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

5.1 Kernels

Calculate features depending on proximity (similarity function) using landmarks ($l^{(i)} = x^{(i)}$) with the *Gaussian kernel* (squared euclidian distance $||x - l^{(i)}||^2$):

$$f_1 = \text{sim}(x, l^{(i)}) = \exp\left(- \frac{||x - l^{(i)}||^2}{2\sigma^2} \right)$$

5.2 Choice of Parameters

- C
  - large C: low bias, high variance (small λ)
  - small C: high bias, low variance (large λ)
- σ^2
  - large σ^2: high bias, low variance (flat gaussian curve)
  - small σ^2: low bias, high variance (abrupt gaussian curve)

6 K-Means

Input: Training Set ($x^{(i)}, x^{(2)}, \ldots, x^{(m)}, x \in \mathbb{R}^n$), number of clusters ($K$); Algorithm:

1. initialize centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$ (pick random training examples)
2. for $i = 1..m$: set $c^{(i)}$ by proximity to μ ($\min_k ||x^{(i)} - \mu_k||$) (assign index of closest centroid)
3. for $j = 1..k$: move μ_j to mean of xs with $c = k$
4. repeat steps 1 to 3

Repeat the algorithm with different random initializations in order to find a global rather than just a local minimum of the cost function ("Distortion of K-Means Algorithm"):

$$J(c^{(1)}, c^{(2)}, \ldots, c^{(m)}, \mu_1, \mu_2, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

7 Principal Component Analysis

Idea: Reduce input matrix $x \in \mathbb{R}^n$ to $z \in \mathbb{R}^k$ with $k < n$ to reduce amount of features while retaining as much variance as possible to save storage, memory, processing power and for easier visualization. Algorithm:

1. preprocess data: mean normalization and feature scaling: $x_j := \frac{x_j - \mu_j}{\sigma}$
2. compute covariance matrix $\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$ (Octave: `Sigma=(1/m)*X'*X;`)
3. compute eigenvectors of Σ (Octave: `[U,S,V]=svd(Sigma);`)
4. take first k vector (i.e. columns) of U (Octave: `Ureduce=U(:,1:k);`)
5. compute $z = U_{\text{reduce}}^T x^{(i)} \in \mathbb{R}^k$ (Compression, Octave: `z=Ureduce'*X;`)
6. reconstruct $x_{\text{approx}} \in \mathbb{R}^n$ from $z \in \mathbb{R}^k$: $x_{\text{approx}} = U_{\text{reduce}} z \approx x$ (Octave: `Xapprox=Ureduce*z`)

The bias unit $x_0 = 1$ is omitted.

7.1 Choosing the Number of Principal Components

Squared Projection Error:

$$\frac{1}{m}\sum_{i=1}^{m}||x^{(i)} - x_{\text{approx}}^{(i)}||^2$$

Total variation in the data:

$$\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}||^2$$

In order to retain 99% of the variance, choose $k = 1..(n-1)$ to be the smallest value, so that:

$$\frac{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)} - x_{\text{approx}}^{(i)}||^2}{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}||^2} \leq 0.01$$

Algorithm (for $k = 1..(n-1)$):

1. compute $U_{\text{reduce}}, z^{(1)}, \ldots, z^{(m)}, x_{\text{approx}}^{(i)}, \ldots, x_{\text{approx}}^{(m)}$
2. compute variance thus retained (see formula above)
3. finish if variance $\leq$ threshold

In Octave, the `S` in `[U,S,V]=svd(Sigma)` is a diagonal matrix that can be used to compute the variance retained:

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01 \quad \text{or} \quad \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

PCA should not be used to address the issue of overfitting; use regularization instead. PCA should only be introduced if really needed.

8 Anomaly Detection

Given a dataset $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, x_{test} is anomalous if:

$$p(x_{\text{test}}) < \varepsilon$$

or *not* anomalous (i.e. normal) if:

$$p(x_{\text{test}}) \geq \varepsilon$$

With $x \sim \mathcal{N}(\mu, \sigma^2)$ (Gaussian):

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The parameters μ and σ^2 can be guessed from the dataset:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \quad \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

Given an *unlabeled* traning set $x \in \mathbb{R}^n, x \sim \mathcal{N}(\mu, \sigma^2)$:

$$p(x) = p(x_1; \mu_1, \sigma_1^2)p(x_2; \mu_2, \sigma_2^2)\dots p(x_n; \mu_n, \sigma_n^2) = \prod_{j=1}^{n} p(x_i; \mu_i, \sigma_i^2) = \prod_{j=1}^{n} \frac{e^{-\frac{(x_j-\mu_j)^2}{2\sigma_j^2}}}{\sqrt{2\pi}\sigma_j}$$

Algorithm:

1. choose indicative features
2. fit parameters $\mu_1, \mu_2, \dots, \mu_n$ and $\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$
3. calculate $p(x)$
4. mark as anomaly if $p(x) < \varepsilon$

For the evaluation using labeled training data, move all anomalous ($y = 1$) examples to the cross validation and test set; only retain normal examples ($y = 0$) in the training set (split usually 60/20/20). Evaluate using precision, recall and F1 Score (accuracy is not indicative due to the skewed distribution of y). Consider finding parameter ε using cross validation (usually 0.05 or 0.01).
The features being used should be normal distributed (plot with Octave: `hist(x, nBins)`). Consider deriving new ($x_3 = \frac{x_1}{x_2}$) or transforming existing features ($x_i = log(x_i + C)$) in order to get normally distributed features.

8.1 Multivariate Gaussian Distribution

If single variables do not qualify a training example as an outlier, but only a combination thereof, using a multivariate Gaussian distribution can help to detect those correctly. With Σ being the covariance matrix, $|\Sigma|$ its determinant, and Σ^{-1} its inverse, the model is defined as:

$$p(x; \mu, \Sigma) = \frac{1}{\sqrt{2\pi}^n |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) \right)$$

If the projection of a two-dimensional univariate Gaussian distribution from the top looks like a circle, a multivariate Gaussian distribution enables to model correlations (elliptic shape denoting the positive or negative correlation).

9 Recommender Systems

9.1 Content-Based Recommenders

Given a table of movies with ratings by user and categorization:

Movie	Alice	Bob	Carol	Dan	Romance	Action
Titanic	4.5	3.5	5.0	3.0	4.5	3.0
Speed	2.0	4.0	2.5	4.5	2.5	5.0
Casablanca	5.0	3.5	4.5	2.5	5.0	2.0
Dawn of the Dead	1.0	5.0	2.0	4.5	1.5	5.0
The Outlaw Josey Wales	2.5	5.0	?	?	2.5	4.5

The following matrices can be derived:

$$Y = \begin{bmatrix} 4.5 & 3.5 & 5.0 & 3.0 \\ 2.0 & 4.0 & 2.5 & 4.5 \\ 5.0 & 3.5 & 4.5 & 2.5 \\ 1.0 & 5.0 & 2.0 & 4.5 \\ 2.5 & 5.0 & ? & ? \end{bmatrix} \in \mathbb{R}^{m_{\text{movies}} \times n_{\text{users}}} \quad \text{and} \quad X = \begin{bmatrix} 4.5 & 3.0 \\ 2.5 & 5.0 \\ 5.0 & 2.0 \\ 1.5 & 5.0 \\ 2.5 & 4.5 \end{bmatrix} \in \mathbb{R}^{m_{\text{movies}} \times k_{\text{genres}}}$$

The matrix R contains the information whether or not a user has rated a movie:

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{m_{\text{movies}} \times n_{\text{users}}} \quad \text{and} \quad \theta \in \mathbb{R}^{n_{\text{users}} \times k_{\text{genres}}}$$

θ is randomly initialized and contains the user's genre preferences to be learned.
Gradient Descent (without regularization):

$$\theta := -\alpha(X\theta^T - Y)X$$

Cost Function (without regularization):

$$J = \frac{1}{2} \sum_{i=1}^{m_{\text{movies}}} (X\theta^T - Y)^2$$

Notice that only entries with $R(i,j) = 1$ must be considered for those calculations.
The missing ratings $R(i,j) = 0$ can be predicted as:

$$X\theta^T$$

9.2 Collaborative Filtering

TODO