ECE 544NA: Pattern Recognition
Lecture 3: September 4

Lecturer: Alexander Schwing Scribe: Matthew Grawe

1 Review

In Lecture 2, we saw that linear regression has utility in describing the underlying functional relationships among the samples within a dataset. It was shown that linear regression penalizes the sum of the squared differences between observed samples and a prescribed model of the data that is linear in its parameters, in addition to (optionally) penalizing the magnitude of the model parameters themselves to curtail any potential overfitting problems (this is called *regularization*). Mathematically, for a dataset of N pairs (x, y), i.e., $\mathcal{D} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{N}$, we saw that linear regression amounted to solving the minimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}} \ \frac{1}{2}\|\mathbf{y} - \mathbf{\Phi}^T\mathbf{w}\|_2^2 + \frac{C}{2}\|\mathbf{w}\|_2^2 \tag{1}$$

where $\mathbf{y} \in \mathbb{R}^N$ $(y_i = y^{(i)})$ is a vector of the y data samples, $\mathbf{\Phi} \in \mathbb{R}^{M\times N}$ is a matrix containing transformations of the x data samples $\mathbf{x} \in \mathbb{R}^N$ $(x_i = x^{(i)})$, $\mathbf{w} \in \mathbb{R}^M$ is a vector of model parameters, and $C \in \mathbb{R}_{\geq 0}$ is the regularization parameter. Each row of $\mathbf{\Phi}$ contains a different transformation of the data samples, i.e., $\Phi_{ki} = [\boldsymbol{\phi}_k(\mathbf{x})]_i$, where $\boldsymbol{\phi}_k$ is the transformation function for row k that inputs the vector $\mathbf{x} \in \mathbb{R}^N$ and outputs a vector having the same dimension. Column i then contains all of the transformations for index i. In general, the transformation functions for a particular index might depend on data from other indices (e.g., gradient transformations). The last row of $\mathbf{\Phi}$ is a vector of ones representing an offset term in the linear data model. Construction of the matrix $\mathbf{\Phi}$ in this manner, in conjunction with Equation 1, implies that we are modeling the data samples $y^{(i)}$ such that

$$y^{(i)} = w_N + \sum_{k=1}^{M} w_k^* \left[\boldsymbol{\phi}_k(\mathbf{x})\right]_i \tag{2}$$

where w_k^* is the k'th element of the vector $\mathbf{w}^* = \left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1}\mathbf{\Phi}\mathbf{y}$ obtained by solving Equation 1 analytically. We also saw that the squared error term in Equation 1 has a perhaps more enlightening probabilistic interpretation in that it maximizes the likelihood of the dataset $\mathcal{D}$ occurring under the assumption that the samples $y^{(i)}$ are independent and identically distributed (i.i.d) Gaussian random variables, i.e., $\underset{\mathbf{w}}{\operatorname{argmax}}\ p(\mathcal{D}) = \underset{\mathbf{w}}{\operatorname{argmin}}\ \frac{1}{2}\|\mathbf{y} - \mathbf{\Phi}^T\mathbf{w}\|_2^2$ when

$$p(\mathcal{D}) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{y} - \mathbf{\Phi}^T\mathbf{w}\|_2^2\right). \tag{3}$$

1.1 Linear Regression for Classification

Lecture 2 also explored the applicability of linear regression to binary classification problems in which the samples $y^{(i)} \in \{-1, 1\}$ represented class labels. We saw that the quadratic loss function

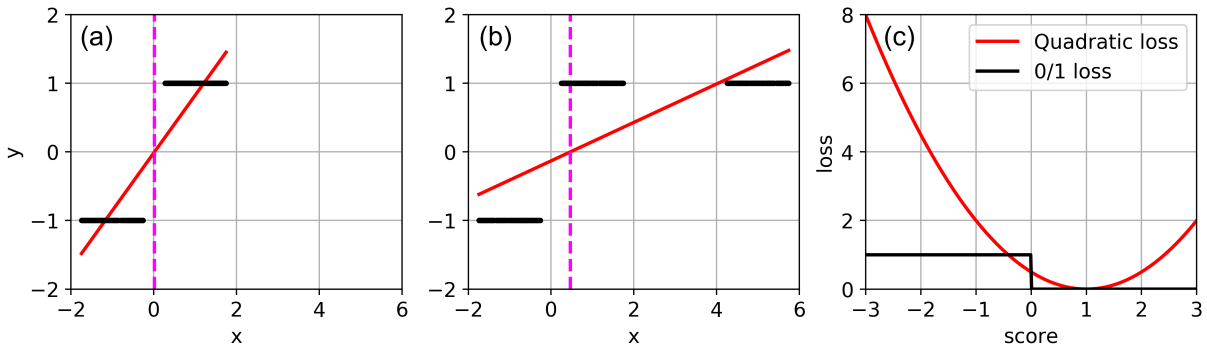

Figure 1: Caveats with using linear regression for classification. Notice that the optimal solution $[x\ 1]\,\mathbf{w}^* = [x\ 1]\,[w_0^*\ w_1^*]^T = w_1^* + w_0^* x$ shown in panel b has been largely influenced by the addition of easily-classified samples with large values of x relative to panel a, causing the decision boundary (the pink vertical line) to undesirably shift to the right. Panel c shows the aforementioned quadratic loss under the $y^{(i)} \in \{-1, 1\}$ constraint, where we see that values of the score function distant from unity are given a large penalization, contrasting perhaps the more desirable "0/1" loss.

inherent to linear regression (i.e., Equation 1) with a decision boundary defined by $y = 0$ thresholding led to unwanted penalization of easily-classified samples (i.e., samples distant from the decision boundary). When y is constrained to ± 1, the squared error cost term in Equation 1, which we denote $J_e(\mathbf{w})$, can be written in the form

$$J_e(\mathbf{w}) = \sum_{i=1}^{N} \ell(y^{(i)}, \mathbf{\Phi}_i^T \mathbf{w}) = \sum_{i=1}^{N} \frac{1}{2} \left[1 - F\left(x^{(i)}, \mathbf{w}, y^{(i)}\right)\right]^2 \tag{4}$$

where $\mathbf{\Phi}_i$ is the i'th column of $\mathbf{\Phi}$, $F\left(x^{(i)}, \mathbf{w}, y^{(i)}\right) = y^{(i)} F(x^{(i)}, \mathbf{w})$ is the *score function*, and $F(x^{(i)}, \mathbf{w}) = \mathbf{\Phi}_i^T \mathbf{w}$ is the *logit function*.

When the cost is written in the form of Equation 4, it is easy to see the unwanted behavior of linear regression when applied to binary classification; larger (in magnitude) elements of $\mathbf{\Phi}_i$ corresponding to values increasingly distant from the decision boundary lead to a large score function and subsequently a large cost. When minimizing $J_e(\mathbf{w})$, the solution $\mathbf{w}^*$ will seek to minimize the distance between these large values and unity, unnecessarily shifting the decision boundary and leading to poor classification performance. This effect is shown in Figure 1. In panel b, notice that the optimal solution $[x\ 1]\,\mathbf{w}^* = [x\ 1]\,[w_0^*\ w_1^*]^T = w_1^* + w_0^* x$ has been largely influenced by the addition of easily-classified samples with large values of x relative to panel a, causing the decision boundary to undesirably shift to the right. Panel c shows the aforementioned quadratic loss under the $y^{(i)} \in \{-1, 1\}$ constraint, where we see that values of the score function distant from unity are given a large penalization, contrasting perhaps the more desirable "0/1" loss.

2 Logistic Regression

The problems underlying the use of linear regression for binary classification are largely overcome by using a more appropriate probabilistic model for the data samples $y^{(i)}$. Because $y^{(i)} \in \{-1, 1\}$, it is perhaps more reasonable to assume that $y^{(i)}$ is distributed according to

$$p(y^{(i)}|x^{(i)}) = \begin{cases} a & y^{(i)} = 1 \\ \frac{1-a}{2} & y^{(i)} = -1 \end{cases} \tag{5}$$

where $a = \left[1 + \exp\left(-\mathbf{w}^T \mathbf{\Phi}_i\right)\right]^{-1}$ is a logistic function. More concisely, this can be written

$$p(y^{(i)}|x^{(i)}) = \frac{1}{1 + \exp\left(-y^{(i)}\mathbf{w}^T \mathbf{\Phi}_i\right)} \qquad y^{(i)} \in \{-1, 1\}. \tag{6}$$

Under the assumption that the data samples are i.i.d, the probability of the dataset becomes

$$p(\mathcal{D}) = \prod_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} p(y^{(i)}|x^{(i)}) = \prod_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \left[1 + \exp\left(-y^{(i)}\mathbf{w}^T \mathbf{\Phi}_i\right)\right]^{-1}. \tag{7}$$

As with linear regression, we seek to choose the value of $\mathbf{w}$ that maximizes the probability that the observed data occurs (call this value $\mathbf{w}^*$). Classification is then performed on a new data sample (x, y) by thresholding Equation 6 evaluated with $\mathbf{w} = \mathbf{w}^*$ at $p(y|x) = 0.5$. Choosing $\mathbf{w}^*$ using this data model is called *logistic regression*. Mathematically, $\mathbf{w}^* = \underset{\mathbf{w}}{\text{argmax}}\ p(\mathcal{D})$, which becomes

$$\mathbf{w}^* = \underset{\mathbf{w}}{\text{argmax}} \prod_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \left[1 + \exp\left(-y^{(i)}\mathbf{w}^T \mathbf{\Phi}_i\right)\right]^{-1}. \tag{8}$$

It turns out that maximization of Equation 8 is rather numerically unstable because individual probabilities within the product are typically small, making the overall product small enough to cause arithmetic underflow in most floating point systems. An equivalent form of Equation 8 can be employed by instead minimizing the negative natural logarithm of $p(\mathcal{D})$ with respect to $\mathbf{w}$. The equivalence follows because the natural logarithm is a monotonically increasing function, and so maximizing $\log[p(\mathcal{D})]$ will yield the same value of $\mathbf{w}$. Inclusion of the negative sign then turns the maximization into a minimization. With this in mind, we rewrite Equation 8 as

$$\begin{aligned}
\mathbf{w}^* &= \underset{\mathbf{w}}{\text{argmax}} \prod_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} p(y^{(i)}|x^{(i)}) \\
&= \underset{\mathbf{w}}{\text{argmax}}\ \log\left(\prod_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} p(y^{(i)}|x^{(i)})\right) \\
&= \underset{\mathbf{w}}{\text{argmax}} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \log\left[p(y^{(i)}|x^{(i)})\right] \\
&= \underset{\mathbf{w}}{\text{argmin}} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} -\log\left[p(y^{(i)}|x^{(i)})\right] \\
&= \underset{\mathbf{w}}{\text{argmin}} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} -\log\left[\left[1 + \exp\left(-y^{(i)}\mathbf{w}^T \mathbf{\Phi}_i\right)\right]^{-1}\right] \\
&= \underset{\mathbf{w}}{\text{argmin}} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \log\left[1 + \exp\left(-y^{(i)}\mathbf{w}^T \mathbf{\Phi}_i\right)\right] \\
&= \underset{\mathbf{w}}{\text{argmin}} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \log\left[1 + \exp\left(-F\left(x^{(i)}, \mathbf{w}, y^{(i)}\right)\right)\right].
\end{aligned} \tag{9}$$

When written in the form of Equation 9, it becomes clear that logistic regression is more appropriate than linear regression for classification problems. Notice that a large value of the score function F leads to small loss, i.e., $\exp(-F) \to 0$ for $F \to \infty$, so that $\log[1 + \exp(-F)] \to \log[1] = 0$ as $F \to \infty$. This means that easily-classified data samples do not contribute much loss, which is the desired behavior.

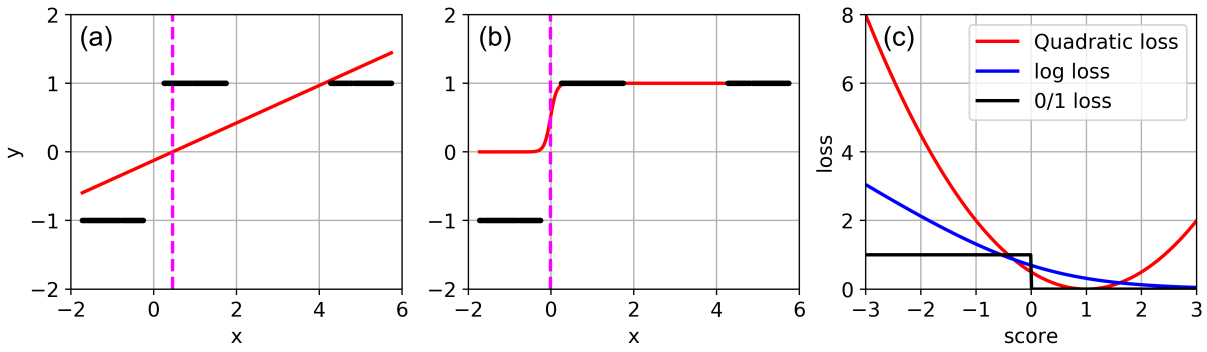

Figure 2: Comparison of linear and logistic regression. In panel b, notice that logistic regression assigns a reasonable decision boundary (the pink vertical line), even when the data contains easily-classified data samples. The same can not be said about linear regression, shown in panel a. Furthermore, the logistic regression loss shown in panel c exhibits the desired behavior (large values of the score function do not yield a large loss), unlike the quadratic loss function with linear regression.

Figure 2 compares linear and logistic regression. In panel b, notice that logistic regression assigns a reasonable decision boundary (the pink vertical line), even when the data contains easily-classified data samples. The same can not be said about linear regression, shown in panel a. Furthermore, the logistic regression loss shown in panel c (the blue curve) exhibits the desired behavior (large values of the score function do not yield a large loss).

2.1 Determining optimal logistic regression parameters

It turns out that Equation 9 has no closed form solution. To see why this is the case, let us attempt to apply straightforward optimization procedures. Denoting the cost function in Equation 9 as $J(\mathbf{w})$, we determine $\frac{\partial J}{\partial \mathbf{w}}$ and set it to zero, i.e.,

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \left[\sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \log \left[1 + \exp \left(-F \left(x^{(i)}, \mathbf{w}, y^{(i)} \right) \right) \right] \right] \\
&= \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \frac{\partial}{\partial \mathbf{w}} \left[\log \left[1 + \exp \left(-F \left(x^{(i)}, \mathbf{w}, y^{(i)} \right) \right) \right] \right] \\
&= \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \frac{\frac{\partial}{\partial \mathbf{w}} \left[1 + \exp \left(-F \left(x^{(i)}, \mathbf{w}, y^{(i)} \right) \right) \right]}{1 + \exp \left(-F \left(x^{(i)}, \mathbf{w}, y^{(i)} \right) \right)} \\
&= \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \frac{-\exp \left(-F \left(x^{(i)}, \mathbf{w}, y^{(i)} \right) \right) \frac{\partial F}{\partial \mathbf{w}}}{1 + \exp \left(-F \left(x^{(i)}, \mathbf{w}, y^{(i)} \right) \right)} \\
&= \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} \frac{-y^{(i)} \exp \left(-y^{(i)} \mathbf{w}^T \mathbf{\Phi}_i \right)}{1 + \exp \left(-y^{(i)} \mathbf{w}^T \mathbf{\Phi}_i \right)} \mathbf{\Phi}_i .
\end{aligned} \tag{10}$$

In this form, it is clear that $\frac{\partial J}{\partial \mathbf{w}} = 0$ cannot be solved for $\mathbf{w}$ and therefore no closed form solution exists. Instead, numerical optimization techniques must be used. In general, a wide selection of

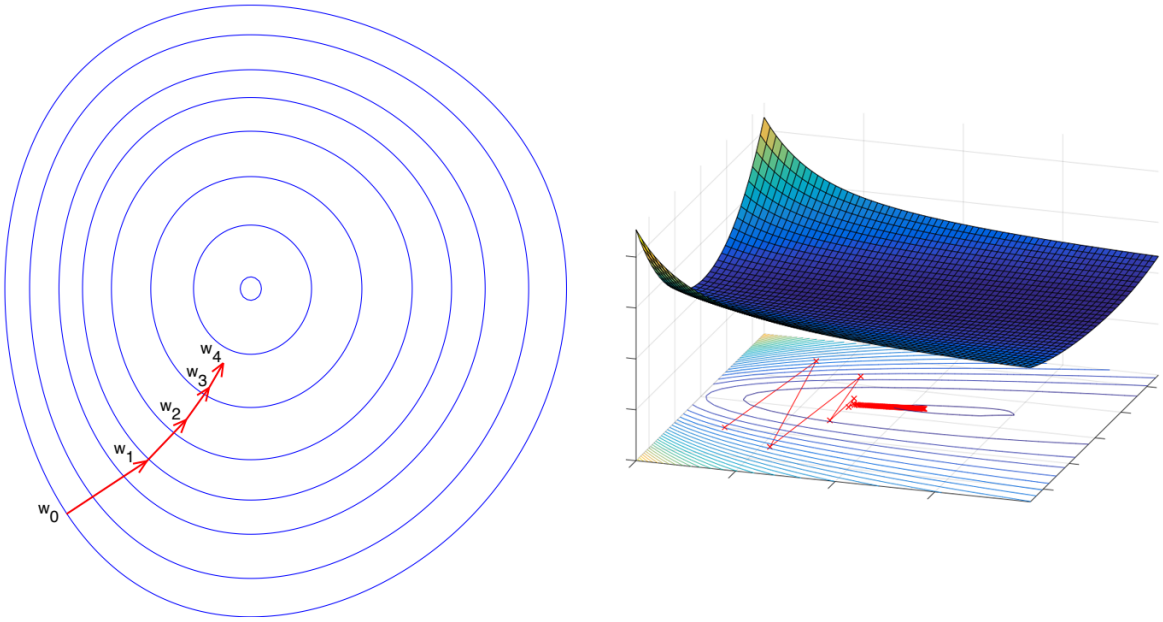

Figure 3: Several iterations of gradient descent. (left) Contours of an arbitrary function are shown, with function minimizing near the center of the figure. (right) Similar to the left panel (for a different error basin) and shown as a 3D surface on top of a contour plot. After [2].

methods could be applied. *Descent methods* are a particular class of these methods that make use of gradient information to iteratively "roll" along the cost manifold towards a nearby minimum. We will discuss *gradient descent*, which is the simplest of the descent methods.

Gradient descent exploits the fact that $-\frac{\partial J}{\partial \mathbf{w}}$ at the point $\mathbf{w}_t$ points "maximally downhill", i.e., in the direction that J decreases the most at $\mathbf{w}_t$. In general, with an appropriate starting point $\mathbf{w}_0$, continuously stepping in the direction of $-\frac{\partial J}{\partial \mathbf{w}}$ will eventually reach the nearest minimum (assuming that the step size is sufficiently small). Thus, a single step of the gradient descent algorithm is written

1. Compute $\mathbf{g}_t = \nabla_{\mathbf{w}} J(\mathbf{w}_t)$
2. Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \mathbf{g}_t$
3. Update $t \leftarrow t + 1$

The iterations would continue until a stopping criterion is reached (e.g., $\mathbf{g}_t \approx 0$ or a maximum iteration cutoff). Several iterations of gradient descent are shown in Figure 3. The left panel shows contours of an arbitrary function, with function minimizing near the center of the figure. Notice that each step of gradient descent "inches" closer to the minimum. Additionally, notice that the steps are smaller with each iteration because the value of $\mathbf{g}_t$ decreases closer to the minimum (α is fixed). Lastly, notice that for any particular iteration, the step direction does not necessarily point directly towards the minimum. The panel on the right is similar to the left panel (for a different error basin), but also shows the loss variation as a 3D surface. Relative to other descent methods, gradient descent often converges quite slowly.

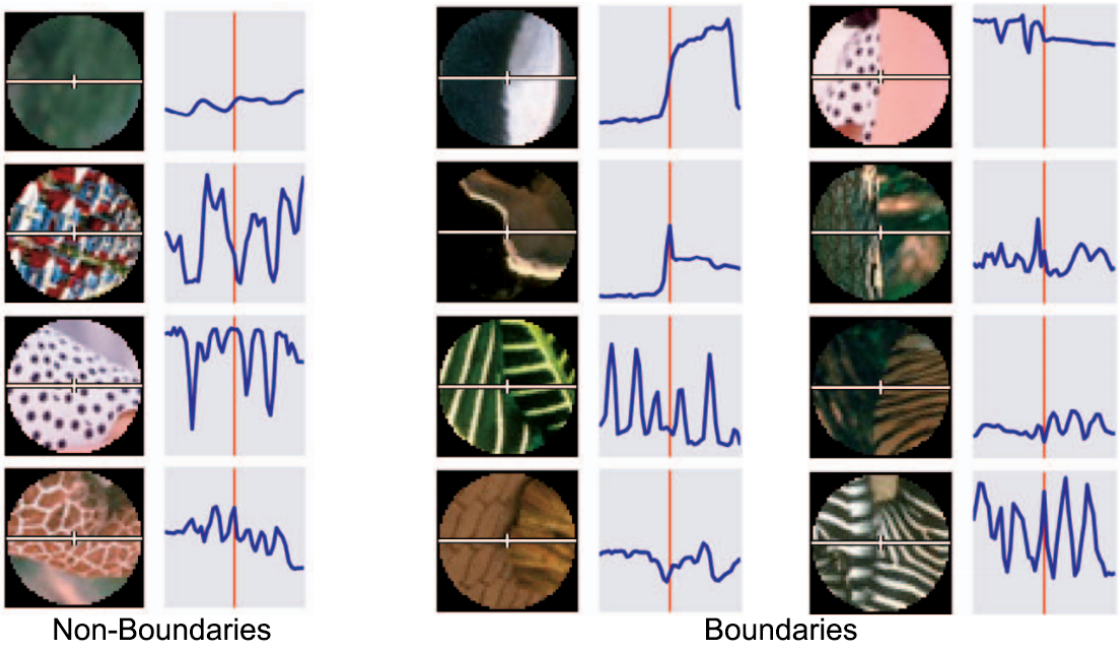

Figure 4: Horizontal intensity profiles cutting through boundaries and non-boundaries for several images. The profiles demonstrate that intensity alone is not a good indicator of whether a particular pixel belongs to a boundary. After [1].

2.2 Application: Edge/Boundary Detection

Use of logistic regression to detect the edges or boundaries in a digital image using various cues (e.g., intensity, boundary gradient, color gradient, texture gradient) has shown success [1]. Intensity alone is not a good indicator of whether a particular pixel belongs to a boundary. This is illustrated in Figure 4, which shows horizontal intensity profiles cutting through both boundaries and non-boundaries for several images. Because images can have a wide array of possible features (e.g., shadows, textures) and/or poor contrast, it is quite difficult to determine any generalizable characteristics of intensity profiles. Gradients, however, often reveal a considerable amount of information about the presence of boundaries [1]. This is illustrated in Figure 5, which shows the boundary gradient (BG), color gradient (CG), and texture gradient (TG) for the images shown at the top of the figure. Notice that particular gradients do a better job at indicating the presence of a particular boundaries.

Logistic regression can serve to predict whether a particular pixel is a boundary pixel. We would assume that the class label (boundary/no boundary) for the pixel is drawn from the distribution in Equation 5 where $y^{(i)} = 1$ corresponds to the pixel being a boundary and $y^{(i)} = -1$ corresponds to the pixel not being a boundary. The samples $x^{(i)}$ would be the pixel values themselves, and the transformations $\boldsymbol{\phi}$ would calculate the various gradients (e.g., BG, CG, TG) using information in the neighborhood of pixel i. Given a training dataset $\mathcal{D}$ of images and annotated pixel labels, the task would then be to determine the optimal linear combination of the aforementioned transformations that maximizes the likelihood of the dataset, i.e., we would determine $\mathbf{w}^*$ in Equation 8. Figure 6 shows the performance of this type of machinery compared to other boundary classifiers (the results of [1] are shown in purple). Although the concepts of precision and recall were not covered in this lecture, it is sufficient to understand that values closer to unity are indicative of better performance.

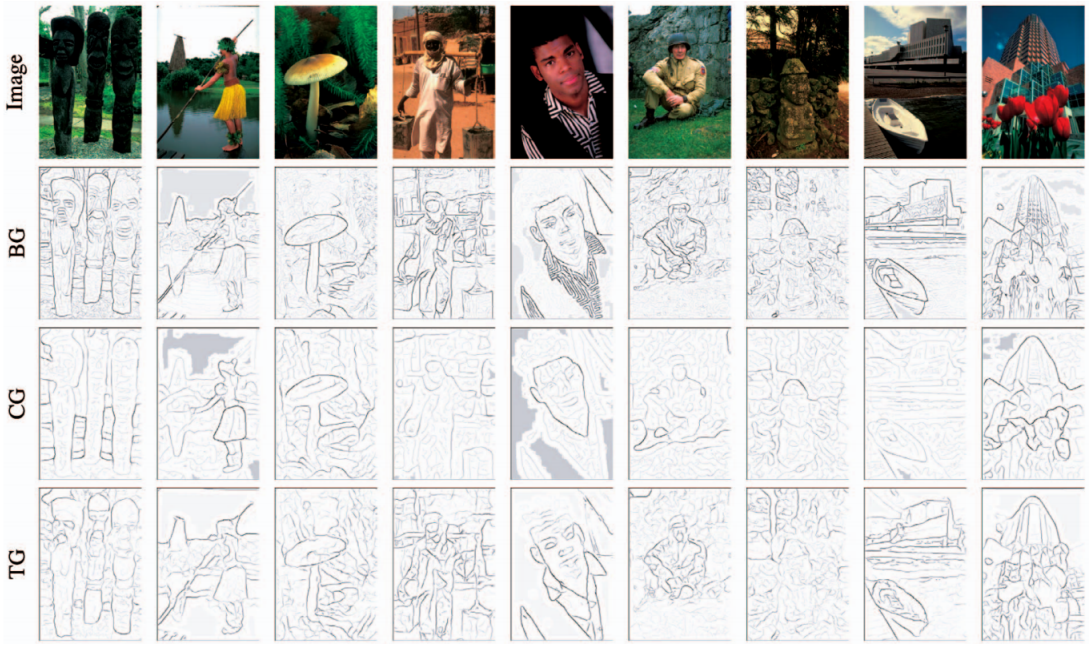

Figure 5: (second row) Boundary gradient (BG), (third row) color gradient (CG), and (fourth row) texture gradient (TG) for the images shown in the first row. Notice that particular gradients do a better job at indicating the presence of particular boundaries. After [1].

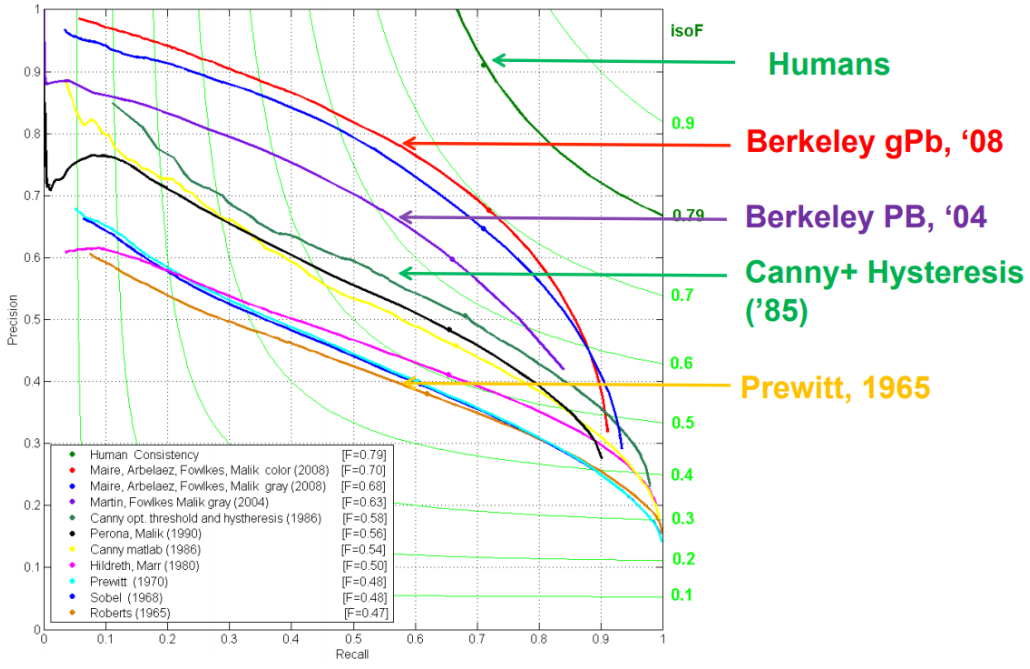

Figure 6: Performance of the technique in [1] and other boundary classifiers (the results of [1] are shown in purple). Although the concepts of precision and recall were not covered in this lecture, it is sufficient to understand that values closer to unity are indicative of better performance. After [2].

```
%pylab inline

# generate some synthetic data
x = []; y = []
x.extend(random.uniform(-2.5, -1.5, 100)); x.extend(random.uniform(1.5, 2.5, 100));
y = np.atleast_2d(np.ones([len(x), 1]))

phi = np.ones([2, len(x)]); phi[0,:] = x
y[phi[0,:] < 0] = -1.0

# linear regression (evaluate from closed form expression)
wlinreg = dot(linalg.inv(dot(phi, phi.T)), dot(phi, y))

# logistic regression (use gradient descent with random initial guess)
wlogreg = np.random.uniform(-1, 1, [phi.shape[0], 1])
gradw   = sum(-y * exp(-y*np.dot(phi.T,wlogreg))/(1.0 + exp(-y*dot(phi.T,wlogreg)))\
              * phi.T, axis = 0, keepdims = True).T
alpha   = 1E-3

while linalg.norm(gradw) >= 0.1:
    wlogreg = wlogreg - alpha*gradw
    gradw   = sum(-y * exp(-y*np.dot(phi.T,wlogreg))/(1.0 + exp(-y*dot(phi.T,wlogreg)))\
                  * phi.T, axis = 0, keepdims = True).T

# plot results
xp    = linspace(min(phi[0,:]), max(phi[0,:]), 1000)
yplin = wlinreg[1,0] + wlinreg[0,0]*xp
yplog = 1.0/(1.0 + exp(-(wlogreg[1,0] + wlogreg[0,0]*xp)))

plot(xp, yplin, linewidth = 5, zorder = 1); plot(xp, yplog, linewidth = 5, zorder = 1)
scatter(phi[0,:], y, s = 5, c = 'k', zorder = 2); grid()
```

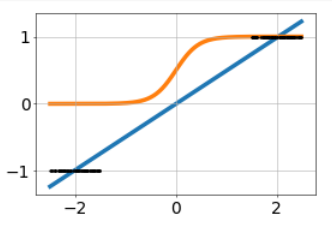

Figure 7: Simple implementation of linear and logistic regression applied to a synthetic two-class dataset using Python, NumPy, and matplotlib. The result of the plotting code is shown as the inset panel.

3 Implementation of Linear and Logistic Regression

Both linear and logistic regression lend themselves to a relatively simple code implementation. If the size of the dataset $\mathcal{D}$ is not too large, the closed form expression for the optimal model parameters (i.e., $\mathbf{w}^* = \left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1}\mathbf{\Phi}\mathbf{y}$) can be used (see Section 1). Otherwise, gradient descent or another numerical technique should be used. For logistic regression, a numerical technique is required to determine $\mathbf{w}^*$. Figure 7 shows a simple implementation of linear and logistic regression applied to a synthetic two-class dataset ($y^{(i)} \in \{-1, 1\}$) using Python, NumPy, and matplotlib. The result of the plotting code is shown as the inset panel. This code was used to generate Figures 1 and 2 with only a few modifications. Packages tailored towards machine learning (e.g., scikit-learn, TensorFlow, pyTorch) allow implementation of linear and logistic regression (in addition to more sophisticated data models) more concisely.

4 Conclusions

Overall, Lecture 3 served as an effective introduction to binary classification. It was shown that using linear regression with zero thresholding leads to poor classification performance because easily-classified samples have an undesirably large influence on the position of the decision boundary. These issues are addressed by instead using logistic regression, which assigns a more appropriate probability distribution (Equation 5 instead of a Gaussian distribution) to the values of y in the dataset (which are constrained to be ± 1). In general, linear regression has a closed form solution and is often well-suited for standard regression problems, but not for classification problems.

Unlike linear regression, however, the solution for the parameters that maximize the likelihood of the dataset (under the probabilistic assumptions made by logistic regression) does not have a closed form expression. Instead, numerical techniques are required in order to determine the

optimal parameters. A simple optimization technique (gradient descent) that could be used to determine the these parameters was introduced qualitatively.

An application of logistic regression for boundary detection was also discussed. It was shown that intensity alone is a poor indicator of whether a particular pixel is or is not a boundary. A more sophisticated approach using a linear combination of image cues (e.g., intensity, boundary gradient, color gradient, texture gradient) leads to better classification performance [1]. Finally, a simple Python implementation of binary classification using linear and logistic regression was presented and tested on a synthetic dataset.

References

[1] D. R. Martin, C. C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):530–549, May 2004.

[2] A. Schwing. ECE 544NA Lecture 3: Logistic Regression (slides). University of Illinois at Urbana-Champaign, 2018.