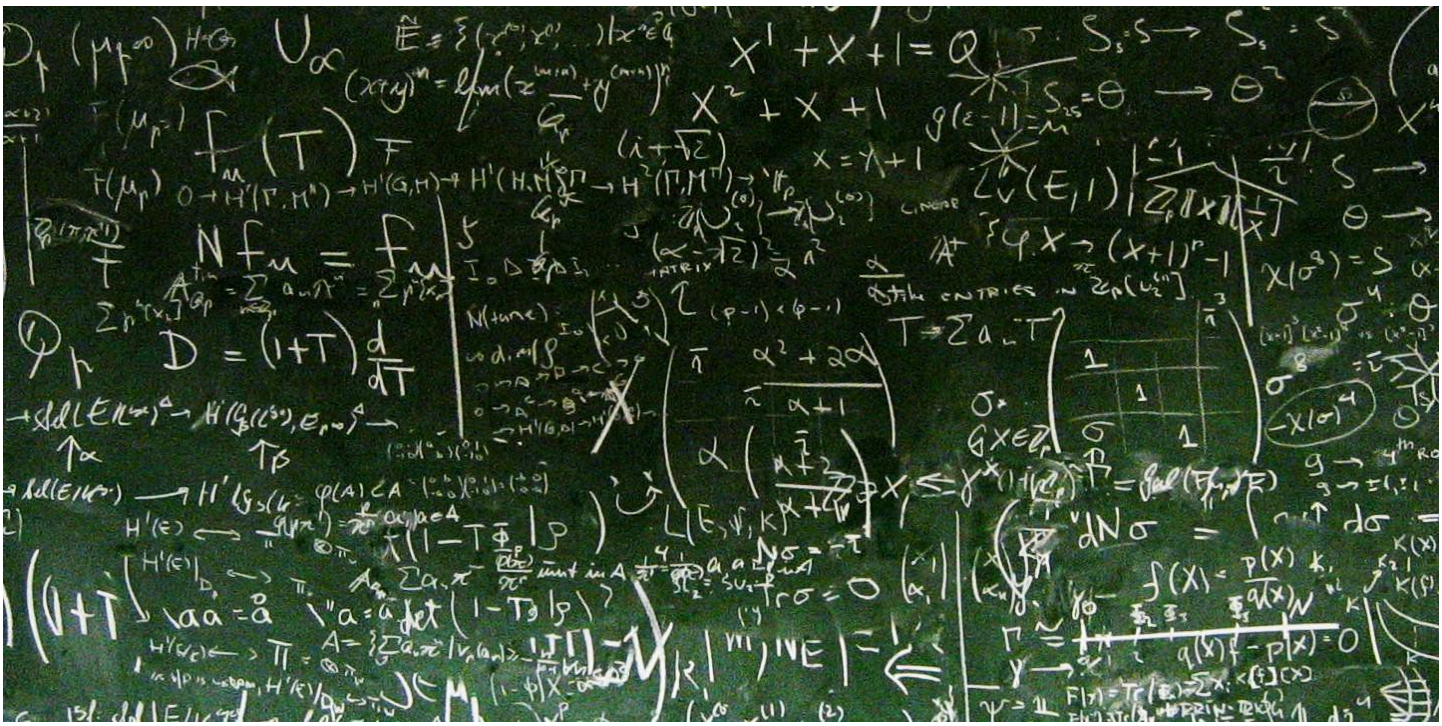


Not really, but this article is definitely the most mathematically involved we've got on AYOAI so far :)
https://theheartthrills.files.wordpress.com/2015/06/blackboard-2.jpg

Rohan #6: The beautiful math behind logistic regression.

What most beginner ML courses won't teach you.

Rohan Kapur
May 21, 2016 · 4 min read

. . .

This is the sixth entry in my journey to extend my knowledge of Artificial Intelligence in the year of 2016. Learn more about my motives in this introduction post.

This blog post assumes sound knowledge of the Logistic Regression algorithm. You can learn more about said algorithm in my first blog post.

. . .

It's been a while since my last article (work), but with summer approaching Lenny and I are going full speed, so expect articles on interesting machine learning algorithms (eg. recurrent and convolutional neural networks), our recent work with using machine learning to analyze fMRI scans, and new research papers from institution/companies like Google's DeepMind we are going to learn about at our trip to ICML 2016 next month. Keep your eyes peeled for some more philosophical write-ups on AI, too!

Keep your eyes peeled!

Today I want to build on my very first article about logistic regression. In particular, I want to discuss the statistical/probabilistic interpretation of logistic regression, which I felt was **missing** from explanations and lectures provided by certain online courses like Andrew Ng's Machine Learning one (which is still wonderful). I will discuss the intuition behind the logistic regression model formulated in the previous article.

UPDATE: The work presented in this article was part of my submission for my school mathematics coursework. Since I submitted it, and don't want to be caught plagiarizing myself (heh), I've replaced the rest of the article with *images* of each page

in the PDF. You can skip the intro and the conclusion + visualization through programming.

Once I get my IB results — around July — I'll put the post back up.

Kapur

Using regression to model conditional probabilities

I. Introduction

A probability is a numerical representation of how likely an event is to occur, from 0 (certain to not occur) to 1 (certain to occur). External factors in the form of information may influence the probability of an event; if wind speed increases we may postulate that the probability of an upcoming storm should increase too, and these are called conditional probabilities. A conditional probability is the probability of an event occurring given some other event has already occurred. Conditional probabilities have wide uses in many industries; data about a cancer tumor, for example, can help practitioners predict the probability of said tumor being either malignant (cancerous) or benign (non-cancerous). Another example may be using the attributes of an email in your inbox to predict the probability that it is spam. In both cases, we can use the probability to make a "choice" about the event, where a probability of at least half can be concluded as "true", or "false" otherwise. In this mathematics investigation, I would like to explore how we can use the data of a variable to predict the probability of an event occurring.

This problem, which is actually called "logistic regression", was borne out of applied mathematics in the context of computer science, most notably artificial intelligence—the art of having computers learn and make choices (like if a patient's tumor is cancerous or not) without explicit rules. As an avid programmer for over five years, I was eager to explore the workings of modern artificial intelligence used in domains ranging from handwriting recognition and self-driving cars to cancer tumor classification. After successfully completing an online course offered by Stanford University on the topic, I was delighted by the sophisticated mathematical content involved and wanted to expand on these findings in my IB mathematics investigation.

Cancer is one of the leading deaths in the world, claiming 600,000 lives in the U.S. alone just last year. My main goal for this investigation is to explore the mathematics with respect to predicting the conditional probability of a patient's cancer tumor being malignant given the tumor's data and attributes, and then ultimately "classifying" said tumor discretely as either malignant or benign to inform said patient. Not only

is this a classic problem used when teaching artificial intelligence, but its altruistic sentiment in contributing

to a tragic issue that impacts so many lives directs my passion for it. I will develop a program in MATLAB (a programming tool) to visualize the results of the algorithm used.

II. Problem Description

In a typical linear regression, we use existing data points to create a model that describes the relationships between different variables. We can then input values that are not already part of our dataset and find a "predicted" output for them. In probabilistic regression we undergo a similar process—we take an existing group of data points and then perform a regression to produce a model that can be used for future predictions, but predictions that are in the form of conditional probabilities instead.

In the example of cancer tumor classification, the existing data points are not continuous probabilities but are instead discrete—either 0 (benign) or 1 (malignant), because they are the outcomes of real patients. However, we can still formulate a model that can output a probability. For example, if the result of the model is 0.4 we can suggest that the data point is somewhat of an anomaly/outlier and the outcome is not certain. This is described as the confidence of the model's prediction. Figure 1 illustrates what our dataset may look like:

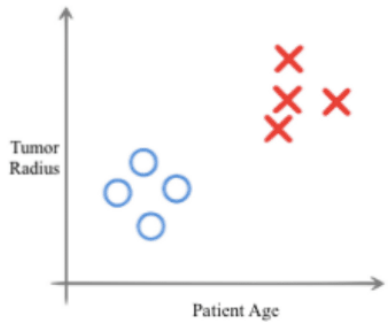


Figure 1

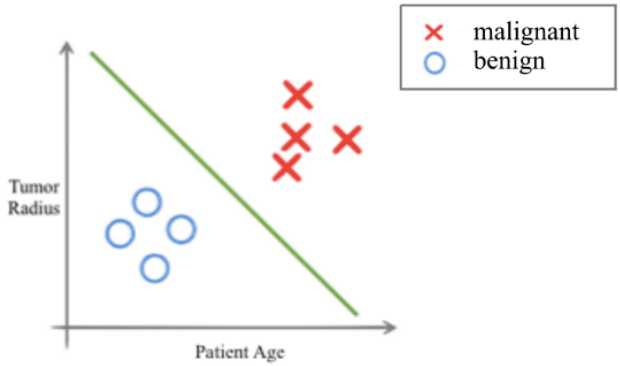


Figure 2

Here, on each axis we plot different features of the variable—patient age and tumor radius, for example. Both of these can be considered as "inputs" or "features", and they are independent variables. The output exists on the data points themselves; they are the "labelling" of the data points as being either malignant or benign, indicated by their shape/color. The red crosses are malignant tumors and the blue circles are benign. There seems to quite clearly exist a separation between the two types of data points. If we apply probabilistic regression, we can in fact create a function that "separates" these two data points, as shown in Figure 2.

This function is a by-product of the probabilistic regression, and does not produce probabilities itself. It separates two discrete regions, where on the right all data points are considered malignant/having a class of 1 and on the left all data points are considered benign/having a class of 0. This exploits the fact that, as mentioned before, probabilities of at least 0.5 can be rounded to 1 and else rounded to 0. Hence, tumor data points that lie on the line have a malignancy probability of 0.5, and have a greater/lesser probability further right/left from the line respectively. Ultimately, although a probability is useful, we need to convert it to a discrete class, an example being for prescribing treatment to the patient. Using probabilistic regression to find such function, denoted the "decision boundary", for predicting the state of cancer tumors will be the ultimate goal of this investigation.

III. Model: Formalism

First, we need to formally define our variables. Looking at Figure 2, it seems that we have three distinct attributes of data: the tumor radius, the patient age, and the classification as either malignant or benign. We consider the tumor radius, patient age, and any other tumor data as the input features. We will now group all these features into a matrix, where each column corresponds to a certain feature and each row corresponds to a single data point with the values of the features (in the form of a vector called a "feature vector" **x**), denoted **X**, $\mathbf{X} \in \mathbb{R}^{(m \times n)}$. With this notation, **m** corresponds to the number of data points while **n** refers to the number of features (which need not be two, we could do this task in 3 or more dimensions). In Figure 2, **m** = 8 and **n** = 2. We will then have a vector called **y**, $\mathbf{y} \in \mathbb{R}^{m}$ which contains all the outputs, either 0 for benign or 1 for malignant. If there is indeed pattern in the data, our job is now to find a model that will map the n-dimensional feature data points (or feature vectors) in **X** to a probability prediction of **y** (the tumor being malignant). We can then round this prediction to a 0 or a 1. For this paper, we will refer to the nth feature vector in matrix **X** by using a subscript of n: $\mathbf{X_n}$, and the nth value in feature vector **x** with a superscript: $\mathbf{x^n}$. Likewise, $\mathbf{y_n}$ refers to the nth value in the output vector **y**.

$$X = \begin{pmatrix} \text{tumorSize1} & \text{patientAge1} \\ \text{tumorSize2} & \text{patientAge2} \end{pmatrix}, \vec{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, X_1 = \vec{x} = \begin{bmatrix} \text{tumorSize1} \\ \text{tumorRadius1} \end{bmatrix}$$

Our feature vector **x** could be considered an n x 1 matrix instead of a vector (since it lacks a "direction" if being mathematically rigorous), but this is the standard terminology used in the field. Since our goal is to use a model to map our features to a probability, our model needs to output values in the range [0, 1] whilst being

continuous and hence differentiable. An example of a function with these properties is the **sigmoid** function, which we denote **P(x)** as follows:

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

The function is called the "sigmoid" function because of its S-shape, as illustrated in the figure below:

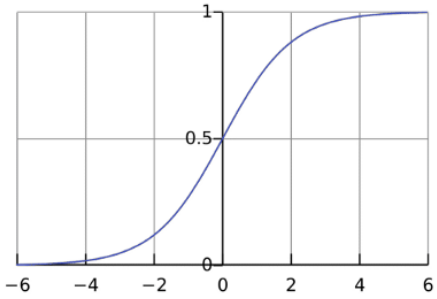


Figure 3

As seen in Figure 3, the range of the function is (0, 1), and the y-intercept is at (0, 0.5). It is a smooth, continuous curve and is differentiable. Horizontal asymptotes exist at P(x) = 0 and P(x) = 1; this becomes apparent when we evaluate the limits to infinity and negative infinity, as shown below. Though it is counter-intuitive that we technically cannot achieve a probability of exactly 0 or 1, it can be suggested that one can never really be certain of an outcome with modeling (this is an epistemological question, ultimately).

$$\lim_{x\to\infty} \frac{1}{1+e^{-x}} = \frac{1}{1+\lim_{x\to\infty} e^{-x}} = \frac{1}{1+0} = 1$$

$$\lim_{x\to-\infty} \frac{1}{1+e^{-x}} = \frac{1}{1+\lim_{x\to-\infty} e^{-x}} = \frac{1}{1+\lim_{x\to\infty} e^{x}} = 0$$

As of now, the input **x** to our probability function is an n-dimensional feature vector, extracted from a single row in the matrix **X**. We need to represent this as a scalar value, however, that we can input to the probabilistic sigmoid function. On top of this, in modeling tasks, we need to modify some parameter that adjusts the function. For example, if we were to perform a linear regression modeling task, we would be finding optimal parameters w_0 and w_1 in the function $f(x) = w_0 + w_1x$. The same can be said for our task; the probabilistic sigmoid function will not automatically fit our data—instead it needs to be transformed to fit our data just like in a typical linear regression. These transformations usually occur as a result of applying coefficients (or "weights") to our input variables, via a regression. Since we are trying to model the outcome

of **y** given **x** (where **y** is a function of **x**), we can use a linear combination of these weight parameters and our input feature values to achieve a scalar value (or overall "score") that, with optimal weights, can be used by the probability function to discriminate between a class of 0 and a class of 1. This will be elaborated in the next section. Thus, we will define a weight vector **w** where $\mathbf{w} \in \mathbb{R}^{\mathbf{n}}$. Essentially, we will have a single weight value (coefficient) for each feature. The weighted sum/linear combination will look like the following:

$$w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_n \cdot x_n$$

where each value in **x** corresponds to the input features patient age, tumor radius, etc. Though this is an accurate representation, we'll want to write this expression in closed form using linear algebra (which would be more preferable), which is shown below:

$$\sum_{i=1}^{m} w_i x_i = w^T x$$

Where $\mathbf{w^T}$ means to "transpose **w**" which is an operation where the rows and columns of a matrix/vector are switched around. The following equation makes this more clear:

$$\begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_n \end{bmatrix}^T \bullet \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} w_1 & w_2 & ... & w_n \end{bmatrix} \bullet \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} = w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

Now, we will use this term as input to the probabilistic logistic (or "sigmoid") function:

$$P(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

If the weighted sum for a data point is at least 0 (remember the y-intercept for the logistic function is at P(x = 0.5)), then we can predict the data point to have a class of 1 (and 0 otherwise). If the data is truly linearly separable, and our weights have been optimized, then our model should be able to make accurate

predictions on the tumor's malignancy (or that **y = 1**). Thus, we can rewrite the function notation as a conditional probability like so:

$$P(y=1\,|\,x;w)=\frac{1}{1+e^{-w^Tx}}$$

Where **x;w** means "**x** parametrized by **w**", because we apply weights **w** to vector **x**. The only pending issue right now is that the notion of a vertical shift in our linear combination does not exist. Though we can stretch the logistic function currently, we cannot shift it. Ideally, we want a term $\mathbf{w_0}$ that can be learned independent of a feature. However, the matrix multiplication $\mathbf{w^Tx}$ does not allow us to add extra terms to **w** without ensuring that **x** increases in size. To achieve this, we will actually prefix **x** with the value 1 (in other words, prefix a column of ones to **X**) so that a new term in **w** is created which "acts" as a horizontal shift ($\mathbf{w_0 \cdot x_0 = 1w_0 = w_0}$). The new length of the vectors **x** and **w** would be **n + 1**.

IV. Model: Interpretation

Now, let's discuss the intuition behind the model formulated in the previous section. First, we will undergo some algebraic manipulation to arrive at an important step that can be interpreted qualitatively (and not just quantitively), where $\mathbf{w^Tx}$ can be isolated:

$$P(y=1\,|\,x;w)=\frac{1}{1+e^{-w^Tx}}$$

$$P(y=1\,|\,x;w)\cdot(1+e^{-w^Tx})=1 \quad \textbf{(1) Multiply both sides by } (1+e^{-wTx})$$

$$P(y=1\,|\,x;w)+P(y=1\,|\,x;w)\cdot e^{-w^Tx}=1 \quad \textbf{(2) Expand the brackets, distribute terms}$$

$$P(y=1\,|\,x;w)\cdot e^{-w^Tx}=1-P(y=1\,|\,x;w) \quad \textbf{(3) Subtract both sides by P(y = 1|x;w)}$$

$$e^{-w^Tx}=\frac{1-P(y=1\,|\,x;w)}{P(y=1\,|\,x;w)} \quad \textbf{(4) Divide both sides by P(y = 1|x;w)}$$

$$(e^{-w^Tx})^{-1}=\left(\frac{1-P(y=1\,|\,x;w)}{P(y=1\,|\,x;w)}\right)^{-1} \quad \textbf{(5) Raise both sides to the power of -1}$$

$$\frac{P(y=1\,|\,x;w)}{1-P(y=1\,|\,x;w)}=e^{w^Tx} \quad \textbf{(6) Compute the result}$$

In our final step, we showed that the conditional probability of the cancer tumor being malignant, divided by one subtracted that probability, is equal to the **e** to the power of $\mathbf{w^Tx}$. Now, **1 - P(y = 1|x;w)** can also be

interpreted as **P(y = 0|x;w)**, since **P(y = 1|x;w) + P(y = 0|x;w) = 1** for a dichotomous variable with two possible classes. The probability of **y = 1** occurring divided by the probability of **y = 0** occurring is known as the "odds-ratio", which is a ratio of how likely an event is to occur to how likely it will *not* occur. For example, if **P(y=1|x;w) = 0.8**, then **P(y=0|x;w) = 1 - 0.8 = 0.2**. Hence the odds-ratio is **0.8/0.2 = 4** (4:1). A log-odds ratio is the logarithm (or natural logarithm) of the odds-ratio value. Let's take the natural logarithm of both sides to achieve such result:

$$\ln\left(\frac{P(y=1\,|\,x;w)}{1-P(y=1\,|\,x;w)}\right)=\ln(e^{w^Tx})=w^Tx$$

Here we show our weighted sum $\mathbf{w^Tx}$ is equal to the log-odds ratio. The sigmoid function is actually derived from the log-odds ratio. To show this, I will first derive the domain and range for the log-odds ratio.

$$\text{Let } F(x)=\ln\left(\frac{x}{1-x}\right),\ \frac{x}{1-x}>0\ \therefore 0<x<1$$
$$D=\{x\,|\,x\in\mathbb{R},\,0<x<1\}$$
$$R=\{y\,|\,y\in\mathbb{R}\}$$

Hence:

$$x=\ln\left(\frac{y}{1-y}\right)$$ **(1) Replace 'y' with 'x' to start inverse calculation**

$$e^x=\frac{y}{1-y};\ e^x(1-y)=y;\ e^x-e^xy=y;\ e^x=y+e^xy;\ e^x=y(1+e^x)$$ **(2) Make y the subject to find the inverse**

$$y=\frac{e^x}{1+e^x}=\frac{e^x}{1+e^x}\bullet\frac{e^{-x}}{e^{-x}}=\frac{1}{e^{-x}(1+e^{-x})}=\frac{1}{1+e^{-x}}$$ **(3) Multiply the fraction by e^{-x}/e^{-x} (which equals one) simplify to get original notation for sigmoid**

$$F^{-1}(x)=\frac{1}{1+e^{-x}}$$
$$D=\{x\,|\,x\in\mathbb{R}\}$$ **(4) Domain and range switch around from log-odds to sigmoid**
$$R=\{y\,|\,y\in\mathbb{R},\,0<y<1\}$$

Thus, the sigmoid function is actually the inverse of the log-odds ratio, which works nicely as the domain and range switch over. As stated before, our linear regression/weighted sum of the input features is equal to the log-odds of a malignant tumor occurring. In classical linear regression, as the value of a weight/ coefficient $\mathbf{w_n}$ increases by 1, the predicted output increases by $\mathbf{x_n}$. In conditional probability modeling, as $\mathbf{w_n}$

increases by 1, the log-odds ratio of having a malignant tumor increases by $\mathbf{x_n}$. Now, the interpretation of using weights is that each feature will have a different importance. If, based on the data patterns/correlations, it is found that the feature of a patient's height (for example) has no bearing on or correlation with the malignancy of a tumor, then the optimal weight for that feature would be zero. The magnitude of a weight hence indicates the importance of the feature it corresponds to, since it will contribute greatly to the weighted sum and override the other contributions to lead to a logistic output closer to 1, increasing the log-odds ratio greatly. If an optimal weight for a feature is very large but negative, then we can assume the presence of said feature strongly indicates that the outcome is unlikely because it "pulls" the weighted sum closer to negative infinity and thus the predicted outcome passed through the sigmoid function to zero. An example would be if we were predicting the probability of a rainy day and one of our features was the amount of sunlight—the optimal weight for that would be highly negative.

The point of this algorithm and investigation is to determine the optimal weights and thus, by analyzing existing datasets, extract patterns to tell us the degrees to which different features are important and not important to formulate a prediction from data. As humans, we cannot by hand determine the relationships between the feature data and the outcomes, but we can measure them and have a computer do it. This is the nature of the field of machine learning and artificial intelligence.

Next, we will talk about the "decision boundary". The decision boundary, a function that separates the two classes, was presented in Figure 2. Here, we will make it concrete. When $\mathbf{w^Tx = 0}$, then $\mathbf{P(y = 1|x;w) = 0.5}$ due to the sigmoid function's y-intercept as mentioned before. Thus, when $\mathbf{w^Tx = 0}$ we are at a "turning point" as we switch from a discrete classification of 0 to 1.

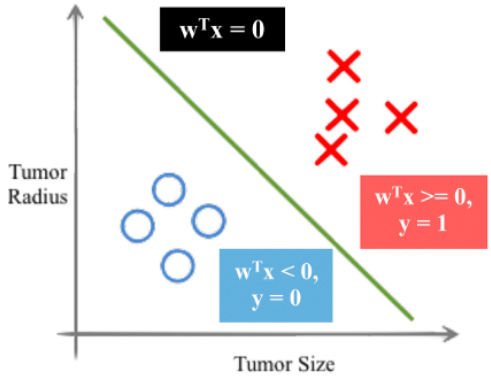


Figure 4

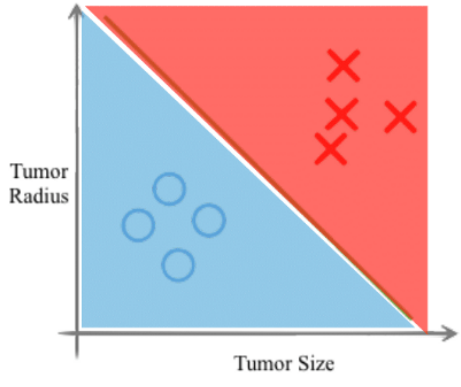


Figure 5

As shown in Figure 4, our separating function/decision boundary is a hyperplane of the equation $\mathbf{w^Tx = 0}$ as it discriminates between two different regions of classification. On the left hand side, where $\mathbf{w^Tx < 0}$, the output of $\mathbf{P(y = 1|x;w) < 0.5}$ thus we predict benign or false. At the line and to the right hand side of the line, we predict malignant or true. This also implies that the log-odds ratio must be at least 0 (where odds-ratio would be 1) for a classification of malignant, and negative for a classification of benign.

If the weights **w** are not optimal, then the separation would be inaccurate, and this could be seen visually in a 2 or 3 dimensional example. We need to "shape" it correctly to create a nice decision boundary. Hopefully, this provides (visual) intuition into why we need to optimize on **w** for our modeling task.

V. Error/Accuracy Metrics

The way we will end up selecting/estimating our parameters **w** is by devising an "error function" (which describes the error created by any arbitrary values for **w**) and then minimizing the output of that function. Obviously, the minimum point **(w, ε)** of an error function implies the lowest possible error **ε** (error will almost always exist for a practical, noisy dataset), and hence the most optimal **w** that can be used for future predictions. Alternatively, the function may describe *accuracy*, in which case we look for the *maximum* value. We will now derive the latter type of (accuracy) function using probability distributions.

The binomial distribution is a discrete probability distribution of exactly **k** binary successes in **n** independent trials (not to be confused with the number of features parameter), with each trial having a success probability of **p** and hence failure probability of **1 - p**. The binomial distribution is then defined as $\mathbf{B(n, p) = {}^nC_r p^k (1-p)^{n-k}}$. Here, $\mathbf{{}^nC_r}$ computes the number of arrangements of the different trials (order is disregarded), $\mathbf{p^k}$ computes the probability of achieving the demanded **k** successes over all trials, and $\mathbf{(1-p)^{n-k}}$ computes the probability of the rest of the trials being failures. In this case, a trial corresponds to observing a data point in our known dataset and predicting a probabilistic output for it based on its features. We set the number of successful trials, **k**, to be the number of times when **y = 1** (for example a tumor is malignant) and thus the number of failures, **n - k**, to the number of times when **y = 0** (for a example a tumor is benign).

Unlike the binomial distribution, however, each of our trials has a separate probability of success, conditioned on our weights **w** (which is constant throughout all trials) and input **x** (which changes). Therefore, we can treat them as *separate* individual trials—known as Bernoulli trials—which have two

possible, mutually exclusive outcomes: success where **y = 1** and failure where **y = 0**. With this, we only either have one success or zero successes. As mentioned before, with a Bernoulli trial, the probability of success **P(y = 1|x;w)** and failure **P(y = 0|x;w)** sum to 1. The probability of failure hence is also equivalent to **1 - P(y = 1|x;w)**. The outputs thus belong to a Bernoulli distribution, which has just one Bernoulli trial and is thus a special case of the binomial distribution where **n = 1** in **B(n, p)**.

The number of successes **k** is either 0 (a failure) or 1 (a success), $\mathbf{k \in \{0, 1\}}$. Since $^{1}\mathbf{C}_{\mathbf{k},\, \mathbf{k} \in \{0,\, 1\}} = \mathbf{1}$, then $\mathbf{B(1, p) = p^{k}(1-p)^{1-k}}$. If **k = 1**, then the outputted probability is **p** since the second term gets raised to the power of 0. When **k = 0**, the outputted probability is **1 - p** (the probability of failure as discussed before). Note that in the case of our Bernoulli distribution the number of successes **k** is equal to the known outcome of each trial **y**. With the example of malignant/benign tumors, we can make the following statement:

$$Y = \text{tumor is malignant}$$
$$Y \sim B(1, p)$$
$$P(Y = y) = p^{y}(1-p)^{1-y} = \begin{cases} p & if \;\; \text{y} = 1 \\ 1\text{-}\,p & if \;\; \text{y} = 0 \end{cases}$$

This is the probability mass function for the Bernoulli distribution; it is the probability that the tumor is malignant. The probability variable **p** is the notation we came up for sigmoid-based prediction **P(y = 1|x;w)**, and therefore **1 - p** = **P(y = 0|x;w)**. This function can hence also be interpreted as an accuracy function—when **y = 1** and we predict with the probability **P(y = 1|x;w) = 0.6**, then the distribution function will return an accuracy of **0.6** (or error of **0.4**). If **y = 0**, the function would return an accuracy of **0.4** (or error of **0.6**).

Now, the function highlighted above for each individual prediction (per feature vector **x**) can be multiplied together to produce the joint probability of all these individual cases occurring together, of which there are **m**. Thus, we multiply all the individual accuracies together to create one composite accuracy over our entire dataset, and we want to maximize this. We can sub in **p** for our formulated conditional probability notation (which is a placeholder for our sigmoid probability mass function) and call this the likelihood function:

$$\text{Y = tumor is malignant}$$
$$L(w \mid y) = \prod_{i=1}^{m} P(Y = y_i)$$
$$= \prod_{i=1}^{m} P(y = 1 \mid x; w)_i^{\,y_i} (1 - P(y = 1 \mid x; w)_i)^{1-y_i} \qquad 10$$

Kapur

The ∏ symbol is similar to summation with ∑ except it is a multiplication; it performs a product on all the

terms inside it such that we can produce this joint probability. The reason we have each separate probability **p** as $\mathbf{P(y = 1|x;w)_i}$ with a subscript **i** outside is because each probability has a separate value for **x**. We will sub in the general notation for that **x**, based on **i**, later on (it uses matrix **X**) when we also sub in the actual sigmoid probability mass function. The likelihood function is in a sense the opposite of our traditional probability prediction function $\mathbf{sigmoid(w^Tx)}$, because it is the probability that a given **w** occurs given our outcomes **y**, which is why our function is written as **L(w|y)**. A higher value for **L(w|y)** implies that the given **w** is more likely to occur given the outcomes (and input data) and the accuracy is greater.

VI. Optimization: Computing Derivatives

Our goal, now, is to maximize **L(w|y)** to find the weights **w** and hence probabilities **P(y = 1|x;w)** that makes our data the most likely, since it is what we observed! Maximizing the likelihood function **L(w|y)** is known as the maximum likelihood estimation. To find the maximum point, we can apply basic differentiation: compute the derivative of **L(w|y)**, set the derivative to zero, and solve for the global maximum. However, an issue is presented:

$$\frac{d}{dw}L(w\,|\,y) = \frac{d}{dw}\prod_{i=1}^{m} P(y=1\,|\,x;w)_i^{\,y_i}\,(1-P(y=1\,|\,x;w)_i)^{1-y_i} = ?$$

It is unclear how we can compute the derivative of a product of terms of arbitrary length. Instead, what we will do is take the log of the likelihood function, denoted as the log-likelihood function. Since the log(x) function is monotonically increasing, the maximum coordinate for the likelihood function **(w, L(w|y))** and log-likelihood function **(w', L'(w|y))** will have equal x-coordinates/optimal values for the weights such that **w = w'** even though $\mathbf{L(w|y) \neq L'(w|y)}$, $\mathbf{L'(w|y) = \log L(w|y)}$. Why we choose log-likelihood, though, becomes clearest after this observation:

$$\log\left(\prod f(x)\right) = \sum \log(f(x))$$

This is because the logarithm of products of terms is equal to the addition of separate logarithms of the separate terms. With this type of function, it's very easy to perform differentiation. Let us now compute the log-likelihood function (we use the natural logarithm):

$$\ln\prod_{i=1}^{m} P(y=1\,|\,x;w)_i^{\,y_i}\,(1-P(y=1\,|\,x;w)_i)^{1-y_i}$$

$$= \ln \prod_{i=1}^{m} P(y=1 \mid x;w)_i^{\,y_i} + \ln \prod_{i=1}^{m} (1-P(y=1 \mid x;w)_i)^{1-y_i}$$ **(1) Apply rule ln(A•B) = lnA + lnB**

$$= \sum_{i=1}^{m} \ln\left(P(y=1 \mid x;w)_i^{\,y_i}\right) + \sum_{i=1}^{m} \ln\left(1-P(y=1 \mid x;w)_i^{\,1-y_i}\right)$$ **(2) Apply rule again, where we can replace product operator with sum operator**

$$= \sum_{i=1}^{m} \left[\ln\left(P(y=1 \mid x;w)_i^{\,y_i}\right) + \ln\left(1-P(y=1 \mid x;w)_i^{\,1-y_i}\right)\right]$$ **(3) Our two summation indices are the same, hence we can group them into one**

$$= \sum_{i=1}^{m} \left[y_i \ln\left(P(y=1 \mid x;w)_i\right) + (1-y_i)\ln\left(1-P(y=1 \mid x;w)_i\right)\right]$$ **(4) Apply rule $\ln A^B = B \ln A$**

This is the final representation that we will use for the log-likelihood function, and we will denote it as $\ell(\mathbf{w})$.

$\ell(\mathbf{w})$ has interpretable properties; if $\mathbf{y = 1}$, the accuracy will tend to negative infinity (not accurate at all!) as $\mathbf{P(y = 1|x;w)}$ tends to $\mathbf{0}$, and similarly for when $\mathbf{y = 0}$ as $\mathbf{P(y = 0|x;w)}$ tends to $\mathbf{1}$.

Note that since the sigmoid function asymptotes horizontally at $\mathbf{y = 0}$ and $\mathbf{y = 1}$, our accuracy value is never undefined. When $\mathbf{P(y = 1|x;w) = y}$ the added accuracy is ln(1)—zero—which is the greatest possible accuracy (others are negative).

Our job is to now compute the derivative so that we can perform optimization. It is important to note that, though this detail was skipped over before, we will be using partial derivatives with respect to **w**. If we used regular derivatives, it would imply that the variables **y** and **x** (among others including **m**) would be non-constant and functions of **w**, and we would need to consider their derivatives with respect to **w** by applying the chain rule. However, in our context, our cancer tumor dataset stays constant over all procedures and are not functions of **w**, and hence we are using partial derivatives to express and imply this. Our first obstruction is the use of the summation, but we can quickly eliminate this using differentiation rules:

$$\frac{\partial}{\partial w} \sum_{i=1}^{m} \left[y_i \ln\left(P(y=1 \mid x;w)_i\right) + (1-y_i)\ln\left(1-P(y=1 \mid x;w)_i\right)\right]$$

(1) Represent closed form summation as the addition of a collection of n terms

$$= \frac{\partial}{\partial w}\left[y_1 \ln\left(P(y=1 \mid x;w)_1\right) + (1-y_1)\ln\left(1-P(y=1 \mid x;w)_1\right) + \ldots + y_m \ln\left(P(y=1 \mid x;w)_m\right) + (1-y_m)\ln\left(1-P(y=1 \mid x;w)_m\right)\right]$$

(2) Apply rule: derivative of a sum of functions of w is equal to the sum of the derivatives of each function. We can group it such that each derivative is of the sum of our two separate ln terms.

$$= \frac{\partial}{\partial w}\left[y_1 \ln\left(P(y=1 \mid x;w)_1\right) + (1-y_1)\ln\left(1-P(y=1 \mid x;w)_1\right)\right] + \ldots + \frac{\partial}{\partial w}\left[y_m \ln\left(P(y=1 \mid x;w)_m\right) + (1-y_m)\ln\left(1-P(y=1 \mid x;w)_m\right)\right]$$

(3) Express P(y = 1|x;w) as our prediction function, and recall nth feature vector in X is denoted $\mathbf{X_n}$

$$= \sum_{i=1}^{m} \left[\frac{\partial}{\partial w} y_i \ln\left(\frac{1}{1+e^{-w^T X_i}}\right) + \frac{\partial}{\partial w}(1-y_i)\ln\left(1-\frac{1}{1+e^{-w^T X_i}}\right)\right]$$

12

Kapur

Before we move ahead, let's precompute the derivative of the probabilistic sigmoid:

$$\frac{d}{\quad} \quad \frac{d}{\quad}\left(\frac{1}{\quad}\right) \quad \frac{d}{\quad}(\quad)$$ **(1) Re-write function 1/x as the power rule**

$$\frac{d}{dx}P(x) = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \frac{d}{dx}\left((1+e^{-x})^{-1}\right) = -(1+e^{-x})^{-2}\cdot e^{-x}\cdot -1$$

(1) Rewrite function 1/x as x^{-1}, apply power and chain rule

$$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2} = \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2}$$

(2) Represent our derivative as a fraction, separate into subtraction of two fractions by adding and subtracting 1

$$= \frac{1}{1+e^{-x}}\left[1 - \frac{1}{1+e^{-x}}\right] = P(x)\left[1-P(x)\right]$$

(3) Recognize that P(x) can be taken out as a factor, where the inner product comes out as (1 - P(x))

This will help us when we apply the chain rule. Now, let's compute our main partial derivatives. We'll sub back in the probability notation **P(y = 1|x;w)** (for the actual function) to simplify the process. One matrix calculus rule is employed:

$$\sum_{i=1}^{m}\left[\frac{\partial}{\partial w}y_i \ln P(y=1|x;w)_i + \frac{\partial}{\partial w}(1-y_i)\ln\left(1-P(y=1|x;w)_i\right)\right]$$

(1) Take out constant terms, apply derivative rule of $\ln(x) = x^{-1}$, apply chain rule

$$= \sum_{i=1}^{m}\left[y_i \frac{1}{P(y=1|x;w)_i}\frac{\partial}{\partial w}P(y=1|x;w)_i + (1-y_i)\frac{1}{1-P(y=1|x;w)_i}\frac{\partial}{\partial w}\left(1-P(y=1|x;w)_i\right)\right]$$

$$= \sum_{i=1}^{m}\left[y_i \frac{1}{P(y=1|x;w)_i}\frac{\partial}{\partial w}P(y=1|x;w)_i - (1-y_i)\frac{1}{1-P(y=1|x;w)_i}\frac{\partial}{\partial w}P(y=1|x;w)_i\right]$$

(2) Take out the partial derivative of P(y = 1|x;w) out as a factor

$$= \sum_{i=1}^{m}\left[\frac{\partial}{\partial w}P(y=1|x;w)_i\left(y_i\frac{1}{P(y=1|x;w)_i} - (1-y_i)\frac{1}{1-P(y=1|x;w)_i}\right)\right]$$

(3) Multiply by precomputed derivative of P(y = 1|x;w), as well as derivative of input (chain rule)

$$= \sum_{i=1}^{m}\left[\frac{\partial}{\partial w}(w^T X_i)\cdot\left(y_i\frac{1}{P(y=1|x;w)_i} - (1-y_i)\frac{1}{1-P(y=1|x;w)_i}\right)\cdot P(y=1|x;w)_i\left(1-P(y=1|x;w)_i\right)\right]$$

(4) Apply matrix calculus rule: Since $a^Tb = b^Ta$ where a and b are equally sized vectors, derivative of w^TX_i with respect to w is the same as the derivative of X_i^Tw with respect to w, which equals X_i^T

$$= \sum_{i=1}^{m}\left[X_i^T\cdot\left(y_i\frac{1}{P(y=1|x;w)_i} - (1-y_i)\frac{1}{1-P(y=1|x;w)_i}\right)\cdot P(y=1|x;w)_i\left(1-P(y=1|x;w)_i\right)\right]$$

(5) Distribute sigmoid derivative $P(y = 1|x;w)_i(1-P(y = 1|x;w)_i)$ inside factor, cancel like terms in numerator and denominator

$$= \sum_{i=1}^{m}\left[X_i^T\cdot\left(y_i\frac{P(y=1|x;w)_i\left(1-P(y=1|x;w)_i\right)}{P(y=1|x;w)_i} - (1-y_i)\frac{P(y=1|x;w)_i\left(1-P(y=1|x;w)_i\right)}{1-P(y=1|x;w)_i}\right)\right]$$

$$= \sum_{i=1}^{m}\left[X_i^T\cdot\left(y_i\left(1-P(y=1|x;w)_i\right) - (1-y_i)P(y=1|x;w)_i\right)\right]$$

(6) Expand inner products/factors, cancel out $-y_iP(y = 1|x;w)_i$ and $+y_iP(y = 1|x;w)_i$

$$= \sum_{i=1}^{m}\left[X_i^T\cdot\left(y_i - y_iP(y=1|x;w)_i - P(y=1|x;w)_i + y_iP(y=1|x;w)_i\right)\right]$$

$$= \sum_{i=1}^{`} \left(y_i - P(y=1 \mid x; w)_i \right) X_i^I$$

(7) Sub back in the actual prediction function for P(y = 1|x;w) to get final representation

$$\frac{\partial \ell(w)}{\partial w} = \sum_{i=1}^{m} \left(y_i - \frac{1}{1+e^{-w^T X_i}} \right) X_i^T$$

The simplified expression for the derivative of our log-likelihood function $\ell(\mathbf{w})$ is presented in the final step. With this first order derivative in hand, we are now equipped to perform numerical maximization to find the most optimal values for **w**.

VII. Optimization: Maximization

Now we set our derivative to 0 to find a maximum point and hence solve for optimum **w**. A convenient property of the derivative of the log-likelihood function $\ell(\mathbf{w})$ is that it is always convex, and hence will only have one turning point (the global maximum) and no other local extremas that we need to evaluate.

$$\frac{\partial \ell(w)}{\partial w} = \sum_{i=1}^{m} \left(y_i - \frac{1}{1+e^{-w^T X_i}} \right) X_i^T = 0$$

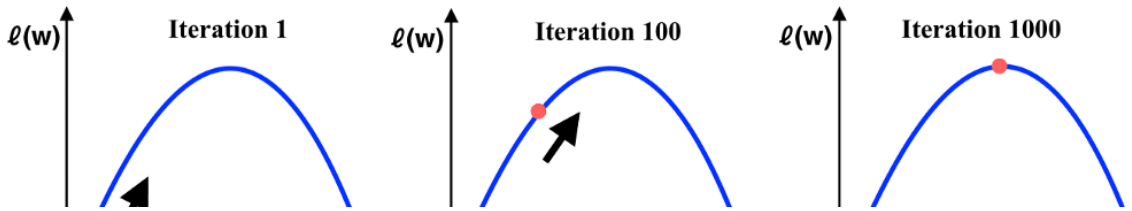
$$w = ?$$

Unfortunately, we hit another roadblock; this function is a transcendental equation and there does not seem to be a way to solve for **w** in closed form. In fact, there is no known solution for this equation except in extremely trivial cases ie. when input features are categorical instead of real (eg. 0 and 1) and there are only two observations/data points. Said cases are so rare in real world datasets that we will not explore further.

So now, instead of solving this equation directly we will perform an iterative optimization (where we "move" towards the maximized value over time). One such example is gradient ascent, where we set **w** to some initial guess (usually just a vector of 0s is fine) and then slowly perturb/update it to a more optimal state, until we "converge" and are at an (almost) completely optimal state where the gradient tends to 0. We denote this optimal state of **w** as $\mathbf{w}_*$. Each weight can either be increased or decreased—and only one of these operations will decrease the gradient and make it closer to 0. Hence we want to, over time, progress **w** to become closer to the global maximum. Figures 7-9 illustrate this notion of an iterative procedure.

14

Kapur



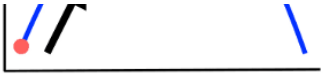


Figure 7

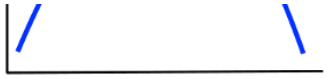


Figure 8

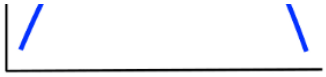


Figure 9

In each of these figures, I illustrate the procedure for a single arbitrary weight $\mathbf{w_j}$, just because it will be easier (and possible!) to visualize in 2-dimensions rather than n-dimensions. Hopefully they provide intuition to the name "gradient ascent", since we slowly but surely climb up the gradient to reach the maximum point. In Figure 7, we are at some wholly suboptimal state; we infer that the maximum point is to the right of our current position (and is far ahead), so we take a fairly large step, which leads into Figure 8. It's important to note that in these two figures we have actually spanned over 99 iterations. This is not a fixed number by any means (it will depend on your problem and dataset), but it provides a general guideline for the magnitude of steps required by this procedure. In Figure 8, we are closer to the maximum but are not quite there yet. Notice that the gradient is much smaller at this point, so we take even smaller steps over a greater number of iterations, until at iteration 1000 (for example) we have "converged" at the global maximum and no longer move.

So, how do we ultimately infer both the direction and magnitude of each step in gradient ascent? Well, at each iteration, we can take the current gradient (using the derivative computed before) to calculate both variables. In particular, note that if $\mathbf{w_j} < \mathbf{w_{j*}}$ then we are to the left of the global maximum and need to move right by increasing the magnitude of $\mathbf{w_j}$. At this point, since $\boldsymbol{\ell}(\mathbf{w})$ with respect to $\mathbf{w_j}$ would be increasing, the gradient would be positive. If $\mathbf{w_j} > \mathbf{w_{j*}}$, we would need to decrease the magnitude of $\mathbf{w_j}$. At this point, since $\boldsymbol{\ell}(\mathbf{w})$ with respect to $\mathbf{w_j}$ would be decreasing, the gradient would be negative. This is a nice property: when we need to increase $\mathbf{w_j}$, the gradient is positive, when we need to decrease it, the gradient is negative. Hence, we can add the gradient vector (called a "Jacobian matrix") to the current value for $\mathbf{w}$ (which would update each $\mathbf{w_j}$) at each update stage. However, these gradients may be too large, causing us to overshoot the maximum point and end up diverging from—instead of converging to—the maximum. So we usually multiply the gradients by a small number $\boldsymbol{\alpha}$, $|\boldsymbol{\alpha}| < \mathbf{1}$. $\boldsymbol{\alpha}$ is a hyper-parameter, meaning that one can try different values for it until a suitable one is found. Thus, our update rule is:

15

Kapur

$$w := w + \alpha \frac{\partial}{\partial w} \ell(x)$$

$$\therefore w := w + \alpha \sum_{i=1}^{m} \left(y_i - \frac{1}{1 + e^{-w^T X_i}} \right) X_i^T$$

When $\mathbf{w} = \mathbf{w_*}$, the gradient would be zero and we would update by 0 (not update at all). Hence, once we

When $\mathbf{w} = \mathbf{w}_*$, the gradient would be zero and we would update by 0 (not update at all). Hence, once we reach the maximum point, we get stuck and converge. It's important to note that, when actually computed, we don't reach the maximum point, because at each step the gradient becomes smaller and smaller. Instead, we tend towards the maximum point so $\mathbf{w} = \mathbf{w}_* - \boldsymbol{\varepsilon}$ or we slightly overshoot so $\mathbf{w} = \mathbf{w}_* + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is an extremely small number. This is rarely an issue, however; we usually set a maximum number of iterations (another hyper-parameter) and then stop after said iterations are complete. Theoretically, however, we can assert the following:

$$\text{perform until } \frac{\partial}{\partial w}\ell(x) = 0:$$
$$w := w + \alpha\sum_{i=1}^{m}\left(y_i - \frac{1}{1+e^{-w^T X_i}}\right)X_i^T$$

And that wraps up gradient ascent. The question, however, remains—how can we compute, by hand, a Jacobian matrix on a proper dataset with complex operations, updating our weights thousands of times? Well, the reality is that we need to have a computer to perform gradient ascent. That's okay—all of these applications (tumor cancer detection, spam detection, image recognition) exist digitally. However, even with this in mind, it can take up to weeks or months to train on a large-scale dataset used in industry, because of the size of the Jacobian matrix to compute at each iteration and the number of iterations usually required. For this reason, other optimization methods like Newton-Raphson are used. The Newton-Raphson method converges in fewer iterations, but at the same time each iteration requires the computation of a Hessian matrix (second-order derivatives), which is more computationally expensive and time taking. Thus, optimization for these non-linear models is actually a major bottleneck for applications being made with this technology.

VIII. Application

Many institutions have published medical breast cancer tumor data online that probabilistic regression can be applied to. The University of Wisconsin, Madison in the United States of America is most notable for this effort. They offer a sample code number for each patient, followed by a series of tumor features, and finally a

field indicating whether the tumor is malignant or benign. I downloaded the dataset and imported it into Microsoft Excel. Figure 10 displays an excerpt (6 out of **m = 699** instances) of the data.

	A	B	C	D	E	F	G	H	I	J	L
1	Sample Code Number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class (Malignant = M, Benign = B)
2	1000025	5	1	1	1	2	1	3	1	1	B
3	1002945	5	4	4	5	7	10	3	2	1	B

4	1015425	3	1	1	1	2	2	3	1	1	B
5	1016277	6	8	8	1	3	4	3	7	1	B
6	1017023	4	1	1	3	2	1	3	1	1	B
7	1017122	8	10	10	8	7	10	9	7	1	M

Figure 10

Column A, the sample code number, can be thought of as simply a reference to the instance and is not a feature in itself because it does not contribute to the outcome. *Columns B-J* for a single row, however, are features and could be thought of as inside a feature vector **x**. All the rows in these columns put together would make matrix **X**. *Column L* is the category/class (malignant or benign) that can be thought of as a value in the output vector **y**. Thus, this is our training set. 458 of the instances are benign and 241 are malignant.

I have provided my algorithm (written in MATLAB) with the data. I will perform probabilistic regression with regards to just two features in two dimensions for visualization purposes. There are ${}^{9}P_{2} = 72$ combinations of feature pairs, and Figure 11 shows a good correlation I found between **Single Epithelial Cell Size** and **Uniformity of Cell Size**, plotted by my program. In said figure, a plus represents a malignant tumor, and a circle represents a benign tumor.

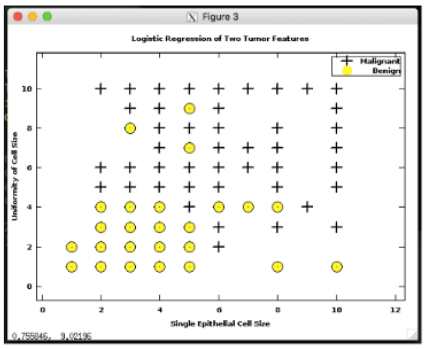
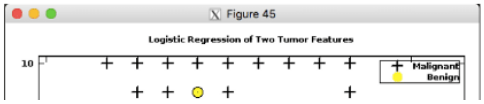


Figure 11

The data is noisy (a few anomalies), but this is to be expected from any real-world/industrial application dataset; we don't need 100% accuracy, we just need to find the general pattern presented since noise is stochastic. More importantly, however, this data is just taking into account two features; later we will look at combining all features. This is also why there seem to be very few data points; many of them overlap/are

duplicates. I ran gradient ascent for 1000 iterations to calculate the optimum weights and decision boundary. I ensured to append a column of 1s to the beginning of matrix **X** so that, as discussed earlier, the ability for the weighted sum $\mathbf{w}^{T}\mathbf{x}$ to experience a vertical shift by enabling a nonzero y-intercept in the decision boundary. Figure 12 shows the decision boundary that was formulated.



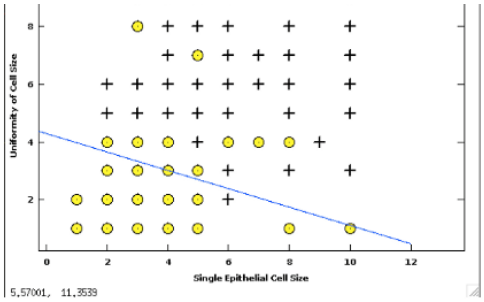


Figure 12

Figure 13 shows statistics regarding the formulated decision boundary.

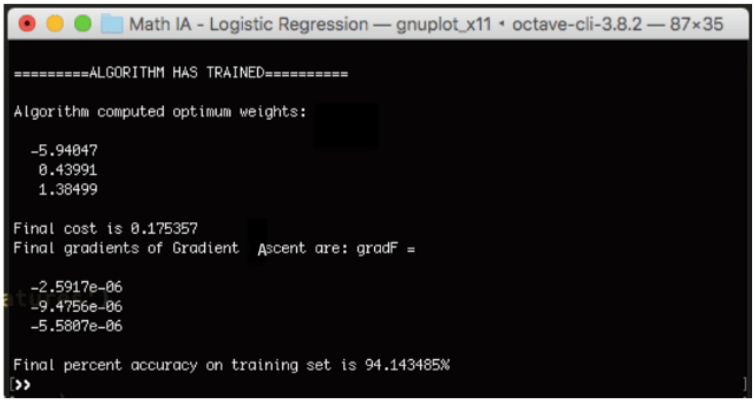


Figure 13

We have three weights, with the first being the constant term and other two corresponding to the "Single Epithelial Cell Size" and "Uniformity of Cell Size" respectively. The algorithm decided that a y-intercept of **-5.9** is suitable, as well as a coefficient of **0.43991** for the "Single Epithelial Cell Size" and **1.30499** for the "Uniformity of Cell Size". This implies that the "Uniformity of Cell Size" variable is more important in indicating the status of a tumor versus the "Single Epithelial Cell Size" variable. Our final percent accuracy is an impressive **94%**! This was calculated based on how many of the discrete predictions (benign or malignant) my algorithm got correct on the data points given. The "cost"/error or the output of $\ell(\mathbf{w})$ is very low at just **0.18**. In addition, we can tell we have converged to the maximum point because the Jacobian

matrix consists of very small values, all of which contain **e-06**, which equates to $\cdot 10^{-6}$. This tells me that the value I chose for the hyper-parameter α, at $\alpha = 0.003$, was suitable.

Though I demonstrated a 2D example, $\mathbf{n = 9}$ for our dataset (9 features) and so we should really be applying probabilistic regression on all of these variables. Though we can't visualize the results on a Cartesian plane, we can certainly analyze the statistics. In fact, I managed to obtain a **97%** accuracy on these 9 features. Now, I will demonstrate how a new patient's tumor data can be inputted to mathematically derive their tumor

status. So, let's say that a patient arrives to the doctor's office and the doctor records the following data for this person:

- Clump Thickness = 6
- Uniformity of Cell Size = 4
- Uniformity of Cell Shape = 4
- Marginal Adhesion = 8
- Single Epithelial Cell Size = 9
- Bare Nuclei = 22
- Bland Chromatin = 4
- Normal Nucleoli = 5
- Mitoses = 10

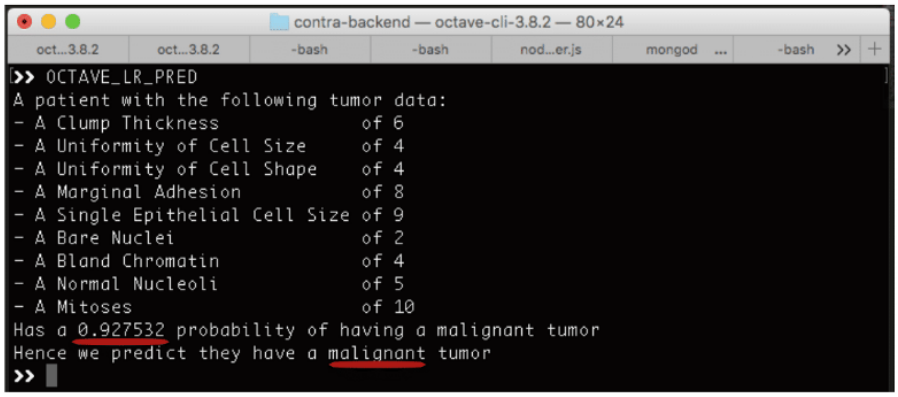
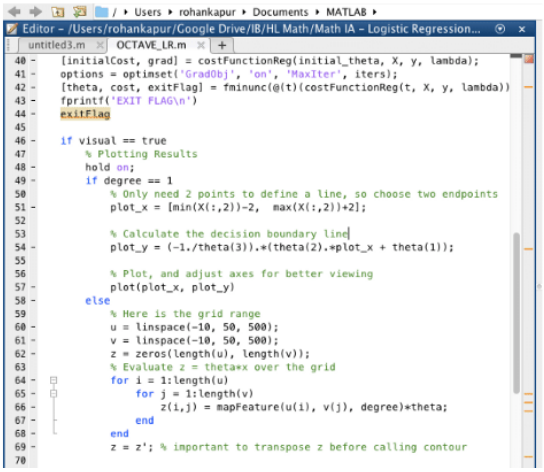


Figure 14

The algorithm informs us that, for this patient, **P(y = 1|x;w) = 0.93**. Hence, we will tell them that they, tragically, have a malignant tumor, and will proceed with medication from there. The following is a screenshot of my programming setup and an excerpt of my code:



```
        % Plot z = 0
        % Notice you need to specify the range [0, 0]
        contour(u, v, z, [0, 0], 'LineWidth', 2)
    end
end
```

IX. Conclusion

Though one may think that a regression is just used to fit models to a linear/polynomial dataset, I have shown that it can be used for a task like predicting probabilities—ultimately performing classification. And this need not be limited to 2-dimensions; instead, decision boundaries in n-dimensions (hyperplanes) can be found that separate the data nicely. When data becomes high-dimensional, we lose the ability to visualize correlation, and matrices of numbers are not at all intuitive to infer relationships from. Thus, we must introduce the correct mathematics that will enable us to solve such a problem. And as I have shown in the application section, this type of mathematical theory is used in our everyday lives from subtle conveniences like filtering spam to larger, more significant and existential matters like detecting and diagnosing cancer—with 97% accuracy!

The intuition behind the mathematics is actually quite simple. We formulate a model that can output probabilities, and pass a linear combination of the input features $\mathbf{x}$ and parameters $\mathbf{w}$ as input—allowing us to aggregate all these varying factors into one "score". As the value of the score increases, the probability of the tumor being malignant (for example) increases. The parameters $\mathbf{w}$ are explanatory, so a greater magnitude at $\mathbf{w_n}$ indicates that variable $\mathbf{x_n}$ is significant to classification. This is important because it allows us—via mathematics—to gain insight into what variables really affect an outcome and to what extent. This type of information can, of course, be used in other contexts. We optimize the model such that $\mathbf{w}$ gets closer and closer to its optimal state $\mathbf{w}_*$, and we do this via an iterative approach because, surprisingly, we are unable to simply equate the derivative to zero. We can further evaluate said approach and its pragmatic advantages and disadvantages. Once we have $\mathbf{w} = \mathbf{w}_*$ or rather $\mathbf{w} \approx \mathbf{w}_*$ we can now use the model to make new predictions. A

"decision boundary" that separates the two classes can be inferred, and it will allow us to turn probabilities into discrete, absolute states. Though probabilities tell us more precise information, we need discrete states so we can make choices ie. giving cancer treatment to a patient.

Although I feel that 97% is an impressive benchmark, 3% error for diagnosing patients of an illness critical to their lives is still an extremely serious matter, and even just 1% would make all the difference. It may in fact be that instead of a linear fit, we would prefer a polynomial (ie. quadratic) decision boundary. Such a decision boundary is shown in Figure 15. Since our logistic input is always a linear combination between feature and weights $\mathbf{w}^T\mathbf{x}$, it is currently unclear how such a decision boundary could be achieved; perhaps we

feature and weights $\mathbf{w}^T\mathbf{x}$, it is currently unclear how such a decision boundary could be achieved, perhaps we could append squared features, cubed features etc. to **x** and extend **w** to accommodate the same size. However, in a large dataset where **n** (number of features) is large, the number of polynomial terms would grow combinatorially, which could make computation infeasible. For this reason, other algorithms like ANNs (Artificial Neural Networks) exist that can find non-linear relationships much more simply. This is something interesting to further investigate.

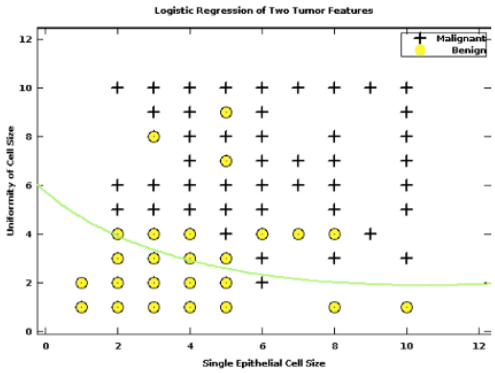


Figure 15

However, a question is posed here—just because the quadratic function fits our data with less error, does it really describe the relationship between the variables? We could have a model with an order of 8 that may "bend" the curve to accommodate all the noise and anomalies in the dataset, reducing error to almost zero. Though this model is accurate, we cannot say it will perform well in predicting future data. Thus, we may be able to perhaps add penalization of model complexity to the likelihood function to prevent this from occurring. Another issue is that not all errors are equal to others, which is a flaw in the current approach. For example, if we predicted a patient had a cancerous tumor when they in reality did not—a false positive—it is not a big deal… "better safe than sorry", if you shall! However, if we predicted a patient did *not* have a cancerous tumor when in fact they did—a false negative—it may cost them their life. Hence, it might be a

good idea to modify our likelihood function to take into account these potential implications, to penalize false positives more than false negatives.

Ultimately, there are many real world factors to take into account when performing these kinds of regressions, and these different issues have independent mathematical solutions. In the future, I would like to continue exploring how I can use mathematics to solve real world problems via my passion for computer science and artificial intelligence.

22

Machine Learning Artificial Intelligence Today I Learned

About Help Legal

Get the Medium app