Week 04: Nonlinear Decision Boundaries - Logistic Regression

1 From Regression to Classification

So far you have become familiar with regression problems and the linear regression model. These topics were introduced not only earlier in this 16ML course, but you should've also been exposed to these ideas from EE16A and EE16B. As review, linear regression is an important model in the **supervised learning** category. Given a vector in feature space, linear regression models solve the regression problem by predicting the response variable using a linear model in the feature space. In regression problems, predictions are always continuous numerical values. However, sometimes, the response variables can only take on a limited number of values. We predict a category that a data point belongs to, not a continuous number as we have in regression problems. We may interpret classification problems as a type of regression problem on probability. It predicts the probability that a data point belongs to some category.

You may think **classification problems** are easier than **regression problems**, since now we only have a finite number of labels to be predicted, instead of an infinite number on a continuous interval. However, with fewer values in the response variable, we will have more restrictions on our model for prediction. In this section, we'll first look at why linear regression models are no longer suitable for solving classification problems. We'll review several classification models and their restrictions you have seen so far. This will give you intuition on how we may improve a classification model.

1.1 Pitfalls of Linear Regression

Recall from the previous module and from your 16 series classes that the linear regression prediction is given by

$$\hat{y} = \vec{\beta}^T \vec{x} = \beta_0 x_0 + ... + \beta_n x_n$$

We first want to consider using linear regression models to directly solve classification problems. There are a few important things to notice here.

- Linear regression models give predictions in continuous values that may go out of label values $\{0, 1\}$ in a binary classification problem. In general, regression predictions violate the property in classification problems that the response variables take on discrete, bound values.

- We may draw a decision boundary for a linear regression model applied in a classification setting. This is usually a hyperplane in the feature space. But we cannot interpret values on the hyperplane or separated by it. We want to interpret our classification model's output as the probability that a data point belongs to a certain class. But linear regression models cannot be understood in this way.
- It's impossible to derive a general decision rule for linear regression model for classification. In the binary case, we may set certain "threshold" value, so that x is predicted as class 1 if linear regression model gives $\hat{y}$ greater than threshold value, and class 0 otherwise. However, we cannot generalize this rule to multiple categories, and the reasoning behind how we determine the threshold isn't universal.

Intuitively, we might want to slightly change our linear regression model so that it gives predictions of different classes, but still preserves its linearity in feature space. This is where some generalized linear models in classification come from, and is a crucial motivator of logistic regression.

1.2 What classification models have we seen so far?

So far we've only talked about linear models; that is, our predictions have been linear transformations of input features. You should recall that linear regression finds the optimal weight vector given input features and observations, and predictions are given by the product of said weight vector and test features. Remember that the closed form solution is given by

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

Given data points with labels, perceptrons and support vector machines (SVM) generate linear decision boundaries in feature space. You should be familiar with properties and limitations of them. Let's review their behavior to motivate an improvement on these methods. This will give us intuition on why we need a logistic regression model in classification.

1.2.1 Perceptrons

Perceptron algorithms work as intended for linearly separable data. Assume we only deal with a binary classification problem. In the feature space in R^d, perceptrons find a $d-1$ dimensional hyperplane that perfectly separates data points from each category. For linearly separable training data, perceptrons can achieve perfect accuracy on the training set, but may not have perfect accuracy on the test set, either because the test set has a

different distribution from the training set, or because the test set is not linearly separable. Perceptrons can easily fail and are highly sensitive to the structure of input data. This can lead to a generalization problem, and is not something we want to find.

If training set is linearly separable, there are infinite separating hyperplanes that perfectly classify all data points. Perceptrons will generate a random one as the solution to binary classification problem. But you should always ask, does the random one that perceptrons select always generalize to the test set? This is why we turn this problem about finding such a separating hyperplane in feature space to an optimization problem.

1.2.2 Support Vector Machine (SVM)

Support vector machines find a similar separating hyperplane as perceptrons. Among all feasible hyperplanes, SVM finds the unique solution that maximizes the margin of the hyperplane to data points in both classes. Intuitively, it maximizes the distance from data points to the hyperplane. Data points that belong to the same class should be close to each other in the feature space. For linearly separable training set, SVM outputs a unique solution. You should also remember that for not linearly separable data, you can add a slack penalty to perform the alternative soft margin SVM, and still get a solution.

However, both perceptrons and SVMs only generate decision boundaries that are linear in the feature space. They fail on training set with nonlinear decision boundaries. This is why we want to introduce a method to generate nonlinear decision boundary in binary classifier. Logistic regression model can be interpreted as a "generalized" linear model. It preserves the linear transformation of the feature vector, but is also able to generate nonlinear decision boundary that can generalize to test set and can be applied to wider ranges of training set.

2 Logistic Regression Theories

The logistic regression model is understood as a "generalized" linear model in classification. More specifically, given a vector $x \in R^d$ in feature space, it generates a "score" as a linear combination of features, and transforms this score to a probability between 0 and 1. The predicted probability is based on logistic function called **sigmoid**, and it is associated with a decision rule to predict the exact category value.

2.1 Logistic Activation Function: Sigmoid

Logistic activation function sigmoid is defined by

$$s(t) = \frac{1}{1 + e^{-t}}$$

Given a raw "score" by a linear combination of features, the sigmoid function turns this continuous numerical value to a probability between 0 and 1. This is the predicted probability given by logistic regression that the data point belongs to class 1, in a binary classification problem. Here are some properties of the sigmoid function:

- Domain of $s(t)$: $-\infty < t < \infty$
- Range of $s(t)$: $0 < s(t) < 1$
- Threshold Value: $s(0) = 0.5$
- Reflection and Symmetry of $s(t)$:

$$1 - s(t) = 1 - \frac{1}{1 + e^{-t}} = \frac{e^{-t}}{1 + e^{-t}} = s(-t)$$

- Inverse of $s(t)$:

$$s(t) = z = \frac{1}{1 + e^{-t}}, 0 < z < 1$$

$$e^{-t} = \frac{1 - z}{z}$$

$$t = -\log(\frac{1 - z}{z}) = \log(\frac{z}{1 - z}), 0 < z < 1$$

- Derivative of $s(t)$ using chain rule:

$$s'(t) = -\frac{-e^{t}}{(1 + e^{-t})^2} = \frac{1}{1 + e^{-t}} \cdot \frac{e^{-t}}{1 + e^{-t}} = s(t)(1 - s(t))$$

2.2 Decision Rule

The logistic regression model takes in a feature vector $x \in R^d$ and outputs a predicted label that the test point belongs to. In a binary classification problem, it outputs the predicted probability that the test point belongs to class label 1. Specifically, the test point is associated with a raw "score", which is given by a linear model of x:

$$t = \theta^T x$$

, where θ is the weight vector of features in the logistic regression model. Then numerical value of t is transformed into a probability by the sigmoid function $s(t)$. The decision rule is described as

- If $s(\theta^T x) \geq \frac{1}{2}$, predict this data point as class 1.
- If $s(\theta^T x) < \frac{1}{2}$, predict this data point as class 0.

In a binary classification problem, the predicted probability that data point x has label 1 is therefore given by

$$f_\theta(x) = s(t_x) = s(\theta^T x)$$

. Since any label $y \in \{0, 1\}$, we can rewrite predicted probabilities as a conditional distribution over label values given a feature vector

$$P(\hat{y} = 1|x) = s(\theta^T x)$$

$$P(\hat{y} = 0|x) = 1 - s(\theta^T x)$$

. Basically, the predicted class of a logistic regression model is

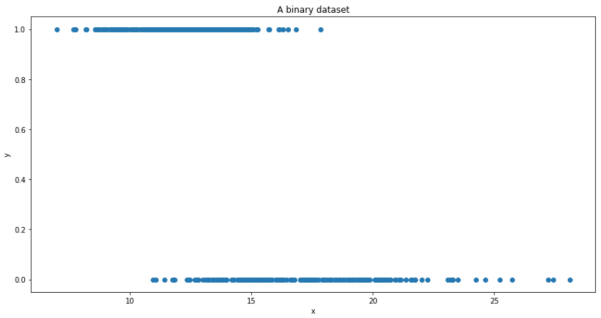
$$\hat{y}_{\text{class}} = \underset{\hat{y}}{\text{argmax}}(P(\hat{y} = 1|x), P(\hat{y} = 0|x)) = \underset{\hat{y}}{\text{argmax}}(s(\theta^T x), 1 - s(\theta^T x))$$

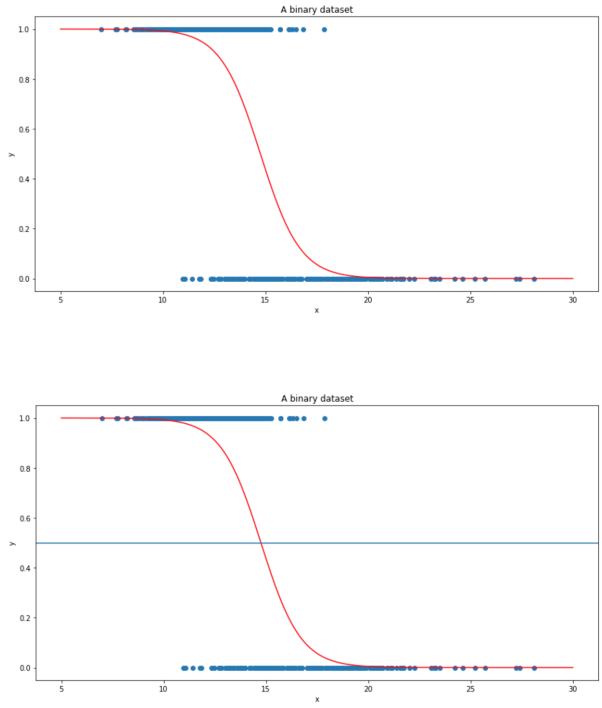
and the predicted probability of x in class 1 is

$$\hat{y}_{\text{probability}} = \max(P(\hat{y} = 1|x), P(\hat{y} = 0|x)) = \max(s(\theta^T x), 1 - s(\theta^T x))$$

.

You can think of our prediction rule as choosing a "threshold" value of $\frac{1}{2}$ based on the shape of sigmoid function. Then this threshold value is a hyperparameter in logistic regression model. As always, you can use cross validation to tune this hyperparameter with custom values. It's not always true that $\frac{1}{2}$ is the best threshold to implement the decision rule on unseen test data.





The figures above should help you visualize what's happening when we're fitting a logistic regression model. The first plot is a bare data plot, that shows the 0 and 1 labels for a single feature x. Below that, we've fitted a logistic regression model shows the shape of the sigmoid function. Clearly, the range is between 0 and 1, and it allows us to interpret the output as probabilities of belonging to the class 1. In the bottom plot, the horizontal line depicts the decision boundary of 0.5. We classify points with sigmoid output above 0.5 as label 1, and below 0.5 as label 0.

2.3 Loss Function

Now we can officially define logistic regression model as a binary classification problem. Given a feature vector of test point $x \in R^d$, logistic model is a regression model on probabilities: $f_\theta(x) = s(\theta^T x)$. Like linear regression, logistic regression has a single model parameter θ, the weight vector for each single element in the feature vector x.

Recall that in linear regression, on continuous values, we find optimal weight vector w by minimizing the mean squared error of our linear regression model on a training set. Intuitively, in binary classification problem, we want the shape of the sigmoid function, as you saw in the previous page, to approximate training points as close as possible. We want to define a loss function that can be generalized to classification problem. We will minimize this loss function to find the optimal parameters of for logistic regression. As we minimize

squared loss in regression problems, we will work with **cross-entropy loss** in classification problems.

2.3.1 Define Cross-Entropy Loss

Mean squared error doesn't work as intended in general classification problems. Now we have labels $\{1, ..., K\}$. You might have different numerical values of predicted probabilities $\frac{1}{2} \leq \hat{p} \leq 1$ which all end up with same predicted class k, but we still want the predicted probabilities to be as close to 1 as possible. Squared loss has limited effect in classification problems, because it doesn't account for these gaps in probability.

For logistic regression, we use the binary cross-entropy looss. Given dataset $X = \{(x_i, y_i)\}_{i=1}^n$, where x_i is input feature vector of the ith data point and X is $n \times d$ matrix, y_i is observed label values in $\{0, 1\}$, $f_\theta(x)$ is logistic regression model, θ is model parameter, cross-entropy loss is defined as

$$L(\theta, X, y) = \frac{1}{n} \sum_{i=1}^{n} -y_i \ln(f_\theta(x_i) - (1 - y_i) \ln(1 - f_\theta(x_i))$$

At each data point (x_i, y_i), cross-entropy loss is given by

$$l(\theta, x_i, y_i) = -y_i \ln(f_\theta(x_i) - (1 - y_i) \ln(1 - f_\theta(x_i))$$

. In binary case, y_i only takes on values $\{0, 1\}$.

- If $y_i = 0$, first term is set to zero

  $$l(\theta, x_i, 0) = \ln(1 - f_\theta(x_i))$$

  . Correct prediction $f_\theta(x_i) = 0$ incurs zero loss. The further the predicted probability $f_\theta(x_i) \geq 0$ is away from true label $y_i = 0$, the higher the loss.

- If $y_i = 1$, second term is set to zero

  $$l(\theta, x_i, 1) = \ln(f_\theta(x_i))$$

  . Correct prediction $f_\theta(x_i) = 1$ incurs zero loss. The further the predicted probability $f_\theta(x_i) \geq 0$ is away from the true label $y_i = 1$, the higher the loss.

Cross-entropy loss not only measures whether the predicted label is correct, it also accounts for the penalty of the predicted probability being too far away from the correct label. To fit an optimal logistic regression model, we want to minimize this cross-entropy loss incurred by model parameter θ.

2.3.2 Gradient of Cross-Entropy Loss

Now, let's take a look at how this cross-entropy loss results in training a model's parameters, such that we arrive at weights that can make accurate predictions. You should've seen how to implement stochastic gradient descent (mini-batches) from Week 2 of this course. EE16B should also have familiarised you with elementary vector calculus from the Vector Differential Equations module. Here, we'll use these principles to see how cross-entropy loss leads to training a logistic regression model.

We use the chain rule to compute the gradient of the cross-entropy loss with respect to θ. The derivative of the sigmoid function is given by

$$s'(t) = s(t)(1 - s(t))$$

. Logistic regression model at the ith data point is defined as

$$f_\theta(X_i) = s(\theta^T X_i) = s(X_i\theta)$$

where X_i is $n \times d$ feature matrix with feature vector x_i^T being in each row vector. Then gradient of f with respect to θ is

$$\nabla_\theta s(X_i\theta) = s(X_i\theta)(1 - s(X_i\theta))\nabla_\theta(X_i\theta) = s_i(1 - s_i)X_i$$

where define $s_i = s(X_i\theta)$ for simplicity.

We can rewrite the cross-entropy loss as

$$L(\theta, X, y) = \frac{1}{n}\sum_{i=1}^{n} -y_i \ln s_i - (1 - y_i)\ln(1 - s_i)$$

Use chain rule to simplify gradient of loss function with respect to θ

$$\nabla_\theta L(\theta, X, y) = \frac{1}{n}\sum_{i=1}^{n} -\frac{y_i}{s_i}\nabla_\theta s_i + \frac{1 - y_i}{1 - s_i}\nabla_\theta s_i$$

$$= -\frac{1}{n}\sum_{i=1}^{n} (\frac{y_i}{s_i} - \frac{1 - y_i}{1 - s_i})\nabla_\theta s_i$$

$$= -\frac{1}{n}\sum_{i=1}^{n} (\frac{y_i}{s_i} - \frac{1 - y_i}{1 - s_i})s_i(1 - s_i)X_i$$

$$= -\frac{1}{n}\sum_{i=1}^{n} (y_i(1 - s_i) - (1 - y_i)(s_i))X_i$$

$$= -\frac{1}{n}\sum_{i=1}^{n}(y_i - s_i)X_i$$

We want to fit the logistic regression model with optimal parameter

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, L(\theta, X, y)$$

. Since this loss function is not convex, we cannot find a closed form formula for $\hat{\theta}$, so we use gradient descent to iteratively approximate the optimal model parameter $\hat{\theta}$.

2.4 Training a Logistic Regression Model

Let's briefly discuss the gradient descent algorithm and its variations: batch gradient descent, mini-batch gradient descent, and stochastic gradient descent. Different methods have its advantages and disadvantages. Batch gradient descent is most accurate and has highest chance of convergence, but it takes the most time complexity, especially when the training set is large. Stochastic gradient descent has the least time complexity to compute, but it suffers from the risk of non-convergence if we use bad data points to compute gradient updates. Mini-batch is a balance between them, so it's most likely to be used in practice. Stochastic gradient descent was covered in Week 2 of this course, so these concepts should not be radically new.

In this section we'll discuss mathematical derivations of three gradient descent algorithms in training a logistic regression model in particular. The loss function we want to minimize is cross-entropy loss

$$L(\theta, X, y) = \frac{1}{n}\sum_{i=1}^{n} -y_i \ln(f_\theta(x_i) - (1 - y_i)\ln(1 - f_\theta(x_i))$$

We assume a learning rate (or step size) in our gradient descent of α. You should always check that the dimension of gradient updates makes sense.

2.4.1 Batch Gradient Descent

The general formula for updating with batch gradient descent is given by

$$\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla_\theta L(\theta^{(t)}, X, y)$$

We compute gradients over all points in the data set, so in batch gradient descent

$$\nabla_\theta L(\theta^{(t)}, X, y) = -\frac{1}{n}\sum_{i=1}^{n}(y_i - s_i)X_i$$

where $s_i = s(X_i\theta)$. Plugging back into the formula, batch gradient descent updates model parameters from the current estimate $\theta^{(t)}$ to the next iteration estimate $\theta^{(t+1)}$ by

$$\theta^{(t+1)} = \theta^{(t)} - \alpha(-\frac{1}{n}\sum_{i=1}^{n}(y_i - s_i)X_i)$$

$$= \theta^{(t)} + \alpha \cdot \frac{1}{n}\sum_{i=1}^{n}(y_i - s_i)X_i$$

Clearly, for batch gradient descent, we need to compute the current gradient of our loss function for every single data point for each iteration.

2.4.2 Stochastic Gradient Descent

The general formula for updating with stochastic gradient descent is given by

$$\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla_\theta l(\theta^{(t)}, X_i, y_i)$$

where i is randomly sampled from the dataset. In each iteration we pick a random data point x_i and compute the gradient update

$$\nabla_\theta l(\theta^{(t)}, X_i, y_i) = -(y_i - s_i)X_i$$

. We use the gradient of one data point to approximate the average over all data points, in order to save computational time complexity.

Plugging back into the formula, the stochastic gradient descent algorithm updates model parameter from the current estimate $\theta^{(t)}$ to the next iteration estimate $\theta^{(t+1)}$ by

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \cdot (-(y_i - s_i)X_i) = \theta^{(t)} + \alpha \cdot (y_i - s_i)X_i$$

where we randomly sample a data point x_i from the training set at each iteration.

2.4.3 Mini-Batch Gradient Descent

Mini-batch gradient descent compromises between the time complexity and accuracy of batch gradient descent and stochastic gradient descent. At each iteration, we randomly sample a subset of data points from the training set, denoted B. We use gradients computed from this smaller subset to approximate the gradients averaged over all data points in the training set.

$$\nabla_\theta L(\theta^{(t)}, X, y) \approx \frac{1}{|B|}\sum_{i\in B}\nabla_\theta l(\theta^{(t)}, X_i, y_i)$$

where at each single data point $x_i \in B$, we compute the gradient as

$$\nabla_\theta l(\theta^{(t)}, X_i, y_i) = -(y_i - s_i)X_i$$

Plugging back into the formula, in logistic regression model, mini-batch gradient descent algorithm updates model parameter from current estimate $\theta^{(t)}$ to next iteration estimate $\theta^{(t+1)}$ by

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \cdot \frac{1}{|B|} \sum_{i \in B} -(y_i - s_i)X_i$$

$$= \theta^{(t)} + \alpha \cdot \frac{1}{|B|} \sum_{i \in B} (y_i - s_i)X_i$$

With small enough sample size, we can make computation time complexity manageable by the machine.

2.5 Evaluation Metrics of Classification Models

2.5.1 Accuracy

Given a set of data points $(x_1, y_1), ..., (x_n, y_n)$, our classifier outputs a set of predictions $\hat{y_1}, ..., \hat{y_n}$, which are predicted labels in $\{1, ..., K\}$. The accuracy of a classifier is defined as the fraction of correct predictions.

2.5.2 Confusion Matrix

The confusion matrix compares what the model predicts with the actual counts in each class. There are two types of error that we are interested in:

- **False Positives**: The actual class is 0 (false) but the algorithm predicts 1 (true).
- **False Negatives**: The actual class is 1 (true) but the algorithm predicts 0 (false).

You should have the intuition that with different sources of error, we want to balance these types of error to minimize over all of them.

2.5.3 Precision and Recall

Intuitively in classification, it's not helpful if the classifier predicts most data points as negative, even though positive cases should be in smaller amount than negative cases. We

should focus on detecting positive cases from given features. So precision and recall are defined to identify how good we are in recognizing positive cases.

- **Precision**:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} = \frac{\text{True Positives}}{\text{Predicted True}}$$

- **Recall**:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} = \frac{\text{True Positives}}{\text{Actually Positive}}$$

You can plot precision vs recall with respect to different threshold values in a decision function. You would compute precision and recall at different classification thresholds, and then plot precision vs recall at each threshold.

2.5.4 AUC-ROC Curve

From our confusion matrix, we have two types of error: false positives (FP) and false negatives (FN). We are interested in **true positive rate (TPR)** and **false negative rate (FNR)**:

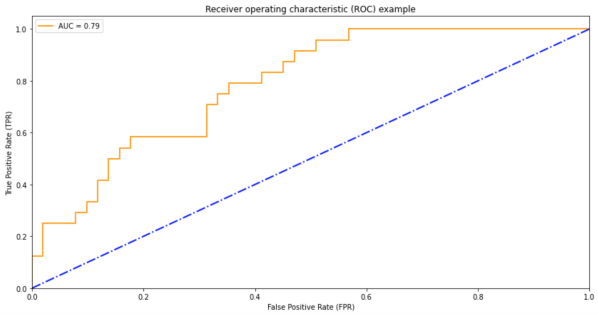
- **TPR**:

$$\text{TPR} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} = \frac{\text{True Positives}}{\text{Actually Positive}}$$

- **FNR**:

$$\text{FNR} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}} = \frac{\text{True Negatives}}{\text{Actually Negative}}$$

Similar to precision vs recall curve, ROC curve plots FNR vs TPR at different thresholds values. It is a probability curve and measures how the model performs in distinguishing different class labels. You should now realize that a random guessing classifier will have a ROC curve as a striaght line with a 45 degree slope. The higher the area under ROC curve (AUC), the better the model correctly recognizes between positive and negative cases.



This figure shows an example receiver operating characteristic (ROC) curve. The closer the curve is to the top right corner, the better our model. This translates to a higher area under the curve (AUC). For a random classifier, we expect a straight line with a 45 degree slop for a ROC curve. Recall that from a previous figure, we set the threshold as 0.5, and plotted a horizontal line with this value on the sigmoid. By varying this threshold from 0 to 1, and taking the TPR and FPR at each threshold, we arrive at the ROC curve above.

3 From Binary to Multiclass Classification Problem

So far we only discussed how logistic regression models solve binary classification problems. That is, the dataset only contain labels $0, 1$. However, in practice, labels may take on multiple categories, more than two. For example, we want to use logistic regression model to classify handwritten digits from 0 to 9. There are 10 categories in total, and binary logistic model doesn't work as intended.

In this section, we want to introduce the **softmax function**, which is based on the logistic activation function sigmoid, and discuss how to expand the binary logistic regression model to solve multi-class classification problems. In particular, we will introduce **one-vs-rest**, which is a common extension from binary classification. Specifically, we will review the sigmoid function, derive how it is a special case of the softmax function, and conclude that logistic regression in binary classification problem is a special case of one-vs-rest classification model.

3.1 One-Vs-Rest Classification and Softmax Function

Recall that logistic regression in binary classification with labels $(0, 1)$ works as follows:

Given a test data point with feature x and a logistic regression model with parameter θ, we compute a linear "score" $\theta^T x$ and convert it to a probability between 0 and 1 with

sigmoid function $s(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$. This is interpreted as the predicted probability of a test sample taking the value 1.

We can generalize the binary classification method to solve multi-class classification problems. The solution is to use the **one-hot-vector encoding** method to represent all labels. If observation y_i takes class value k among all possible label values $1, 2, ..., K$, instead of having scalar $y_i = k$, we enforce $y_i = e_k$ and thus $y_i \in R^K$. For example, in the example of classifying digits, an observation of digit 3 and be interpreted as $y_i = [0001000000]$.

Binary logistic regression model has d parameters represented by θ. Now we want to modify model parameters to conform with one-hot-vector encoding in multi-class classification. For each class in $1, ..., K$, we have d parameters as we treat each of them as a binary classification problem. In total we have $K \times d$ parameters represented by $\theta_1, ..., \theta_K$. For each feature input $x_i \in R^d$, each class k gives a "score" based on weight vector θ_k: $z_k = \theta_k^T x_i$. In total there are K raw scores of data point x_i:

$$z_1 = \theta_1^T x_i, ..., z_K = \theta_K^T x_i$$

Intuitively, the higher the score of class k is, the higher the probability x_i takes label value k, the more likely logistic regression chooses class k.

In binary classification, we take two raw scores $\theta_0^T x_i$ and $\theta_i^T x_i$, and transform them into probabilities between 0 to 1 using the sigmoid function $s(w)$. Generalization to the multi-class case is the **softmax function**. It takes all K scores of x_i in each class label j, and outputs a probability distribution corresponding to value j:

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

You can check this is indeed a probability distribution, because all entries are between 0 and 1 and all entries sum up to 1.

This only leaves the question of predicting a new test point in multi-class classification. In the binary case, given two probabilities $P(\hat{y} = 1|x) = s(\theta^T x)$ and $1 - P(\hat{y} = 1|x) = 1 - s(\theta^T x)$, we pick the label value with larger probability. Similarly, using softmax function, we generate a probability distribution on given test point x: $\sigma(z_1), ..., \sigma(z_K)$. Logistic regression model prediction takes the label value with the highest probability:

$$\hat{y}_i = \underset{j \in \{1,...,K\}}{\text{argmax}}\ \sigma(z_j) = \underset{j \in \{1,...,K\}}{\text{argmax}}\ \sigma(\theta_j^T x)$$

This is predicted class label $\hat{y}_i$ in $1, ..., K$. The predicted probability that test point is in class $\hat{y}_i$ is given by

$$\hat{y}_i = \max_{0 \le j \le 1} \sigma(z_j) = \max_{0 \le j \le 1} \sigma(\theta_j^T x)$$

Basically, by using one-hot encoding for categorical variables, we train K different weight vectors θ for each class, and obtain K different raw linear scores. We transform them into a probability distribution given by the sigmoid function, and predict with the class label with the highest probability among K classes. In multi-class classification, this probability distribution is proportional to $s(\theta^T x)$, similar to the binary case. So we conclude it as an extension and generalization of binary classification.

3.2 Review Sigmoid Function: Special Case of Softmax Function

We showed in the previous part that one-vs-rest multi-class classification problem can be viewed as an extension and generalization of binary classification. In fact, we can further show that the sigmoid function in the binary case outputs a probability distribution as a special case of the softmax distribution.

Suppose we train a logistic regression model with parameters θ_0 and θ_1 for class label 0 and 1, then our predicted probabilities given by softmax function are

$$P(\hat{y}_i = 1|x_i) = \sigma(\theta_1^T x_i) = \frac{e^{\theta_1^T x_i}}{e^{\theta_0^T x_i} + e^{\theta_1^T x_i}} = s((\theta_1 - \theta_0)^T x_i)$$
$$P(\hat{y}_i = 0|x_i) = \sigma(\theta_0^T x_i) = \frac{e^{\theta_0^T x_i}}{e^{\theta_0^T x_i} + e^{\theta_1^T x_i}} = 1 - s((\theta_1 - \theta_0)^T x_i)$$

We can use change of variables with $\theta = \theta_1 - \theta_0$ as the model parameter we learn in binary logistic regression. With only two classes, we are interested in the difference between them so there's no need to further define two model parameters.

4 Bonus: Probabilistic Interpretation

For those who are interested, we introduce **KL divergence** here, also called **relative entropy**, and demonstrate how minimizing average KL divergence in the empirical observed distribution of our data set and predicted probability distribution after fitting a logistic regression model is equivalent to minimizing cross entropy loss in binary classification. While some of the topics in this Bonus section are out of scope from what you've learned in the 16 series and 16ML so far, we leave this here for you to optionally gain a deeper mathematical understanding of the probabilistic interpretation of logistic regression.

Logistic regression in binary classification is given by the sigmoid function. It outputs a probability based on some linear transformation of features vector $x \in R^d$ weighted by optimal parameter $\hat{\theta}$. Specifically, it estimates the probability that the label y of a test point takes value 1 based on x.

Consider a simple 1-dimensional example. Suppose x denotes the mean radius of cells in the Wisconsin Breast Cancer dataset and $y = 1$ denotes the probability that the patient was diagnosed with benign breast cancer. A logistic regression model finds an optimal scalar parameter $\hat{\theta}$ to predict the probability that a patient has benign breast cancer given mean radius of cells, i.e. $P_\theta(\hat{y} = 1|x)$. Collecting data with different values of mean radius provides a method to construct an empirical probability distribution $P(y = 1|x)$. For each value of $x = k$ in dataset, we can compute $P_\theta(\hat{y} = 1|x = k)$. Intuitively, we want to fit a logistic model that for all values of x in the dataset, $P_\theta(\hat{y} = 1|x) \approx P(y = 1|x)$.

4.1 Definition of KL Divergence in Binary Classification

Given two probability distribution, we want to measure the amount they diverge from each other. Given probability distributions P and Q defined on the same outcome space, the KL divergence from Q to P is given by

$$D_{KL}(P||Q) = \sum_x P(x) \ln \frac{P(x)}{Q(x)}$$

This can be interpreted as the average log difference between P and Q weighted by the probability distribution of P.

In binary classification, we use KL divergence to quantify the difference between the probability distribution $\hat{P}_\theta$ computed by our logistic model with parameters $\hat{\theta}$ and the actual distribution P based on the dataset. Intuitively, it calculates how imprecisely the logistic model estimates the distribution of labels in data.

For each data point $(\vec{x}, y)$, KL divergence between P and P_θ is given by

$$D_{KL}(P(y)||P_\theta(\hat{y})) = P(y = 1|\vec{x}) \ln \frac{P(y = 1|\vec{x})}{P_\theta(\hat{y} = 1|\vec{x})} + (1 - P(y = 1|\vec{x})) \ln \frac{1 - P(y = 1|\vec{x})}{1 - P_\theta(\hat{y} = 1|\vec{x})}$$

Therefore, our goal in binary classification problem is to find $\hat{\theta}$ parameter in logistic regression that minimizes the sum of KL divergence of all $(\vec{x_i}, y_i)$ in the dataset given by

$$\sum_{i=1}^{n} P(y_i = 1|\vec{x_i}) \ln \frac{P(y_i = 1|\vec{x_i})}{P_\theta(\hat{y_i} = 1|\vec{x_i})} + (1 - P(y_i = 1|\vec{x_i})) \ln \frac{1 - P(y_i = 1|x_i)}{1 - P_\theta(\hat{y_i} = 1|\vec{x_i})}$$

4.2 Derive Cross Entropy Loss from KL Divergence

For each data point $(\vec{x}, y)$ and for label value l, rewrite

$$P(y = l|\vec{x}) \ln \frac{P(y = l|\vec{x})}{P_\theta(\hat{y} = l|\vec{x})} = P(y = l|\vec{x}) \ln P(y = l|\vec{x}) - P(y = l|\vec{x}) \ln P_\theta(\hat{y} = l|\vec{x})$$

For simplicity, define $y_i = P(y_i = 1|\vec{x_i})$ and $\hat{y_i} = P_\theta(\hat{y_i} = 1|\vec{x_i})$, rewrite objective function sum of KL divergence across all data points

$$\sum_{i=1}^{n} -y_i \ln(\hat{y_i}) - (1 - y_i) \ln(1 - \hat{y_i}) + y_i \ln y_i + (1 - y_i) \ln(1 - y_i)$$

Notice here $C = y_i \ln y_i + (1 - y_i) \ln(1 - y_i)$ is a constant observed from the dataset and is therefore independent of θ. It's equivalent to minimizing

$$\sum_{i=1}^{n} -y_i \ln(\hat{y_i}) - (1 - y_i) \ln(1 - \hat{y_i}) + C$$

This is exactly the sum of cross entropy loss between observed empirical distribution P and predicted probability distribution P_θ given by a logistic regression model across each data point $(\vec{x_i}, y_)$ in the dataset.

Now, we have another interpretation of minimizing cross entropy loss on training set in logistic regression model. We actually minimize the KL divergence between empirically observed distribution of the dataset and estimated distribution given by our logistic model. The optimal weight parameter $\hat{\theta}$ minimizes both of them.

5 Takeaways

In this note, we hope you gained both understanding and intuition for the motivation for logistic regression, the mathematical theory, the training/optimization process for a basic logistic regression model, the evaluation metrics, and (optionally) a probabilistic interpretation. In the following module, you will be introduced to regularization methods, which apply to all machine learning techniques in general, including logistic regression. The concepts you will learn in the regularization module can be combined with the concepts we taught here to make more robust logistic regression models for prediction.

References

[1] https://www.eecs189.org/static/notes/n17.pdf

[2] http://www.textbook.ds100.org/ch/17/classification$_intro.html$

[3] http://www.ds100.org/sp20/resources/assets/lectures/lec23/part1.html

[4] http://www.ds100.org/sp20/resources/assets/lectures/lec24/LogisticRegressionPart2.html

[5] https://github.com/topspinj/diabetes-ml-workshop/blob/master/notebooks/walkthrough.ipynb