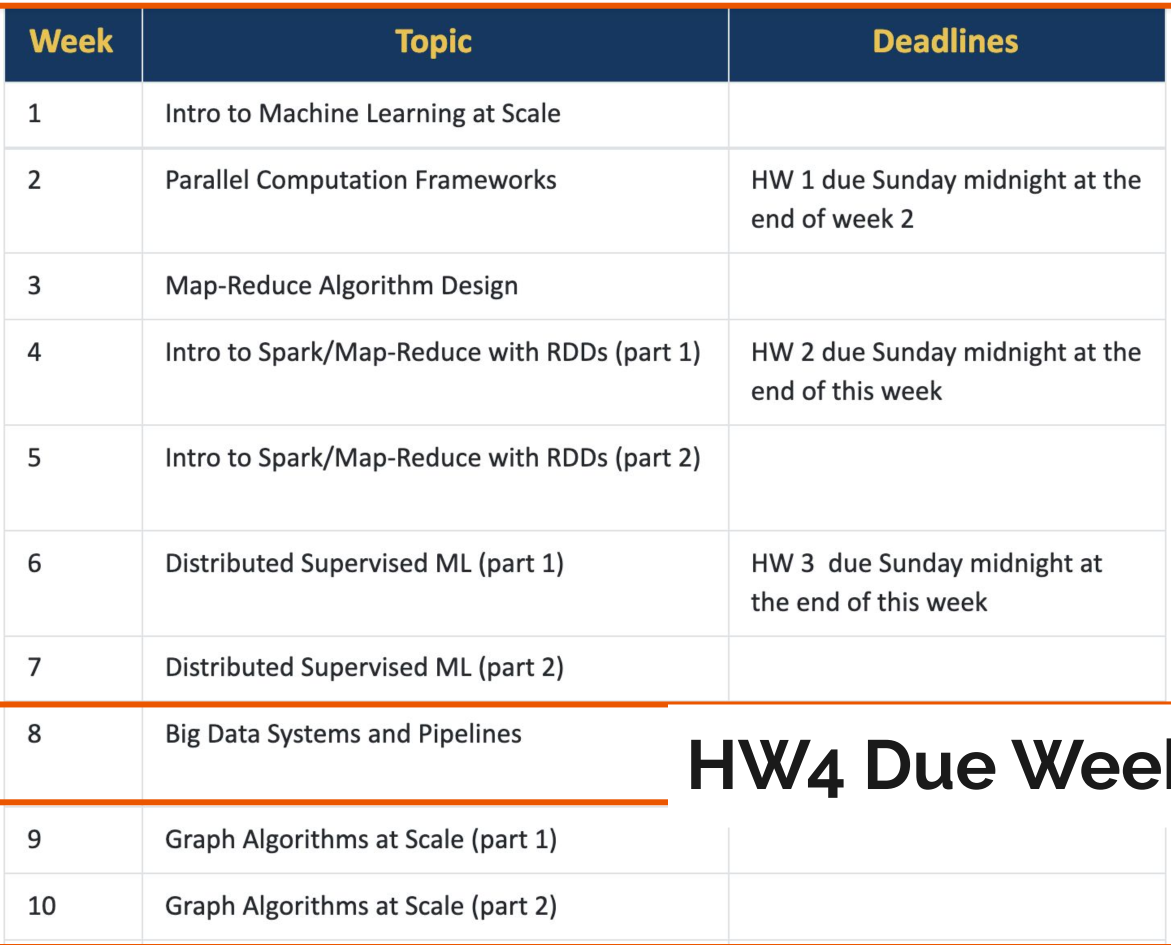
W261 - Week 7
Distributed Supervised Machine Learning: Part 2
Regularized Linear models

UC Berkeley - MIDS
Summer 2023

Table of Contents

HW4 release and Due date



Week	Topic	Deadlines
1	Intro to Machine Learning at Scale	
2	Parallel Computation Frameworks	HW 1 due Sunday midnight at the end of week 2
3	Map-Reduce Algorithm Design	
4	Intro to Spark/Map-Reduce with RDDs (part 1)	HW 2 due Sunday midnight at the end of this week
5	Intro to Spark/Map-Reduce with RDDs (part 2)	
6	Distributed Supervised ML (part 1)	HW 3 due Sunday midnight at the end of this week
7	Distributed Supervised ML (part 2)	
8	Big Data Systems and Pipelines	
9	Graph Algorithms at Scale (part 1)	
10	Graph Algorithms at Scale (part 2)	

HW4 Due Week 8

MidTerm Course Evaluation eek)

Please fill out the course evals over the next 24 sible) here. These are important for the course and for the program. Thank you!!

Thank you!

https://course- s.berkeley.edu/berkeley/

Take 10 minutes to complete now (due this week)

Customer Satisfaction Survey

Customer Satisfaction Survey

[Title of Project]

Goals: Explain what you want to get out of conducting this customer survey.

Timeline: What is your timeline for completion?

Team: List out the team members involved with this project.

Directions:

Hi Team,

Let's collaborate on our [project name] customer satisfaction survey. We need to finalize our list of survey questions, create a survey form (we can use either Google Docs or Typeform). Let's track the results via Google Spreadsheet.

Your Team Leader

Step 1: Finalize Survey Questionnaire

1) How likely are you to recommend this company to a friend or colleague? (NPS)

- Scale 0-10 (not likely to extremely likely)

2) How satisfied or dissatisfied are you with our company?

Week 7

7: Distributed Supervised Machine Learning: Part II

1h 0m Total Video Time

Course Content	Description
7.1 Weekly Introduction 7	Sequence | 1m 6s | View Sequence Outline
7.2 Loss Functions and General Framework for Gradient Descent	Interactive Video | 16m 8s
7.3 Quiz 7-1	Sequence | View Sequence Outline
7.4 Logistic Regression at Scale	Interactive Video | 17m 23s
7.5 SVMs	Interactive Video | 8m 34s
7.6 Quiz 7-2	Sequence | View Sequence Outline
7.7 SGD-Based SVMs at Scale	Interactive Video | 16m 22s

Regularized Linear Regression

3.5. LASSO Regression update via GD

Lasso Regression cost function

$$\text{LASSO\_MSE}(\theta) = \text{MSE}(\theta) + \eta \sum_{i=1}^{n} |\theta_i|$$

$$\begin{aligned} MSE(\boldsymbol{\theta}) &= \frac{1}{m} \|\mathrm{X} \cdot \boldsymbol{\theta} - \mathbf{y}\|_2^2 \text{ (i.e., the euclidean distance }^2) \\ &= \frac{1}{m} [(\mathrm{X} \cdot \boldsymbol{\theta} - \mathbf{y}) \cdot (\mathrm{X} \cdot \boldsymbol{\theta} - \mathbf{y})] \end{aligned}$$

Lasso Regression subgradient vector

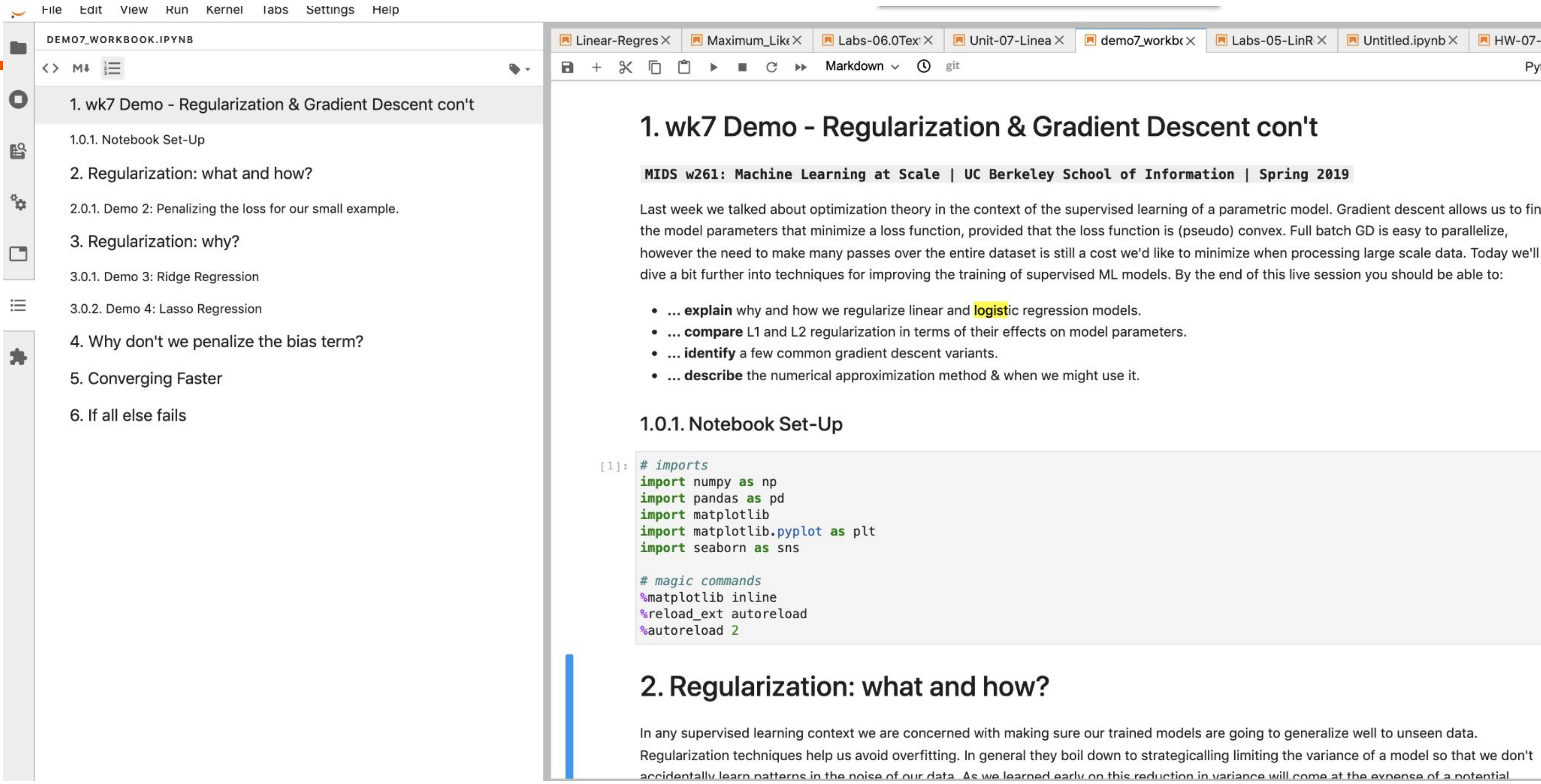
$$\nabla_{\text{LASSO\_MSE}}(\theta) = \nabla_{\text{MSE}}(\theta) + \eta \begin{pmatrix} 0 \\ \text{sign}(\theta_1) \\ \text{sign}(\theta_2) \\ \vdots \\ \text{sign}(\theta_n) \end{pmatrix} \quad \text{where } \text{sign}(\theta_i) = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{cases}$$

The gradient for Lasso Regression results in a vector from adding two vectors:

$$\begin{aligned} \nabla_{\text{LASSO\_MSE}}(\theta) &= \nabla_{MSE} & &+\eta * \nabla_{L1} \\ &= \begin{bmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{bmatrix} & &+\eta * \begin{bmatrix} 0 \\ \text{sign}(\theta_1) \\ \vdots \\ \text{sign}(\theta_n) \end{bmatrix} \\ &= \begin{bmatrix} \frac{2}{m} \mathbf{X_0}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y}) \\ \frac{2}{m} \mathbf{X_1}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y}) \\ \vdots \\ \frac{2}{m} \mathbf{X_n}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y}) \end{bmatrix} & &+\eta * \begin{bmatrix} 0 \\ \text{sign}(\theta_1) \\ \vdots \\ \text{sign}(\theta_n) \end{bmatrix} \\ &= \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y}) & &+\eta * \nabla_{L1} \end{aligned}$$

Gradient Descent step

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\text{LASSO\_MSE}}(\theta)$$



1. wk7 Demo - Regularization & Gradient Descent con't

`MIDS w261: Machine Learning at Scale | UC Berkeley School of Information | Spring 2019`

Last week we talked about optimization theory in the context of the supervised learning of a parametric model. Gradient descent allows us to fin
the model parameters that minimize a loss function, provided that the loss function is (pseudo) convex. Full batch GD is easy to parallelize,
however the need to make many passes over the entire dataset is still a cost we'd like to minimize when processing large scale data. Today we'll
dive a bit further into techniques for improving the training of supervised ML models. By the end of this live session you should be able to:

- **... explain** why and how we regularize linear and logistic regression models.
- **... compare** L1 and L2 regularization in terms of their effects on model parameters.
- **... identify** a few common gradient descent variants.
- **... describe** the numerical approximization method & when we might use it.

1.0.1. Notebook Set-Up

```
[1]: # imports
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

# magic commands
%matplotlib inline
%reload_ext autoreload
%autoreload 2
```

2. Regularization: what and how?

In any supervised learning context we are concerned with making sure our trained models are going to generalize well to unseen data.
Regularization techniques help us avoid overfitting. In general they boil down to strategicalling limiting the variance of a model so that we don't
accidentally learn patterns in the noise of our data. As we learned early on this reduction in variance will come at the expense of a potential

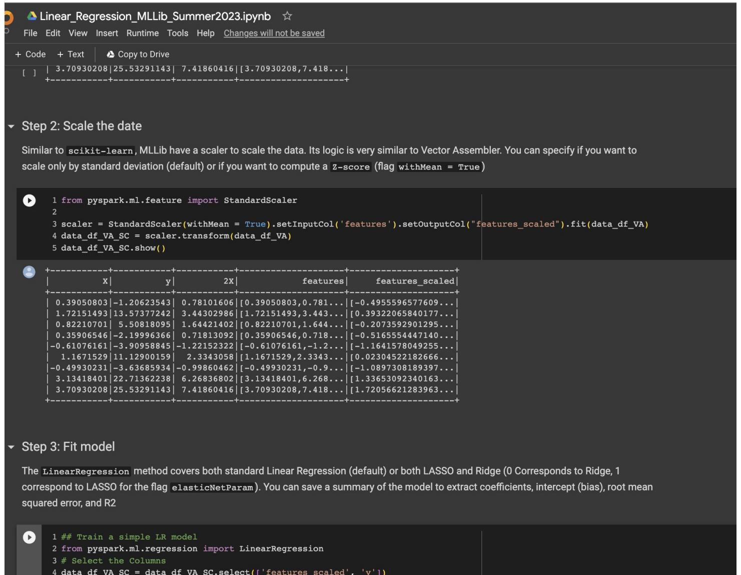
Important links for live session

The slides for module 7 are located here:

https://docs.google.com/presentation/d/1dbjllRxGnb7pcEx-HGT4igZxlidSvOV5jsnesSbIUKw/edit?usp=sharing

And a short supplemental Colab notebook on Dataframes, mllib/ml is located here:

https://colab.research.google.com/drive/1z-eHrD7Wu03DiX5hNros6lOinDgc3RkD?usp=sharing


Linear_Regression_MLLib_Summer2023.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
+ Code + Text
Copy to Drive

```
| 3.70930208|25.53291143| 7.41860416|[3.70930208,7.418...|
+-----------+-----------+-----------+--------------------+
```

Step 2: Scale the date

Similar to `scikit-learn`, MLLib have a scaler to scale the data. Its logic is very similar to Vector Assembler. You can specify if you want to scale only by standard deviation (default) or if you want to compute a `Z-score` (flag `withMean = True`)

```
from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(withMean = True).setInputCol('features').setOutputCol("features_scaled").fit(data_df_VA)
data_df_VA_SC = scaler.transform(data_df_VA)
data_df_VA_SC.show()
```

```
+-----------+-----------+-----------+--------------------+--------------------+
|          X|          y|         2X|            features|     features_scaled|
+-----------+-----------+-----------+--------------------+--------------------+
| 0.39050803|-1.20623543| 0.78101606|[0.39050803,0.781...|[-0.4955596577609...|
| 1.72151493|13.57377242| 3.44302986|[1.72151493,3.443...|[0.39322065840177...|
| 0.82210701| 5.50818095| 1.64421402|[0.82210701,1.644...|[-0.2073592901295...|
| 0.35906546|-2.19996366| 0.71813092|[0.35906546,0.718...|[-0.5165554447140...|
|-0.61076161|-3.90958845|-1.22152322|[-0.61076161,-1.2...|[-1.1641578049255...|
|  1.1671529|11.12900159|  2.3343058|[1.1671529,2.3343...|[0.02304522182666...|
|-0.49930231|-3.63685934|-0.99860462|[-0.49930231,-0.9...|[-1.0897308189397...|
| 3.13418401|22.71362238| 6.26836802|[3.13418401,6.268...|[1.33653092340163...|
| 3.70930208|25.53291143| 7.41860416|[3.70930208,7.418...|[1.72056621283963...|
+-----------+-----------+-----------+--------------------+--------------------+
```

Step 3: Fit model

The `LinearRegression` method covers both standard Linear Regression (default) or both LASSO and Ridge (0 Corresponds to Ridge, 1 correspond to LASSO for the flag `elasticNetParam`). You can save a summary of the model to extract coefficients, intercept (bias), root mean squared error, and R2

```
## Train a simple LR model
from pyspark.ml.regression import LinearRegression
# Select the Columns
data_df_VA_SC = data_df_VA_SC.select(['features_scaled', 'y'])
```

Linear Classifiers and Maximum Likelihood

9.1. Deriving Maximum-Likelihood Estimates

Consider a training data set of the form $(X^{(1)}, y^{(1)}, \ldots, (X^{(m)}, y^{(m)})$, where each $X^{(i)}$ is a an n-dimensional vector, and $y^{(i)}$ is the true label (aka ground truth label) of the corresponding example. Given a (learnt) logistic regression model one can calculate a probability for the ground truth class of each training example (as was defined above and presented here in context for convenience) as follows:

$$p(Y = y^{(i)} \mid x_1^{(i)}, \ldots, x_n^{(i)})$$

Let's assume the training data are independently sampled from the same distribution; this is known as the *iid assumption*, which stands for “independent and identically distributed”. Given an IID dataset, we can estimate the joint probability of the dataset as a function of the parameters of the learnt model as follows using the class probability associated with the ground truth class of each training example:

$$P(D|\theta) = \prod_{i=1}^{m} p(Y = y^{(i)} \mid x_1^{(i)}, \ldots, x_n^{(i)})$$

We refer to this joint probability distribution, $P(D|\theta)$, as the **(conditional) likelihood** $L(\theta)$. Here for convenience we use θ to refer to the vector (for the binary classifier and a matrix for an n-ary classifier) of model coefficients, and bias term for all parameters $i \in 1, \ldots, n.i \in 1, \ldots, n$. For a multinomial classifier, θ will be an $n \times K$ matrix for features $i \in 1, \ldots, n$ and classes $j \in [1, \ldots, K]$. *The likelihood is a function of the parameter values, and the training examples.* We usually work with the **log likelihood** (because of numerical underflow issues), which is given by:

$$LL(\theta) \triangleq \log P(D|\theta) = \sum_{i=1}^{m} log P(y^{(i)}|x^{(i)})$$

Here $\triangleq$ denotes *"is defined by"*. This decomposes into a sum of terms, one per example. Generally, we wish to maximize this quantity, i.e., for each example, we would desire for $p(Y = y^{(i)} \mid x_1^{(i)}, \ldots, x_n^{(i)})$ to be maximum (or as close to 1 as possible for the ground truth class). This then leads to Maximum Likelihood Estimation (MLE) objective function that is given by:

$$\hat{\theta}_{MLE} = \arg\max_{\theta;D} \sum_{i=1}^{m} log P(y^{(i)}|x^{(i)})$$

$$\hat{\theta}_{MLE} = \arg\max_{\theta;D} \sum_{i=1}^{m} logP(y^{(i)}|x^{(i)})$$

Since most optimization algorithms and software implementations are designed to minimize objective functions, the MLE objective function is flipped to a minimization problem, commonly known as the (conditional) negative log likelihood or NLL as follows:

$$NLL(\theta; D) \triangleq -\log p(D|\theta) = -\sum_{i=1}^{m} logP(y^{(i)}|x^{(i)})$$

Minimizing this will also give the maximum likelihood estimate (MLE) of the model parameters.

$$\hat{\theta}_{MLE} = \arg\min_{\theta;D} -\sum_{i=1}^{m} logP(y^{(i)}|x^{(i)}) \tag{1}$$

In Equation (1), Vector θ, in the case of Naive Bayes, could refer to the model parameters $Pr(C_j)$ and $Pr(x_{ij|C=y_j})$ for $j \in [1, \ldots, K]$ and $i \in 1, \ldots, n$. In the case of logistic regression, θ would correspond to the the vector of coefficients and bias term for a binary classification proble,.

As an aside, if the model is unconditional (unsupervised, as in, say, the case of the EM clustering algorithm), the MLE becomes:

$$\hat{\theta}_{MLE} = \arg\min_{\theta;D} -\sum_{i=1}^{m} logP(x^{(i)}, \theta) \tag{2}$$

In summary, maximum likelihood estimation (MLE) is a method of estimating the parameters of a model (of probability distributions in the case of Naive Bayes machine learnt model) by maximizing a likelihood function, so that under the assumed statistical model the observed data is most probable (i.e., the joint probability of the data is highest given by the likelihood function). The model, a point in the parameter space, that maximizes the likelihood function is called the maximum likelihood estimate. The logic of maximum likelihood is both intuitive and flexible, and as such the method has become a dominant means of statistical inference (think machine learning).

Most machine learning algorithms can be reframed probabilistically to leverage the maximum likelihood estimation framework. If the likelihood function is differentiable, the derivative test for determining minima (or maxima) can be applied. In some cases, the

Recall that in linear regression, the goal is to make predictions of the form

$$\hat{y} = \boldsymbol{x}^T \boldsymbol{w} + b,$$

where $\hat{y}, b \in \mathbb{R}$ and $\boldsymbol{w}, \boldsymbol{x} \in \mathbb{R}^n$. In contrast, in the case of classification, the interest is in asking the question "*what is the probability that example* $\boldsymbol{x}$ *belongs to the positive class?*" A regular linear model is a poor choice here because it can output values greater than 1 or less than 0. To coerce reasonable answers from our model (in the $[0, 1]$ interval), the linear regression model can be modified slightly, by running the linear function through a sigmoid activation function σ:

$$\hat{y} = \sigma(\boldsymbol{x}^T \boldsymbol{w} + b).$$

The sigmoid function σ, sometimes called a squashing function or a *logistic* function - thus the name logistic regression - maps a real-valued input to the range 0 to 1. Indeed, the logistic function $\sigma(z)$ is a good choice since it has the form of a probability, i.e. $\sigma(-z) = 1 - \sigma(z)$ and $\sigma(z) \in (0, 1)$ as $z \to \pm\infty$. Here z can be viewed as the perpendicular distance from a data point to a class hyperplane in the binary case. If target labels are set $y \in (0, 1)$ this will result in the following:

$$\mathbb{P}(y = 1|z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$
$$\mathbb{P}(y = 0|z) = 1 - \sigma(z) = \frac{1}{1 + e^{z}}$$

which can be written more compactly as $\mathbb{P}(y|z) = \sigma(z)^y(1 - \sigma(z))^{1-y}$. The logistic function can be visualized as follows:

liner 1/2

$$\begin{aligned} L(\mathbf{w}) &= P(\mathbf{y} \mid \mathbf{x}; \mathbf{w}) \\ &= \prod_{i=1}^{n} P\left(y^{(i)} \mid x^{(i)}; \mathbf{w}\right) \\ &= \prod_{i=1}^{n} \left(\sigma\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}} \end{aligned}$$

In practice, it is easier to maximize the (natural) log of this equation, which is called the log-likelihood function:

$$\begin{aligned} l(\mathbf{w}) &= \log L(\mathbf{w}) \\ &= \sum_{i=1}^{n} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right)\right] \end{aligned}$$

Negative Log-Likelihood Loss

In practice, it is even more convenient to minimize negative log-likelihood instead of maximizing log-likelihood:

$$
\begin{aligned}
\mathcal{L}(\mathbf{w}) &= -l(\mathbf{w}) \\
&= -\sum_{i=1}^{n} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right) \right]
\end{aligned}
$$

(in code, we also usually add a $1/n$ scaling factor for further convenience, where n is the number of training examples or number of examples in a minibatch)

MSE)), as a measure the model's performance.

Since now linear classification is focussed on outputing class probabilities, **one natural objective is to choose the weights (or parameters θ) that give the actual labels in the training data highest probability**. For n samples $\{x_i, y_i\}$ the objective would boil down to maximizing the joint probability of the all the training data labels given their inputs:

$$\operatorname*{argmax}_{\theta} \mathbb{P}_{\theta}\left(y_1, \ldots, y_n \middle| \boldsymbol{x}_1, \ldots \boldsymbol{x}_n\right)$$

Because each example is independent of the others, and each label depends only on the features of the corresponding examples, the above can be simplified/decomposed and rewritten as follows:

$$\operatorname*{argmax}_{\theta} \prod_{i}^{n} \mathbb{P}_{\theta}(y_i \,|\boldsymbol{x}_i) = \max_{\theta} \mathbb{P}_{\theta}\left(y_1 \,|\boldsymbol{x}_1\right)\mathbb{P}_{\theta}\left(y_2 \,|\boldsymbol{x}_2\right) \cdots \mathbb{P}_{\theta}\left(y_n \,|\boldsymbol{x}_n\right)$$

where y_i corresponds to the class of the example, either positive or negative, $\hat{p}_1$ or $\hat{p}_0$.

This function is a product over the training data. Since these predicted probabilies are less than or equal to one the product of many of such small numbers will get extremely small, so small that the numerical precision of modern day computers will run into numerical precision problems. To avoid such complications it is a lot easier to work with a loss function that, instead of working with the product of small numbers, works as a sum. This is accomplished by taking the log of the above equation. This results in the following:

$$\operatorname*{argmax}_{\theta} \log\left(\prod_{i}^{n} \mathbb{P}(y_i \,|\boldsymbol{x}_i)\right) = \sum_{i}^{m} \log\left(\mathbb{P}(y_i \,|\boldsymbol{x}_i)\right) = \log\left(\mathbb{P}(y_1 \,|\boldsymbol{x}_1)\right) + \cdots + \log\left(\mathbb{P}(y_n \,|\boldsymbol{x}_n)\right)$$

The log transformation is monotonic and therefore does not change the dynamics/location of the solution to a maximization problem.

4.2. Learning a binomial logistic regression model via gradient descent using BXE (Binomial cross entropy)

The objective function for the learning a binomial logistic regression model (log loss) can be stated as follows:

$$\underset{\theta}{\operatorname{argmin}}\,[\mathrm{BXE}] = \underset{\theta}{\operatorname{argmin}} \left[-\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} log\left(\hat{p}^{(i)}\right) + (1 - y^{(i)}) log\left(1 - \hat{p}^{(i)}\right) \right] \right]$$

- where probability of class 1: | $\hat{p} = \mathbb{P}(y = 1|X) = \sigma(X \cdot w + b) = \hat{p} = \sigma(t) = \dfrac{1}{1 + \exp(-t)}$

The corresponding gradient function of partial derivatives is as follows (after a little bit of math):

$$\nabla_{\mathrm{CXE}}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{BXE}(\theta) \\ \frac{\partial}{\partial \theta_1} \mathrm{BXE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{BXE}(\theta) \end{pmatrix} = \frac{1}{m} \mathbf{X}^T \cdot (\hat{p} - \mathbf{y})$$

For completeness learning a binomial logistic regression model via gradient descent would use the following step iteratively:

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\mathrm{BXE}}(\theta)$$

3.6. The gradient for Ridge Logistic Regression results is a vector from adding two vectors:** ¶

Binary Cross Entropy (BXE) function (log loss)

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

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

$$\nabla_{\text{BXE}}(\theta) = \frac{1}{m}\mathbf{X}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y})$$

$$\nabla_{\text{RIDGE\_BXE}}(\theta) = \nabla_{BXE} \quad + \eta * \nabla_{L2}$$

$$= \begin{bmatrix} \frac{\partial}{\partial\theta_0}\text{BXE}(\theta) \\ \frac{\partial}{\partial\theta_1}\text{BXE}(\theta) \\ \vdots \\ \frac{\partial}{\partial\theta_n}\text{BXE}(\theta) \end{bmatrix} + \eta * \begin{bmatrix} 0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{m}\mathbf{X_0}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y}) \\ \frac{1}{m}\mathbf{X_1}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y}) \\ \vdots \\ \frac{1}{m}\mathbf{X_n}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y}) \end{bmatrix} + \eta * \begin{bmatrix} 0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad \text{where } \mathbf{X_i} \text{ denotes column } i \text{ of } \mathbf{X}$$

$$= \frac{1}{m}\mathbf{X}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y}) \quad + \eta * \nabla_{L2}$$

For more details on deriving the Gradient Equation for logistic regression see:

- Dan Jurafsky's Chapter on Logistic Regression is located here
- Or locally here.

Ridge Logistic Regression Gradient Descent step

$$\theta^{(\text{next step})} = \theta - \eta\nabla_{\text{RIDGE\_BXE}}(\theta)$$

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

$$\frac{\partial}{\partial\theta_j}\text{BXE}(\theta) = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}\left(\sigma(\mathbf{x}^{(\mathbf{i})}\,\theta) - y^{(i)}\right)$$

$$\nabla_{\text{BXE}}(\theta) = \frac{1}{m}\mathbf{X}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y})$$

3.7. The gradient for Lasso Logistic Regression results i a vector from adding two vectors:**

$$\begin{aligned}
\nabla_{\text{LASSO\_BXE}}(\theta) &= \nabla_{BXE} && +\eta * \nabla_{L1} \\
&= \begin{bmatrix} \frac{\partial}{\partial\theta_0}\text{BXE}(\theta) \\ \frac{\partial}{\partial\theta_1}\text{BXE}(\theta) \\ \vdots \\ \frac{\partial}{\partial\theta_n}\text{BXE}(\theta) \end{bmatrix} && +\eta * \begin{bmatrix} 0 \\ \text{sign}(\theta_1) \\ \vdots \\ \text{sign}(\theta_n) \end{bmatrix} \\
&= \begin{bmatrix} \frac{1}{m}\mathbf{X_0}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y}) \\ \frac{1}{m}\mathbf{X_1}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y}) \\ \vdots \\ \frac{1}{m}\mathbf{X_n}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y}) \end{bmatrix} && +\eta * \begin{bmatrix} 0 \\ \text{sign}(\theta_1) \\ \vdots \\ \text{sign}(\theta_n) \end{bmatrix} \quad \text{where } \mathbf{X_i} \text{ denotes column } i \text{ of } \mathbf{X} \\
&= \frac{2}{m}\mathbf{X}^T \cdot (\sigma(\mathbf{X}\cdot\theta) - \mathbf{y}) && +\eta * \nabla_{L1}
\end{aligned}$$

For more details on deriving the Gradient Equation for logistic regression see:

- Dan Jurafsky's Chapter on Logistic Regression is located here
- Or locally here.

Gradient Descent step

$$\theta^{(\text{next step})} = \theta - \eta\nabla_{\text{LASSO\_BXE}}(\theta)$$