LOGISTIC REGRESSION AND KERNEL LOGISTIC REGRESSION

A comparative study of logistic regression and kernel logistic regression for binary classification

Ezukwoke K.I[1], Zareian S.J[2]

[1, 2] Department of Computer Science
Machine Learning and Data Mining
{ifeanyi.ezukwoke, samaneh.zareian.jahromi}@etu.univ-st-etienne.fr
University Jean Monnet, Saint-Etienne, France

Abstract

Logistic regression is a linear binary classification algorithm frequently used for classification problems. In this paper we present its kernel version which is used for classification of non-linearly separable problems. We briefly introduce the concept of multiple kernel learning and apply it to kernel logistic regression. We elaborate the performance differences between classical, kernel logistic regression and its stochastic variant (both classical and kernel logistic regression).

Keywords
Classification, logistic regression, kernel logistic regression, multi-kernel learning.

1 INTRODUCTION

Linear regression is a statistical method used for univariate and multivariate analysis. Given a set of observations $\{x_i, y_i\}^n$, where $\{x_i\}^N$ is the independent variables (*feature space*) and y_i is the dependent (*response variable*- usually continuous or discrete). Linear regression models estimate parameter β that best maps the predictors to the response variable y_i.

$$y = X_1\beta_1 + X_2\beta_2 + \cdots + X_N\beta_N \tag{1}$$

Using Ordinary Least Squares (**OLS**) we can estimate the unknown parameters in the linear regression problem [1]. It does this by minimizing the sum of square differences between the predictors and the response variable. [1]

$$\min_{\beta} \|X\beta - y\|_2 \tag{2}$$

We minimize this objective function using maximum likelihood estimation and derive the following closed form solution.

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y \tag{3}$$

This returns a model that produces as straight line that maps the predictors to the response [1].

However, linear regression is only sufficient for explaining the relationship in observations with continuous variable. For observations with categorical variables it becomes impossible to adopt this model.

Logistic regression solves the limitation of linear regression for categorical variable using maximum likelihood estimation of probability log function. This idea is further

[1]source code for project is available on github

explained in the next sections. Our focus however is on its kernel version and how we explore the inner product of the independent variable to classify non-seperable data.

2 CLASSIFICATION

Classification is a supervised machine learning approach to categorize data into **distinct number of classes** where we can assign label to each class. Given a set of data $\{x^{(i)}, y^{(i)}\}$ where x is the feature space in $m \times (n+1)$ dimension and y is the classfication output such $y \in \{0, 1\}$ for binary output or $\{1, 2, ...n\}$ for multiclass output. Classification algorithms are most used for Spam detection, Voice and image recognition, sentiment analysis, fraud detection and many more.

Logistic regression is a linear binary classification algorithm that maps a set of predictors to their corresponding categorical response variables. The algorithm is capable of classifying linearly separable datasets. However, linear logistic regression is not able to accurately classify non-linear data, therefore we use kernel logistic regression for non-linearly separable data classification.

Kernel logistic regression is similar to support vector machines in its operational output [4]. Existing papers already implement kernel logistic regression using the Newton-Ralphson method [3], Sequential Minimal Optimization (**SMO**) [4] and Truncated Newton-method [5].

In this paper however we solve logistic regression and kernel-logistic regression using gradient descent (**GD**) and stochastic gradient descent (**SGD**) optimization techniques.

2.1 LOGISTIC REGRESSION

Logistic regression is a discriminative model since it focuses only on the posterior probability of each class $Pr(Y|x;\beta)$. It is also a generalized linear model, mapping output of linear multiple regression to posterior probability of each class $Pr(Y|x;\beta) \in \{0, 1\}$ [2]. The probabilty of a data-sample belonging to class 1 is given by:

$$Pr(Y = 1|X = x; \beta) = \sigma(z), \, where \; z = \beta^T x \tag{4}$$

$$P(Y = 1|X = x; \beta) = \sigma(\beta^T x) \tag{5}$$

where

$$Pr(Y = 1|X = x; \beta) + Pr(Y = 0|X = x; \beta) = 1 \tag{6}$$

$$Pr(Y = 0|X = x; \beta) = 1 - Pr(Y = 1|X = x; \beta) \tag{7}$$

Hence, the probability of a data-sample belonging to class 0 is given by:

$$Pr(Y = 0|X = x; \beta) = 1 - \sigma(z) \tag{8}$$

$\sigma(z)$ is called the **logistic sigmoid function** and is given by

$$\sigma(z) = \frac{1}{1 + \exp^{-z}} \tag{9}$$

The uniqueness of this function is that it maps all real numbers $\mathbb{R}$ to range { 0, 1}.

Again, we know

$$\begin{aligned} &log(odds(Pr(Y = 1|X = x; \beta))) \\ &\quad = \frac{Pr(Y = 1|X = x; \beta)}{Pr(Y = 0|X = x; \beta)} \\ &\quad = \frac{Pr(Y = 1|X = x; \beta)}{1 - Pr(Y = 1|X = x; \beta)} \end{aligned}$$

Assuming $P(Y = 1|X = x; \beta) = p(x)$, the next most obvious idea is to let $log p(x)$ be a linear function of x, so that changing an input variable multiplies the probability by a fixed amount. This is done by taking a log transformation of $p(x)$.

Formally, $logit(p(x)) = \beta_0 + \beta^T x$ making

$$logit(p(x)) = log\left(\frac{p(x)}{1 - p(x)}\right) = \beta_0 + \beta^T x \tag{10}$$

Simplifying for $p(x)$ and $1 - p(x)$ we have

$$\frac{p(x)}{1 - p(x)} = \exp(\beta_0 + \beta^T x) \tag{11}$$

$$p(x) = (1 - p(x)) \exp(\beta_0 + \beta^T x) \tag{12}$$

$$p(x) = \exp(\beta_0 + \beta^T x) - p(x) \cdot \exp(\beta_0 + \beta^T x) \tag{13}$$

$$p(x) + p(x) \cdot \exp(\beta_0 + \beta^T x) = \exp(\beta_0 + \beta^T x) \tag{14}$$

$$p(x)(1 + \exp(\beta_0 + \beta^T x)) = \exp(\beta_0 + \beta^T x) \tag{15}$$

$$\begin{aligned} p(x) &= \frac{\exp(\beta_0 + \beta^T x)}{1 + \exp(\beta_0 + \beta^T x)} \\ &= \frac{\frac{1}{exp(\beta_0 + \beta^T x)} \cdot \exp(\beta_0 + \beta^T x)}{\frac{1}{exp(\beta_0 + \beta^T x)} + \frac{exp(\beta_0 + \beta^T x)}{exp(\beta_0 + \beta^T x)}} \\ &= \frac{1}{1 + \exp -(\beta_0 + \beta^T x)} \end{aligned}$$

$$1 - p(x) = \frac{1}{1 + \exp(\beta_0 + \beta^T x)} \tag{16}$$

2.2 Learning Logistic regression

We assume that $P(Y = 1|X = x; \beta) = P(x; \beta)$, for some probability function $P(x; \beta)$ parameterized by β the conditional likelihood function is given by Bernoulli sequence:

$$\Pi_{i=1}^n Pr(Y = y_i | X = x_i; \beta) = \Pi_{i=1}^n p(x_i; \beta)^y (1 - p(x_i; \beta)^{(1-y_i)})$$

The probability of a class is p, if $y_i = 1$, or $1 - p$, if $y_i = 0$. The likelihood is then

$$L(\beta_0, \beta) = \Pi_{i=1}^n p(x_i)^y (1 - p(x_i)^{(1-y_i)}) \tag{17}$$

taking the log of this likelhood we have

$$\begin{aligned} l(\beta_0, \beta) &= \sum_{i=1}^n y_i log p(x_i) + (1 - y_i) log(1 - p(x_i)) \\ &= \sum_{i=1}^n y_i log p(x_i) - y_i log(1 - p(x_i)) + log(1 - p(x_i)) \\ &= \sum_{i=1}^n log(1 - p(x_i)) + y_i log \left(\frac{p(x_i)}{1 - p(x_i)} \right) \end{aligned} \tag{18}$$

we replace $log\left(\frac{p(x)}{1-p(x)}\right)$ with $\beta_0 + x \cdot \beta$ as seen in equation (8) and $(1 - p(x))$ with $\frac{1}{1+\exp(\beta_0 + x \cdot \beta)}$. Hence,

$$\begin{aligned} l(\beta_0, \beta) &= \sum_{i=i}^n log \left(\frac{1}{1 + \exp(\beta_0 + x \cdot \beta)} \right) + y(\beta_0 + x \cdot \beta) \\ &= \sum_{i=i}^n -log(1 + \exp(\beta_0 + x \cdot \beta)) + y(\beta_0 + x \cdot \beta) \end{aligned} \tag{19}$$

$$\nabla l(\beta_0, \beta) = -\sum_{i=1}^n \frac{1}{1 + \exp(\beta_0 + x \cdot \beta)} x_{ij} \exp(\beta_0 + x \cdot \beta) + \sum_{i=1}^n y_i x_i$$

$$\nabla l_\beta = \sum_{i=1}^n (y_i - p(x_i; \beta_0, \beta)) x_{ij} \tag{20}$$

Since this is a transcendental equation with no closed-form solution, we apply the gradient descent optimization algorithm. We take first the first order derivative of the objective function $\nabla_\beta l = \sum_{i=1}^n (y_i - p(x_i; \beta_0, \beta)) x_{ij}$

Algorithm 1: Logistic regression via Gradient Descent **GD**

```
   Input  : $x \in \mathcal{X}$ where $y \in \mathcal{Y}$
   Output : $\beta$
1  begin
2      $\beta_j \leftarrow \beta^0$;
3      while not converged do
4          $\beta_{j+1} = \beta_j - \alpha \nabla_\beta l$;
5      end
6  end
```

We also solve it using stochastic approach

Algorithm 2: Logistic regression via Stochastic Gradient Descent **SGD**

```
   Input  : $x \in \mathcal{X}$ where $y \in \mathcal{Y}$
   Output : $\beta$
1  begin
2      $\beta_j \leftarrow \beta^0$;
3      while not converged do
4          for $i \in$ randshuffle($\{1, \dots, N\}$) do
5              for $k \in \{1, \dots, i\}$ do
6                  $\beta_{j+1} = \beta_j - \alpha \nabla_{beta} l_k$;
7              end
8          end
9      end
10 end
```

2.3 KERNEL LOGISTIC REGRESSION

Classical logistic regression will fail to classify accurately non-linearly separable data, therefore, we prefer to use its kernel version. It also has a direct probabilistic interpretation that makes it suited for Bayesian design [4].

The vector space can be expressed as a linear combination of the input vectors such that

$$\beta = \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x_i}) \tag{21}$$

where $\alpha \in \mathbb{R}^{n \times 1}$ is the dual variable. The function $\phi(x_i)$ maps the data points from lower dimension to higher dimension.

$$\phi : \mathbf{x} \in \mathbb{R}^{\mathbb{D}} \rightarrow \phi(x) \in \mathbb{F} \subset \mathbb{R}^{\mathbb{D}'} \tag{22}$$

Let $\kappa(x_i, x)$ be a kernel function resulting from the inner product of $\phi(x_i)$ and $\phi(\mathbf{x_j})$, such that

$$\kappa(x_i, x) = \left\langle \phi(x_i)\phi(x_j) \right\rangle \tag{23}$$

From **representer theorem** we know that

$$\begin{aligned} F = \beta^T \phi(x) &= \alpha \left\langle \phi(\mathbf{x_i})\phi(\mathbf{x_j}) \right\rangle \\ &= \alpha \kappa(x_i, x_j) \end{aligned}$$

We can now express $p(x; \beta)$ is subspace of input vectors only such that

$$p(\phi; \alpha) = \frac{1}{1 + e^{-\alpha_i \kappa(x_i, x_j)}} \tag{24}$$

and

$$1 - p(\phi; \alpha) = \frac{1}{1 + e^{\alpha_i \kappa(x_i, x_j)}} \tag{25}$$

The logit function is mapped into the kernel space as

$$logit(\frac{p(\phi; \alpha)}{1 - p(\phi; \alpha)}) = \alpha \kappa(x_i, x) \tag{26}$$

Deriving the equation of kernel logistic regression requires the regularized logistic regression, precisely the $l2 - norm$ of the log-likelihood. This is in comparison to the SVM objective function used in [3].

$$\begin{aligned} L_\alpha = \sum_{i=1}^{n} y_i log p(x_i) + (1 - y_i) log(1 - p(x_i)) \\ - \frac{\lambda}{2} \alpha^{\mathbf{T}} \kappa(\mathbf{x_i}, \mathbf{x}) \alpha \end{aligned}$$

2.4 Learning kernel logistic regression

As mentioned earlier, some of the methods for finding the maximum likelihood estimate include gradient descent (**GD**) and iterative re-weighted least sqaures (**IRLS**) method. Here we employ the use of IRLS which is based on the Newton-Ralphson algorithm.

2.4.1 Optimization problem

$$\begin{aligned} L_\alpha = \sum_{i=1}^{n} y_i log p(x_i) + (1 - y_i) log(1- \\ p(x_i)) - \frac{\lambda}{2} \alpha^{\mathbf{T}} \kappa(\mathbf{x_i}, \mathbf{x}) \alpha \end{aligned}$$

We can expand the objective function as follows

$$\begin{aligned} L_\alpha &= ylog\left(\frac{p}{1-p}\right) + log(1-p(x_i)) \\ &\quad -\frac{\lambda}{2}\alpha^{\mathbf{T}}\kappa(\mathbf{x_i},\mathbf{x})\alpha \\ &= ylog\left(\frac{p}{1-p}\right) + log\left(\frac{1}{1+e^{\alpha\kappa(x_i,x)}}\right) \\ &\quad -\frac{\lambda}{2}\alpha^{\mathbf{T}}\kappa(\mathbf{x_i},\mathbf{x})\alpha \\ &= y\alpha\kappa(x_i,x) - log(1+e^{\alpha\kappa(x_i,x)}) \\ &\quad -\frac{\lambda}{2}\alpha^{\mathbf{T}}\kappa(\mathbf{x_i},\mathbf{x})\alpha \end{aligned}$$

First order derivative of the log-likelihood

$$\begin{aligned} \nabla_\alpha L &= y\kappa(x_i,x) - \frac{\kappa(x_i,x)e^{\alpha\kappa(x_i,x)}}{1+e^{\alpha\kappa(x_i,x)}} \\ &\quad -\lambda\alpha\kappa(x_i,x) \\ &= y\kappa(x_i,x) - p\kappa(x_i,x) - \lambda\alpha\kappa(x_i,x) \\ \nabla_\alpha L &= \kappa(x_i,x)(y-p) - \lambda\alpha\kappa(x_i,x) \end{aligned}$$

We apply gradient descent and stochastic gradient descent algorithms

Algorithm 3: Kernel logistic regression using **Gradient descent**

```
  Input  : $\kappa, y, \alpha$
  Output : $\alpha$
1 begin
2     $\alpha_j \leftarrow \alpha^{0}$';
3     while not converged do
4         $\alpha_{j+1} = \alpha_j - lr\nabla_\alpha L$;
5     end
6 end
```

In algorithm 4 of Algorithm 3, lr is the learning rate.

Algorithm 4: Kernel logistic regression using **Stochastic Gradient descent**

```
   Input  : $\kappa, y, \alpha_j$
   Output : $\alpha$
1  begin
2      $\alpha_j \leftarrow \alpha^{0}$';
3      while not converged do
4          for
             $i \in$ randshuffle($\{1, \dots, N\}$)
           do
5              for $k \in \{1, \dots, i\}$ do
6                  $\alpha_{j+1} = \alpha_j - lr\nabla_\alpha L_k$;
7              end
8          end
9      end
10 end
```

2.4.2 Prediction

Still using the representer theorem, we compute the posterior probability of a new data point

$$y = sign\left(\frac{1}{1+\exp^{-\alpha\kappa(x_i,x)}}\right) \tag{27}$$

Here, the prediction is dependent only on α and the kernel.

2.5 Kernels

We introduce the commonly used kernels and a brief overview of Multiple kernels used. The radius R is computed from the

- **Linear kernel**

$$\kappa(x_i,x_j) = \mathbf{x}_i\mathbf{x}_j^T \tag{28}$$

- **Polynomial kernel**

$$\kappa(x_i,x_j) = (\mathbf{x}_i\mathbf{x}_j^T + c)^d \tag{29}$$

  where $c \geq 0$ and d is the degree of the polynomial usually greater than 2.

- **RBF(Radial Basis Function) kernel**

Sometimes referred to as the **Gaussian kernel**.

$$\kappa(x_i, x_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2) \quad (30)$$

where $\gamma = \frac{1}{2\sigma^2}$.

- **Sigmoid kernel**

$$\kappa(x_i, x_j) = \tanh(\gamma \mathbf{x}_i \mathbf{x}_j^T + c) \quad (31)$$

where $c \geq 0$ and $\gamma = \frac{1}{2\sigma^2}$.

- **Laplace kernel**

$$\kappa(x_i, x_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||) \quad (32)$$

where $\gamma = \frac{1}{2\sigma^2}$.

2.6 Multi-kernel

The reason behind the use of multiple kernel is similar to the notion of multiclassification, where cross-validation is used to select the best performing classifier [6]. By using multiple kernel, we hope to learn a different similar in the kernel space not easily observed when using single kernel.

We can prove from **Mercer's Theorem** that a kernel is **Positive Semi-Definite (PSD)** if $u^T\kappa(x_i, x_j)u \geq 0$. Hence by performing arithmetic or any mathematical operation on two or more kernel matrix, we obtain a new kernel capable of exploiting different property or similarities of training data.

Given a kernel κ, we prove that κ is PD if

$$\langle u, \kappa u\rangle \geq 0 \quad (33)$$

Proposition: *A symmetric function κ: $\chi \to \mathbb{R}$ is positive semi-definite if and only if* $\langle u, \kappa u\rangle \geq 0$
Proof:
Suppose that κ is a kernel which is the inner product of the mapping functions $\big\langle \phi(x_i)\phi(x_j)\big\rangle$. κ is a kernel if its inner product are positive and the solution of $\kappa u = \lambda u$ gives non-negative eigenvalues.

So that,

$$\langle u, \kappa u\rangle = \sum_{i=1}^{N} u_i \cdot (\kappa u_i) \quad (34)$$

$$= \sum_{i=1}^{N} u_i \sum_{j=1}^{N} \big\langle \phi(x_i)\phi(x_j)_{\mathcal{H}}\big\rangle\, u_j \quad (35)$$

Where $\mathcal{H}$ represent the Hilbert space we project the kernel [7].

$$= \left\langle \sum_{i=1}^{N}\sum_{j=1}^{N} u_i\phi(x_i), u_j\phi(x_j) \right\rangle_{\mathcal{H}} \quad (36)$$

$$\langle u, \kappa u\rangle = \left\|\sum_{i=1}^{N} u_i\phi(x_i)\right\|_{\mathcal{H}}^2 \geq 0 \quad (37)$$

Therefore κ is positive definite.

Using this property of the kernel κ we introduce multiple kernel combination as follow

- **LinearRBF**
  Here we combine two kernels, precisely **Linear and RBF kernel** using their inner product.

$$\hat{\mathbf{K}}_{\mathbf{linrbf}} = \kappa(x_i, x_j) \times \kappa(x_i, x_l) \quad (38)$$

- **RBFPoly**
  Here we combine **RBF and Polynomial kernel** using their inner product.

$$\hat{\mathbf{K}}_{\mathbf{rbfpoly}} = \kappa(x_i, x_j) \times \kappa(x_i, x_l) \quad (39)$$

- **EtaKernel**
  The **EtaKernel** is a composite combination of **LinearRBF, RBFPoly and RBFCosine** and it is given by

$$\hat{\mathbf{K}}_{\mathbf{etarbf}} = \hat{\mathbf{K}}_{\mathbf{linrbf}} \times \hat{\mathbf{K}}_{\mathbf{rbfpoly}} + \hat{\mathbf{K}}_{\mathbf{rbfpoly}} \times \hat{\mathbf{K}}_{\mathbf{rbfcosine}}$$

3 EXPERIMENT

3.1 Dataset

We perform a comparison between classical logistic regression and its kernel version using benchmark datasets including moons, blob, circle and classification. Given N, the total number of samples. Circle ($N = 1000$), Moon ($N = 1000$) and Classification ($N = 1000$), we split each data into training and test sample of $70\% - 30\%$ each.

3.1.1 Data description

Each data contains binary class (exactly 2 groups of data), and each samples data contains exactly *two* feature vectors each.

3.2 Logistic and kernel logistic regression result (Non-stochastic)

We begin by passing all data into one pipeline and runs this procedure N number of times. We do this because of the non-deterministic result we get from random initialization of β and α-for stochastic version.

Using the configuration such that *learning rate* $= 10$, $\gamma = 1$ and $\lambda = 0.00001$ the algorithm returns the result in figure 1.

3.3 Logistic and kernel logistic regression result (Stochastic)

The figure 1 below shows the result of the stochastic version for logistic regression and its kernel versions. After N amount of runs using the configuration such that *learning rate* $= 10$, $\gamma = 1$ and $\lambda = 0.00001$ the algorithm returns the result in figure 2.

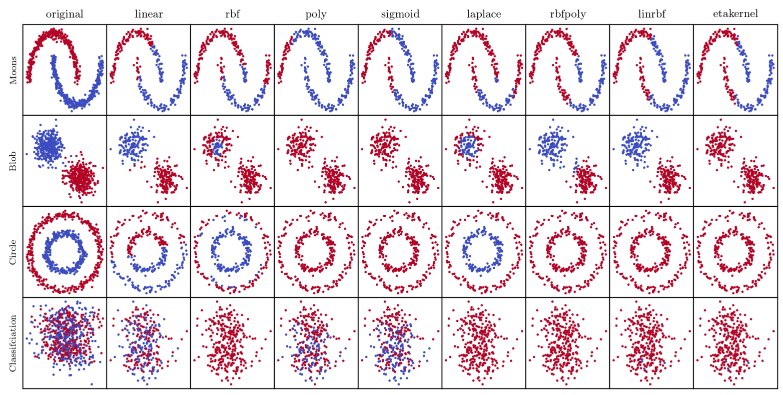


Figure 1: Non-Stochastic logistic and kernel logistic regression. 3 iterations for all except blob data (10 iterations), 0.01 learning rate.

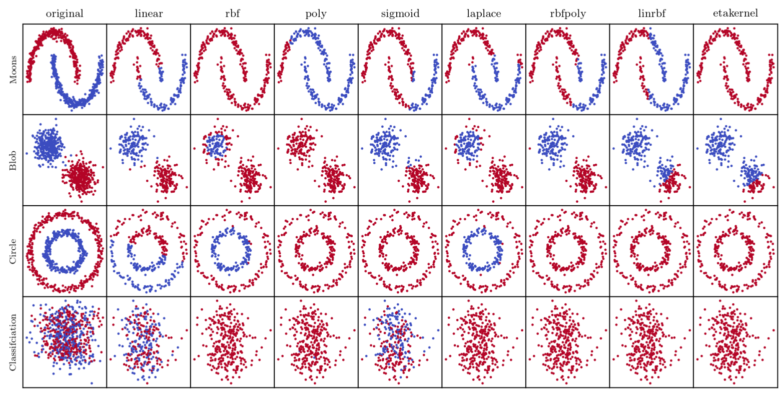


Figure 2: Stochastic logistic and kernel logistic regression. 3 iterations for all except blob data (50 iterations), 0.01 learning rate.

3.4 Performance Analysis

We compare the performance of Logistic regression and its kernel version. We also show that although stochastic logistic regression gives us a better stable result compared to non-stochastic logistic regression, its takes considerable amount of time to compute and hence, less faster compared to non-stochastic version.

3.4.1 Evaluation metric

We use f1-score as our evaluation metric to compare the performance of classical logistic regression and its kernel versions. F1-Score is the harmonic mean of precision and recall and it is given by

$$F1 - score = \frac{2 \times precision \times recall}{precision + recall} \tag{40}$$

where precision is given by

$$precision = \frac{TP}{TP + FP} \tag{41}$$

$$recall = \frac{TP}{TP + FN} \tag{42}$$

TP: True Positives, **TN**: True Negatives, **FP**: False Positives **FP**: False Negatives.

Non-Stochastic F1-Score (%)

kernels	linear	rbf	poly	sigmoid	laplace	rbfpoly	linrbf	eta kernel
Moons	**81**	72	68	72	63	67	69	66
Blobs	98	41	0	66	68	**1**	97	0
Circle	50	93	0	69	**1**	0	0	0
Classification	**88**	0	79	64	0	5	0	4

Non-Stochastic Running Time (*secs*)

kernels	linear	rbf	poly	sigmoid	laplace	rbfpoly	linrbf	eta kernel
Moons	**0.003**	0.02	0.04	0.006	0.02	0.6	0.02	0.37
Blobs	**0.003**	0.02	0.04	0.007	0.03	0.05	0.03	0.12
Circle	**0.003**	0.02	0.04	0.05	0.02	0.06	0.02	0.38
Classification	**0.003**	0.08	0.04	0.005	0.12	0.17	0.08	0.81

Stochastic F1-Score (%)

kernels	linear	rbf	poly	sigmoid	laplace	rbfpoly	linrbf	eta kernel
Moons	86	23	68	69	**87**	67	69	66
Blobs	98	75	0	66	95	88	**99**	84
Circle	48	49	0	17	**69**	0	0	0
Classification	89	0	**78**	17	0	0	0	0

Stochastic Running Time (*secs*)

kernels	linear	rbf	poly	sigmoid	laplace	rbfpoly	linrbf	eta kernel
Moons	0.62	0.05	0.07	**0.02**	0.09	0.21	0.08	0.27
Blobs	0.60	0.37	0.06	**0.05**	0.38	0.06	0.05	0.08
Circle	0.61	0.08	0.250	**0.01**	0.09	0.29	0.07	0.35
Classification	0.63	0.25	0.04	**0.01**	0.28	0.13	0.15	0.23

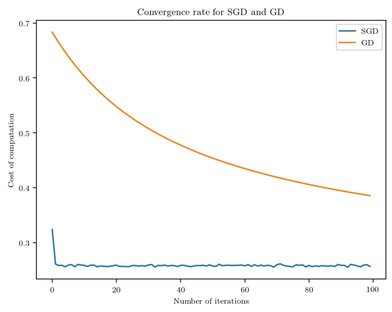
We observer from the $F1-score$ table that linear logistic regression is almost suitable for all datasets except circle data. This is because since circle data is not a linearly seperable data, precisely **rbf kernel** logistic regression is most suitable for classifying it with an score of 93%.

In terms of running time, it is obvious classical or linear logistic regression is the fastest in computation compared to its kernel versions. This is due to the time taken in computing the kernel matrix $\mathbf{O(m \times n)^d}$ where $\mathbf{d}$ is the degree (used for rbf and its variants with $d = 2$ and polynomial with $d \geq 2$). **Sigmoid kernel** still has the fastest running time of all kernels.

Note however, that we have considered different datatypes and the performance of the algorithm can be better evaluated when each dataset is considered individual with different configuration of *learning rate*, γ and *polynomial degree.*

3.5 Convergence rate

We compare the convergence rate for logistic regression and its kernel and stochastic kernel version to speed of convergence.



We observe the speed of convergence for stochastic logistic regression is faster than its gradient descent version. in other words, stochastic gradient version of LR reaches the optimum solution faster than its gradient descent version. We can make the same argument for kernel logistic regression (stochastic) and logistic regression (non-stochastic). Its stochatic version converges faster towards zero than its gradient descent version.

4 CONCLUSION

We demonstrate the use of logistic regression, kernel logistic regression and stochastic version of logistic and kernel logistic regression. We conclude that kernel logistic regression is the best performing algorithm for classifying non-linearly separable data. Its classical version however, has a faster computational time but only serves best for linear binary classification. This is one of the advantages of using stochastic gradient descent version over non-stochastic version. Specifically because it does required large number of iteration to converge.

We introduced the notion of multiple kernel learning and see that that can also outperform classical logistic regression using the F1-score evaluation metric. Stochastic logistic and kernel logistic regression both behave alike with non-stochastic version but can be much stable than their non-stochastic counterpart. We noted that the convergence rate of stochastic logistic and kernel logistic regression is faster than its non-stochastic version.

References

[1] Goldberger, Authur S. Classical Linear Regression. Econometric Theory. John Wiley & Sons. pp. 158. ISBN- 04471-31101-4, 1964.

[2] Scott Menard. Applied Logistic Regression Analysis. SAGE PUBLICATIONS. ISBN- 0-7619-2208-3, 2001.

[3] Zhu J, Hastie T. Kernel logistic regression and import vector machine. *J Comput Graphic Stat*, 14:185–205, 2005.

[4] Keerthi, S., Duan, K., Shevade, S., and Poo, A. A Fast Dual Algorithm for Kernel Logistic Regression. *International Conference on Machine Learning, 19*, 2002.

[5] Maher Maalouf, Theodore B. Trafalis, Indra Adrianto., Kernel logistic regression

using truncated Newton method. *Computer Management Science*, 8:415-428, 2009.

[6] Mehmet Gönen, Ethem Alpaydin. Multiple Kernel Learning Algorithms. *Journal of Machine Learning Research*, 12:2211-2268, 2011.

[7] John Shawt-Taylor, Nello Cristianini. Kernel Methods for Pattern Analysis. Cambridge University Press, ISBN:9780511809682, pg47-83, 2011.