Separating Data Points

K.Padayachee

September 2020

Abstract

A sequence of procedures to calculate hyperplanes that separate data points is examined. These data points represent vectors in multi-dimensional space; subsets of which are anomalous to the rest of the data points. By this we mean, there is some element (or elements) within a vector that differ in some way from the rest of vectors associated with the data points. The objective is to separate the anomolous points from the 'regular points'. Various techniques may be used; in particular, we consider support vector machines, logistic regression and a constrained optimization technique called Sequential Least Squares Quadratic Programs.

1 Hyperplanes used to Separate Data Points

Let $X_1, ..., X_m$ be a set of vectors in $\mathbb{R}^n$ and suppose the $Y_1, ..., Y_m$ are a set $\{0, 1\}$ - variables that indicate whether the vectors $X_1, ..., X_m$ are anomalous or not. We wish to construct a hyperplane that separates the designated anamolous data points from those data points that are not. A hyperplane, if found, will be used to predict the nature of future data points; namely those that are anomalous from those that are not.

2 Linear Separability

We begin our discussion by discussing the notion of *linear separability* which is then tied to the notion of separating hyperplanes. The vectors $X_1, ..., X_m$ are **linearly separable** if there exists a vector $w \in \mathbb{R}^m$ such that $X.w > 0$ where X is the $n \times m$ matrix $[X_1, ..., X_m]$.

In order to link the notion of linear separability with that of separating hyperplanes we begin by introducing the concept of a *perceptron*. A

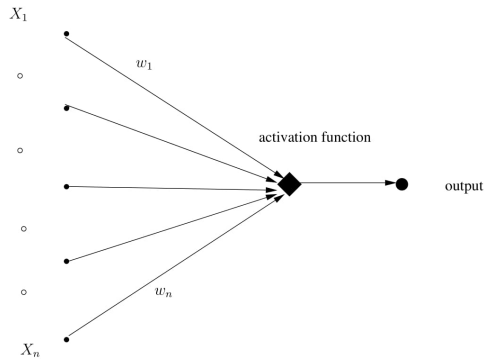

Figure 1: An example of a Perceptron

perceptron is the simplest form of Artificial Neural Net and may be diagrammatically represented as in Figure 1

In Figure 1 , the $\{X_i, i = 1, \ldots, m\}$ are input vectors; the $\{w_i, i = 1, \ldots, m\}$ are weights and the activation function may take several forms some of which are listed here:

- The Signum function:
  $$\text{signum}(X.w) = \begin{cases} 0 & \text{if } X.w <= 0 \\ 1 & \text{else} \end{cases}$$

- The Sigmoid function: $\text{sigmoid}(X.w) = \frac{1}{1+exp(X.w)}$

- The Tanh function: $\tanh(X.w)$

- The ReLu function:
  $$\max(0, X.w) = \begin{cases} 0 & \text{if } X.w <= 0 \\ X.w & \text{else} \end{cases}$$

A Perceptron has a single *activation function* and at least one output. The inputs could be m n-dimensional vectors $\{X_i, i = 1, \ldots, m\}$ where $m, n \in \mathbb{N}$.

If the vectors $X_i, i = 1, \ldots, m$ are linearly separable, we may use the *Perceptron Convergence Algorithm* which finds a $w \in mathbbR^n$ such that $X.w > 0$. An implementation of the Perceptron Convergence Algorithm may be found at `https://github.com/krispad/Machine_Learning`.

$$
\begin{array}{llll}
(Primal) & \max & 0^T w + 1^T e & \\
& \text{s.t.} & Xw - Ie > 1 & \\
& & e >= 0 &
\end{array}
\qquad
\begin{array}{llll}
(Dual) & \min & v^T 1 & \\
& \text{s.t.} & v^T X = 0 & \\
& & v >= 1 &
\end{array}
$$

Table 1: Linear Programming Formulation [1]

We observe that **non convergence** of the *Perceptron Convergence Algorithm* implies non (linear) separability of the vectors X_i. This implies that there does not exist a w such that $Xw > 0$ and $w \in \mathbb{R}^n$. Furthermore we observe that the given definition of *linear separability* is equivalent to finding a w such that $Xw > 1$. With this definition of linear separability, we may formulate the following the problem of finding a w such that $Xw > 1$ as a Linear Programming feasibility problem. The Primal and Dual Linear Programs are depicted in Table 1.

A non-trivial feasible solution to the Dual Linear Program in Table 1 implies a Primal solution. Such a solution w^* results in a w satisfying the conditions of linear separability of the vectors $X_i, i = 1, \ldots, m$. Note that infeasibility or unboundedness of the dual problem results in infeasibility of the primal problem which implies linear non-separability of the vectors $\{X_1, \ldots, X_m\}$.

2.1 Comparing Packaged Versions of Support Vector Machines (SVMs) , Logistic Regression and Convex Optimization Algorithms

We compare the algorithms contained within the **scikit-learn package** for Support Vector Machines and Logistic Regression with the Convex Optimization Algorithms contained within the **SciPy package**.

These algorithms are applied to simulated data consisting of a mixture of bivariate normal data, and data from a combination of the Dirichlet and Gamma distributions. Further details on the specifics of the simulated data will be provided when the algorithms are employed.

We consider three methods to tackle the hyperplane separation problem, namely Support Vector Machines (SVMs), Logistic Regression and Convex Optimization. We note that the methodologies employed for SVMs and Logistic Regression are in widespread use and well understood hence we will not be examining the mathematical details associated with these methodologies. We do point out that the description given below of the Convex

[1] The variables e are artificial variables

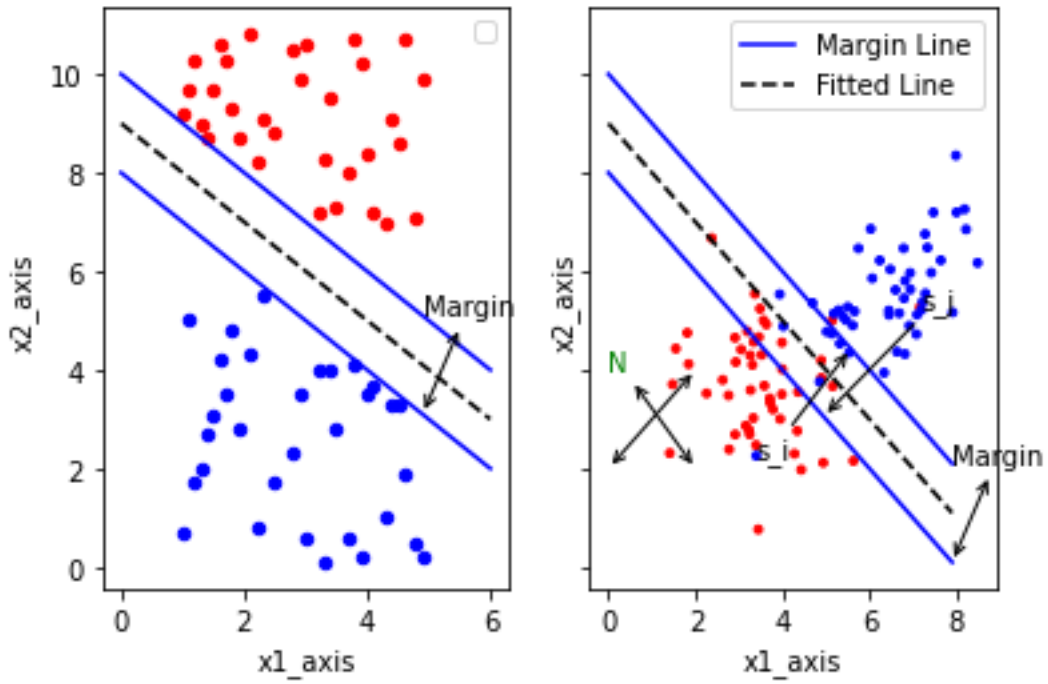

Figure 2: Linearly/Non-Linearly Separable Points

Optimization formulation is the same as the SVM formulation presented by Friedman *et. al* in [1] where in the case of the Convex Optimization problem they implement the Lagrangian dual (more about this concept later).

2.2 Convex Optimization

We wish to find coefficients $(\beta, \beta_0); \beta \in \mathbb{R}^n, \beta_0 \in \mathbb{R}$, of a hyperiplane that separates data points $\{(x_i, y_i)\}, i = 1, y_i \in \{-1, 1\}, i = 1, \ldots, m$; into two groups. In the case where the points are **linearly separable**, there exists a $w \in \mathbb{R}^m$ such that $X.w > 0$. Such a w can be found using the Perceptron Convergence Algorithm or the linear programming formulation in Table 1. It should be pointed out that the w found by either of these methods is not unique. This observation provides some motivation for the following approach.

We wish to find a w such that the closest given data point on either side of the hyperplane is at least M units away from the hyperplane. This region, at a perdindicular distance of M units on either side of the fitted hyperplane, is called a ***margin***. The left-side of Figure 2 provides an illustration.

It follows from vector algebra and properties of hyperplanes that the closest points on either side of a fitted hyperplane are proportional to a normal to the surface of the fitted hyperplane. And hence a formal description of the problem may be given:

$$\begin{array}{rl} & \max_{\{\|\beta\|=1, \beta_0\}} M \\ where & y_i.(\beta^T x_i + \beta_0) \geq M, i = 1, \ldots, m; \beta \in \mathbb{R}^n \end{array}$$

In the above formulation , β, β_0 and M are the unknowns. If we set $\|\beta\| = 1/M$ the problem may be reformulated as:

$$\begin{array}{rl} & \min_{\{\beta, \beta_0\}} \|\beta\| \\ where & y_i.(\beta^T x_i + \beta_0) \geq 1, i = 1, \ldots, m; \beta \in \mathbb{R}^n \end{array}$$

In the reformulation, (β, β_0) are unknowns.

In the case where the data points are **not linearly separable**, we adjust the above formulation in an attempt to minimize the number of points from both groups that are on the wrong side of the fitted hyperplane contained within the margin. Colloquially speaking, we wish to find a hyperplane separating the set of points $\{(x_i, y_i)\}$ such that 'most' of the points with $y_i = -1$ lie on one side of the hyperplane and 'most' of the points with $y_i = 1$ lie on the other. We also wish to control the number of points associated with either $y_i = -1$ or $y_i = 1$ that lie on the wrong side of the hyperplane.

With this in mind, we introduce variables, $s_i, i = 1, \ldots, m$, called slack variables to modify, without loss of generality, the problem as follows:

$$\begin{array}{ll} & \min \frac{\|\beta\|^2}{2} \\ where & y_i.(\beta^T x_i + \beta_0) \geq 1 - s_i, i = 1, \ldots, m; \beta \in \mathbb{R}^n \\ & \sum s_i \leq C \\ & s_i \geq 0, i = 1, \ldots, m \end{array}$$

Note that when β is unscaled the assumption $\|\beta\| = \frac{1}{M}$ implicitly implies that $y_i.(\beta^T x_i + \beta_0) \geq M(1 - s_i), \forall i$. The constant C is positive and forms a hyperparameter that may be 'tuned' by the user.

From the constraints above, we observe that the points x_i lying on the correct side of *their* margin are such that $s_i = 0$ suffices. Points x_i on the wrong side of *their* margin miss the margin by an amount Ms_i. Points that lie outside *and* on the wrong side of *their* margin, would have $Ms_i > 2M$ - we want as few as these points as possible. Colloquially, the formulation above indicates that an optimization procedure which finds a hyperplane that separates points x_i and is determined by controlling the total amount that misses the correct side of the hyperplane, may 'optimally' separate most of the points x_i.

The right-side of the figure above provides an illustration. For example, the blue point labelled s_i and lying on the wrong side of the margin will have a non-zero slack value in the output of the optimization algorithm. The red point labelled s_j will also have a non-zero slack value in the optimal solution.

2.3 The Data Used in Comparing Support Vector Machines (SVMs), Logistic Regression and Convex Optimization Algorithms

We simulate two sets of data points, the first from a bivariate normal distribution and the second a mixture of data pairs; the first set of pairs has the first and second coordinate from a lognormal and gamma distribution respectively and the second consists of data points from a bivariate Dirichlet distribution. The first set, that is the bivariate normal data will be regarded as the 'fraudulent' data and the combined lognormal, gamma, and bivariate dirichlet data the 'non-fraudulent' data. We emphasize that the non-fraudulent data is produced using the data pair distributed as (lognormal, gamma) or bivariate Dirichlet whereas the fraudulent data comes from a bivariate normal distribution.The code to produce the input or

training data is contained in the file *simdata_fraud1.py* found at `https://github.com/krispad/Machine_Learning`.

2.3.1 A Solution to the Hyperplane Separation Problem

Derivation of the Separating Hyperplane using Convex Optimization We consider an equivalent convex optimization problem expressed in matrix form:

$$\begin{array}{lrcl} min & \frac{\beta^T.\beta}{2} + C.\mathbf{1^T}.\mathbf{s} & & \\ where & I^*.(X^T.\beta + \beta_0.\mathbf{1}) & \geq & \mathbf{1} - \mathbf{s} \\ & s & \geq & 0 \end{array}$$

Here β, s are $n\times 1, m\times 1$ real vectors respectively and $\mathbf{1}$ a $m\times 1$ vector of ones; $\mathbf{I}^*, \mathbf{X}$ are $m \times m, n \times m$ real matrices respectively; β_0, C are constants. The matrix X consists of m input data vectors $x_i, i = 1, \ldots, m$ and the diagonal matrix I^* is defined as follows:

For each $i = 1, \ldots, m$:

$I^*(i,i) = \begin{cases} 1 & \text{if } y_i = 1 \\ -1 & \text{if } y_i = -1 \end{cases}$ The problem above forms a convex optimization problem. A conventional approach to solving the above convex optimization problem is to begin with the (Primal) Lagrangian. Namely,

$$L_P(\beta, \beta_0, s, a) = \frac{\beta^T.\beta}{2} + C.\mathbf{1^T}.\mathbf{s} + \mathbf{a^T}.((\mathbf{1}-\mathbf{s}) - \mathbf{I}^*.(\mathbf{X^T}.\beta + \beta_\mathbf{0}.\mathbf{1})) + \mu^\mathbf{T}.\mathbf{s} \quad (1)$$

Here a, μ are non-negative $m \times 1$ real vectors. We see that the (Primal) Lagrangian is itself convex in the parameters $(\beta, \beta_0, s, a, \mu)$ and forms a lower bound to the convex optimization problem for any feasible tuple $(\beta, \beta_0, s, a, \mu)$.

If we fix a and μ and optimize L_P wrt. to (β, β_0, s) we obtain the dual Lagrangian, that is

$$L_D(a, \mu) = min_{(\beta, \beta_0, s)} L_P(\beta, \beta_0, s, a, \mu) \quad (2)$$

with a, μ fixed we have $\bigtriangledown L_p(\beta^*, \beta_0^*, s^*, a, \mu) = 0$ at an optimal solution $(\beta^*, \beta_0^*, s^*)$ to the primal Lagrangian L_P given by equation (1). As a result the following equalities are obtained:

$$\begin{array}{lrcl} \frac{\delta L_p}{\delta \beta} & = & \beta^{*T} - a^T.\mathbf{I}^*.\mathbf{X^T} & = 0 \\ \frac{\delta L_p}{\delta \beta_0^*} & = & -a^T.\mathbf{I}^*.\mathbf{1} & = 0 \\ \frac{\delta L_p}{\delta s} & = & C.\mathbf{1^T} - \mathbf{a^T} - \mu^\mathbf{T} & = 0 \end{array}$$

Substituting the above equalities into the (Primal) Lagrangian results in the Lagrangian Dual, L_D.

$$L_D(a) = -\frac{a^T(X^T.X).a}{2} + a^T.\mathbf{1} \tag{3}$$

where $a^T.\mathbf{I}^*.\mathbf{1} = \mathbf{0}$ and $0 \leq a_i \leq C, \forall i = 1, \ldots, m$

We note that the Lagrangian Dual depends on the input vectors $\{x_i, i = 1, \ldots, m\}$. Maximizing L_D with respect to a and satisfying the constraints in 3 provides us with the **best** or **closest** lower bound to the convex optimization problem. Furthermore the lower bound is equal to the optimum of the convex optimization problem if the following conditions hold at a fixed tuple (β, β_0, s).

$$\begin{aligned} a^T.(I^*.(X^T.\beta + \beta_0.\mathbf{1}) - (\mathbf{1} - \mathbf{s})) &= 0 & (4)\\ s.\mu^T &= 0 & (5)\\ I^*.(X^T.\beta + \beta_0.\mathbf{1}) - (\mathbf{1} - \mathbf{s}) &\geq 0 & (6) \end{aligned}$$

Therefore if we solve for $max_a L_D(a)$ to obtain a and find (β, β_0, s) that satisfy the above equations we are able to construct the hyperplane that optimally separates the vectors $\{x_i, i = 1, \ldots, m\}$. In passing, we note that the condition $\bigtriangledown L_p(\beta, \beta_0, s, a, \mu) = 0$ and the six equalities above are known as the Karush-Kuhn-Tucker $(KKT))$ conditions.

If for a feasible $(\beta, \beta_0, s, a, \mu)$ the Karush-Kuhn-Tucker conditions hold, then the tuple $(\beta, \beta_0, s, a, \mu)$ forms an optimal solution to both the Lagrangian Primal and its Dual. We note that one of the KKT conditions, namely $\bigtriangledown L_p(\beta, \beta_0, s, a, \mu) = 0$ implies that at optimality of the Lagrangian Dual we obtain the equation

$$\beta^T = a^T.\mathbf{I}^*.\mathbf{X^T} \tag{7}$$

for all non-zero values of $a = \{a_i, i = 1, \ldots, m\}$. That is, only the non-zero values of a contribute to the construction of the optimal β.

The constant β_0 is taken to be $\frac{1}{m}(\mathbf{1} - \mathbf{X^T}\beta)^\mathbf{T}.\mathbf{1}$, where we average over all the $x_i, i = 1, \ldots, m$ associated with non-zero a_i.

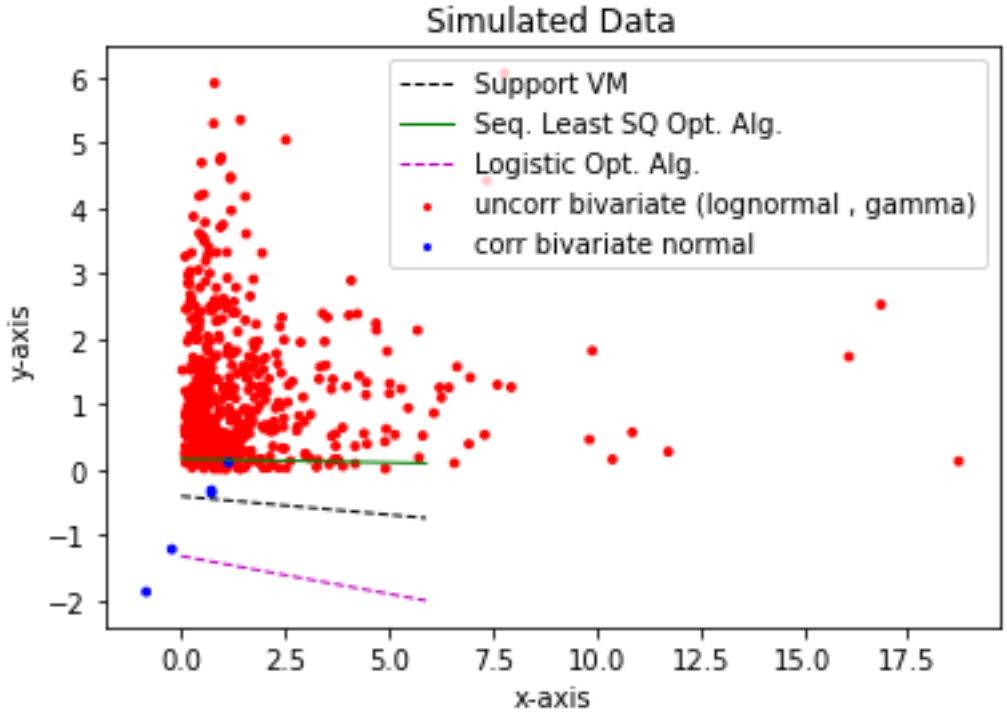

Figure 3: SLSQ, SVM, LOGISTIC

2.4 Analysis of the TrainingData

2.4.1 Application of Convex Optimization, SVM and Logistic Regression Routines

We use the SciPy optimization package to optimally separate two groups of data points with a hyperplane. In the Training Data example, the two groups of data points represent 'fraudulent' and 'non-fraudulent' data respectively. The results are compared to SVM and Logistic Regression implementations from the **ski-learn** package. In our implementation of the convex optimization routine, we use the Sequential Least Squares Quadratic Algorithm (SLSQ) and observe in the figure (3) that SLSQ has superior performance to either SVM or Logistic Regression. The code to generate Figure(3) may be found in `https://github.com/krispad/Machine_Learning`

References

[1] Friedman, Hastie, Tibshirani. *The Elements of Statistical Learning, Second Edition 2009, Springer Verlag.*