Método de proyección de gradiente en el entrenamiento de Support Vector Machines

Juan Sánchez[a], David Halliday[b], David Molina[c], Christian Pabón[d]

[a] *jusanchezmu@unal.edu.co*
[b] *dhalliday@unal.edu.co*
[c] *demolinar@unal.edu.co*
[d] *ccpabonu@unal.edu.co*

Abstract

Los modelos de *Support Vector Machine* en su mayoría utilizan librerías que entrenan dichos modelos con métodos de gradiente en descenso. En este artículo desarrollamos una comparación de diferentes métodos de entrenamiento para un modelo de *Support Vector Machine*, capaz de clasificar dos tipos de frutas o más a partir de imágenes brindadas por un dataset de uso libre.
Comparamos el desempeño de modelos entrenados con métodos usuales, con respecto al entrenado con el método de proyección de gradiente. Aplicamos estos modelos en Python utilizando las librerías SKLearn, SciPy, PyOptim y ProjGrad.
Para realizar dicha comparación entre modelos, usamos los tipos de evaluación *Accuracy*, la curva ROC y la matriz de confusión.

Keywords: Método de proyección del gradiente, Machine Learning, Support Vector Machines, Optimización.

1. Introducción

Estamos en la era de la Big Data. Actualmente existen más de 50 billones de páginas indexadas por Google [1], para el mes de junio del 2022 Youtube ya tiene más de 2.6 billones de usuarios activos, adicionalmente se envían más de 8.67 millones de tweets a diario y las bases de datos de Amazon registran más de 500,000 transacciones por segundo [2]. Los avances tecnológicos han hecho posible almacenar, leer y escribir tanta información a costos sostenibles. El mundo entero genera cada vez más datos que brindan mejores modelos de negocio si dichos datos son almacenados, manipulados y estudiados de manera práctica, es por ello que la ciencia de datos toma tanta importancia en lo que académicos llaman la cuarta revolución industrial. No obstante, cada vez requerimos mejores modelos, prácticas e investigación para manipular dichos datos y para así tomar mejores decisiones respecto a los resultados que aquellos modelos nos proveen, esto es útil en la resolución de problemas en cualquier campo de estudio que se requiera.

En la actualidad, gracias a la cantidad de datos y modelos existentes, es posible crear un modelo matemático que sea capaz de generalizar y predecir de manera óptima modelos que aunque parezcan bastante contemporáneos ya habían sido estudiados e implementados desde mucho antes, como fue el caso en 1805 por Legendre y por Gauss en 1809 quienes implementaron la primera forma de regresión que fue el método de mínimos cuadrados para el problema de determinar a partir de observaciones astronómicas, la órbitas de cuerpos alrededor del Sol (en su mayoría cometas, pero también más tarde los planetas menores recién descubiertos), incluso en 1943 Warren S. McCulloch y Walter Pitts en el boletín de matemáticas publicaron *A logical calculus of the ideas immanent in nervous activity* [3] e introdujeron el primer perceptrón, lo que sería después la unidad mínima en una capa de una red neuronal, sin embargo, pocos modelos tienen la suficiente rigurosidad matemática como para predecir qué tantos datos requerimos para llegar a una generalización óptima o qué tan factible es para cierta estructura de datos. Por ello trabajaremos con el modelo

de *Support Vector Machine* (SVM), el cual es capaz realizar regresiones o clasificaciones dado un conjunto de datos etiquetado previamente, que utilizaremos para entrenarlo.
Los modelos de *Support Vector Machine* se utiliza sobre todo para realizar la tarea de clasificación, además dicho modelo funciona por medio de un método de optimización que minimiza una función de costo y es en la variedad de estos métodos de optimización en los que profundizaremos, comparándolos con con nuestra implementación del método de proyección de gradiente, el cual es poco utilizado en los SVM.

A continuación vamos a ver varios conceptos que nos van a permitir desarrollar de forma teórica y práctica un modelo de SVM.

2. Machine Learning y Support Vector Machine

Machine Learning (ML) es una forma de Inteligencia Artificial (IA) que permite a un sistema aprender un conjunto de reglas a partir de datos en lugar de mediante la programación explícita de dichas reglas. En la mayoría de casos, el *machine learning* es usado para obtener una función $g(x)$ que reproduzca el comportamiento de una función objetivo $f(x)$ que es parcialmente conocida.

El *machine learning* no es un proceso sencillo. Conforme el algoritmo ingiere datos de entrenamiento, es posible producir modelos más precisos basados en esta información. Después del entrenamiento, al proporcionar al modelo una entrada, obtendrá un resultado acorde a los ajustamientos del modelo entrenado.
Una Support Vector Machine(SVM) es un modelo de aprendizaje supervisado (técnica usada que consiste para deducir una función a partir de varios datos de entrenamiento) con algoritmos de aprendizaje asociados que analizan datos para clasificación y análisis de regresión. Las SVM son uno de los métodos de predicción más robustos y se basan en marcos de teoría Vapnik–Chervonenkis (VC) [4], el cual cubre al menos cuatro partes:

- Teoría de la consistencia de los procesos de aprendizaje
  ¿Cuáles son las condiciones (necesarias y suficientes) para la consistencia de un proceso de aprendizaje basado en el principio empírico de minimización de riesgos?
- Teoría no asintótica de la tasa de convergencia de los procesos de aprendizaje
  ¿Qué tan rápida es la tasa de convergencia del proceso de aprendizaje?
- Teoría del control de la capacidad de generalización de los procesos de aprendizaje
  ¿Cómo se puede controlar la tasa de convergencia (la capacidad de generalización) del proceso de aprendizaje?
- Teoría de la construcción de máquinas de aprendizaje
  ¿Cómo se pueden construir algoritmos que puedan controlar la capacidad de generalización?

Como se menciono anteriormente, las técnicas de *machine learning* buscan emular el comportamiento de una función objetivo $f(x)$ que conocemos parcialmente. Es importante notar que el dominio de está función es complejo, es decir, de elevada dimensión, por lo que se suele reducir la dimensión de este espacio mediante técnicas de *feature extraction*, es decir, mapear una función a un espacio de dimensión menor perdiendo la menor cantidad de información posible.

Los datos dados para entrenar el algoritmo se representan como un punto único en un espacio donde cada punto está representado por algún vector de características x donde $x \in \mathbb{R}^n$.
Una representación gráfica del mapeo de un punto en un espacio de características complejo x es el que se puede observar en la Figura 1.

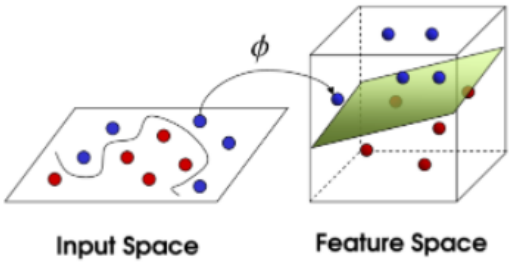

Figura 1: Representación gráfica del mapeo en un espacio de características complejo.

Ahora, si separamos estos puntos usando una función lineal, hallamos la ecuación de una línea recta con pendiente m e intersección c, la ecuación sería $mx + c = 0$ en $\mathbb{R}^2$.
Para una dimensión n tendríamos la ecuación del hiper-plano que divide los puntos (para clasificar) como $H := wT(x) + b = 0$ donde b es el término de intersección y sesgo de la ecuación del hiper-plano, el cual siempre es de orden $n - 1$.
Al ajustar la línea de separación,queremos una línea que pueda clasificar los puntos de la mejor manera posible y con la menor cantidad de errores. Para tener pocos errores en la clasificación de los puntos se requiere medir la distancia entre un punto y la línea de separación.
La distancia de un hiper-plano $wT\phi(x) + b = 0$ a un vector $\phi(x_0)$ se puede escribir como:

$$d_H\left(\phi\left(x_0\right)\right) = \frac{\left|w^T\left(\phi\left(x_0\right)\right) + b\right|}{\|w\|_2} \qquad \text{con} \qquad \|w\|_2 =: \sqrt{w_1^2 + w_2^2 + w_3^2 + \ldots w_n^2} \tag{1}$$

Ahora bien, podría haber muchos hiper-planos con diferentes inclinaciones separando los datos, entonces para elegir el hiper-plano óptimo, se debe colocar el hiper-plano justo en el centro donde la distancia es máxima desde los puntos más cercanos y proporcione la menor cantidad de errores (véase 2).

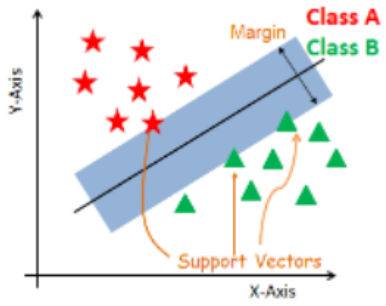

Figura 2: Representación gráfica una división óptima.

Así que ahora debemos encontrar un hiper-plano con margen máximo, donde el margen es un espacio protegido alrededor del hiper-plano. El algoritmo de SVM intenta tener un margen máximo con los puntos más cercanos, conocidos como vectores de soporte, es decir el objetivo es maximizar la distancia mínima para la distancia. Esto es,

$$w^* = \arg_w \max \left[\min_n d_H\left(\phi\left(x_n\right)\right)\right]. \tag{2}$$

$$y_n \left[w^T \phi(x) + b \right] = \begin{cases} \geq 0 \text{ Si es correcto} \\ < 0 \text{ Si es incorrecto} \end{cases} \tag{3}$$

A continuación vamos a revisar algunos conceptos que nos permiten clasificar conjuntos de datos.

- **Datos perfectamente separables** Para conjuntos de datos perfectamente separables, el hiper-plano óptimo clasifica todos los puntos correctamente, sustituyendo además los valores óptimos en la ecuación de peso.

$$w^* = \arg_w \text{máx} \left[\min_n \frac{\left| w^T \left(\phi \left(x_n \right) \right) + b \right|}{\|w\|_2} \right] = \arg_w \text{máx} \left[\min_n \frac{y_n \left| w^T \left(\phi \left(x_n \right) \right) + b \right|}{\|w\|_2} \right]. \tag{4}$$

  El término $\arg_{\text{máx}}$ es una abreviatura de argumentos máximos, es decir los puntos del dominio de una función en los que se maximizan los valores de esta.
  Podemos reescribir la ecuación (4) como

$$w^* = \arg_w \text{máx} \frac{1}{\|w\|_2} \left[\min_n y_n \mid w^T (\phi(x) + b \mid \right]. \tag{5}$$

  El término interno $\left(\text{mín}_\text{n} \, \text{y}_\text{n} \left| \text{w}^\top \boldsymbol{\phi}(\text{x}) + \text{b} \right| \right)$ representa la distancia mínima de un punto al límite de decisión y el punto más cercano al límite de decisión H.
  Así que podemos modificar un poco la función a maximizar y transformarla convenientemente para usar multiplicadores de Lagrange y continuar con la optimización de la siguiente forma :

$$\text{máx} \frac{1}{\|w\|} \equiv \text{mín} \|w\| \quad \text{acomodamos por conveniencia} \quad \text{mín} \frac{1}{2} \|w\|^2 \text{ así } \frac{\mathrm{d}}{\mathrm{d}x} \frac{1}{2} x^2 = x. \tag{6}$$

  Para maximizar la función usamos el Lagrangiano, el cual nos dice que restemos la función de costo por la suma de todas las restricciones donde cada una de esas restricciones se multiplicará por algún alfa constante como se ve a continuación:

$$\begin{aligned} L &= \frac{1}{2} \|\vec{w}\|^2 - \sum_i^n \alpha_i \left[y_i (\vec{w} \cdot \vec{x} + b) - 1 \right] \\ \frac{\partial L}{\partial w} &= \vec{w} - \sum_i^n \alpha_i y_i x_i = 0 \\ \vec{w} &= \sum_i^n \alpha_i y_i x_i \\ \frac{\partial L}{\partial b} &= - \sum_i^n \alpha_i y_i = 0 \\ \sum_i^n \alpha_i y_i &= 0. \end{aligned} \tag{7}$$

de ahí podemos deducir:

$$
\begin{aligned}
L &= \frac{1}{2}\|\vec{w}\|^2 - \sum_i^n \alpha_i \left[y_i(\vec{w}\cdot\vec{x}+b)-1\right] \quad \vec{w} = \sum_i^n \alpha_i y_i x_i \\
L &= \frac{1}{2}\left(\sum_i^n \alpha_i y_i \vec{x}_i\right)\cdot\left(\sum_i^n \alpha_j y_j \vec{x}_j\right) - \sum_i^n \alpha_i y_i \vec{w}\cdot\vec{x} - \sum_i^n \alpha_i y_i b + \sum_i^n \alpha_i \\
\vec{w} &\equiv \sum_i^n \alpha_i y_i x_i \qquad\qquad \sum_i^n \alpha_i y_i = 0 \\
L &= \frac{1}{2}\sum_i^n\sum_j^n \alpha_j y_j \alpha_i y_i \vec{x}_i\cdot\vec{x}_j - \sum_i^n\sum_j^n \alpha_j y_j \alpha_i y_i \vec{x}_i\cdot\vec{x}_j - 0 + \sum_i^n \alpha_i \\
L &= \sum_i^n \alpha_i - \frac{1}{2}\sum_i^n\sum_j^n \alpha_i\alpha_j y_i y_j x_i\cdot x_j.
\end{aligned}
\tag{8}
$$

Antes de que podamos continuar, necesitamos expresar la ecuación en términos de matrices en lugar de sumas. La razón es que la función *qp* de la biblioteca CVXOPT [5], que usaremos para resolver el Lagrangiano, acepta argumentos muy específicos. Por lo tanto, tenemos que pasar de:

$$
L = \sum_i^n \alpha_i - \frac{1}{2}\sum_i^n\sum_j^n \alpha_i\alpha_j y_i y_j x_i\cdot x_j \text{ Dónde: } \alpha_i \geq 0,\ \sum_i^n \alpha_i y_i = 0,
$$

a la forma:

$$
\begin{array}{ll}
\text{minimize} & (1/2)x^T P x + q^T x \\
\text{subject to} & Gx \preceq h \\
& Ax = b.
\end{array}
\tag{9}
$$

Podemos lograr esto usando las siguientes identidades:

- $\left(\mathbf{A}^{\mathrm{T}}\right)^{\mathrm{T}} = \mathbf{A}$.
- $a \cdot b = \sum_i^n ab$.
- $\mathbf{v}^T\mathbf{w} = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}\begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = v_1 w_1 + \cdots + v_n w_n = \mathbf{v}\cdot\mathbf{w}$.
- $\boldsymbol{\alpha} = [\alpha_1, \alpha_2 \ldots \alpha_N]^{\boldsymbol{T}}$.
- $\boldsymbol{y} = [y_1, y_2 \ldots y_N]^T$.
- $\mathbf{1}_N = [1, 1 \ldots 1]^T$.
- $-\boldsymbol{X}_{(d\times N)} = [x_1 y_1, \boldsymbol{x}_2 y_2 \ldots \boldsymbol{x}_N y_N]$.

Aplicando obtenemos :

$$
\begin{aligned}
L &= \sum_i^n \alpha_i - \frac{1}{2}\left(\sum_i^n \alpha_i y_i x_i\right)\cdot\left(\sum_i^n \alpha_j y_j x_j\right) \\
L &= 1^T\alpha - \frac{1}{2}\left(\alpha^T X\right)\cdot\left(\alpha^T X\right) \qquad \text{pues } X = [x_1 y_1 \ldots] \\
L &= 1^T\alpha - \frac{1}{2}\left(\alpha^T X\right)^T\left(\alpha^T X\right) \\
L &= 1^T\alpha - \frac{1}{2}\alpha X^T\alpha^T X \\
L &= 1^T\alpha - \frac{1}{2}X^T\alpha^T X\alpha.
\end{aligned}
\tag{10}
$$

Entonces quedarían asignadas así para usar nuestra función:

$$\begin{array}{lll} \text{minimize} & (1/2)x^T Px + q^T x & \longleftarrow \quad SVMProblem \\ \text{subject to} & Gx \preceq h & \quad \text{máx}_\alpha \quad \mathbf{1}^T\alpha - \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\alpha} \\ & Ax = b & \quad \text{Subject to:} \quad \alpha \geqslant 0 \\ & & \qquad\qquad\quad y^T\alpha = 0. \end{array} \tag{11}$$

$$\begin{gathered} X = \alpha \\ P = X^T X \\ q = -1_N \\ G = -1_{NxN} \\ h = 0_N \\ A = y^T \\ v = 0 \end{gathered}$$

3. Métodos de evaluación

La **Precisión** de un modelo se define como:

$$\text{Precisión} = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}}, \tag{12}$$

sin embargo, para problemas de clasificación binaria se puede calcular mediante el número de positivos y negativos,

$$\text{Precisión} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{VN} + \text{FP} + \text{FN}}, \tag{13}$$

donde VP =Verdaderos positivos, VN =Verdaderos negativos, FP =Falsos positivos y FN =Falsos negativos.

En términos binarios, un verdadero positivo es cuando la etiqueta real corresponde a un 1 y el algoritmo asigna la etiqueta de 1, un verdadero negativo es cuando nuestro la etiqueta corresponde a un 0 y el modelo le asigna una etiqueta de 0. De manera análoga se definen los falsos positivos y falsos negativos .

Una **Matriz de confusión** o matriz de error es una tabla desarrollada para visualizar información relevante de un modelo de aprendizaje supervisado. La tabla está compuesta por dos filas y dos columnas que reportan el número de falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos, esto permite un análisis más detallado a comparación de solo observar la precisión del modelo[6].

Una **Curva ROC** es un gráfico de dos dimensiones, donde el eje y corresponde al ratio de falsos positivos fp y el eje x al ratio de verdaderos positivos tp

$$fp = \frac{\text{FN}}{\text{FN} + \text{VN}} \quad , tp = \frac{\text{VP}}{\text{FP} + \text{VP}} \tag{14}$$

La curva ROC ayuda a describir la relación entre los beneficios(verdaderos positivos) y el costo(falsos positivos) [6].

Ya que sabemos como evaluar un modelo de SVM, solo nos queda ver como se entrena el modelo usando optimización, para ello debemos ver algunos conceptos de optimización convexa.

4. Problemas cuadráticos

Un problema de optimización convexo se dice cuadrático o QP(quadratic problem) si la función objetivo es convexa cuadrática y las restricciones son afines. Recordemos que una función $f : \mathbb{R}^n \to \mathbb{R}^m$ se dice

afín si es una suma de una función lineal y una constante, es decir, es de la forma $f(x) = Ax + b$ donde $A \in \mathbb{R}^{m\times n}$ y $b \in \mathbb{R}^m$, y una función $g : \mathbb{R}^n \to \mathbb{R}$ es convexa si dom(f) es un conjunto convexo y si para todo $x, y \in$ dom(f), con $0 \leq \theta \leq 1$ se tiene

$$f(\theta x + (1\theta)y) \leq \theta f(x) + (1-\theta)f(y). \tag{15}$$

Un problema cuadrático se expresa de la forma

$$\begin{array}{ll} \text{minimizar} & (1/2)x^T Px + q^T x + r \\ \text{sujeto a} & Gx \preceq h \\ & Ax = b, \end{array} \tag{16}$$

donde $P \in S_+^n$, es decir P es una matriz simétrica semi definida positiva, $G \in \mathbb{R}^{m\times n}$, y $A \in \mathbb{R}^{p\times n}$. Así que estamos minimizando una función convexa cuadrática sobre un poliedro.

En caso que P no fuese una matriz semi definida positiva, se dice que el problema es no convexo QP. Este tipo de problemas resulta ser más difícil de solucionar ya que hay varios mínimos locales [7].
Un ejemplo de problema QP sin restricción es el problema de mínimos cuadrados, que consiste en minimizar

$$\|Ax - b\|_2^2 = x^T A^T Ax - 2b^T Ax + b^T b, \tag{17}$$

este problema surge en diferentes campos, como el análisis de regresión o la aproximación por mínimos cuadrados [8]. Se conoce solución analítica para este problema, esta es $x = A^{\dagger}$b, donde $A^{\dagger}$ es la pseudo inversa de A [9].

Veamos los métodos para solucionar el problema de minimización que presenta la SVM.

5. Método de gradiente descendiente estocástico

El método de gradiente en descenso es un proceso iterativo de aprendizaje, donde una función objetivo es minimizada de acuerdo a la dirección opuesta del gradiente, dado que está es la de mayor crecimiento. Es importante notar que el gradiente se computa una vez para todo el conjunto de datos, por lo que esto puede dar resultados no muy buenos, en la practica se utilizan *batches* o subconjuntos de datos para calcular el gradiente en cada uno de ellos [10].

El método de gradiente descendiente estocástico(SGD) se caracteriza por computar el gradiente en cada dato del conjunto,de esta forma se reduce las posibilidades de caer en un mínimo local a cambio de aumentar el costo computacional [11].

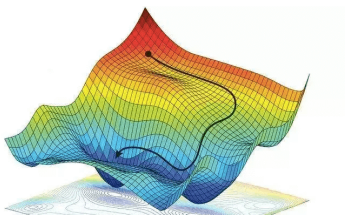

Figura 3: Representación gráfica del método de gradiente descendiente

6. Método de proyección del gradiente

Ahora que ya estamos más familiarizados con los problemas cuadráticos consideremos un caso específico de estos, que consiste en la solución[12] numérica de Serafini et al. programa cuadrático

$$\min_{x\in\Omega} f(x) = \frac{1}{2}x^T Gx + q^T x, \tag{18}$$

En el cual G es una matriz simétrica de dimensiones $n \times n$, q, x son vectores de $\mathbb{R}^n$ y la minimización está sujeta al conjunto $\emptyset \neq \Omega \subset \mathbb{R}^n$ que debe ser un conjunto cerrado convexo, definido con condiciones simples, se suele llamar a la región factible de este problema como un çaja", dado a que define una región rectangular en el espacio.

El método de proyección del gradiente se puede describir mediante dos etapas. En la primera, nos movemos en la dirección contraria al gradiente, es decir $-g$ donde $g = Gx + q$, si se encuentra con una restricción entonces la dirección se altera para que se mantenga en la región factible. Ahora buscamos en el camino obtenido al alterar nuestra dirección.

El segundo paso es explorar la cara de la caja definida por la región factible y usarla para resolver el subproblema, es decir 6 sujeto a la región de la cara de la caja [7].

El algoritmo del método de proyección del gradiente propuesto en [12] es
ALGORITMO MPG (Métodos de Proyección del Gradiente)

Paso 1. *Inicialización.* Tome $x^{(0)} \in \Omega$, defina $\alpha_0, \alpha_{\text{mín}}, \alpha_{\text{máx}}$ tales que $0 < \alpha_{\text{mín}} < \alpha_{\text{máx}}$ y $\alpha_0 \in [\alpha_{\text{mín}}, \alpha_{\text{máx}}]$, establezca $k = 0$.

Paso 2. *Proyección.* Finalizar si $x^{(k)}$ cumple con el criterio de terminación. De lo contrario, calcule

$$d^{(k)} = P_\Omega(x^{(k)} - \alpha_k[Gx^{(k)} + q]) - x^{(k)}, \tag{19}$$

donde $P_\Omega(\cdot)$ denota la proyección ortogonal sobre Ω.

Paso 3. *Búsqueda de línea.* Calcule

$$x^{(k+1)} = x^{(k)} + \lambda_k d^{(k)}, \quad \lambda_k \in (0, 1], \tag{20}$$

donde λ_k es el tamaño del paso determinado por un procedimiento de búsqueda de línea.

Paso 4. *Actualización.* Calcule $\alpha_{k+1} \in [\alpha_{\text{mín}}, \alpha_{\text{máx}}]$, declaramos ahora k como $k+1$ y reiteramos desde el Paso 2.

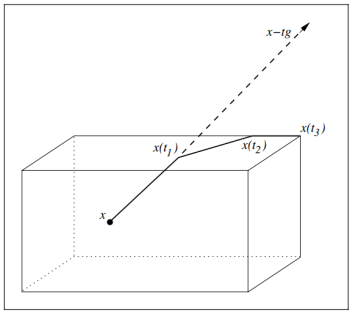

Figura 4: Búsqueda de línea de una función definida en $\mathbb{R}^3$

7. Planteamiento del problema

En la práctica, el entrenamiento de *SVM* se realiza mayoritariamente con métodos basados en el gradiente en descenso [13]. Queremos comparar el desempeño de dos modelos supervisados de clasificación binarios entrenados a partir del mismo *dataset* usando el método usual (gradiente descendiente estocástico) y el método de proyección del gradiente.

Dado un conjunto de entrenamiento con datos etiquetados

$$\mathcal{D} = \{(z_i, y_i), i = 1, \ldots, n, z_i \in \mathbb{R}^m, y_i \in \{-1, 1\}\}, \tag{21}$$

la *SVM* realiza la clasificación de datos z no pertenecientes al conjunto de entrenamiento a través de la función de decisión $\mathcal{F} : \mathbb{R}^m \to \{-1, 1\}$, de la forma

$$\mathcal{F}(z) := \text{sign}\left(\sum_{i=1}^{n} x_i^* y_i K(Z, Z_i) + b^*\right), \tag{22}$$

donde K es un producto interno y $x^* = (x_1^*, \ldots, x_n^*)^T$ es la solución del problema QP con restricciones simples

$$\begin{array}{ll} \text{minimizar} & \mathcal{F}(x) = \frac{1}{2}x^T G x - \sum_{i=1}^{n} x_i \\ \text{sujeto a} & \sum_{i=1}^{n} y_i x_i = 0, \\ & 0 \leq x_j \leq C, j = 1, \ldots, n, \end{array} \tag{23}$$

donde las entradas de G se definen como $G_{ij} = y_i y_j K(z_i, z_j)$, $i, j = 1, 2, \ldots, n$ y C es un parámetro del algoritmo *SVM*.

Note que el problema planteado en (23) corresponde a un problema QP con una restricciones de caja y una sola restricción de igualdad lineal, para ver esto de una forma más clara, reescribimos el problema de la *SVM* como

$$\begin{array}{ll} \text{minimizar} & \mathcal{F}(x) = \frac{1}{2}x^T G x + (-1)^T x \\ \text{sujeto a} & y^T x = 0, \\ & 0 \preceq x \preceq C. \end{array} \tag{24}$$

En este punto es cuando se realiza la escogencia de un método de optimización para solucionar 23, en nuestro caso vamos a utilizar el método de proyección del gradiente que se presento en la sección anterior y el método de gradiente descendiente estocástico. Este método funciona bien en problemas con restricciones simples [12].

Note que el gradiente de $\mathcal{F}(x)$ es

$$\nabla \mathcal{F}(x) = Gx + (-1), \tag{25}$$

podemos observar que el computo de este gradiente no es costoso, puesto que no tiene operaciones con alta carga computacional.

8. Especificación y Modelización del Experimento

Para entender como se realiza el entrenamiento de la *SVM* con los diferentes métodos de gradiente, veamos algunos particularidades acerca de nuestro conjunto de datos

8.1. Especificaciones y Datos

Entrenaremos un modelo de *SVM* para la detección de frutas y vegetales mediante imágenes de estas. El conjunto de datos esta separado en 67692 datos de entrenamiento y 22688 para la comprobación, dentro de estas imágenes hay 131 clases de frutas y vegetales. Las imágenes tienen tamaño de 100x100 píxeles.

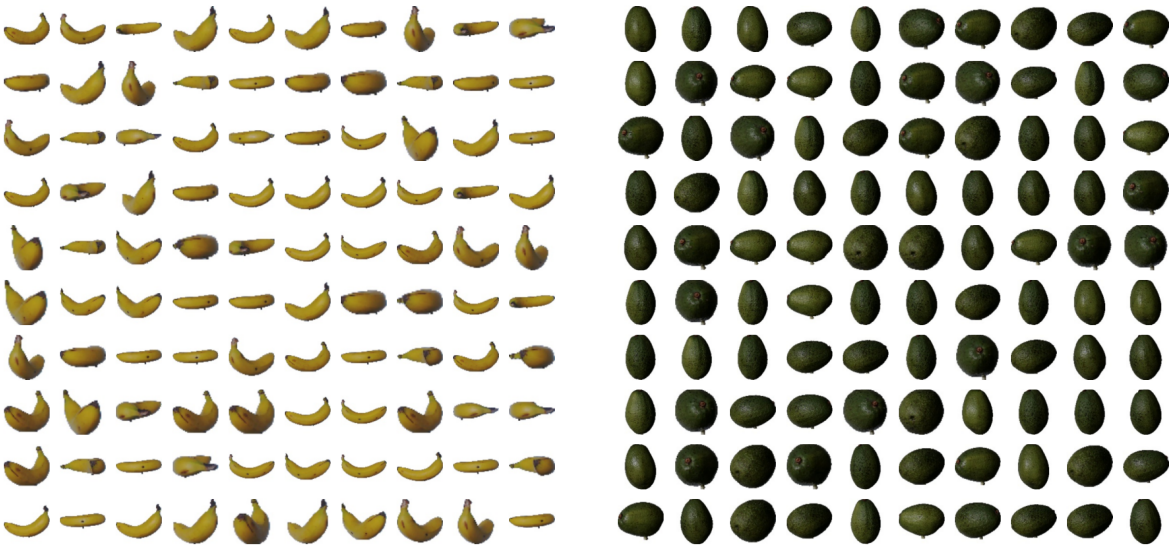

Figura 5: Muestra del tipo de imágenes pertenecientes al dataset.

El *dataset* que utilizamos nos facilita dos carpetas distintas con los datos de entrenamiento que corresponde a un 75 % y los de comprobación que corresponden al 25 % restante de los datos totales por cada tipo de fruta. Anexamos en (Github Repositorio) el conjunto de datos correspondiente y además el código en Python.
Las librerías que utilizamos, junto con una pequeña descripción, son las siguientes.

- Projgrad : Librería de Python para la optimización de gradientes proyectados.
- PyOptim: Librería de optimización numérica.
- SciKit-Learn: Librería para problemas de aprendizaje de máquina.

8.2. Modelización del Experimento

El problema de clasificación es un problema que se presenta en nuestro día a día. En nuestro problema de clasificación vamos a recibir como entrada conjuntos de imágenes, del conjunto de datos mencionado anteriormente, ya sea para entrenar o para comprobar, dichas imágenes las descomponemos a un mediante sus valores *RGB* (véase 6).

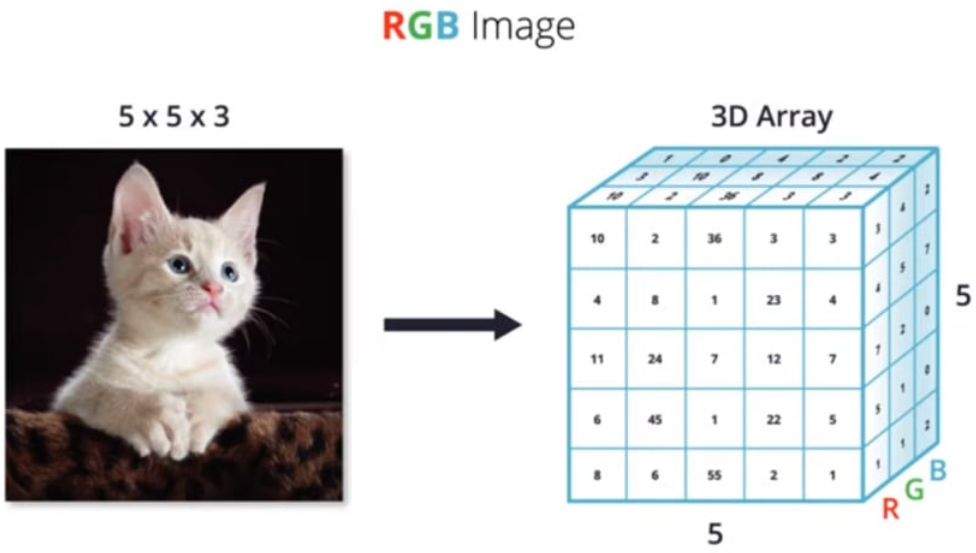

Figura 6: Apoyo gráfico explicativo del modelo RGB.

Primero que todo, se implementan funciones en *Python* que nos permitan elegir las frutas que deseamos y las guardamos en una matriz de tipo *Numpy* de 100x100 con 3 valores en su interior para los colores RGB, la aplanamos para que sea un arreglo que es nuestro vector de características. Luego de tener nuestras imágenes listas para ser manipuladas es apropiado reducir la dimensión de nuestro conjunto de datos, ya que tiene más de 100 dimensiones, para ello realizamos el análisis de componentes principales (PCA) [14], una técnica utilizada para reducir la dimensionalidad de un conjunto de datos mientras se conserva la mayor cantidad de información posible, donde los datos se re-proyectan en un espacio dimensional más bajo, en particular, encontrando una proyección que minimice el error cuadrático en la reconstrucción de los datos originales.

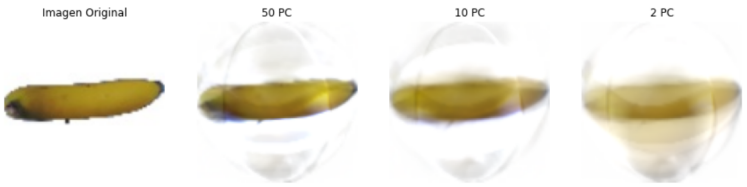

Figura 7: Efecto del PCA sobre una imagen

Así disminuimos la complejidad de nuestro modelo y también nos facilita visualizar nuestros datos en 2 o 3 dimensiones, dependiendo del PCA, como en las siguientes imágenes donde los círculos azules representan las bananas y las equis verdes representan los aguacates (Avocados).

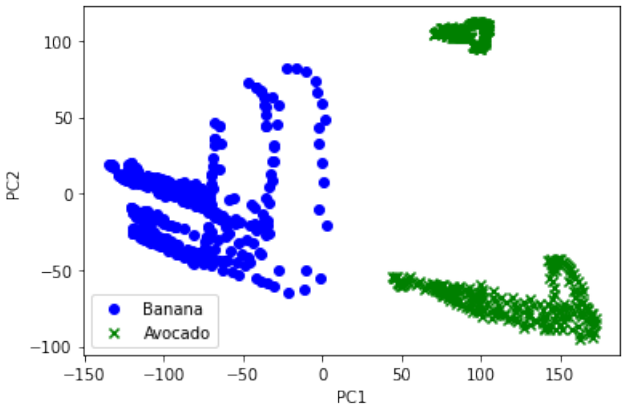

Figura 8: Gráfica ejemplo del mapeo de los objetos tras el PCA en dos dimensiones.

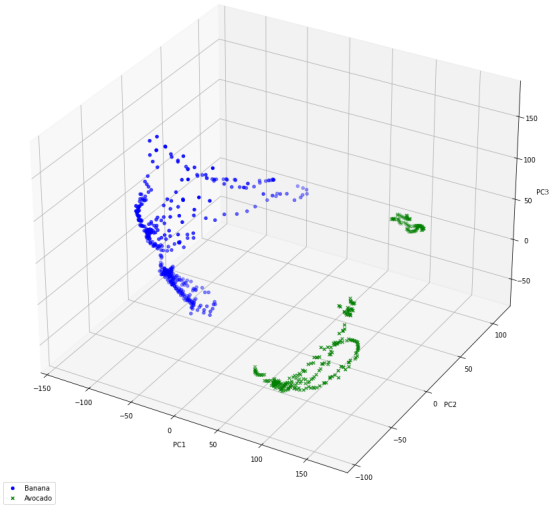

Figura 9: Gráfica ejemplo del mapeo de los objetos tras el PCA en tres dimensiones.

Como podemos ver, nuestros datos son separables, por lo que ahora revisaremos de acuerdo al grado del PCA qué tanta varianza capturan nuestros nuevos datos.

A continuación verificamos con diferentes grados, ya que si no capturamos un buen porcentaje de varianza, nuestro modelo difícilmente será capaz de generalizar el comportamiento de la función que distingue las frutas. Esto ocurre ya que nuestro nuevo conjunto de datos puede contener más ruido que información realmente útil [14].

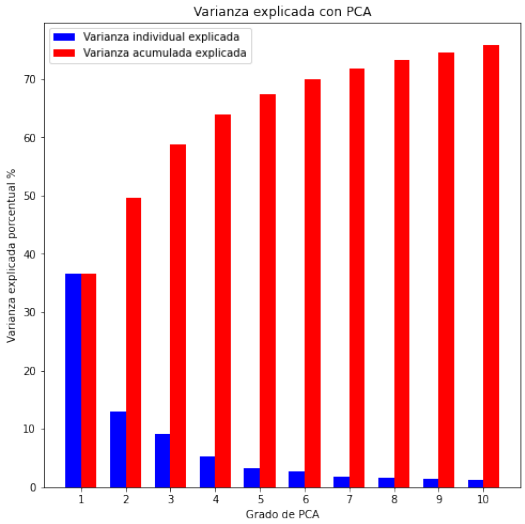

Figura 10: Gráfica comparativa de la varianza individual y acumulada.

Después de un PCA de grado 2 obtenemos que se explica más del 50 %, esta gráfica puede cambiar dependiendo del tipo de fruta que se tome, por ejemplo tomando Orange y Tangelo obtenemos que el crecimiento porcentual no es es tan alto, como se observa en la figura 11, esto puede deberse a que son frutas con características visuales muy similares.

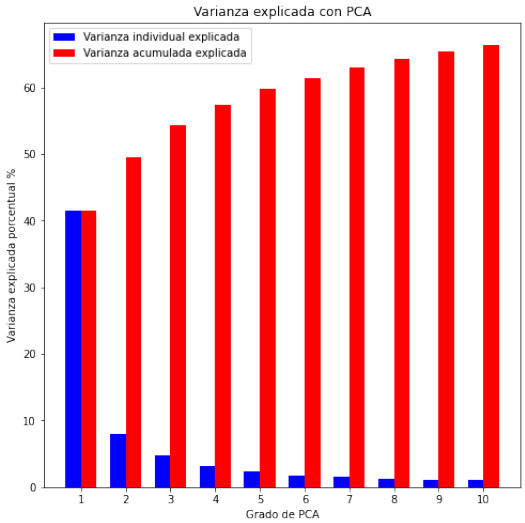

Figura 11: Gráfica comparativa de la varianza individual y acumulada para Orange y Tangelo.

9. Conclusiones

Después de entrenar el modelo de SVM con el método de gradiente descendiente de la librería *cvxopt* [5], obtenemos el hiper-plano separador mostrado a continuación junto con los puntos usados para el entrenamiento:

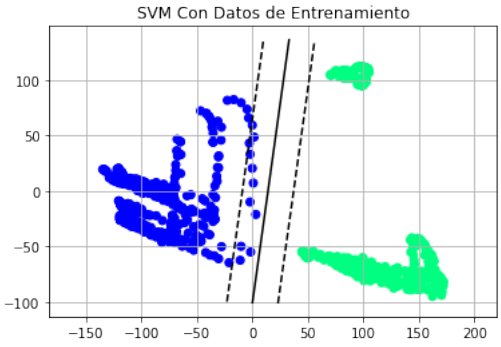

Figura 12: Hiper-plano separador con método de gradiente descendiente en el entrenamiento.

Ahora veamos como se ve la SVM con los datos de prueba:

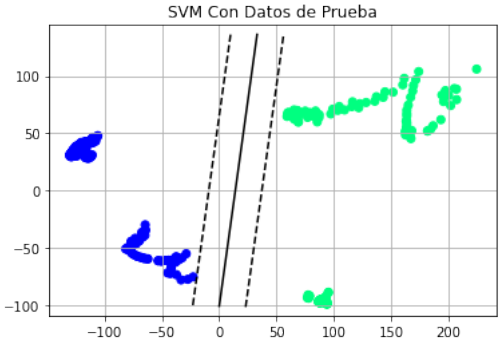

Figura 13: Hiper-plano separador con método de gradiente descendiente en la prueba.

Como podemos ver, la SVM separa los datos sin error, esto se debe a la fuerte agrupación de los datos de un mismo tipo. Esto puede confirmarse mediante la matriz de confusión asociada.

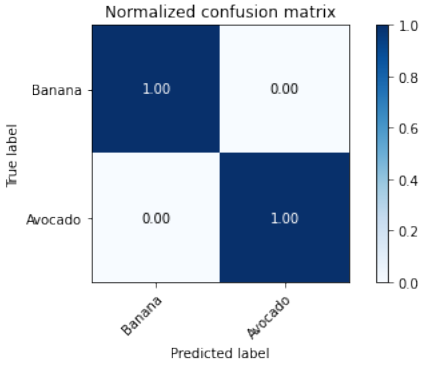

Figura 14: Matriz de confusión de la SVM entrenada con SGD.

Finalmente vemos la curva ROC:

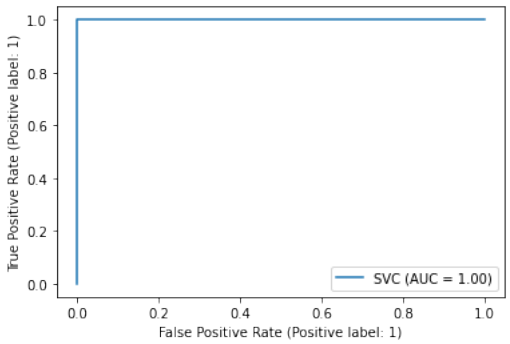

Figura 15: Curva ROC de la SVM entrenada con SGD.

Observamos que la curva ROC tiene una forma similar a un cuadrado, esto es debido a que no se presentan falsos positivos, como se vio en la matriz de confusión.

Para la SVM entrenada con el método de proyección del gradiente notamos que el proceso de entrenamiento es mas rápido que el de la entrenada con gradiente descendiente estocástico, sin embargo resulta difícil en la práctica encontrar las restricciones que vuelvan la región factible del problema de optimización asociado a la SVM una caja, es decir, restricciones formadas con desigualdades simples.

```
          PGD
Required tolerance achieved!
Convergence in 10 iterations
Function value = 42.03228121230303
Difference in function values = 2.448432759649677e-06
Difference in argument = 1.5247668655744236e-06
--- 0.043305158615112305 seconds ---
```

Figura 16: Tiempo y convergencia del método de proyección del gradiente.

```
-------- SVM Sklearn ---------
--- 13.176594257354736 seconds ---
```

Figura 17: Tiempo del método de gradiente descendiente estocástico.

Para profundizar más en las comparaciones de estos métodos sobre las SVM, concluimos que para alcanzar una mayor comprensión sobre la aplicación de estos métodos en el entrenamiento, es menester realizar análisis comparativos, en específico sobre problemas no separables. Otra forma de mejorar el trabajo realizado, es mediante un algoritmo que permita encontrar las condiciones de la región factible, es decir, las constantes que definen la caja sobre la que se resuelve el problema de optimización.

Referencias

[1] E. p. L. D. Seta, Google alcanza el billón de páginas indexadas.
URL https://dosideas.com/noticias/actualidad/146-google-alcanza-el-trillon-de-paginas-indexadas#:~:text=En20una20entrada20reciente20en,Un20billC3B3n203D2012C0002C0002C0002C000).

[2] M. Lahtela, P. P. Kaplan, Es (1966).
URL https://aws.amazon.com/es/rds/aurora/faqs/

[3] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics 5 (4) (1943) 115–133.

[4] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (3) (1995) 273–297.

[5] Cvxopt, https://cvxopt.org/index.html.

[6] T. Fawcett, An introduction to roc analysis, Pattern recognition letters 27 (8) (2006) 861–874.

[7] J. Nocedal, S. J. Wright, Numerical optimization, Springer, 1999.

[8] S. P. Boyd, Convex Optimization, Cambridge University Press, 2009, pp. 152–160.

[9] A. J. Laub, Notes on moore-penrose pseudoinverse (2012).

[10] S. Khirirat, H. R. Feyzmahdavian, M. Johansson, Mini-batch gradient descent: Faster convergence under data sparsity, in: 2017 IEEE 56th Annual Conference on Decision and Control (CDC), IEEE, 2017, pp. 2880–2887.

[11] O. Shamir, T. Zhang, Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes, in: International conference on machine learning, PMLR, 2013, pp. 71–79.

[12] T. Serafini, G. Zanghirati, L. Zanni, Gradient projection methods for quadratic programs and applications in training support vector machines, Optimization Methods and Software 20 (2-3) (2005) 353–378.

[13] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.

[14] H. Abdi, L. J. Williams, Principal component analysis, Wiley interdisciplinary reviews: computational statistics 2 (4) (2010) 433–459.