DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Final

Support Vector Machines

Reconocimiento de patrones

Integrante	LU	Correo electrónico
Rodrigo Oscar Kapobel	695/12	rokapobel35@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

En este trabajo práctico se implementa una solución para $Support\ Vector\ Machines$ (SVMs) [1]. La teoría que ocupa las soluciones de vectores de soporte es muy amplia, bien estudiada y fundamentada.

Intentaré en este informe dar una aproximación general de lo que puede encontrarse en la literatura a modo de resumen mencionando algunos de los métodos utilizados y la matemática de fondo que los sustentan.

Para aclarar, la implementación del algorítmo para este trabajo final no fué compleja de llevar a cabo, pero si de encontrar y comprender en ciertos aspectos. En casi todas las bibliografias leídas y para cada función a minimizar o maximizar, sea el problema primario o el dual respectivamente, el método se introducía de forma matemática solo mencionando mediante que método podría resolverse.

Existen diferentes implementaciones pero no todas son fáciles de realizar. La más común es resolver el problema dual mediante programación cuadrática, pero esto requiere primero un mapeo del problema dual a uno de programación cuadrática y luego la implementación de un algorítmo que haga optimización cuadrática. Esto último no es sencillo de realizar. En general lo más sencillo en este punto es utilizar una librería que resuelva este tipo de algorítmos como puede ser scikit-learn para python aunque la idea siempre fue dar una implementación sin utilizar una libreria.

En base a esto mismo intentaré abarcar de manera general cuales son estos métodos diferenciandolos claramente, pero solo daré la implementación de uno de ellos.

1.1. SVMs

En Machine Learning, un SVM es un algorítmo supervisado que se utiliza para clasificación y análisis de regresion.

Dado un data set de entrenamiento, donde cada dato puede verse como un punto en el espacio euclideano y está marcado como perteneciente a una de dos clases (en este informe simplificaré a dos clases, pero puede extenderse a más clases). Cada punto puede verse como un vector p-dimensional. El algorítmo construye un modelo que asigna nuevos datos a cada clase. Este modelo es un hiperplano de dimensión p-1 que separa con mayor margen ambas clases y sin errores de clasificación. En casos donde el data set no admite un hiperplano sin errores de clasificación, el algorítmo en su versión más simple no podrá encontrar el hiperplano. Para solventar este problema, se realizan ciertas modificaciones al algorítmo que permite encontrar un hiperplano separador, aceptando un número razonable de errores de clasificación.

Cuando el set de datos no es separable bajo ningún concepto de manera lienal, por ejemplo, para datos circulares, se utilizan los llamados trucos de kernel ($kernel\ tricks$) que transforman el problema en uno de mayor dimensión para poder realizar una separación lineal de los datos.

Cuando el set de datos de entrenamiento no está categorizado, se suele recurrir a $Support\ Vector\ Clustering$ [2], que es un algorítmo no supervisado creado por Hava Siegelmann y Vladimir Vapnik, que no será estudiado en este informe pero es ampliamente utilizado en la industria.

Como es costumbre en este tipo de problemas, se plantea una función a maximizar mediante Multiplicadores de Lagrange que es respaldada por una base teórica matemática muy fuerte y que bajo ciertas propiedades permite implementar diferentes soluciones de SVM, algunas mejores que otras en determinadas situaciones.

1.2. SVM Lineal

Dado un data set $(\vec{x}_1, y_1)$ ... $(\vec{x}_n, y_n)$ donde y_i es 1 o -1, cada uno indicando a que clase pertenece $\vec{x}_i$ y $\vec{x}_i$ es un vector p-dimensional buscamos el hiperplano de mayor margen separador de ambas clases. Es decir, que maximiza la distancia entre los puntos de cada clase.

Cualquier hiperplano puede escribirse como como un set de puntos $\vec{x}$ que satisface

$$\vec{w} \times \vec{x} + b = 0$$

donde $\vec{w}$ es el vector normal al hiperplano y b es el corrimiento respecto al origen del hiperplano (este valor se conoce como $bias$ y cuando es cero el hiperplano será aquel que pase por el origen de coordenadas)

1.3. Hard margin

Si el data set de entrenamiento es linealmente separable podemos encontrar dos hiperplanos que separan ambas clases con el mayor margen. Estos hiperplanos pueden ser definidos como

$$\vec{w} \times \vec{x} + b = 1 \text{ (clase 1)}$$

$$\vec{w} \times \vec{x} + b = -1 \text{ (clase -1)}$$

Geometricamente la distancia entre estos dos hiperplanos será $\frac{2}{||\vec{w}||}$ Por lo que para maximizar la distancia de los planos, tenemos que minimizar la norma de $\vec{w}$.

Como queremos evitar que ningún punto caiga dentro del margen de los hiperplanos, tenemos que escribir las siguientes restricciones

$$\vec{w} \times \vec{x} + b \geq 1 \text{ (si } y = 1)$$

$$\vec{w} \times \vec{x} + b \leq -1 \text{ (si } y = -1)$$

Esto puede reescribirse como

$$\forall\, i \quad g(\vec{x_i}) = y_i(\vec{w} \times \vec{x_i} + b) \geq 1 \ (1)$$

1.4. Problema Primario

El problema de optimización se reduce a minimizar una expresion de la forma

$$f(\vec{w}) = \tfrac{1}{2}||\vec{w}||^2 \text{ sujeto a } g(\vec{x_i})$$

Notar que los multiplicadores de Lagrange estan definidos para restricciones con igualdad. Pero el método también puede aplicarse con restricciones basadas en desigualdades.

La clase de $\vec{x}$ se determina como el signo de $\vec{w} \times \vec{x_i} + b$

Como es un problema de optimización con restricciones puede resolverse con multiplicadores de Lagrange. Al ser cuadrático, la superficie es un paraboloide y por lo tanto el problema es convexo y tiene un solo mínimo global. También puede probarse que sus restricciones dan una región factible convexa. Por estas razones es que puede resolverse aplicando $programación\ cuadrática$ [5].

El Lagrangiano de este problema será

$$L_p(\vec{w}, b, \vec{\lambda}) = f(\vec{w}) - \sum_{i=1}^{n} \lambda_i(y_i(\vec{w} \times \vec{x_i} + b))$$

donde $\vec{\lambda} = (\lambda_1, .., \lambda_n) \geq 0$. A este Lagrangiano se lo denomina problema primario (por eso lo notamos como L_p). Ya que intenta minimizar una función convexa para encontrar los valores de $\vec{w}$ y b.

Una consecuencia importante de esta descripción geométrica es que el hiperplano de margen máximo está completamente determinado por aquellos $\vec{x_i}$ que se encuentran más cercanos a él. Estos $\vec{x_i}$ se denominan vectores de soporte.

Para minimizar L_p hay que obtener el gradiente ∇L e igualar a cero.

Para ello debemos calcular las derivadas parciales de $\vec{w}$ y b

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^{n} \lambda_i y_i \vec{x_i} = 0$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \lambda_i y_i = 0$$

1.5. Problema Dual

Ahora podemos reemplazar en la formula original las derivadas parciales encontradas previamente. De esta manera se obtiene el siguiente Lagrangiano que es utilizado para resolver el $problema\ dual$ [3].

$$G(\vec{\lambda}) = L_d(\vec{w}, b, \vec{\lambda}) = \sum_{i=1}^{n} \lambda_i - \tfrac{1}{2} \sum_{i,j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \vec{x_i}, \vec{x_j} \rangle$$

Donde $\langle \vec{x}_i, \vec{x}_j \rangle$ representa el producto interno.

El producto interno es una función definida en un espacio vectorial que cumple una serie de propiedades [6].

Notar que G es cuadrático en $\vec{\lambda}$ por lo que es posible resolver este problema mediante $programación\ cuadrática$ al igual que el problema primario. Dado que el Hessiano de G es semidefinido positivo, G es concavo.

El problema dual consiste entonces en maximizar G sujeto a

$$\vec{\lambda} \geq 0$$

$$\sum_{i=1}^{n} \lambda_i y_i = 0$$

Esto es posible ya que la función dual de Lagrange G es una cota inferior del valor óptimo p^* del problema primario, donde

$$p^* = inf_{\vec{w},b}\{f(\vec{w}) - \sum_{i=1}^{n} \lambda_i (y_i(\vec{w} \times \vec{x_i} + b))\}$$

La mejor cota inferior, a la que notaremos como d^* cumplirá que $d^* \leq p^*$. Condición que es válida aún cuando el problema primario no es convexo.

A esta propiedad se la llama $dualidad\ debil$. A la diferencia $p^* - d^*$ se la llama $brecha\ de\ dualidad\ óptima$ y es siempre positiva. Si esta brecha es nula, entonces $d^* = p^*$. En este caso decimos que hay $dualidad\ fuerte$.

A las condiciones bajo las cuales existe dualidad fuerte se las denomina $constraints\ qualifications$. Son condiciones suficientes para la optimalidad, pero no necesarias. Una condición suficiente conocida es que hay dualidad fuerte cuando el problema primario es convexo y valen las condiciones de $Slater$ [7] en las restricciones.

Sabemos que nuestro problema primario es convexo (restricciones incluidas) y se cumplen las condiciones de $Slater$ ya que puede probarse que las restricciones son $funciones\ afines$ [8].

Además, las restricciones son diferenciables, por lo que para el óptimo del problema primario $\vec{w}^*$ y b^* valen las condiciones KKT (Karush-Kuhn-Tucker [9]) y luego el gradiente de $L_p(\vec{w}, b, \vec{\lambda}^*)$ evaluado en $\vec{w}^*$ ($\vec{\lambda}^*$ valor para el cual el problema dual alcanza el óptimo) es nulo. Por lo cual el valor obtenido es un óptimo local y más aún global por ser convexo el problema primario.

Las condiciones KKT son necesarias y suficientes (siempre que se cumplan las constraints qualifications), por lo que encontrar una solución para ellas es encontrar una solución para el problema primario.

Para probar que todo lo mencionado es válido sería necesario dar las definiciones a todas las propiedades mencionadas. Esto es vital para comprender el problema en su magnitud. Dado que esto conlleva una ardua tarea se dejan referencias al lector que lo llevaran a todas las definiciones.

1.6. Soft margin

En el caso de que los datos no sean linealmente separables, se introduce una funcion de perdida llamada $Hinge\ Loss$ [4]

$$max(0, 1 - y_i(\ \vec{w} \times \vec{x_i} + b))$$

La función es cero si (1) se satisface, es decir, si $\vec{x}_i$ cae del lado correcto del margen. En caso contrario, el valor de la función es proporcional a la distancia del punto al margen.

Esta función en principio nos permitirá aceptar un número de errores de clasificación en pos de obtener una separación lineal de los datos. Es decir, seguimos encontrando un hiperplano separador pero ahora con un margen de error.

A partir de esta función puede plantearse un Lagrangiano, pero la función de $Hinge$ no es diferenciable. Esto puede solventarse reescribiendo la misma de la siguiente forma

$$L_p(\vec{w}, b, \vec{\xi}) = f(\vec{w}) - C \times \sum_{i=1}^{n} \xi_i$$

sujeto a

$$\forall\ i \quad y_i(\vec{w} \times \vec{x}_i + b)$$

$$\forall\ i \quad \xi_i \geq 0$$

Donde ξ_i representa el error cometido y se conocen como $Slack\ variables$. Representan por cuanto se viola el margen establecido para cada dato.

La variable C es un hiperparámetro de regularización que controla si esta violación de margenes será muy grande o muy pequeña. Cuanto más pequeño su valor, no se tendrá en cuenta el error y por lo tanto el problema será como en $hard\ margin$. Cuanto más grande su valor, el problema será como en $soft\ margin$.

Para este Lagrangiano tambien puede plantearse el problema dual y la única diferencia con hard margin será que cada λ_i estará acotado superiormente por C.

1.7. Método del subgradiente

Dada la dificultad para implementar un algoritmo de optimización de $programación\ cuadrática$ y además realizar el mapeo del problema dual al mismo, se decidió ir por una solución basada en el algoritmo $Pegasos$ [12]. Esta idea combina $Stochastic\ Subradient\ Descent$ [11] con la idea de $subgradiente$ [10].

Dado que la implementación de $Pegasos$ parte del problema primario pero con algunas diferencias del planteado en este informe, el algorítmo final se basa en las diapositivas de Support Vector Machines: Training with Stochastic Gradient Descent de la universidad de Utah [13].

El método de subgradiente es un algorítmo de optimización convexa y además un $proceso\ estocástico$ [14] ya que elige de manera randomizada vectores del dataset para realizar un avance en la dirección del subgradiente del mismo.

Como su nombre lo índica, lo que se calcula en este método no es un gradiente sino, un subgradiente, ya que la función de $Hinge$ no es diferenciable.

El subgradiente queda definido como

$$\frac{\partial L_i^{hinge}}{\partial \vec{w}} = \begin{cases} \vec{w} & : y_i(\vec{w} \times \vec{x}_i + b) \geq 1 \\ \vec{w} - y_i\vec{x}_i & : y_i(\vec{w} \times \vec{x}_i + b) < 1 \end{cases}$$

$$\frac{\partial L_i^{hinge}}{\partial b} = \begin{cases} 0 & : y_i(\vec{w} \times \vec{x}_i + b) \geq 1 \\ -y_i & : y_i(\vec{w} \times \vec{x}_i + b) < 1 \end{cases}$$

Con esto, podremos realizar un algorítmo de descenso utilizando el subgradiente de la siguiente manera.

Require: $[y_1..y_n]$, $[\vec{x}_1..\vec{x}_n]$, lr, C
Ensure: $something$?
  $\vec{w} = \vec{0}$
  $b = 0$
  $Y = [y_1..y_n]$
  $X = [\vec{x}_1..\vec{x}_n]$
  $data = zip(X, Y)$
  $it = 0$
  while $it < maxIter$ **do**
    $shuffle(data)$ ▷ Es importante mezclar los datos para cumplir con la aleatoriedad del proceso
    $(\vec{x}, y) = choice(data)$ ▷ Elegir de manera aleatoria un vector
    $b = y - \vec{w} \times \vec{x}$
    for $(\vec{x}, y)\ in\ data$ **do**
      $v = y \times (\vec{w} \times \vec{x} + b)$
      $\vec{w} -= \frac{lr}{it+1} \times C \times (v < 1\ ?\ \vec{w} - \vec{x} \times y : \vec{w})$
      $b -= \frac{lr}{it+1} \times C \times (v < 1\ ?\ -y : 0)$
    end for
    $it += 1$
  end while

Donde

- lr es el learning rate utilizado para el descenso y decrece en cada iteracion.

- C es el hiperparámetro de regularización previamente definido.
- El bias es parte del hiperplano separador que encuentra SVM y es necesario calcularlo para que el descenso se haga correctamente. Puede realizarse una reescritura del vector normal al hiperplano $\vec{w'} = [b, \vec{w}]$ y los datos $\vec{x'}_i = [1, \vec{x}_i]$

  En [12] pag. 15 incorporating the bias term se explica el bias puede incluirse como parte del vector normal y ser regularizado como parte del mismo subgradiente planteado para el algorítmo original aunque esto implica que se está resolviendo un problema de optimización relativamente diferente o no hacerlo y calcular el subgradiente para el bias por separado como se ha elegido para este trabajo práctico.

  Una razon para incluir el bias según se explica es la siguiente:

  «El término de sesgo a menudo juega un papel crucial cuando la distribución de las etiquetas es desigual, como suele ser el caso en las aplicaciones de procesamiento de texto donde los ejemplos negativos superan ampliamente a los positivos.»

  Para la implementación realizada, no incluir el bias parecía no ser una opción viable ya que los resultados obtenidos no fueron los deseados.

2. Implementación

El test se ejecuta utilizando datos sintéticos generados mediante una distribución de medias aleatorias para cada clase y con misma covarianza, siendo esta la identidad en ambos casos.

Debe ingresar el siguiente comando en el terminal desde el directorio TP_Final:

```
python SVMTest.py
```

para más opciones:

```
python SVMTest.py -h
```

El algorítmo se realizó para clasificar 2 clases. El número de iteraciones, el learning rate, valores para el hiperparámetro de regularización, y si desea utilizarse o no datos nuevos para clasificar luego del entrenamiento se pueden espeficiar ingresando los comandos indicados por help.

```
  -lr LEARNINGRATE      Learning rate of the gradient descent.
  -c_list REGLISTPARAMC [REGLISTPARAMC ...]
                        A list of regularization parameter
                        for the slack variables:
                        too small == hard margin |
                        too large == soft margin.
  -i MAXNUMITER         Max number of iterations for the loss
  function of the gradient descent.
  -e TESTUSINGTRAININGDATA
                        1: Test the classifier using a
                        different data set.
                        0: Test using the training data set.
```

Por ejemplo,

python SVMTest.py -e 0 -c_list 0.001 0.25 2 -lr 0.1 -i 100

Ejecuta el test, entrenando para cada valor especificado en c_list que en este caso serán 0.001, 0.25 y 2.

3. Casos de estudio

3.1. Sobre los gráficos y tests

Dado que el método es un proceso estocástico, la función de costo (que es la suma del valor de la función de perdida $Hinge$ para cada dato $\vec{x}_i$) no desciende suavemente en cada iteración ya que el algorítmo no se calcula utilizando un gradiente, si no, un sugradiente ($Hinge$ no es derivable).

Esto provoca picos en los gráficos que hacen que no ofrezca demasiada información sobre la disminución del costo.

Debido al tiempo estimado para el desarrollo del informe, no se calcula el Accuracy de clasificación para los datos de entrenamiento ni de test por lo que no se realizan gráficos comparativos de los mismos.

Tampoco se realizan gráficos comparativos analizando la convergencia cuando se realizan retoques al learning rate o a la cantidad de iteraciones, pero es interesante notar que el learning rate complementa al hiperparámetro C y que el algorítmo en general siempre converge rápido para un número razonable de datos de entrenamiento y más hablando de que solo se requiere separar dos clases. Sería muy interesante ver como escala el algorítmo cuando se trata de set de datos mayores y con más de dos clases.

3.2. Influencia del hiperparámetro C

Para mostrar la influencia del hiperparámetro C se ha realizado un pequeño test donde se entrena el algorítmo y se testea con el mismo set de datos.

Se podrá ver que cuanto menos separación haya entre las clases, el valor de C deberá ser más grande y cuanto más separación, alcanzará con un valor pequeño, ya que un valor grande permitirá un soft margin cuando las clases son perfectamente separables.

En ambos casos, un valor muy pequeño de C conllevará a no encontrar un hiperplano separador. Se estima que esto se debe a que se le da demasiada poca importancia a la restricción que impone el subgradiente y el proceso de descenso acaba no siendo adecuado.

Para el primer test se utilizan los siguientes datos

- c_list: 0.001 0.15 2
- lr: 1
- i: 100
- $means$: [[0, 0], [0, 4]]
- $covariance$: identity

Figura 1: Clasificación para $C = 0{,}001$

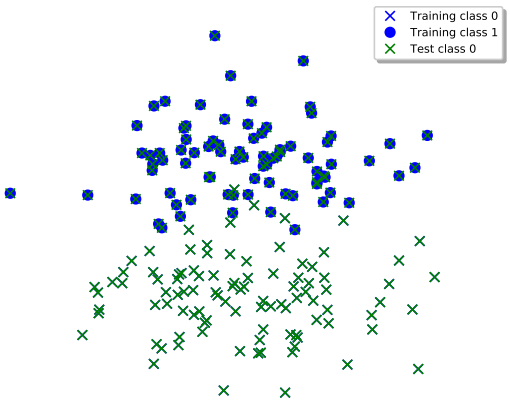


Figura 2: Clasificación para $C = 0{,}15$

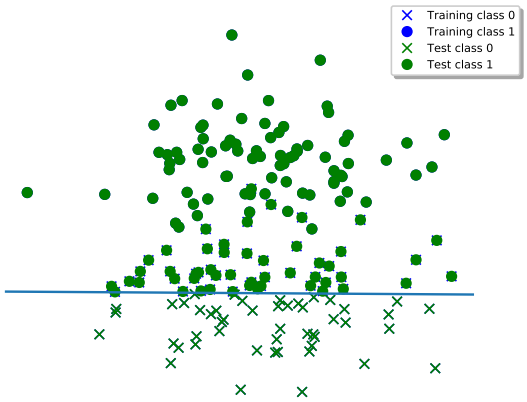
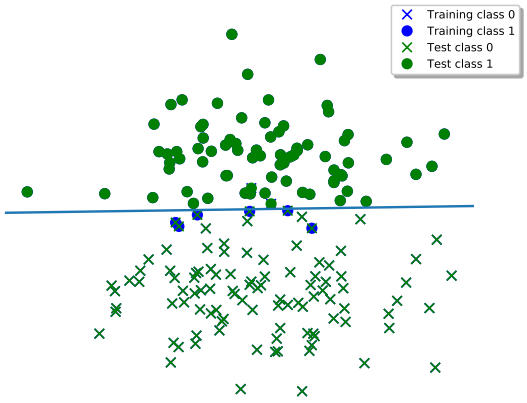


Figura 3: Clasificación para $C = 2$



- c_list: 0.001 0.15 2
- lr: 1
- i: 100
- $means$: [[0, 0], [0, 8]]
- $covariance$: identity

Figura 4: Clasificación para $C = 0{,}001$

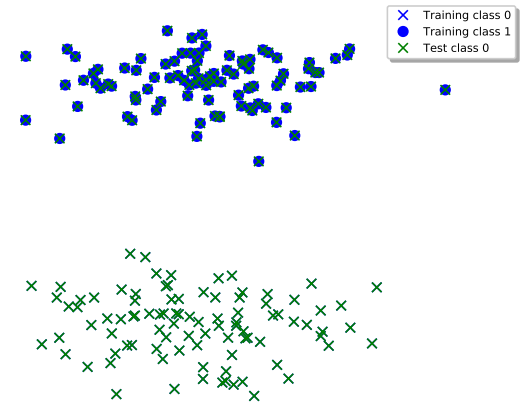


Figura 5: Clasificación para $C = 0{,}15$

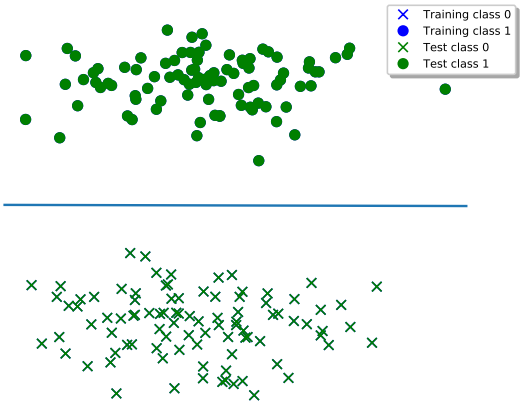
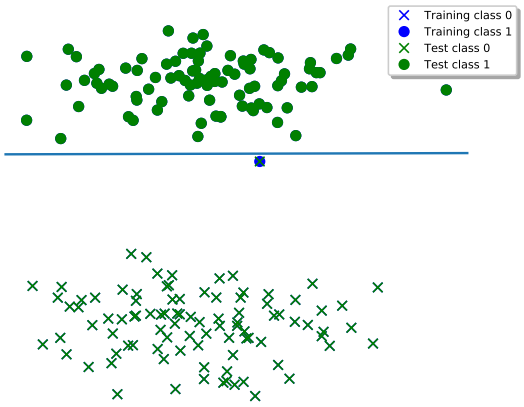


Figura 6: Clasificación para $C = 2$



4. Conclusiones

1. El método del subgradiente resulta una opción sencilla de implementar y probar. La función de costo, al contrario, no ofrece demasiados detalles sobre el avance del descenso del gradiente.

2. El hiperparámetro C resulta indispensable a la hora de buscar un $hard\ margin$ o un $soft\ margin$ y tiene importancia cuando las clases son perfectamente separables por la recta de mayor margen y cuando no lo son.

3. La matemática que cubre toda la teoría general de SVM es muy amplia y compleja. Requirió bastante tiempo y esfuerzo de mi parte comprender la mayor parte de los aspectos y hubo otros que tuve que asumir como válidos por falta de base teorica. No quise encarar el trabajo final implementando directamente el algorítmo realizado sin primero dar un paneo general de lo que se conoce naturalmente como un SVM.

4. La teoría que fundamenta los $Kernel\ Tricks$ es mucho más compleja. En este informe no se dan demasiados detalles al respecto pero hay mucha literatura sobre ellos. Implementar un $Kernel$ quedaba fuera de mi alcance para este informe.

5. De los métodos aprendidos en la materia me ha parecido uno de los mas interesantes por la variedad de formas de encararlo y algorítmos que ofrece igualmente válidos. Aún me quedan muchas areas por aprender para comprender detalles más específicos sobre SVM.

Referencias

[1] Support Vector Machines
`https://en.wikipedia.org/wiki/Support-vector_machine`

[2] Support Vector Clustering

Ben-Hur, Asa; Horn, David; Siegelmann, Hava; and Vapnik, Vladimir; "Support vector clustering"(2001) Journal of Machine Learning Research, 2: 125–137 Support Vector Machines

[3] Dual Problem
`https://en.wikipedia.org/wiki/Duality_(optimization)`

[4] Hinge Loss
`https://en.wikipedia.org/wiki/Hinge_loss`

[5] Quadratic Programming
`https://en.wikipedia.org/wiki/Quadratic_programming`

[6] Dot product properties
`https://en.wikipedia.org/wiki/Dot_productProperties`

[7] Slater's conditions
`https://en.wikipedia.org/wiki/Slater's_condition`

[8] Affine Transformation
`https://en.wikipedia.org/wiki/Affine_transformation`

[9] Karush Kuhn Tucker Conditions
`https://en.wikipedia.org/wiki/Karush-Kuhn-Tucker_conditions`

[10] Subgradient method
`https://en.wikipedia.org/wiki/Subgradient_method`

[11] Stochastic Subradient Descent
`https://en.wikipedia.org/wiki/Stochastic_gradient_descent`

[12] Pegasos algorithm: An Stochastic gradient descent method for SVM
https://www.cs.huji.ac.il/ shais/papers/ShalevSiSrCo10.pdf

[13] Support Vector Machines: Training with Stochastic Gradient Descent
https://svivek.com/teaching/machine-learning/fall2018/slides/svm/svm-sgd.pdf

[14] Stochastic Process
https://en.wikipedia.org/wiki/Stochastic_process