PLANTILLA OFICIAL PARA LA PRESENTACIÓN DE TRABAJOS
II CONVENCIÓN CIENTÍFICA INTERNACIONAL
"II CCI UCLV 2019"

DEL 23 AL 30 DE JUNIO DEL 2019.
CAYOS DE VILLA CLARA. CUBA.

II CONFERENCIA INTERNACIONAL DE PROCESAMIENTO DE LA INFORMACION (CIPI 2019)

Scalable Large-Margin Distance Metric Learning usando Gradiente Descendente en Apache Spark

Scalable Large-Margin Distance Metric Learning using Descendant Gradient in Apache Spark

Christian Ariel Isac Palma[1], Carlos Morell[2]

1- Christian Ariel Isac Palma. Datys, Cuba. E-mail: crisitian.isac@datys.cu

2- Carlos Morell. UCLV, Cuba. E-mail: cmorellp@uclv.edu.cu

Resumen:

El aprendizaje de una función de distancia (DML) es una técnica efectiva para aprender una función de distancia basada en los datos y permite mejorar el rendimiento de algoritmos de aprendizaje automático ante problemas de clasificación, regresión, agrupamiento entre otros. La mayoría de los algoritmos empleados en esta área necesitan aprender una matriz de Mahalanobis, que es semidefinida positiva y escala cuadráticamente con el número de características de los datos de entrada. Además, si sumamos a esto el volumen creciente de información en problemas reales, el costo computacional durante la etapa de aprendizaje es muy alto. En este trabajo, aprovechando el poder de la computación distribuida y específicamente las bondades de Apache Spark en problemas de aprendizaje automatizado se propone una variante distribuida del algoritmo LMDML-A que logra acelerar el proceso de aprendizaje y que puede ser utilizado en problemas con grandes volúmenes de datos.

Abstract:

The learning of a distance function (DML) is an effective technique to learn a distance function based on the data and allows to improve the performance of algorithms of automatic learning for problems of classification, regression, grouping among others. Most of the algorithms used in this area need to learn a Mahalanobis matrix, which is positive semi-definite and scales quadratically with the number of characteristics of the input data. In addition, if we add to this the increasing volume of information in real problems, the computational cost during the learning stage is very high. In this work, taking advantage of the power of distributed computing and specifically the benefits of Apache Spark in automated learning problems, we propose a distributed variant of the LMDML-A algorithm that accelerates the learning process and can be used in problems with large volumes of data.

Palabras Clave:
Aprendizaje de distiancia

Keywords:
Metric learning

1. Introducción

La selección de una medida de distancia apropiada es fundamental para muchos algoritmos de aprendizaje automático, tales como agrupamiento, *k* vecinos más cercanos (k-NN), entre otros. La selección de esta medida puede dictar el éxito o el fracaso del algoritmo. El aprendizaje de una función de distancia (DML), consiste en ajustar una función de distancia usando la información contenida en los datos. Una buena medida de distancia debe mantener más cercanos entre sí los ejemplos de la misma clase, mientras que ejemplos de clases distintas deben estar más alejados.

La mayoría de los estudios se enfocan en aprender una distancia de Mahalanobis, debido a sus diversos usos en muchas aplicaciones reales. La distancia de Mahalanobis es

parametrizada por una matriz simétrica, semidefinida positiva (PSD) $M \in R^{DxD}$ donde la distancia entre dos ejemplos $u,v \in R^{D}$es calculada por $d_{M=\sqrt{(u-v)^T M(u-v)}}$.

Recientemente, la eficiencia computacional necesaria para aprender una métrica de distancia ha sido sustancialmente mejorada. Sin embargo, este tipo de distancia tiene grandes retos, especialmente en grandes volúmenes de información donde todavía persisten problemas de escalabilidad.

El primer problema de escalabilidad está relacionado con el número de características, ya que se requiere estimar una matriz *DxD*, donde *D* es el número de características. Esto es un reto para problemas que involucran miles de características, ya que el rendimiento del algoritmo se degrada según aumenta este número.

El segundo problema viene dado por la restricción de que la matriz de Mahalanobis sea semidefinida positiva, lo que requiere en la mayoría de las propuestas existentes una complejidad computacional de $O(D^3)$ para hacer una proyección dentro del cono PSD.

Otro problema de escalabilidad está relacionado con el número de ejemplos de entrenamiento y considerando el creciente aumento de datos, la complejidad computacional es muy crítica. Una solución puede ser usar algoritmos de aprendizaje en línea, particularmente Gradiente Descendiente Estocástico (SGD) y alguna estrategia de paralelización. En (Nguyen, Morell, & Bats, 2019) se propone una estrategia basada en SGD en la cual cada iteración requiere menos poder computacional para mantener la solución dentro del cono PSD. En el mismo se usa una función de pérdida para aprender la distancia de Mahalanobis y se reduce el número de modificaciones y proyecciones.

La principal contribución de este trabajo es proponer una variante distribuida de este algoritmo que pueda ser aplicada a grandes volúmenes de datos, con un número discreto de características, logrando buenos niveles de escalabilidad de manera tal que se logre acelerar el proceso de aprendizaje de la función de distancia.

1.1. Trabajo relacionado

Desde que fue formulado el problema de aprender una función de distancia como un problema de optimización (Xing, Jordan, Russell, & Ng, 2002), han aparecido muchos algoritmos en la literatura desarrollados para lograr diferentes objetivos. Entre ellos podemos mencionar NCA (Goldberger, Roweis, Hinton, & Salakhutdinov, 2005), MCML (Roweis, Globerson, & T., 2006), ITML (Davis, Kulis, Jain, Sra, & Dhillon, 2007), LMNN (Saul, Weinberger, & K., 2009) y DMLMJ (Nguyen, Morell, & Baets, 2017). Sin embargo, la utilización de estos algoritmos es un reto cuando el número de instancias de entrenamiento es grande o la dimensionalidad es alta.

Otros tratan de acelerar el proceso de entrenamiento restringiendo que la matriz aprendida sea una matriz diagonal; logrando disminuir la complejidad computacional, pero se pierde la posible correlación entre las características de las instancias.

Otra área de investigación se centra en aprender una matriz de rango bajo. Desafortunadamente, muchas propuestas en esta dirección caen en un problema de optimización no convexo, lo que puede llevar a la obtención de un mínimo local. En el caso de (Nguyen, Morell, & Bats, 2019), que es la propuesta tomada como base para este trabajo, se busca una matriz de Mahalanobis de rango bajo y se garantiza la convergencia global, al ser formulado como un problema de optimización convexo.

Recientemente, se ha utilizado la computación distribuida para atacar este problema, línea que seguimos en nuestro trabajo. En este sentido en (Su, Yang, King, & Lyu, 2016) se hace una propuesta de implementación de ITML (Davis, Kulis, Jain, Sra, & Dhillon, 2007) usando Apache Spark (Apache spark website, n.d.). En la misma se logra un equilibrio entre el desempeño y el tiempo de ejecución del método.

1.2. Formulación del problema

En problemas de clasificación usando k-NN si se garantiza que los vecinos de la misma clase estén más cerca que vecinos de clases diferentes, entonces se tendrá una clasificación exitosa. Adoptando la terminología empleada en (Nguyen, Morell, & Bats, 2019), sea x un ejemplo del universo, entonces se define a T_x como el conjunto de k ejemplos del universo más cercanos a x que comparten su misma clase y lo llamamos conjunto objetivo; y se define M_x como el conjunto de m ejemplos del universo más cercanos a x que no comparten su clase. Por tanto, el objetivo de la función de distancia a aprender es lograr que T_x se convierta en la vecindad de x.

Definimos el margen de una instancia x_i correspondiente a la función de distancia de Mahalanobis como:

$$\theta_M(x_i) = d_M^2(x_i,x_i^-) - d_M^2(x_i,x_i^+) \quad (1)$$

donde

$$x_i^+ = \text{argmax}_{xj \in Tx\ i}(d_M^2(x_i,x_j)) \text{ y } x_i^- = \text{argmin}_{xj \in Mx\ i}(d_M^2(x_i,x_j)) \quad (2)$$

El margen de una instancia x_i no es más que la diferencia de las distancias entre x_i y su vecino más cercano de diferente clase y x_i y su vecino más lejano en T_x. Recientemente, el margen ha sido muy usado en problemas de aprendizaje de funciones de distancia (DML).

En (Nguyen, Morell, & Bats, 2019) se propone un novedoso algoritmo denominado LMDML-A que usando SGD no necesita el paso de proyección para garantizar que la nueva solución se mantenga en el cono PSD, logrando un menor costo computacional de una iteración a otra.

Formalmente, el algoritmo LMDML-A resuelve el siguiente problema de optimización:

$$min_{M \geq 0} f(M) = \frac{1}{n}\sum_{i=1}^{n}[1 + d_M^2(x_i, x_i^+) - d_M^2(x_i, x_i^-)]_+ \quad (3)$$

s.t. $tr(M) \leq B,\ B > 0$

En cada iteración t, se selecciona una instancia x_i aleatoriamente. Si $1 + d_M^2(x_i, x_i^+) \leq d_M^2(x_i, x_i^-)$, entonces el gradiente de la función objetivo $\nabla f_i(M_t)$ es 0 y no es necesario modificar la matriz de Mahalanobis. En otro caso el gradiente viene dado por:

$$\nabla f_i(M_t) = (x_i - x_i^+)(x_i - x_i^+)^T - (x_i - x_i^-)(x_i - x_i^-)^T \quad (4)$$

Luego, se modifica la matriz M_t en la dirección del gradiente con paso $n_t = \min(\alpha_t, \frac{c}{\sqrt{t}})$, donde $c > 0$ es una constante y α_t es calculado usando M_t y su pseudo inversa $M_t^\dagger$ de forma tal que $M_t - \alpha_t \nabla f_i(M_t)$ sea PSD, ver más detalles en (Nguyen, Morell, & Bats, 2019).

$$M_{t+1/3} = M_t + n_t(x_i - x_i^-)(x_i - x_i^-)^T \quad (5)$$

$$M_{t+1/3}^\dagger = (M_t + n_t(x_i - x_i^-)(x_i - x_i^-)^T)^\dagger \quad (6)$$

$$M_{t+2/3} = M_{t+1/3} - n_t(x_i - x_i^+)(x_i - x_i^+)^T \quad (7)$$

$$M_{t+2/3}^\dagger = (M_{t+1/3} - n_t(x_i - x_i^+)(x_i - x_i^+)^T)^\dagger \quad (8)$$

$$M_{t+1} = \min(B/tr(M_{t+2/3}),1)M_{t+2/3} \quad (9)$$

$$M_{t+1}^\dagger = \max(tr(M_{t+2/3})/B,1)M_{t+2/3}^\dagger \quad (10)$$

2. Metodología

El método empleado fue la experimentación y la comparación con otros algoritmos que resuelven el mismo problema o problemas similares.

2. Resultados y discusión

2.1. Propuesta

En (Wang, Zhang, Liu, Huang, & Liqiang, 2018) se propone una versión distribuida de SGD que explota el principio de hacer una serie de iteraciones locales en las distintas particiones de datos, y luego, en cada iteración global, promediar los valores. Basados en este principio y en la probada factibilidad y análisis de convergencia realizado en dicho estudio tomamos esta idea para aplicarlo en nuestra propuesta.

Dado un conjunto de instancias de entrenamiento, las mismas se distribuyen uniformemente en p particiones y se construye para cada instancia x el conjunto T_x y M_x usando solamente instancias de esta partición. Estos conjuntos se pueden construir usando la función de distancia euclideana, nótese además que estos conjuntos no cambian durante la etapa de entrenamiento. En cada partición se aplican T iteraciones del algoritmo LMDML-A, posteriormente se promedian todas las matrices resultantes de cada partición y así sucesivamente durante E iteraciones. A este algoritmo le llamamos Scalable Large-Margin Distance Metric Learning (S-LMDML-A). El pseudo código del mismo es proporcionado en el Algoritmo 1.

Entrada: Instancias etiquetadas $\{x_1,x_2,...x_n\}$

Parámetros: B,c,T,E

Salida: M

1. Construir los conjuntos T_x y M_x para cada instancia x usando solamente información de la partición donde se encuentra contenida.
2. Sea $M_1 = I$ y $M_1^\dagger = I^\dagger$
3. Para $e \leftarrow 1,2,...,E$
    - Distribuir M_e y $M_e^\dagger$ para todas las particiones.
    - En cada partición, independientemente ejecutar:
        - $M_1 = M_e$ y $M_1^\dagger = M_e^\dagger$
        - *Para* $t \leftarrow 1,2,...,T$

- Seleccionar aleatoriamente $i \in \{1,2,...,m\}$ donde m es el número de instancias en cada partición.
- Buscar x_i^+ y x_i^- según (2)
- Si $1 + d_M^2(x_i,x_i^+) \succ d_M^2(x_i,x_i^-)$
  - Buscar α_t de forma tal que $M_t - \alpha_t \nabla f_i(M_t) \succcurlyeq 0$
  - Seleccionar $n_t = \min(\alpha_t, \frac{c}{\sqrt{t}})$
  - Calcular M_{t+1} y $M_{t+1}^\dagger$ según (9) y (10)
- $M_{e+1} = M_t$ y $M_{e+1}^\dagger = M_t^\dagger$

- Promediar las matrices M_{e+1} y $M_{e+1}^\dagger$ resultantes de cada partición para obtener las nuevas M_{e+1} y $M_{e+1}^\dagger$.

Algoritmo 1: Scalable Large-Margin Distance Metric Learning

Nótese que el promediar las distintas matrices obtenidas en cada partición de datos es una operación válida, ya que la suma de matrices PSD da lugar a una nueva matriz PSD y la multiplicación de una matriz PSD por una constante positiva da lugar a una matriz PSD.

2.2. Detalles de implementación

El algoritmo está implementado en Apache Spark (Apache spark website, n.d.), un framework distribuido de propósito general basado en Apache Hadoop (Apache hadoop, n.d.). El mismo logra una mayor eficiencia debido a que mantiene los datos en memoria, en una estructura de datos distribuida y tolerante a fallos (RDD), entre las fases de mapeo y reducción. Ha sido adoptado rápidamente debido a una mejora en el rango de 10x a 100x sobre Hadoop, en problemas de aprendizaje automatizado, tales como clasificación, regresión, clusterización y más recientemente ha tomado auge para problemas de aprendizaje profundo.

Para medir el desempeño obtenido al utilizar la función de distancia aprendida se usa la implementación de k-NN basada en Spark provista en (Maillo, Ramirez, Triguero, & Herrera). El mismo aprovecha la utilización de operaciones en memoria para clasificar

grandes volúmenes de datos. La fase de mapeo calcula para cada instancia de prueba los k vecinos más cercanos en las distintas particiones de datos del conjunto de entrenamiento y posteriormente a través de múltiples reducciones se obtienen los k vecinos más cercanos definitivos. La clave de este algoritmo es el manejo del conjunto de datos de pruebas, manteniendo el mismo en memoria, siempre que se pueda. En caso contrario, el mismo se particiona en un número mínimo de partes, donde cada parte pueda ser almacenada en memoria, y se aplica a cada una el algoritmo descrito anteriormente, usando la capacidad de Spark para mantener los datos en memoria y reusarlos tantas veces como sea necesario hasta completar la clasificación del conjunto de pruebas.

2.3. Experimentos

Para evaluar la efectividad y eficiencia del algoritmo propuesto se condujeron una serie de experimentos para comparar dicho algoritmo con la propuesta original (Nguyen, Morell, & Bats, 2019).

Se usaron bases de casos estándares con diferentes tamaños. Todos los conjuntos de datos fueron descargados del repositorio de KEEL. La tabla 1 describe cada uno de los conjuntos de datos utilizados. Todas las características fueron normalizadas en el intervalo [0,1]. En el experimento se usó validación cruzada de 10 iteraciones para estimar el desempeño del método propuesto.

#	Base de casos	# Características	# Instancias
1	appendicitis	7	106
2	balance	4	625
3	Bupa	6	345
4	Iris	4	150
5	Letter	16	20000
6	Magic	10	19020
7	Monk-2	6	432
8	Optdigits	64	5620
9	Ring	20	7400
10	Wdbc	30	569
11	Wine	13	178
12	Wisconsin	9	683

Tabla 1. Descripción de las bases de casos

Para el método LMDML-A y S-LMDML-A, el parámetro B se busca entre los valores 0.1,1,10,100; el número de iteraciones globales E usado es 10, el número de iteraciones locales T es 1000 y 1 para el parámetro c.

La tabla 2 sumariza el desempeño de clasificación y el tiempo de entrenamiento de ambos algoritmos en cada base de casos. En cada base de casos se asigna rango 1 al algoritmo que mejor desempeño tiene y rango 2 al otro. En la última fila de la tabla 2 se muestra el promedio de los rangos del desempeño de cada algoritmo.

Basado en los resultados experimentales y en la realización de la prueba de los rangos con signos de Wilcoxon se puede concluir que el algoritmo propuesto logra resultados de desempeño similares a la variante original (no distribuida) al no presentar diferencias significativas, incluso en bases de casos que son pequeñas, por lo que se puede concluir que el método propuesto es válido. El tiempo de ejecución para bases de casos pequeñas es mayor, pero en las bases de casos más grandes si se puede apreciar una mejora; y este precisamente es el resultado esperado, pues se logra acelerar el proceso de aprendizaje cuando el volumen de instancias de entrenamiento es grande.

	Desempeño		Tiempo de ejecución	
#	LMDML	SLMDML	LMDML	SLMDML
1	**86.91**	83.43	**0.08**	0.41
2	90.7	**92.19**	**0.07**	0.64
3	**67.49**	64.7	**0.07**	0.85
4	95.33	**95.46**	**0.06**	0.41
5	**96.83**	96.4	60.03	**20.16**
6	**84.93**	83.36	60.43	**8.85**
7	**98.37**	97.75	**0.05**	0.68
8	98.74	**99.17**	**1.17**	7.54
9	74.07	**75.53**	**0.66**	8.63
10	**97.54**	95.8	**0.08**	0.62
11	97.75	**98.12**	**0.06**	0.69
12	96.63	**97.12**	**0.04**	0.37
Avg. Rango	1.2857	1.2857		

Tabla2: Desempeño y tiempo de entrenamiento en bases de casos estándares.

4. Conclusiones

En este trabajo se ha propuesto una variante de paralelización del algoritmo LMDML-A implementado usando Apache Spark. Los resultados experimentales han validado la factibilidad del mismo para ser empleado en grandes volúmenes de datos. En trabajos futuros se analizará la posibilidad de aplicar este algoritmo a datos con mayor número de características, ya que uno de los requerimientos del mismo es usar un número de características discreto ya que la matriz de Mahalanobis es de tamaño DxD y debe poderse almacenar en memoria. Además, se analizará la influencia de usar otra estrategia de particionado para los datos de entrada.

Referencias bibliográficas

1. *Apache hadoop*. (n.d.). Retrieved from Apache hadoop: http://hadoop.apache.org/
2. *Apache spark website*. (n.d.). Retrieved from Apache spark website: https://spark.apache.org/
3. Davis, J. V., Kulis, B., Jain, P., Sra, S., & Dhillon, I. S. (2007). Information-theoretic metric learning. *Proceedings of the 24th International Conference on Machine Learning*.
4. Goldberger, J., Roweis, S., Hinton, G., & Salakhutdinov, R. (2005). Neighbourhood components analysis. *Advances in Neural Information Processing Systems 17*.
5. Maillo, J., Ramirez, S., Triguero, I., & Herrera, F. (n.d.). knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data.
6. Nguyen, B., Morell, C., & Baets, B. D. (2017). Supervised distance metric learning through maximization of the jeffrey divergence. *Pattern Recognition*.
7. Nguyen, B., Morell, C., & Bats, B. D. (2019). Scalable large-margin distance metric learning using stochastic gradient descent.
8. Roweis, Globerson, A., & T., S. (2006). Metric learning by collapsing classes. *Advances in Neural Information Processing System 18*.

9. Saul, Weinberger, K. Q., & K., L. (2009). Distance metric learning for large margin nearest neighboar clasification. *The Journal of Machine Learning Research*.

10. Su, Y., Yang, H., King, I., & Lyu, M. (2016). Distributed information-theoretic metric learning in apache spark. *International Joint Conference on Neural Networks*.

11. Wang, Zhang, H., Liu, Z., Huang, H., & Liqiang. (2018). FTSGD: An Adaptive Stochastic Gradient Descent Algorithm for Spark MLlib.

12. Xing, E. P., Jordan, M. I., Russell, S., & Ng, A. Y. (2002). Distance metric learning with application to clustering with side-information. *Advances in neural information processing systems*.

13. Zhang, H., Liu, Z., Huang, H., & Wang, L. (2018). Ftsgd: An adaptive stochastic gradient descent algorithm for spark mllib.