Un Framework de Selección de Características basado en la Teoría de la Información para Big Data sobre Apache Spark

S. Ramírez-Gallego[1], Héctor Mouriño-Talín[2], David Martínez-Rego[2], Verónica Bolón-Canedo[2], Jose M. Benitez[1], Amparo Alonso-Betanzos[2], and Francisco Herrera[1]

[1] Departamento de Ciencias de la Computación e Inteligencia Artificial, CITIC-UGR, Universidad de Granada, 18071 Granada, España
{sramirez,J.M.Benitez,herrera}@decsai.es
[2] Departmento de Ciencias de la Computación, Universidad de A Coruña, 15071 A Coruña, España. {h.mtalin,dmartinez,veronica.bolon,ciamparo}@udc.es

Resumen Con la llegada de la alta dimensionalidad a los datos, las técnicas de reducción se han convertido en elemento imprescindible para su procesamiento. Entre ellas, la selección de características se erige como una herramienta importante para la identificación de características relevantes en conjuntos de gran tamaño. El objetivo de este trabajo es demostrar que los métodos estándar de selección pueden paralelizarse haciendo uso de plataformas Big Data, como Apache Spark, mejorando así la rapidez de estos métodos. Nuestra propuesta se basa en el desarrollo de un framework distribuido para selección de características, el cual incluye un amplio grupo de métodos de selección basados en la Teoría de la Información. Los resultados experimentales muestran cómo nuestro framework es capaz de ofrecer resultados competitivos para conjuntos de datos de alta dimensionalidad, así como conjuntos con un alto número de ejemplos.

Keywords: Alta dimensionalidad · Métodos de filtrado · Selección de características · Apache Spark · Big Data

1. Introducción

Durante las últimas décadas, la dimensionalidad de los datos en el campo del Aprendizaje Automático (AA) se ha incrementado de manera significativa. Esto presenta un desafío para los investigadores ya que los algoritmos existentes no siempre son capaces de responder con rapidez cuando se enfrentan a estas dimensionalidades extremadamente grandes. La identificación de características relevantes se presenta, por tanto, como una tarea imprescindible en este campo. Las técnicas de reducción de dimensionalidad (entre ellas, la Selección de Características o SC) pueden ser aplicadas para reducir el tamaño de los datos y a la vez mejorar la capacidad de aprendizaje de los algoritmos [4]. Sin embargo, los métodos de SC no están diseñados para trabajar con Big Data por lo que su

eficiencia puede verse significativamente deteriorada o incluso imposibilitar su aplicación.

EN las últimas años, han aparecido nuevos protocolos y frameworks de programación distribuida y escalable para manejar el problema del Big Data. Recientemente, Apache Spark [6] se presenta como una nueva herramienta para el procesamiento de datos a gran escala, la cual ha ganado popularidad entre la comunidad científica debido a su idoneidad para procesos iterativos. De hecho, Spark posee una biblioteca de AA llamada MLlib [7]. A pesar de esta evolución, pocos métodos de SC han aparecido.

Este trabajo tiene como objetivo abordar este problema, demostrando que los algoritmos de SC estándar pueden ser paralelizados también usando las plataformas Big Data, y así mostrar su utilidad al enfrentarse a conjuntos de gran tamaño. En este trabajo, proponemos una implementación distribuida de un framework genérico para SC basado en la Teoría de la Información [1] usando el paradigma de programación Apache Spark.

El resto del paper se organiza de la siguiente manera: la Sección 2 describe el framework distribuido propuesto para SC en Big Data. La Sección 3 presenta los experimentos llevados a cabo y analiza los resultados. Finalmente, la Sección 5 enumera las conclusiones del trabajo.

2. Métodos de Filtrado basados en la Teoría de la Información

Dentro de la SC, podemos definir los métodos de filtrado como aquellos que usan medidas de separabilidad o dependencias estadísticas para valorar la relevancia de los atributos. Estos métodos están basados en el uso de un índice o medida cuantitativa. Existe un amplio espectro de métodos de filtrado en la literatura construidos sobre las medidas de relevancia (interacción de un atributo con la clase) y redundancia (información similar entre atributos de entrada). Para homogeneizar el uso de todos estos criterios, Brown et al. [1] propusieron una expresión genérica que permite agrupar varios métodos de filtrado basados en la Teoría de la Información en un único framework para SC. Dicho framework está basado en un proceso de optimización voraz que evalúa los atributos usando un simple criterio genérico. Haciendo uso de algunas asunciones de independencia, este criterio permite transformar el conjunto de métodos como combinaciones lineales de términos de la teoría de la entropía de Shannon: información mutua simple (IM) y condicionada (IMC) [3]. La fórmula propuesta por estos autores es:

$$J = I(X_i; Y) - \beta \sum_{j \in S} I(X_j; X_i) + \gamma \sum_{j \in S} I(X_j; X_i | Y), \tag{1}$$

dónde X_i representa el conjunto de atributos no-seleccionados, X_j el conjunto de atributos seleccionados, S el conjunto de índices seleccionados, Y el atributo de clase, I las medidas de información mutua, β el peso de la redundancia simple y γ el peso de la redundancia condicional. La fórmula se divide en tres partes: la

primera representa la relevancia, la segunda la redundancia entre los atributos seleccionados y los no-seleccionados, y la tercera la redundancia condicional entre las variables anteriores y la clase. A través de las citadas asunciones, muchos criterios pueden ser transformados para que sigan la expresión genérica propuesta por Brown et al. con ligeras variaciones. La lista completa de los criterios transformados puede ser consultada en [1].

3. SC basada en Filtrado para Big Data

En esta sección, describimos cómo hemos remodelado el framework de Brown et al. para el paradigma distribuido. Esta versión contiene una implementación genérica de varios métodos de filtrado como: minimum Redundancy Maximum Relevance (mRMR), Conditional Mutual Information Maximization (CMIM), o Joint Mutual Information (JMI), entre otros. Más allá de una simple implementación de métodos para Spark, hemos re-diseñado el framework de Brown et al. añadiendo mejoras importantes a la eficiencia del algoritmo clásico:

- **Formato por Columnas:** El patrón de acceso que presentan la mayoría de métodos de SC está pensado para trabajar por atributos/columnas, mientras que para el resto de algoritmos el patrón suele ser por instancia/fila. Un uso de un formato correcto en los datos puede mejorar el rendimiento de cálculos como los derivados de la IM. Esto es especialmente importante en sistemas distribuidos como Spark, donde el esquema de particionamiento elegido es determinante para su rendimiento.
- **Cacheo de variables:** El primer elemento que aparece en la Ecuación 1 es el cálculo de la relevancia. Dicha operación se realiza sólo una vez al inicio del algoritmo, siendo su resultado almacenado para su posterior uso en las siguientes evaluaciones de la citada fórmula. De igual manera, se almacenan las probabilidades marginales y combinadas derivadas de esta operación para evitar realizar operaciones extra.

3.1. Algoritmo principal

El algoritmo principal de nuestro método es un procedimiento secuencial que se encarga de calcular las relevancias iniciales e iterar sobre X_i, seleccionando atributos de acuerdo a la Ecuación 1 y a los valores de IM y IMC derivados de esa expresión.

El primer paso consiste en trasformar los datos a formato por columnas. Una vez transformado, el algoritmo obtiene el valor de relevancia entre cada atributo de entrada y la clase, inicializando el criterio asociado a cada atributo. Los valores de relevancia se almacenan como parte de la expresión previa y se re-utilizan en los siguientes pasos para actualizar los criterios. Después, se selecciona el atributo más relevante p_{best} y se añade al conjunto S (vacío). La fase iterativa comienza calculando la IM y la IMC entre p_{best}, los atributos en X_i e Y. Los valores derivados servirán para actualizar las redundancias acumuladas

de los criterios. En cada iteración, se selecciona el atributo no-seleccionado más relevante como nuevo p_{best} y añadido a S. El bucle termina cuando S alcanza un tamaño igual al establecido como parámetro o no hay más atributos para seleccionar.

3.2. Operaciones Distribuidas: Formato por Columnas y Cálculo de IM

La estimación de la IM y la IMC son, sin ninguna duda, las operaciones más costosas de este framework. Esta sección describe cómo han sido paralelizadas haciendo uso de conjunto de operaciones de Spark[3].

Formato por Columnas La Figura 1 detalla la transformación por columnas definida como primer paso del anterior algoritmo. La idea en la que se basa esta transformación es la de transponer la matriz de datos local de cada partición. Esta operación mantendrá el esquema de particionamiento sin necesidad de barajar los datos por la red, y además, permitirá re-utilizar este formato en el resto de operaciones. El resultado es una nueva matriz con una fila por cada atributo. El algoritmo genera así una tupla, dónde la clave está formada por el índice del atributo, y la clave por el índice de la partición/bloque y la matriz local transpuesta de ese bloque.

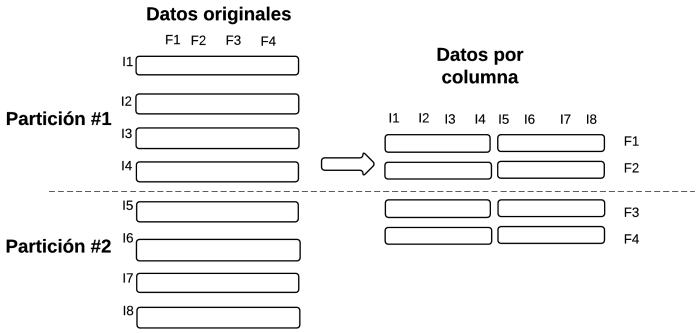

Figura 1: Esquema de transformación por columnas. F indica atributo e I instancia. Cada rectángulo a la izquierda representa un registro en el conjunto original. Cada rectángulo a la derecha representa la matriz transpuesta para cada atributo y bloque.

Aprovechando la localidad de los datos, el algoritmo coloca todas las instancias del mismo atributo en un mismo conjunto de particiones (a ser posible, sólo

[3] Para una descripción completa de estas operaciones, visitar el siguiente enlace: `https://spark.apache.org/docs/latest/api/scala/index.html`

una). Para esto, es necesario ordenar las nuevas instancias por clave (operación distribuida), limitando el número de particiones al parámetro facilitado. Así, en las siguientes fases, las particiones son mapeadas con el objetivo de generar histogramas por atributo. Dichos histogramas contabilizan el número de ocasiones en los que aparece una determinada combinación de variables.

Escoger un número adecuado de particiones es muy importante para las siguientes operaciones. Así, si el número de particiones es igual o menor que el número de atributos, el número total de histogramas por atributo será como máximo de dos. Por el contrario, si el número de particiones es mayor que el número de atributos, el número de histogramas puede dispararse al tener un mismo atributo distribuido en varias particiones. Lo ideal es establecer este parámetro por debajo de dos veces el número de atributos.

Cálculo de la Relevancia Tras la transformación de los datos, esta parte describe cómo calcular la relevancia (IM) entre X_i y la clase Y de manera distribuida. Este método es ideado como un método de inicialización, de manera que todas las variables que aparecen pueden ser re-utilizadas en los siguientes métodos.

La idea principal que deriva del cálculo de la relevancia y la redundancia es la realización de estas operaciones de manera independiente para cada atributo en X_i. Esto se realiza replicando y copiando sólo aquellos atributos variables en cada iteración, aprovechando así la localidad de los datos ya transformados. En este caso, el primer paso consiste en obtener todos los bloques de Y y reunirlos todos en un mismo vector que será replicado mediante la operación *broadcast* (operación distribuida que replica una variable de sólo lectura en cada nodo con una sóla copia de ésta). Después de esto, el algoritmo calcula los histogramas para cada atributo en X_i con respecto a Y. El algoritmo calcula después las probabilidades marginales y combinadas también a partir de los histogramas anteriores a través de una serie de operaciones matriciales: agregando proporciones por fila para las combinadas, y calculando simples proporciones sobre la matriz para las marginales. Finalmente, usando toda esta información podemos calcular los valores de relevancia (IM) para X_i.

Cálculo de la Redundancia En este caso, el cálculo de la redundancia y la redundancia condicional se realiza entre el conjunto p_{best}, X_i, e Y. La redundancia condicional introduce una tercera variable condicional, siguiendo la fórmula $I(X_j; X_i|Y)$ (ver Ecuación 1). Esta operación es repetida hasta que se llega a la condición de parada (número de atributos seleccionados). Este algoritmo es una extensión del algoritmo para el cálculo de la relevancia. Primero, éste obtiene los bloques para el atributo p_{best} y los envía a cada nodo. Después la función de cálculo de histogramas es llamada con dos atributos extra[4], para así obtener los histogramas tridimensionales. Finalmente, se calculan los dos tipos de redundancia haciendo uso de la función de cálculo de la IM y la IMC.

[4] La variable Y ya está almacenada tras ejecutar la fase del cálculo de la redundancia.

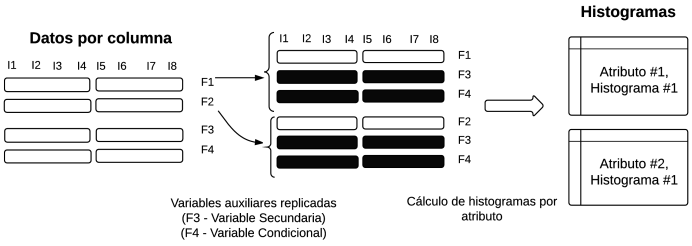

Figura 2: Esquema de generación de histogramas. F indica atributo e I instancia. Cada rectángulo blanco representa un bloque para un atributo en el formato por columnas. Cada rectángulo negro representa una variable secundaria replicada. En este caso, el número de particiones se corresponde con el número de atributos.

Generación de Histogramas La Figura 2 muestra un esquema del algoritmo que tiene como objetivo calcular los histogramas para X_i con respecto a las variables p_{best} y Y. Dichos histogramas se utilizarán luego para el cálculo de la IM y la IMC. En el caso de no aportar una tercera variable, este algoritmo calcula matrices con la tercera dimension igual a uno.

Primero, se inicia una operación de mapeo sobre cada partición del conjunto transformado. Esta operación itera sobre los bloques generados en la transformación inicial. Para cada una de estas tuplas, se inicializa una matriz, y posteriormente se incremeta según la combinación de p_{best}, X_i, y Y. El algoritmo replica después las matrices asociadas a los atributos variables (p_{best} e Y) a todos los nodos; estas matrices tienen como primera dimension el índice de bloque, y como segunda el índice parcial del valor en el bloque. Esta operación de actualización se repite para cada valor en el vector. Después de esto, se genera una nueva tupla con el índice de atributo como clave y la matriz resultante como valor ($< x, mat >$). Esta operación de mapeo continúa con el siguiente bloque de datos hasta que se termina la iteración de la partición. Finalmente, se agregan todos los histogramas sumando aquellos con la misma clave. Como se ha especificado anteriormente, normalmente una única partición contiene todos los bloques para un mismo atributo, y por tanto, no se generan demasiadas matrices, siendo la posterior agregación menos pesada.

Cálculo de la IM y la IMC La Figura 3 detalla el proceso que unifica el cálculo de la IM y la IMC. El algoritmo define como parámetros de entrada el índice de los dos atributos variables (Y o p_{best} para la IM, y p_{best} e Y para la IMC), y el conjunto de histogramas anteriormente calculados. Antes de comenzar, el algoritmo obtiene y replica las matrices marginales y combinadas de dichos atributos. Esta información se envia porque no se puede derivar de cada histograma de manera independiente y ya está precalculada desde la fase de relevancia.

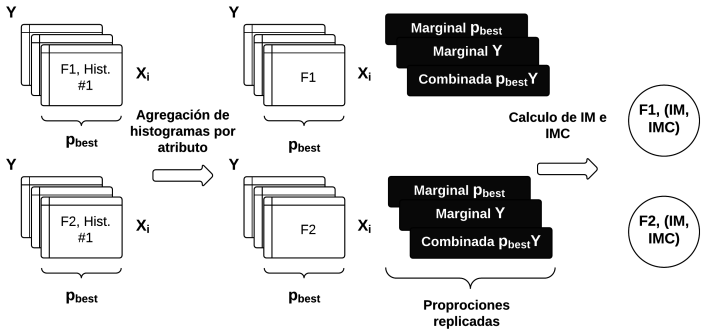

Figura 3: Esquema del cálculo de la IM y la IMC. F indica atributo e I instancia. Las matrices tridimensionales representan los histogramas para cada atributo-bloque. Los rectángulos negros las probabilidades marginales y combinadas para cada atributo variable (p_{best} e Y) (pre-calculadas y replicadas).

Se inicia entonces una fase de mapeo sobre cada tupla atributo-histograma. En esta fase, el algoritmo genera todos los valores de IM e IMC entre X_i y el resto de atributos variables. Antes de esto, es necesario calcular las proporciones marginales para X_i y las combinadas entre X_i e Y, y X_i y p_{best} (usando operaciones de agregación de matrices). Una vez todas las probabilidades marginales y combinadas son calculadas, se inicia un bucle sobre todas las combinaciones entre las tres variables para calcular la probabilidad de cada combinación, y finalmente, el resultado final para cada combinación. Todos los resultados por instancia se agregan para obtener el valor final de IM y IMC por atributo.

Versión Dispersa para Conjuntos Altamente Dimensionales La versión previamente explicada funciona bien para conjuntos con gran número de instancias y un número reducido de atributos. Sin embargo, cuando nos enfrentamos a conjuntos con millones de atributos y un alto índice de dispersión en sus datos, la estrategia cambia. En nuestro caso, hemos añadido alguna modificación al algoritmo original para permitir ejecutar nuestro framework sobre este tipo de datos. Básicamente, la idea es eliminar el procesamiento por bloques y optar por crear un único vector por atributo.

4. Framework Experimental y Análisis

Esta sección describe los experimentos llevados a cabo para evaluar la utilidad de nuestra solución de SC sobre varios conjunto de datos de gran tamaño (tanto en número de atributos, como de instancias).

4.1. Conjuntos de Datos y Métodos

En nuestros experimentos, hemos utilizado dos conjuntos de datos en una tarea de clasificación para medir la calidad de las soluciones desarrolladas. El primer conjunto, *ECBDL14*, es un conjunto denso con un alto número de ejemplos, que fue usado como referencia en la conferencia internacional GECCO-2014[5]. El segundo, *kddb*, es un conjunto disperso de alta dimensionalidad, proveniente del repositorio LibSVM [2][6]. La Tabla 1 ofrece información sobre estos conjuntos.

Tabla 1: Descripción de los conjuntos de datos empleados. Para cada conjunto se muestra: el número de instancias para entrenamiento y test (#Entr.) (#Test), el número total de características (#Cr.), el número total de clases (#Cl) y su condición de dispersión (Disp.).

Data Set	**#Entr.**	**#Test**	**#Cr.**	**#Cl.**	**Disp.**
ECBDL14	65 003 913	2 897 917	630	2	No
kddb	19 264 097	748 401	29 890 095	2	Sí

Cómo método de selección hemos escogido mRMR [5], al ser uno de los más relevantes en SC. Para todas las ejecuciones, se ha establecido un nivel de paralelismo de 864 y se ha cacheado los conjuntos en memoria al tratarse de un proceso iterativo. Para evaluar la eficiencia de la SC, hemos utilizado el tiempo de modelado en entrenamiento como medida en tiempo. En cuanto al cluster utilizado, éste se compone de 18 nodos esclavo y 1 maestro. Los nodos poseen las siguientes características: 2 procesadores x Intel Xeon CPU E5-2620, 6 núcleos por procesador, 2.00 GHz, 15 MB cache, Red QDR InfiniBand (40 Gbps), 2 TB HDD, 64 GB RAM, Hadoop 2.5.0-cdh5.3.1, Apache Spark and MLlib 1.2.0, 432 núcleos utilizados (24 núcleos/nodo), 864 RAM GB utilizados (48 GB/nodo). Finalmente, existe un paquete asociado a este trabajo que puede ser consultado en el repositorio de Apache Spark: *http://spark-packages.org/package/sramirez/spark-infotheoretic-feature-selection*.

4.2. Análisis de Resultados

En esta sección evaluamos el tiempo empleado por nuestra solución usando distintos límites de selección, y así mismo, comparamos nuestra versión distribuida con la secuencial desarrollada por el autor original[7]. La Tabla 2 presenta los resultados de nuestro framework usando diferentes límites de características seleccionadas. Como se puede apreciar, nuestro algoritmo obtiene resultados competitivos para todos los casos sin importar el número de iteraciones utilizadas (representadas por el límite). Es importante destacar que para todos los casos se obtienen soluciones por debajo de una hora.

[5] http://cruncher.ncl.ac.uk/bdcomp/
[6] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
[7] FEAST toolbox (python version): http://www.cs.man.ac.uk/~gbrown/software/

Tabla 2: Tiempo de selección según el límite de atributos (en segundos)

Conjunto	10	25	50	100
kddb	283.61	774.43	1365.82	2789.55
ECBDL14	332.90	596.31	1084.58	2420.94

Finalmente, hemos realizado una estudio comparativo entre nuestra versión distribuida y la versión secuencial[8]. Hemos generado subconjuntos con distinto número de instancias para estudiar la escalabilidad de nuestro algoritmo en contraposición a la de la versión secuencial. El nivel de paralelismo se ha establecido a un valor de 200 para la versión distribuida. La Figura 4 muestra los resultados de tiempo para la comparativa[9]. Los resultados muestran que nuestra nueva implementación distribuida mejora la versión clásica en todos los casos. Especialmente remarcable es el caso de mayor tamaño, dónde conseguimos acelerar el proceso en un factor de 71.11. Cabe también destacar que el algoritmo distribuido selecciona las mismas características que el algoritmo secuencial, salvo en aquellos casos en los que varias caracterísiticas presentan el mismo valor de utilidad.

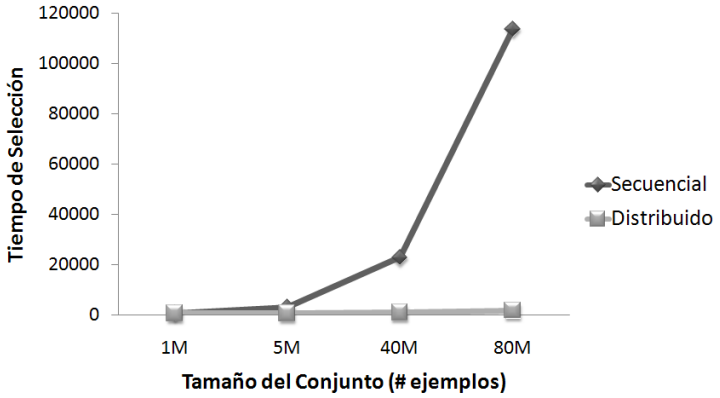

Figura 4: Comparativa de tiempos de selección (secuencial vs. distribuida) en segundos. 200 atributos originales, 100 seleccionados.

[8] La versión secuencial ha sido ejecutada en un nodo de nuestro cluster, con las características anteriormente comentadas.

[9] El resultado para los dos últimos valores de la versión secuencial ha sido estimado debido a problemas de límite de memoria

5. Conclusiones

A pesar del creciente interés en el campo de la reducción de dimensionalidad para Big Data, muy pocos métodos de SC han sido desarrollados para tratar problemas de alta dimensionalidad. Aquí, proponemos un framework genérico para SC en Big Data. Este es una adaptación del framework basado en la Teoría de la Información desarrollado por Brown et al. en [1]. La adaptación de este framework no ha sido trivial y ha supuesto un re-diseño completo para su adaptación al paradigma distribuido. Con este trabajo también hemos querido contribuir con un módulo de SC a la emergente plataforma de Spark y MLlib, dónde no existen métodos complejos de SC actualmente.

Los resultados competitivos que hemos obtenido demuestran la validez de nuestro método. Destacamos el hecho de que en ambos casos, nuestro método obtiene resultados por debajo de una hora incluso en conjuntos tan grandes (tanto en número de instancias como de atributos) como los utilizados. Además, nuestra solución es capaz de obtener mejores resultados que la versión secuencial en todos los casos estudiados, habilitando la aplicación de la SC a muchos problemas en los que antes era inabordable. En un futuro cercano, se extenderá este trabajo con un marco experimental más amplio que permita validar las conclusiones anteriormente expuestas.

Referencias

1. Brown, G., Pocock, A., Zhao, M.J., Luján, M.: Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. J. Mach. Learn. Res. 13, 27–66 (Jan 2012)
2. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011), datasets available at `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`
3. Cover, T.M., Thomas, J.A.: Elements of Information Theory (1991)
4. Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L.A.: Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing). Springer-Verlag New York, Inc. (2006)
5. Peng, H., Long, F., Ding, C.: Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. Pattern Analysis and Machine Intelligence, IEEE Transactions on 27(8), 1226–1238 (2005)
6. Spark, A.: Apache Spark: Lightning-fast cluster computing (2015), `https://spark.apache.org/`, [Online; accessed March 2015]
7. Spark, A.: Machine Learning Library (MLlib) for Spark (2015), `http://spark.apache.org/docs/latest/mllib-guide.html`, [Online; accessed March 2015]