ISSN 1870-4069

Calibración de hiperparámetros para la clasificación de requisitos de software mediante evolución diferencial

J. Manuel Pérez-Verdejo[1], Efrén Mezura-Montes[2],
Ángel Juan Sánchez-García[1], Jorge Octavio Ocharán-Hernández[1]

[1] Universidad Veracruzana,
Facultad de Estadística e Informática,
México
[2] Universidad Veracruzana,
Centro de Investigacion en Inteligencía Artificial,
México

manuel.vrdjo@gmail.com, {emezura,jocharan}@uv.mx,
angelsg89@hotmail.com

Resumen. La clasificación automática de requisitos representa un área de interés dentro de la Ingeniería de Software. Es a partir del tipo de requisito, que se toman estrategias para todo el ciclo de vida de un sistema. Trabajos previos han presentado enfoques de clasificación por aprendizaje máquina sobre este tópico, sin embargo, la selección de los hiperparámetros y su efecto sobre la precisión de estos modelos permanecen como problemas abiertos. En este estudio, se presenta una propuesta para la calibración de estos parámetros a través del algoritmo de evolución diferencial DE/rand/1/bin. Este procedimiento se probó con un modelo de Naïve Bayes y se comparó contra la herramienta para la optimización TPOT. Los resultados preliminares muestran un incremento de la precisión del modelo hasta por un 7 %, evaluado por validación cruzada, y una precisión máxima de 69.7 % con el modelo generado por TPOT.

Palabras clave: Evolución diferencial, aprendizaje máquina automático, ingeniería de requisitos, algoritmos evolutivos, clasificación de requisitos.

Calibration of hyperparameters for the classification of software requirements using differential evolution

Abstract. Automatic classification of requirements represents an area of interest within Software Engineering. It is from the type of requirement that strategies are taken for the entire life cycle of a system. Previous

work has presented machine learning classification approaches on this topic, however, the selection of hyperparameters and their effect on the precision of these models remain open problems. In this study, a proposal is presented for the calibration of these parameters through the differential evolution algorithm DE/rand/1/bin. This procedure was tested with a Naïve Bayes model and compared against the TPOT optimization tool. Preliminary results show an increase in model precision of up to 7%, evaluated by cross-validation, and a maximum precision of 69.7% with the model generated by TPOT.

Keywords: Differential evolution, machine learning, requirements engineering, evolutionary algorithms, classification of requirements.

1. Introducción

Uno de los objetivos fundamentales de la Ingeniería de Requisitos (IR) se enfoca en entender y documentar las necesidades y deseos de los interesados en un producto de software [7]. Dentro de las actividades de la IR, se encuentra la especificación, en la que un analista plasma los intereses y necesidades de los usuarios en enunciados estructurados conocidos como requisitos de software [21].

Para la especificación y posterior administración de los requisitos identificados, es necesario que sean separados a partir de la categoría a la que pertenecen, pues esta determina la estrategia que se sigue para que se pueda satisfacer [25]. De estas categorías, los Atributos de Calidad resultan especialmente críticos, pues su correcta identificación afecta de forma directa la arquitectura del sistema [5].

Este análisis suele consumir mucho tiempo y ser propenso a errores, además de repetitivo, pues se efectúa de nuevo cuando se agregan o modifican los requisitos [2]. Es por estos motivos que se han desarrollado diferentes enfoques para la clasificación automática de requisitos de software [8,10,11]. De la misma forma, la implementación de estos modelos de aprendizaje máquina implican una combinación de parámetros que determinan su composición. Estos, denominados hiperparámetros, influyen en el rendimiento y precisión de los clasificadores [12]. Es por eso que la calibración de estos datos se representa un paso importante en el entrenamiento de modelos de aprendizaje máquina.

Este tipo de problemas combinatorios son viables para ser abordados por técnicas de optimización como los algoritmos evolutivos. Estos algoritmos ya ha sido explorados con anterioridad en el área de la Ingeniería de Software, específicamente para la configuración de sistemas auto-adaptativos [6].

Los algoritmos evolutivos realizan la búsqueda de soluciones competitivas mediante conjuntos de soluciones potenciales llamados poblaciones. Estos algoritmos están inspirados en el proceso de la evolución natural [1]. Especialmente, los algoritmos de evolución diferencial son fáciles de implementar para la optimización, pues utilizan pocos parámetros que suelen mantenerse fijos a lo largo del proceso evolutivo [22].

Por lo anterior, este estudio plantea la aplicación de algoritmos evolutivos, específicamente de evolución diferencial para la calibración de los hiperparámetros de un clasificador de requisitos de software. Lo anterior con el objetivo de maximizar el rendimiento y precisión del mismo.

El resto de este documento está organizado de la siguiente forma: la Sección 2 presenta el problema de optimización de hiperpárametros y su aplicación en la clasificación de requisitos. La Sección 3 expone la contribución, una propuesta de la aplicación del algoritmo DE/rand/1/bin para la optimización de hiperparámetros. En la Sección 4 se describen los experimentos y los resultados obtenidos. Finalmente, la Sección 5 resume los hallazgos identificados y el trabajo futuro.

2. Definición del problema

La configuración de modelos de aprendizaje máquina afecta en gran medida su desempeño [12]. La selección de estas características se complica en la medida del clasificador que se selecciona. La exploración de las posibles combinaciones de hiperparámetros, representan una oportunidad para mejorar el rendimiento de un modelo, cuando este es comparado con una comparación con valores por defecto.

Calibrar este tipo de valores para maximizar el rendimiento de modelos de aprendizaje máquina a través de métodos evolutivos ha sido explorado anteriormente. En [4] se exploran diferentes algoritmos evolutivos para incrementar el rendimiento de modelos de Árboles de Decisión; a partir de este estudio, se identifican ventajas de utilizar este tipo de optimizaciones para adaptar algoritmos a dominios específicos. De la misma forma, en [12] se exploran estrategias de calibración de hiperparámetros para clasificadores en el contexto de información biomédica.

Sin embargo, y hasta donde los autores han podido verificar, no se han identificado estrategias de calibración de hiperparámetros para clasificadores en el dominio de la Ingeniería de Requisitos. Dado que estás estrategias se han explorado para otros ámbitos de la Ingeniería de Software [6], se realiza la propuesta de aplicar este conocimiento para clasificar requisitos de software.

De esta forma, el objetivo principal de este estudio es aplicar el algoritmo de evolución diferencial DE/rand/1/bin [22] para explorar combinaciones de hiperparámetros que permitan maximizar la precisión de un clasificador de requisitos de software.

3. Contribución

En este estudio, se presenta una propuesta de optimización de hiperparámetros para un modelo de clasificación Naïve Bayes. Para este fin se estableció un procedimiento de extracción de características para el entrenamiento del modelo de aprendizaje máquina, y posteriormente se realizó la calibración del mismo utilizando Evolución diferencial. La Figura 1 muestra el proceso llevado a cabo.

3.1. Clasificación de requisitos

La compatibilidad de texto con el clasificador por aprendizaje máquina se realizó a través de un proceso de vectorización y codificado de etiquetas. Este procedimiento de extracción de características se realizó de la siguiente manera:

1. Remoción de caracteres especiales. De esta forma se quitan caracteres de salto de línea o tabulaciones.
2. Conversión de texto a minúsculas.
3. Remoción de signos de puntuación. Tales como puntos, comas y signos de admiración o interrogación.
4. Lematización. Así, se reemplazan los sufijos de las palabras para obtener una palabra raíz más básica. De esta forma se reduce la variabilidad de palabras [9].
5. Remoción de palabras vacías (*stop words*). Se retiran palabras predefinidas que no enriquecen semánticamente el enunciado.
6. Vectorización por TF-IDF. *Term Frequency - Inverse Document Frecuency* por sus siglas en inglés es una técnica de representación de términos a través de su peso en una determinada categoría [24]. Esta representación de términos por pesos está definida por la Ecuación 1:

$$W(d,t) = TF(d,t) \times log(\frac{N}{df(t)}). \tag{1}$$

A partir de este modelo es posible calcular el peso de un término con respecto a la frecuencia con la que aparece en un conjunto de texto $W(d,t)$. Donde t corresponde al término y d al documento. Como primer factor en esta ecuación se calcula la frecuencia del término (*Term Frequency*), al mismo tiempo que la

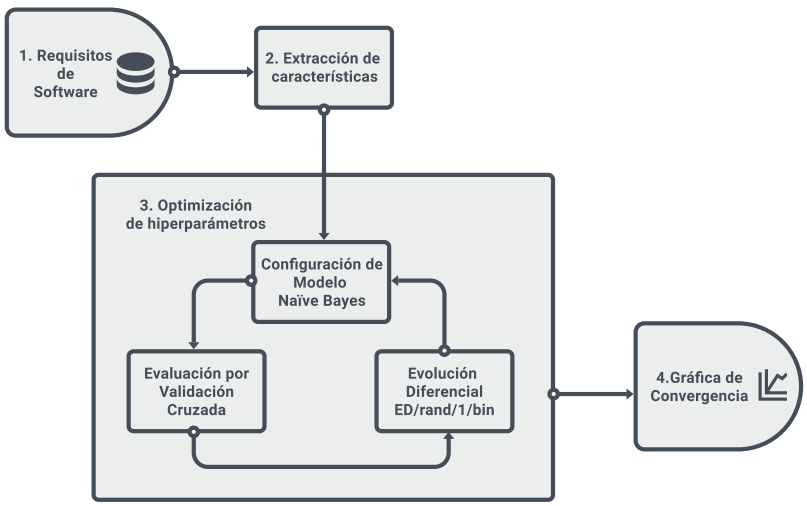

Fig. 1. Procedimiento para la optimización del modelo de aprendizaje máquina.

frecuencia inversa en relación a los documentos (*Inverse Document Frequency*) se representa en el segundo. En este factor se calcula qué tan específico es un término, donde N es el número total de textos y $df(t)$ es el número de textos que contienen al término t [19].

Una vez aplicada la transformación de los conjuntos de datos, se realiza la selección y entrenamiento con valores por defecto del clasificador. Se selecciona para este estudio el modelo MultinomialNB incluido en la biblioteca sklearn [17]. Para este fin, el clasificador seleccionado únicamente cuenta con dos hiperparámetros, denominados `alpha` y `fit_prior`, siendo valores de tipo real y booleano respectivamente. Para corroborar el proceso de optimización, se utilizó el modelo de Naïve Bayes con su configuración por defecto, y se comparó con un modelo con el mismo entrenamiento, pero con los parámetros resultantes del proceso de evolución diferencial.

3.2. Optimización por evolución diferencial

Con el objetivo de maximizar la precisión del modelo, se implementó el algoritmo de evolución diferencial de DE/rand/1/bin [22]. La representación para el conjunto de soluciones se realiza de forma vectorial, y se inicializa de forma aleatoria a lo largo de una población inicial.

Para la ejecución de este algoritmo, únicamente se definen cuatro valores. El número de individuos presentes en cada generación, la cantidad de generaciones que se ejecutarán; un factor de escala que representa la diferencia para calcular el vector de mutación, y finalmente el porcentaje de cruza, que determina el parecido entre individuos padres e hijos.

Este proceso evolutivo se realiza a lo largo de cada generación, en la que prevalecen aquellas combinaciones de hiperparámetros que generan una precisión mayor. La evaluación se realiza través de la configuración de un modelo MultinomialNB con los parámetros de los individuos. Para este proceso se define como función de aptitud a maximizar, la validación cruzada. Lo anterior con el fin de minimizar el sobre ajuste al conjunto de datos de entrenamiento.

Debido a que el proceso de mutación y cruza resulta en individuos con parámetros aleatorios que pueden exceder los límites deseados, se utilizó la reflexión como técnica de manejo de restricciones de límite. Esta estrategia se expresa por la Ecuación 2, así se reingresa cualquier valor fuera de los límites [18]:

$$x_i^c = \begin{cases} x_i & \text{si } l_i \leq x_i \leq u_i, \\ 2 \times l_i - x_i & \text{si } x_i < l_i, \\ 2 \times u_i - x_i & \text{si } x_i > u_i. \end{cases} \tag{2}$$

3.3. TPOT

Adicionalmente, se comparó el proceso de optimización aplicado con la biblioteca TPOT. Esta herramienta realiza la configuración automática de una cadena de modelos de *machine learning* denominada *pipeline* [16]. A través de

un proceso de optimización por programación genética [3], esta herramienta es capaz de diseñar y configurar *pipelines* de transformaciones de datos y modelos de aprendizaje máquina con el fin de maximizar la precisión de un modelo de clasificación [15]. La Figura 2 muestra el procedimiento que involucra esta herramienta y su uso para la clasificación de requisitos de software.

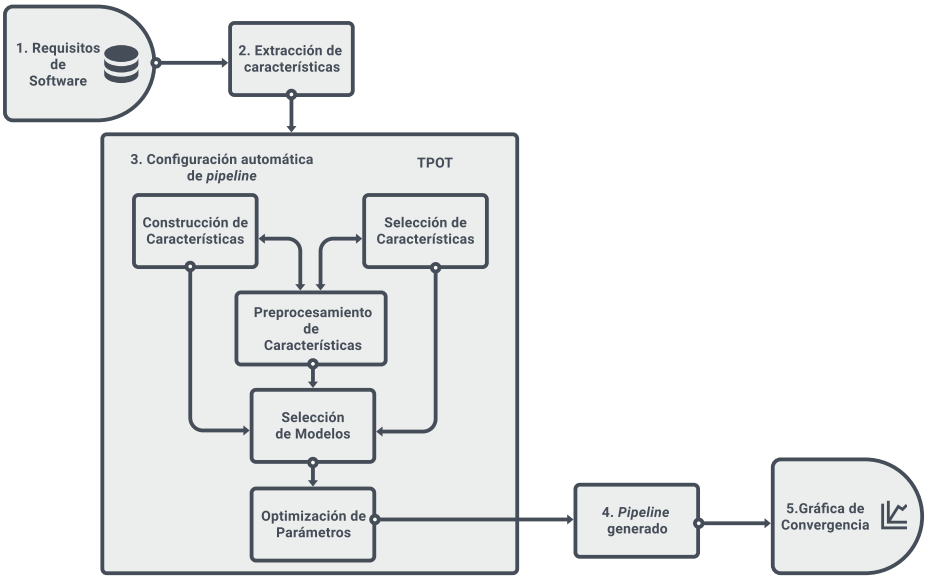

Fig. 2. Proceso de configuración de modelo generado por la herramienta TPOT.

4. Experimentos y resultados

Para evaluar la precisión del clasificador con los requisitos de software, se utilizaron dos bases de datos con enunciados de este tipo, la base de PROMISE [20] y 40 ejemplos recolectados de *The Quest for Software Requirements* [14]. Este proceso de extracción y optimización propuesto se puede observar en la Figura 1. De estas dos bases de datos, se recolectaron aquellos requisitos que estuvieran asociados con los Atributos de Calidad siguientes: Disponibilidad, Tolerancia a Fallos, Mantenibilidad, Rendimiento, Escalabilidad, Seguridad y Usabilidad.

A partir de los requisitos en formato de texto, se realizó el proceso de extracción de características antes descrito, con el cual se representó el formato de lenguaje en espacios de vectores para su subsecuente uso en el entrenamiento del modelo Naïve Bayes configurado con parámetros por defecto.

Table 1. Distribución de las clases del conjunto de entrenamiento.

Atributo de Calidad	Frecuencia
Tolerancia a Fallos	10
Mantenibilidad	25
Disponibilidad	29
Escalabilidad	29
Rendimiento	54
Seguridad	74
Usabilidad	75

Estos valores, de acuerdo a la documentación de la biblioteca sklearn [17] son $alpha = 1.0$ y $fit\_prior = True$. Evaluado a través de validación cruzada por 5 dobleces, este modelo obtuvo una precisión ($accuracy$) promedio de 0.546.

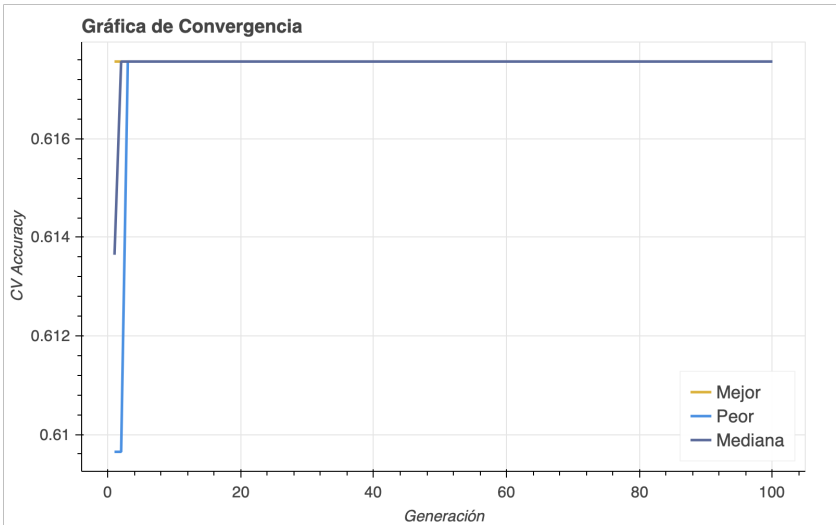

Fig. 3. Gráfica de convergencia del algoritmo DE/rand/1/bin.

Para la aplicación del algoritmo de evolución diferencial, se realizó una configuración basada en el estudio [13]. De esta forma, los parámetros con los que se realizaría el proceso de evolución son un tamaño de población de 100 individuos, a través de 100 generaciones; usando un factor de escala de 0.9 y un porcentaje de cruza de 1.0. Como función de aptitud, los individuos fueron evaluados en los mismos términos que el modelo por defecto. De esta forma, el proceso evolutivo se llevó a cabo con el objetivo de maximizar la precisión promedio del modelo, evaluada por validación cruzada.

Así mismo, y dado que la aplicación de Evolución Diferencial se enfoca en exploración de espacios continuos [23], se limitaron espacios de búsqueda de los parámetros, siendo el espacio $[0, 100]$ para el valor *alpha* y $[0, 1]$ para fit_prior. Para este último dado que es de tipo booleano, se determinó que aquellas soluciones con un valor de 0.5 o superior se consideraría como "Verdadero", mientras que las soluciones menores a 0.5 serían interpretadas como un valor "Falso".

Finalmente, y dada la naturaleza no determinista de este tipo de algoritmos [23], se realizaron 30 ejecuciones del proceso de Evolución Diferencial. De estas, se identificaron y compararon aquellas con la mejor, peor y mediana aptitud obtenida. La Figura 3 muestra la gráfica de convergencia de estas ejecuciones. En esta representación se muestra el proceso de optimización con respecto de la precisión del mejor individuo de una generación, a lo largo del proceso evolutivo. En esta gráfica, se observa que el algoritmo alcanza una convergencia durante las primeras generaciones. Notablemente, todas las ejecuciones alcanzaron este estado con una precisión de 0.617, aún con combinaciones de parámetros diferentes. Esto se puede deber a muchos factores, tales como el espacio de búsqueda de los parámetros, la cantidad de los mismos (pues en el caso de este modelo son dos), el método de representación de lenguaje seleccionado o el dataset mismo.

De la misma forma, se realizaron 30 ejecuciones de la herramienta TPOT con el mismo dataset preprocesado. El objetivo de optimización del algoritmo se configuró para ser el mismo que el aplicado para el DE/rand/1/bin, siendo maximizar el valor de precisión por validación cruzada, en 5 dobleces. La gráfica de convergencia en la Figura 4 muestra la comparación de las ejecuciones mejor, peor y mediana. En esta gráfica se puede apreciar la forma en la que gradualmente se incrementa el valor de precisión. En el caso de la mejor ejecución, se logró obtener una precisión de 0.697, mientras que en la peor se logra 0.661, ambas puntuaciones superando al máximo alcanzado por el clasificador Naïve Bayes optimizado. Adicionalmente, la información de las 30 ejecuciones de ambos algoritmos, así como su comparación son accesibles en el repositorio del proyecto[3].

5. Conclusiones y trabajo futuro

Las aplicaciones de aprendizaje automático (*AutoML*) así como de optimización son de especial interés en el área de Ingeniería de Software, pues su uso ha mostrado resultados favorables en diferentes etapas del desarrollo de sistemas, dada la cantidad de procesos aptos para ser optimizados [6,26]. Con el interés de clasificar de forma automática requisitos de software, se realizó el entrenamiento de un modelo Naïve Bayes. De la misma forma, se aplicó el algoritmo de Evolución Diferencial DE/rand/1/bin, que como objetivo tuvo maximizar la precisión para clasificar este tipo de información. Adicionalmente, se probó el uso de una herramienta de generación de secuencias de modelos, conocida como TPOT. Esta última implementación mostró una estrategia de

[3] https://github.com/quality-attributes/optimization

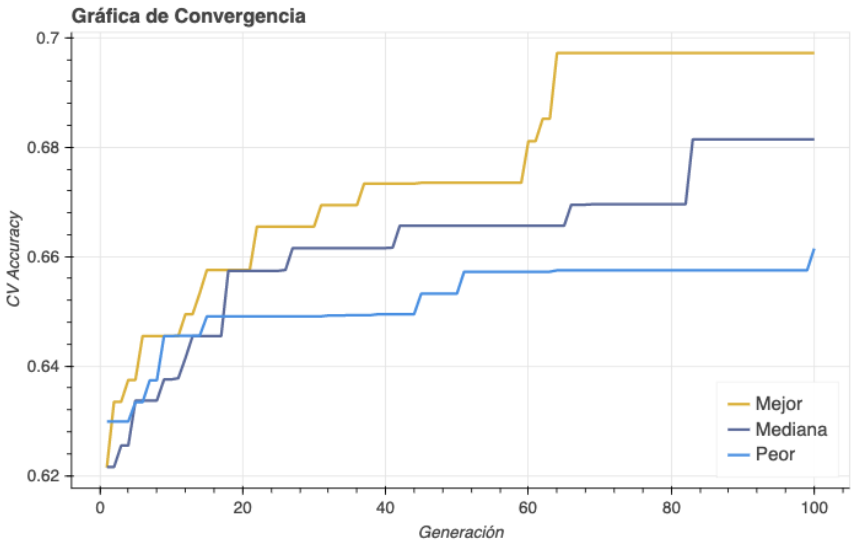

Fig. 4. Gráfica de convergencia de TPOT.

diseño generación de modelos más elaborados, calibrados a partir de programación genética. Ambos acercamientos mostraron resultados favorables para la calibración de parámetros de modelos de aprendizaje máquina, pues demostraron capacidad para incrementar su precisión. De está forma, fue posible mejorar la precisión promedio del clasificador con los parámetros por defecto, pasando de 54.6% a 61.7% una vez ejecutado el proceso de calibración Naïve Bayes mediante Evolución Diferencial. No obstante, el *pipelane* generado por la herramienta de TPOT mostró una puntuación superior al modelo optimizado, obteniendo 69.7% en términos de precisión.

Durante la conducción del presente estudio, se identificaron áreas de oportunidad y trabajo futuro para la aplicación de algoritmos de optimización en la clasificación de requisitos de software. Particularmente, se identifica la implementación de modelos de aprendizaje máquina adicionales, con el objetivo de complementar los resultados obtenidos con el modelo Naïve Bayes. De esta forma, se pueden examinar modelos con más hiperparámetros, permitiendo la exploración en más espacios de búsqueda, y posiblemente de más combinaciones que incrementen la precisión de los mismos.

Finalmente, se plantea la aplicación de las técnicas de optimización en más etapas del proceso de clasificación de requisitos propuesto, especialmente en el procedimiento de representación de texto. Así, se propone la exploración de más procedimientos de procesamiento de lenguaje [9], así como sus respectivas configuraciones. Lo anterior el objetivo de incrementar la compatibilidad de los modelos con el conjunto de datos de requisitos de software.

References

1. Alcalá, R., Gacto, M.J., Alcalá-Fdez, J.: Evolutionary data mining and applications: A revision on the most cited papers from the last 10 years (2007–2017). WIREs Data Mining and Knowledge Discovery 8(2) (mar 2018), https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1239
2. Ambriola, V., Gervasi, V.: On the Systematic Analysis of Natural Language Requirements with CIRCE. Automated Software Engineering 13(1), 107–167 (jan 2006), http://link.springer.com/10.1007/s10515-006-5468-2
3. Banzhaf, W., Francone, F.D., Keller, R.E., Nordin, P.: Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
4. Barros, R.C., Basgalupp, M.P., de Carvalho, A.C.P.L.F., Freitas, A.A.: A Survey of Evolutionary Algorithms for Decision-Tree Induction. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42(3), 291–312 (may 2012)
5. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Professional, 3rd edn. (2012)
6. Chen, T., Li, M., Li, K., Deb, K.: Search-Based Software Engineering for Self-Adaptive Systems: One Survey, Five Disappointments and Six Opportunities (jan 2020), http://arxiv.org/abs/2001.08236
7. Glinz, M.: A Glossary of Requirements Engineering Terminology. Tech. rep., International Requirements Engineering Board (2017)
8. Jindal, R., Malhotra, R., Jain, A.: Automated classification of security requirements. In: 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI). pp. 2027–2033 (2016)
9. Kowsari, K., Meimandi, K.J., Heidarysafa, M., Mendu, S., Barnes, L., Brown, D.: Text classification algorithms: A survey (2019)
10. Kurtanović, Z., Maalej, W.: Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning. In: 2017 IEEE 25th International Requirements Engineering Conference (RE). pp. 490–495 (2017)
11. Lu, M., Liang, P.: Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering. pp. 344–353. EASE'17, ACM, New York, NY, USA (2017)
12. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Network Modeling Analysis in Health Informatics and Bioinformatics 5(1), 18 (dec 2016)
13. Mezura-Montes, E., Miranda-Varela, M.E., del Carmen Gómez-Ramón, R.: Differential evolution in constrained numerical optimization: An empirical study. Inf. Sci. 180(22), 4223–4262 (Nov 2010)
14. Miller, R.E.: The Quest for Software Requirements. MavenMark Books, Oconomowoc, WI, USA (2009)
15. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 485–492. GECCO '16, ACM, New York, NY, USA (2016)
16. Olson, R.S., Urbanowicz, R.J., Andrews, P.C., Lavender, N.A., Kidd, L.C., Moore, J.H.: Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings,

Part I, chap. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pp. 123–137. Springer International Publishing (2016)
17. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011)
18. Ronkkonen, J., Kukkonen, S., Price, K.: Real-Parameter Optimization with Differential Evolution. In: 2005 IEEE Congress on Evolutionary Computation. vol. 1, pp. 506–513. IEEE (2005)
19. Salton, G.: Automatic text processing: The transformation, analysis, and retrieval of information by computer (1989)
20. Sayyad Shirabad, J., Menzies, T.J.: The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada (2005), http://promise.site.uottawa.ca/SERepository
21. Sommervile, I., Sawyer, P.: Requirements Engineering: A Good Practice Guide. John Wiley & Sons, Inc., New York, United States (1997)
22. Storn, R.: On the usage of differential evolution for function optimization. In: Proceedings of North American Fuzzy Information Processing. pp. 519–523 (jun 1996)
23. Storn, R., Price, K.: Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. Journal of Global Optimization 23 (01 1995)
24. Tokunaga, T., Iwayama, M.: Text categorization based on weighted inverse document frequency (1994)
25. Wiegers, K.E., Beatty, J.: Software Requirements. Microsoft Press, USA, 3rd edn. (2013)
26. Zhang, D., Tsai, J.J.P.: Machine Learning and Software Engineering. Software Quality Journal 11(2), 87–119 (2003)