ISSN 2007-9737

Detección automática de similitud entre programas del lenguaje de programación Karel basada en técnicas de procesamiento de lenguaje natural

Grigori Sidorov[1], Martín Ibarra Romero[1], Ilia Markov[1], Rafael Guzman-Cabrera[2], Liliana Chanona-Hernández[3], Francisco Velásquez[4]

[1] Instituto Politécnico Nacional (IPN), Centro de Investigación en Computación (CIC), México D.F., México

[2] Universidad de Guanajuato, División de ingenierías, Campus Irapuato-Salamanca, México

[3] Instituto Politécnico Nacional (IPN), Escuela Superior de Ingeniería Mecánica y Eléctrica (ESIME), México D.F., México

[4] Universidad Politécnica de Querétaro, México

sidorov@cic.ipn.mx, maibarraromero@yahoo.com.mx, markovilya@yahoo.com, guzmanc81@gmail.com, lchanona@gmail.com, francisco.castillo@upq.mx

Resumen. Este artículo presenta un método para calcular la similitud entre programas (código fuente). La tarea es útil, por ejemplo, para la clasificación temática de programas o detección de reuso de código (digamos, en el caso de plagio). Usamos para los experimentos el lenguaje de programación Karel. Para determinar la similitud entre programas y/o ideas de soluciones similares utilizamos un enfoque basado en técnicas de procesamiento de lenguaje natural y de recuperación de información. Estas técnicas usan la representación de un documento como un vector de valores de características. Usualmente, las características son *n*-gramas de palabras o de caracteres. Posteriormente, se puede aplicar el análisis semántico latente para reducir la dimensionalidad de este espacio vectorial. Finalmente, se usa el aprendizaje automático supervisado para la clasificación de textos (o programas que son textos también) parecidos. Para validar el método propuesto, se compiló un corpus de programas para 100 tareas diferentes con un total de 9,341 códigos y otro corpus para 34 tareas adicionalmente clasificado por la idea de solución, formado por 374 códigos. Los resultados experimentales muestran que para el corpus con ideas de solución es mejor la representación con trigramas de caracteres, mientras que para el corpus completo los mejores resultados se obtienen con trigramas de términos y la aplicación del análisis semántico latente.

Palabras clave. Similitud, *n*-gramas, programa, código fuente, análisis semántico latente, recuperación de información, procesamiento de lenguaje natural.

Automatic Detection of Similarity of Programs in Karel Programming Language based on Natural Language Processing Techniques

Abstract. In this paper, we present a method for calculating similarity between programs (source codes). One of the applications of the task is detection of code reuse, for example, in the case of plagiarism. The Karel programming language is used for experiments. In order to determine similarity between Karel programs and/or similar software solutions, we make use of techniques from the fields of natural language processing and information retrieval. These techniques use representations of documents as vectors of features and their values. Usually, the features are *n*-grams of words or *n*-grams of characters. In addition, we consider application of the latent semantic analysis for reduction

of the number of dimensions of the vector space. Finally, we use a supervised machine learning approach for classification of texts (or programs, which are texts as well) based on their similarity. For evaluation of the proposed method, two corpora were developed: the first corpus is composed of 100 different programs with a total of 9,341 source codes. The second corpus consists of 34 tasks with a total of 374 codes, which are grouped by the proposed solution. Our experiments showed that for the first corpus, the best results were obtained using trigrams of terms (words) accompanied with application of latent semantic analysis, while for the second corpus, the best representation was achieved using character trigrams.

Keywords. Similarity, *n*-grams, program, source code, latent semantic analysis, information retrieval, natural language processing.

1. Introducción

El cálculo de similitud es una tarea básica y primaria para las áreas donde se manipula la información simbólica, como procesamiento de lenguaje natural o análisis de imágenes.

Los programas en cualquier lenguaje de programación en cierto sentido son textos, por lo que se puede aplicar a ellos las técnicas de procesamiento de lenguaje natural o de recuperación de información [3], por ejemplo, calcular su similitud. Tal cálculo de similitud puede ser útil, por ejemplo, para detectar programas que resuelvan el mismo problema (clasificación temática de código de programas). Esta tarea también es útil para apoyar a los docentes que se dedican a enseñar técnicas de programación, por ejemplo, para detectar el reuso no autorizado de código (un posible plagio). En el presente trabajo utilizamos los programas desarrollados en el lenguaje Karel [11] para probar nuestro método de detección de similitud de códigos fuente.

Hay que notar que ambos problemas (clasificación y reuso de código) son sumamente difíciles y no existe algún método simple que los pueda resolver. Digamos, una línea base (un método simple), para la tarea de clasificación, sería el método aleatorio, que da un resultado relacionado con el número de clases. Por ejemplo, si tenemos 10 clases con frecuencias similares, la precisión de este método base sería 10 %.

Los primeros programas para detección de similitud o plagio se desarrollaron por Halstead [7] y McCabe [10]. Posteriormente, el programa de Wise [17] hace comparación línea a línea usando la distancia de Levenshtein (similitud de cadenas). Gitchell y Tran [4] trabajaron en base al análisis del árbol que genera un programa. Trabajos más recientes [1] además utilizan el análisis semántico latente (*latent semantic analysis*, LSA). En fechas recientes la idea de relacionar la semántica del código fuente con la de la documentación, definición de variables, nombres de procedimientos y/o comentarios del mismo ha surgido para ayudar a mejorar el entendimiento de datos, los requerimientos y diseño de los códigos fuente [8]. Un ejemplo es el programa “Moss” [13]. “Moss” compara dos programas según la similitud de sus “huellas”. Para medir la similitud también a menudo se usa la medida de coseno o su modificación, el coseno suave [15], que permite tomar en cuenta la similitud entre características en un modelo de espacio vectorial.

Cabe mencionar que el uso de los *n*-gramas (secuencias de palabras o caracteres) como características para la clasificación permite obtener muy buenos resultados en tareas relacionadas con el aprendizaje automático para textos. Existen otras opciones como *n*-gramas sintácticos [12], grafos sintácticos integrados [5], distancia de edición de árboles [16], etc., que pueden ser utilizados en el modelo de espacio vectorial [14].

Desde el punto de vista de aprendizaje automático el problema de clasificación de códigos fuente se formula de la siguiente manera: utilizando un conjunto de códigos como datos de entrenamiento (note que cada código pertenece a una clase dependiendo de su idea de solución, por ejemplo), necesitamos entrenar un clasificador. Después para un nuevo código debemos decidir a qué clase él pertenezca (qué idea de solución se expresa en él).

Un programa “P” se define por un conjuntos de elementos (rutinas, procedimiento, funciones, métodos). Decimos que un programa “P” es un conjunto $\{p_1, p_2, \ldots, p_n\}$. De la misma manera existe otro programa “Q” con su conjunto $\{q_1, q_2, \ldots, q_m\}$. Necesitamos encontrar elementos similares, es decir, una correspondencia entre los

elementos de “P” y “Q”. Note que la similitud entre programas se puede dar total o parcial, esta parcialidad la podemos ver cuando los procedimientos y/o rutinas entre dos o más programas presentan cierto grado de similitud.

2. Preparación de los datos

Para realizar las mediciones de similitud entre programas se formaron tres corpus de prueba, basados en el sitio con programas en lenguaje Karel (Karelotitlan[1]).

La selección fue hecha de la siguiente manera: de cada problema se cuantificó la cantidad de programas que diferentes personas habían enviado, este valor se dividió entre 100 dando un factor “X”, posteriormente en base a la lista de programas se tomaron 100 veces (siempre que fuera posible) 1 programa cada “X” programas (ósea se tomaba 1 y nos saltábamos “X”, tomábamos otro y nos saltábamos “X”, etc.).

De esta madera conformamos los tres siguientes corpus de datos:

Corpus **“K_ideas_11”** consta de 374 programas, aproximadamente 34 por problema. Los programas son clasificados por problema y por idea de solución. Se tomaron 11 problemas y en cada problema se agruparon por idea de solución.

Corpus **“K_100”** consta de 100 problemas y de aproximadamente 100 programas por problema dando un total de 9,341 programas. Note que aquí los programas solo están clasificados por problema. Los programas dentro del sitio web de “Karelotitlan” se encuentran precisamente agrupados por persona/alumno y problema, de aquí tomamos los 100 problemas.

Corpus **“K_100_50”** está constituido por un subconjunto de 4,598 programas de los 100 problemas seleccionados anteriormente con la finalidad de poder realizar los cálculos con mayor velocidad.

Con la finalidad de ilustrar el flujo de un programa implementado en Karel, a continuación se muestra el problema “Isla” que consiste en encontrar el camino desde el lugar donde nos encontramos hasta otro lugar que se encuentra en la esquina inferior izquierda. En el problema de la isla tenemos dos ideas de solución etiquetadas como “idea 2” e “idea 3”, las cuales se muestran en las Figuras 1 y 2 respectivamente.

“1” en la figura significa el zumbador (un elemento de lenguaje Karel, cuando el robot puede poner el zumbador), con líneas se marcan las “paredes”.

En la idea 2 agrupamos todos los programas que realizan lo siguiente: para buscar la salida llegan a una esquina y de cada esquina, marcándola con zumbadores intentan salir, en caso de lograrlo se ubican en la casilla (1,1).

En la idea 3 agrupamos todos los programas que rodean la isla con zumbadores de manera que hace un recorrido alrededor de la isla buscando salir perpendicularmente y si se encuentran con una pared sin zumbadores significa que están fuera de la isla y basta con dirigirse a la casilla (1,1), los dos primeros corpus están disponibles desde nuestro sitio Web[2].

Tabla 1. Ejemplo de código original y después de la etapa de preprocesamiento

Programa original	Programa preprocesado
inicia-ejecucion (* Programa de ejemplo *) mientras no-orientado- al-norte inicio hacer gira-izquierda; fin; mientras frente-libre hacer inicio cuenta-compara(0); fin; (* Termina *) apagate; termina-ejecucion	inicia-ejecucion mientras no-orientado- al-norte inicio hacer gira-izquierda fin mientras frente-libre hacer inicio cuenta-compara fin apagate termina-ejecucion

Antes de procesar el texto usualmente se realiza su preprocesamiento para eliminar los elementos que se consideran como inútiles. Eliminamos los términos de sintaxis de lenguaje: “(”, “)” “;” y las palabras del propio lenguaje como “entonces”, “veces”, “hacer”, “como”, “define-nueva-instrucción”, “iniciar-programa”, “finalizar-programa”. Cabe hacer mención que todos los preprocesamiento

[1] http://www.cmirg.com/Karelotitlan/

[2] http://www.cic.ipn.mx/∼sidorov

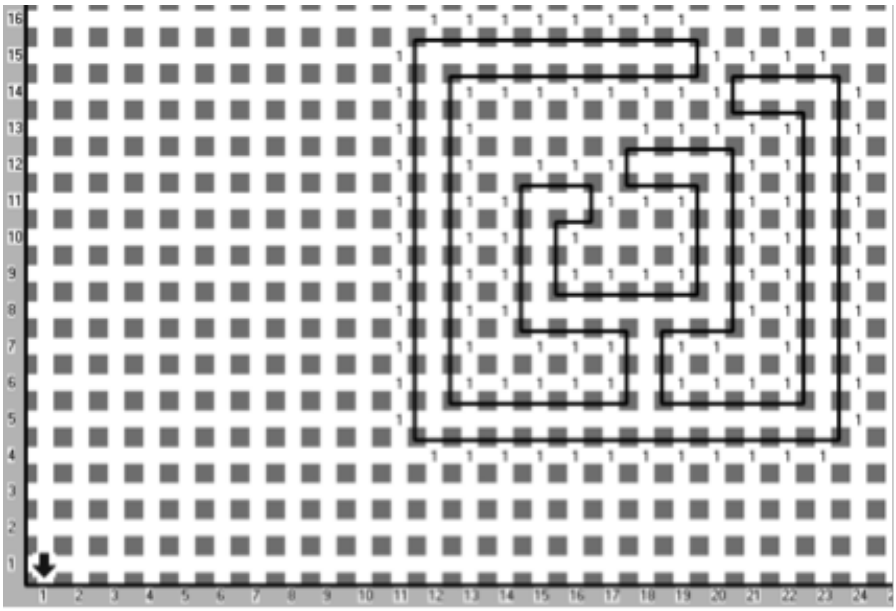

Fig. 1. Idea de solución 2

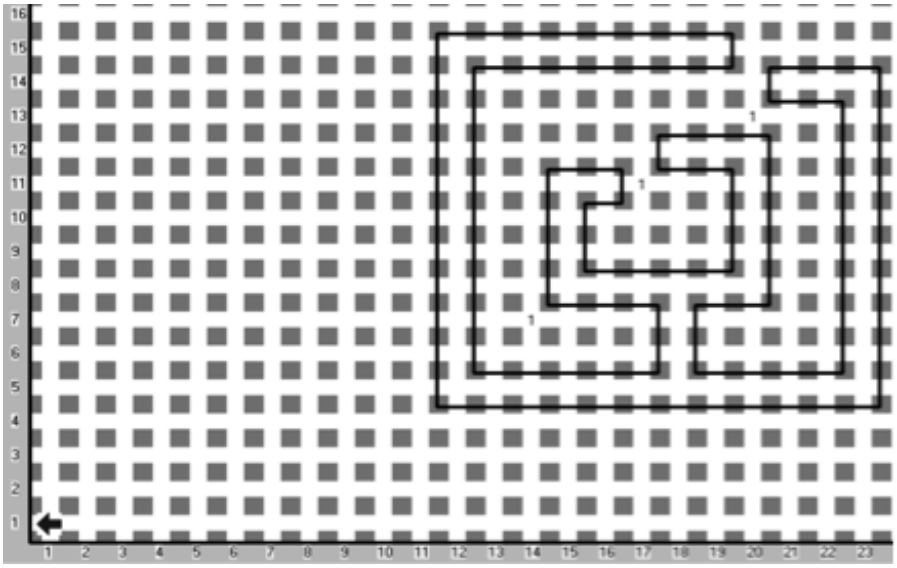

Fig. 2. Idea de solución 3

incluían la eliminación de los comentarios de programas.

También para los experimentos realizados en el presente trabajo, únicamente se consideraron las palabras con frecuencia de dos y mayor. Los términos únicos (con frecuencia de uno) fueron eliminados con la finalidad de disminuir la dimensión del vector y tener un menor coste computacional.

En la Tabla 1 se muestra, a manera de ejemplo, una sección de código original y después de la etapa de preprocesamiento.

Tabla 2. Características de los experimentos realizados donde la columna "N" se refiere al número dado al experimento, "TS" es "Término Sintáctico", "Norm." es "Normalizado", "Núm." es "Incluye Números", "TU" es "Términos Únicos", "PC" es "Palabras Clave", "SW" es "Stopwords", "+" significa que se incluyó y "–" que no se incluyó

N	Tamaño de n-gramas	Pesado	Término	TS	Norm.	Núm.	Términos Longitud 1	TU	PC	SW
1	1	tf	caracter	+	–	+	+	+	+	+
2	3	tf	caracter	+	–	+	+	+	+	+
Línea base (3)	1	tf	palabra	+	–	+	+	+	+	+
4	3	tf	palabra	+	–	+	+	+	+	+
5	2	tf	palabra	+	–	+	+	+	+	+
6	3	tf	palabra	–	–	–	–	–	–	–
7	4	tf	palabra	+	–	+	+	+	+	+
8	1	tf-idf	caracter	+	–	+	+	+	+	+
9	3	tf-idf	caracter	+	–	+	+	+	+	+
10	1	tf-idf	palabra	+	–	+	+	+	+	+
11	3	tf-idf	palabra	+	–	+	+	+	+	+
12	2	tf-idf	palabra	+	–	+	+	+	+	+
13	3	tf-idf	palabra	–	–	–	–	–	–	–
14	4	tf-idf	palabra	+	–	+	+	+	+	+
15	3	tf-idf	palabra	–	+	–	+	+	–	–
16	2	tf	palabra	–	+	+	+	+	–	–
17	3	tf-idf	palabra	+	+	+	+	+	–	–

3. Cálculo de similitud entre los códigos fuente

Para el cálculo de similitud entre los códigos fuente se usan los valores estándar en el espacio vectorial construido: tf y tf-idf [9] y se aplica (o no) una reducción de dimensiones, para lo cual utilizamos el análisis semántico latente (LSA) [2]. Se hicieron los experimentos [6] con aplicación de algoritmos *Naive Bayes* (NB) y *Support Vector Machines* (SVM) en su configuración estándar.

A continuación se presentan los resultados arrojados por los experimentos así como la descripción de los mismos. En la Tabla 2 describimos las características de preprocesamiento de cada experimento. Como línea base (*base line*), se utilizó el conjunto de características de palabras con el pesado tf sin reducción de dimensiones.

Ahora presentamos los resultados obtenidos en los experimentos realizados así como la descripción de los mismos para los corpus utilizados. Es necesario mencionar que se presentan únicamente los resultados más representativos.

Los resultados para el corpus **"K_ideas_11"** se presentan en la Tabla 3. La columna "N" corresponde al número de experimento tomado de la Tabla 2. LSA (número) significa el número de las dimensiones. Los mejores resultados para cada experimento y para el algoritmo correspondiente están marcados con negrita.

Para el corpus **"K_ideas_11"**, el algoritmo SVM muestra mejores resultados de clasificación que *Naive Bayes*, los tres más altos valores de precisión son 55.91 %, 53.70 % y 53.63 %. El mejor resultado se obtiene cuando no hacemos reducción de dimensiones y realizamos el pesado

Tabla 3. Resultados (exactitud, %) para el corpus "K_ideas_11"

N	Algoritmo	Sin LSA	LSA (50)	LSA (100)	LSA (150)	LSA (200)	LSA (300)	LSA (400)	LSA (500)	LSA (600)
1	NB	**47.23**	44.72	44.61	44.50	44.61	44.50	44.61	44.72	44.45
1	SVM	37.06	39.20	39.25	38.98	39.15	39.30	**39.36**	39.30	39.15
2	NB	**50.40**	49.63	44.14	41.59	38.26	38.26	37.86	49.63	37.99
2	SVM	**55.62**	45.44	48.00	45.46	30.89	30.89	31.28	45.44	31.02
3	NB	**50.54**	47.07	46.53	44.28	42.13	36.53	36.93	36.53	36.53
3 (Línea base)	SVM	40.37	**39.43**	46.13	44.65	41.71	32.76	33.28	32.89	32.49
4	NB	**51.75**	44.90	46.13	46.66	43.05	39.03	39.30	39.43	39.43
4	SVM	**51.72**	41.30	50.41	48.81	45.20	36.24	36.10	36.23	35.97
5	NB	**49.88**	46.67	45.60	46.26	42.27	37.17	36.90	36.90	38.10
5	SVM	51.21	44.25	**53.70**	50.53	45.06	34.63	34.37	34.90	34.63
6	NB	**47.20**	42.12	42.40	42.64	40.27	34.78	34.24	34.24	34.78
6	SVM	**45.70**	35.81	44.50	44.25	42.90	37.56	37.70	37.03	37.57
7	NB	**50.12**	45.32	47.86	47.46	43.32	42.38	41.72 4	2.25	41.72
7	SVM	**51.73**	40.36	50.53	47.20	43.59	35.59	35.32	35.58	35.45
8	NB	25.52	26.20	25.39	26.20	26.20	25.53	25.39	26.34	**26.34**
8	SVM	29.41	29.54	29.68	29.81	29.81	**29.94**	29.54	29.54	29.81
9	NB	**50.92**	45.60	40.66	38.11	35.16	33.16	33.43	33.43	32.89
9	SVM	**55.91**	50.28	50.53	49.46	48.13	40.53	40.80	40.80	40.53
10	NB	**49.05**	39.05	38.00	35.31	34.64	33.96	33.56	33.96	34.10
10	SVM	**45.33**	42.50	46.40	43.18	41.31	39.85	39.71	39.58	39.58
11	NB	**52.81**	47.20	45.34	42.54	38.90	38.37	37.97	37.97	37.83
11	SVM	53.62	51.20	**53.08**	50.27	45.59	38.39	38.25	38.39	38.12
12	NB	**52.14**	45.47	43.88	40.54	37.60	37.04	37.18	37.44	37.18
12	SVM	**54.55**	52.01	51.74	49.86	48.12	42.53	42.40	42.53	42.27
13	NB	**45.85**	39.86	39.04	35.96	35.54	33.71	33.57	33.45	33.84
13	SVM	**47.98**	43.17	45.83	44.90	44.24	39.72	40.25	40.12	40.12
14	NB	**49.47**	43.59	45.99	41.59	39.32	38.36	38.36	38.63	38.63
14	SVM	51.60	51.07	**53.63**	50.15	46.53	41.45	41.72	41.71	41.45
15	NB	**44.65**	40.67	39.59	37.33	35.31	35.71	35.18	35.85	35.98
15	SVM	**47.72**	44.24	46.67	43.98	41.18	37.83	37.83	37.43	37.70
16	NB	**47.86**	43.61	44.54	43.20	41.07	37.17	37.17	37.03	37.17
16	SVM	**47.99**	34.22	45.31	45.18	42.51	37.97	37.97	37.97	37.84
17	NB	**49.99**	45.73	45.21	41.74	39.60	38.79	39.05	38.51	38.65
17	SVM	**50.95**	49.47	51.35	47.46	44.79	39.73	39.73	39.73	39.86

Tabla 4. Tres resultados más altos para el corpus "K_ideas_11"

Algoritmo	Término	Tamaño de *n*-gramas	Pesado	Todas dimensiones (sin LSA)	LSA 100
Línea base (NB)	palabra	1	tf	**50.54**	46.53
SVM	caracter	3	tf-idf	**55.91**	50.53
SVM	palabra	2	tf	51.21	**53.70**
SVM	palabra	4	tf-idf	51.60	**53.63**

Tabla 5. Resultados (exactitud, %) para el corpus "K_100"

N	**Algoritmo**	**Sin LSA**	**LSA (50)**	**LSA (100)**	**LSA (150)**	**LSA (200)**	**LSA (300)**	**LSA (400)**	**LSA (500)**	**LSA (600)**
1	NB	13.07	**21.86**	19.20	19.20	19.19	19.19	19.19	19.19	19.19
1	SVM	4.58	8.25	9.27	9.35	9.24	9.35	9.28	9.32	**9.39**
2	NB	**49.05**	29.93	32.40	33.91	34.55	34.34	34.22	33.51	32.48
2	SVM	58.59	11.18	21.34	32.70	40.32	52.07	56.21	58.27	**59.42**
3	NB	**39.27**	24.24	23.71	21.51	20.99	21.99	22.50	22.93	22.06
3 (Línea base)	SVM	**43.25**	6.97	12.44	15.02	17.93	23.33	27.31	31.66	34.35
4	NB	**55.36**	24.76	30.34	32.77	33.81	35.00	35.93	36.32	36.29
4	SVM	55.69	5.99	16.71	29.07	38.08	51.00	56.65	61.25	**63.89**
11	NB	**58.81**	21.96	30.84	36.53	39.54	43.04	44.74	45.29	45.17
11	SVM	68.56	34.50	52.23	60.35	65.03	68.42	69.85	**69.94**	69.88

Tabla 6. Tres resultados más altos para el corpus 'K_100"

Algoritmo	Término	Tamaño de *n*-gramas	Pesado	Todas dimensiones (sin LSA)	LSA 400	LSA 500	LSA 600
Línea base (SVM)	palabra	1	tf	**43.25**	27.31	31.66	34.35
SVM	palabra	3	tf-idf	68.56	**69.85**	**69.94**	**69.88**

de términos de manera tf-idf sobre trigramas de caracteres.

Como ya hemos mencionado, elegimos como la línea base la idea más natural de utilizar palabras como características. En este caso el mejor resultado fue obtenido por el algoritmo *Naive Bayes* que dio 50.54 % con el pesado tf, lo que es más de 5 % peor que el resultado obtenido con trigramas de caracteres. La comparación de los tres mejores resultados con la línea base se puede observar en la Tabla 4.

Los resultados para el corpus **"K_100"** se presentan en Tabla 5, donde LSA (número) es el número de las dimensiones. Los mejores resultados para cada experimento y algoritmo están marcados con negrita. Una vez más recordamos que sólo se muestran los resultados más representativos.

Para el corpus **"K_100"** tenemos nuestro mejor resultado con el algoritmo de SVM (69.94 %) usando trigramas de palabras, el pesado tf-idf y reducción de dimensiones a 500 (con LSA). Los

Tabla 7. Resultados (exactitud, %) para el corpus "K_100_50"

N	Algoritmo	Sin LSA	LSA (50)	LSA (100)	LSA (150)	LSA (200)	LSA (300)	LSA (400)	LSA (500)	LSA (600)
2	NB	**50.91**	34.17	38.93	40.43	39.89	38.23	35.84	33.86	32.51
2	SVM	65.42	35.38	56.39	61.59	64.53	65.88	**65.96**	64.64	63.25
4	NB	33.93	37.34	38.17	38.12	38.10	29.43	37.10	36.58	**62.27**
4	SVM	39.08	50.76	60.20	67.36	69.42	21.27	**69.77**	68.99	64.92
7	NB	**62.27**	29.43	33.93	37.34	38.17	38.12	38.10	37.10	36.58
7	SVM	64.92	21.27	39.08	50.76	60.20	67.36	69.42	69.77	**68.99**
9	NB	**52.50**	33.65	42.41	44.74	46.43	46.61	45.30	43.39	42.34
9	SVM	**71.77**	46.19	58.40	63.18	64.20	65.42	65.55	65.46	65.68
11	NB	**65.66**	27.16	41.43	45.15	47.80	49.35	48.17	47.87	46.65
11	SVM	**74.14**	44.80	63.11	69.81	72.12	73.60	73.34	73.31	72.40
12	NB	**57.33**	33.32	42.56	45.48	46.48	45.35	44.19	41.74	38.87
12	SVM	68.96	45.87	61.57	66.81	68.40	**69.62**	69.55	68.03	67.68
14	NB	**63.72**	24.45	35.52	40.93	43.69	45.24	45.54	44.59	43.80
14	SVM	72.55	35.17	58.79	66.64	70.75	73.38	**73.88**	72.99	72.16

Tabla 8. Tres resultados más altos para el corpus "K_100_50"

Algoritmo	Término	Tamaño de *n*-gramas	Pesado	Todas dimensiones (sin LSA)	LSA 300	LSA 400
SVM	palabra	3	tf-idf	**74.14**	**73.60**	73.34
SVM	palabra	4	tf-idf	72.55	73.38	**73.88**

parámetros de esta configuración se encuentran en la Tabla 2 con el número 11. Se puede observar que la reducción de dimensiones dio buenos resultados para el algoritmo SVM (una mejora de 1.4 %), sin embargo para el algoritmo NB la aplicación de LSA da resultados mucho peores para la misma configuración del experimento: 58.81 % sin LSA y casi 15 % peor al aplicar LSA en el mejor caso (de 500 dimensiones). El resultado de 69.94 % supera la línea base con 43.25 % por 26.69 %, que es una mejora inobjetable, ver Tabla 6.

Finalmente, presentamos los resultados obtenidos para el corpus **"K_100_50"**, ver Tabla 7. Para este corpus, el mejor resultado se obtiene sin reducción de dimensiones, utilizando el pesado $tf\text{-}idf$ sobre trigramas de palabras, ver Tabla 8. Es curioso, que en este corpus que es un subconjunto del corpus anterior, la aplicación de LSA ya no de los mejores resultados (aunque la diferencia es menos de 0.5 %). Posiblemente, más grande es el corpus, más tiene sentido aplicar el LSA.

4. Conclusiones

En este artículo hemos considerado la tarea de cálculo de similitud entre códigos de programas utilizando los métodos de aprendizaje automático y de procesamiento de lenguaje natural. Desarrollamos tres corpus de códigos de programas en el lenguaje de programación Karel que se encuentran disponibles para su uso y pruebas.

Hemos mostrado que con *n*-gramas de palabras o caracteres podemos obtener mejores resultados en detección de similitud en comparación con el uso de las palabras solamente. Además se observa que estos resultados se mejoran aplicando el análisis semántico latente en caso del corpus más grande.

Hay que notar una situación interesante que en el corpus donde se comparan las ideas de solución, el mejor resultado se obtiene con trigramas de caracteres, mientras que en un corpus claramente temático, los mejores resultados se obtienen con trigramas de palabras.

Agradecimientos

Agradecemos el apoyo del Gobierno de México (SNI, IPN, COFAA-IPN), proyecto CONACYT número 240844, proyectos SIP-IPN 20161947, 20151406, 20144274.

Referencias

1. **Cosma, G.** (**2008**). An approach to source-code plagiarism detection and investigation using latent semantic analysis. *IEEE Transactions on Computers*, Vol. 61, No. 3, pp. 379–394.
2. **Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R.** (**1990**). Indexing by latent semantic analysis. *Journal of the American society for information science*, Vol. 41, No. 6, pp. 391–407.
3. **Flores, E., Ibarra, M., Moreno, L., Sidorov, G., & Rosso, P.** (**2014**). Modelos de recuperación de información basados en n-gramas aplicados a la reutilización de código fuente. *Proceedings of the 3rd Spanish Conference on Information Retrieval*, CERI '14, pp. 185–188.
4. **Gitchell, D. & Tran, N.** (**1999**). Sim: A utility for detecting similarity in computer programs. *SIGCSE Bull.*, Vol. 31, No. 1, pp. 266–270.
5. **Gómez-Adorno, H., Sidorov, G., Pinto, D., & Markov, I.** (**2015**). A graph based authorship identification approach. *Working Notes Papers of the CLEF 2015 Evaluation Labs*, volume 1391 of *CLEF '15*, CEUR.
6. **Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H.** (**2009**). The WEKA data mining software: An update. *SIGKDD Explorations*, Vol. 11, No. 1, pp. 10–18.
7. **Halstead, M. H.** (**1977**). *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA.
8. **Hsu, S. & Lin, S.** (**2011**). A block-structured model for source code retrieval. *Proceedings of Intelligent Information and Database Systems: Third International Conference, ACIIDS 2011*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 161–170.
9. **Manning, C. D., Raghavan, P., & Schütze, H.** (**2008**). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.
10. **McCabe, T. J.** (**1976**). A complexity measure. *EEE transaction on software Engineering*, Vol. 2, No. 4, pp. 308–320.
11. **Pattis, R. E., Reoberts, J., & Stehlik, M.** (**1994**). *Karel the Robot: Gentle Introduction to the Art of Programming*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition.
12. **Posadas-Durán, J. P., Markov, I., Gómez-Adorno, H., Sidorov, G., Batyrshin, I., Gelbukh, A., & Pichardo-Lagunas, O.** (**2015**). Syntactic n-grams as features for the author profiling task. *Working Notes Papers of the CLEF 2015 Evaluation Labs*, volume 1391 of *CLEF '15*, CEUR.
13. **Schleimer, S., Wilkerson, D. S., & Aiken, A.** (**2003**). Winnowing: Local algorithms for document fingerprinting. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, ACM, New York, NY, USA, pp. 76–85.
14. **Sidorov, G.** (**2013**). *Construcción no lineal de n-gramas en la lingüística computacional: n-gramas sintácticos, filtrados y generalizados*. México.
15. **Sidorov, G., Gelbukh, A., Gómez-Adorno, H., & Pinto, D.** (**2014**). Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, Vol. 18, No. 3, pp. 491–504.
16. **Sidorov, G., Gómez-Adorno, H., Markov, I., Pinto, D., & Loya, N.** (**2015**). Computing text similarity using tree edit distance. *Proceedings of the Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing (WConSC), 2015 Annual Conference of the North American*, NAFIPS '15, IEEE, pp. 1–4.

17. Wise, M. J. (**1996**). YAP3: Improved detection of similarities in computer program and other texts. *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '96, ACM, New York, NY, USA, pp. 130–134.

Grigori Sidorov es Doctor en Ciencias (PhD) por la Universidad Estatal Lomonosov de Moscú, Rusia, 1996. Actualmente es profesor-investigador del Laboratorio de Lenguaje Natural y Procesamiento de Texto del CIC-IPN, México. Es Investigador Nacional de México (miembro del SNI) nivel 3 y miembro de la Academia Mexicana de Ciencias. Sus áreas de interés científico son lingüística computacional e IA.

Martin Ibarra Romero es Licenciado en Ciencias de la Informática, por UPIICSA–IPN en 1986, Maestro en ciencias de la computación por el CIC-IPN en 2013. Ha trabajado como consultor de informática por 30 años. Ha sido delegado de la olimpiada de informática del D.F. y Estado de México desde el 2003. Sus principales áreas de investigación incluyen procesamiento del lenguaje natural, lingüística computacional y recuperación de información.

Ilia Markov recibió su grado de Licenciado en Ingeniería Informática en 2001 de la Universidad Técnica Estatal de Kaliningrado, Rusia. Obtuvo el grado de Maestro en Ciencias de Lenguaje en 2012 de la Universidad de Algarve, Portugal. Actualmente es estudiante de doctorado en el CIC-IPN, México. Sus principales áreas de investigación incluyen procesamiento del lenguaje natural, lingüística computacional y recuperación de información.

Rafael Guzmán-Cabrera es Profesor Titular del departamento de Ingeniería Eléctrica de la División de Ingenierías del campus Irapuato-Salamanca de la Universidad de Guanajuato desde hace 18 años, Dr. en Reconocimiento de Formas e Inteligencia Artificial por la Universidad Politécnica de Valencia, España. Miembro del grupo de NanoBioFotónica. Miembro de la Academia Mexicana de Ciencias, SNI-1.

Liliana Chanona-Hernández es ingeniera en sistemas computacionales por el Instituto Tecnológico de Tuxtla Gutierrez, Chiapas, México; Maestra en Ciencias por el CIC-IPN, México. Actualmente es profesora de ESIMEZ-IPN. Sus áreas de interés son procesamiento automático de lenguaje natural, lingüística computacional, minería de datos.

Francisco Antonio Castillo Velásquez es Licenciado en Informática por la Universidad Veracruzana y Maestro y Doctor en Ciencias de la Computación por el Instituto Politécnico Nacional. Su área de investigación es el aprendizaje automático (*machine learning*) aplicado en el área del procesamiento de lenguaje natural. Ha sido docente en diversas universidades privadas y actualmente es Profesor Investigador de Tiempo Completo en la Universidad Politécnica de Querétaro.

Artículo recibido el 05/04/2016; aceptado el 20/05/2016.
Autor para correspondencia es Grigori Sidorov.