ISSN 1870-4069

Identificación de potenciales “Hotspots” usando métodos de clasificación

Humberto Reyes San Pedro, Humberto Cervantes Maceda,
Abel García Nájera

Universidad Autónoma Metropolitana,
Posgrado Ciencias y Tecnologías de la Información,
México

{hreyes7, hcm}@xanum.uam.mx, agarcian@cua.uam.mx

Resumen. El desarrollo continuo del software es una realidad en la actualidad. Los cambios en el software pueden deberse al desarrollo de nuevas funcionalidades, la corrección de defectos o la reparación de algunas decisiones de diseño o de desarrollo, convenientes a corto plazo, pero que causan problemas más adelante. Esto último se conoce como pago de la deuda técnica del sistema. La gestión de la deuda técnica es una actividad fundamental en el desarrollo, y en este trabajo se propone utilizar modelos de software que permitan identificar de manera eficiente los archivos propensos al cambio y propensos al error, a los que llamamos “Hotspots” que son fuente de deuda técnica. Este estudio examina la efectividad de nuestro modelo basado en un algoritmo de clasificación, en métricas de software de cada archivo del sistema y en el manejo de los valores de las métricas, con el fin de encontrar una combinación de ellas que mejor nos ayuden a determinar Hotspots.

Palabras clave: Deuda técnica, hotspots, métricas de software, clasificadores.

Potential "Hotspots" Identification Using Classification Methods

Abstract. Continuous software development is a reality these days. Changes in software may be due to the development of new functionalities, the correction of defects or the repair of some design or development decisions, convenient in the short term, but that later causes problems. These decisions are known as the technical debt of the system. Its management is a fundamental activity in development, and in this work we propose the use of software models that support the identification of change prone and error prone files, which we call "Hotspots", which are sources of technical debt. This study examines the effectiveness of our model based on classification algorithms, on software metrics for each file in the system, and on managing the values of the metrics to find a combination of them that best helps us determine Hotspots.

Keywords: Technical debt, hotspots, software metrics, classifiers.

1. Introducción

En 1992 fue introducida la metáfora de Deuda Técnica (DT) desde un punto de vista financiero, para reconocer los posibles efectos negativos a largo plazo y de largo alcance que tienen ciertas decisiones que se toman durante el desarrollo [2]. La DT es generalizada, afecta a todos los aspectos de la ingeniería del software: desde el levantamiento de requerimientos, diseño arquitectónico elegido y hasta como es implementado. El término "deuda técnica" definido por Howard Cunningham en 1992, no es nuevo y tampoco los conceptos que cubre.

Durante 35 años los ingenieros de software lo han conocido con otros nombres: mantenimiento de software, evolución, envejecimiento, decadencia, reingeniería, sostenibilidad, entre otros. Pero el progreso ha sido poco sistemático, el tema no se considera atractivo y rara vez es enseñado en las escuelas hasta la fecha. La deuda técnica arquitectónica (DTA) se presenta al hacer decisiones de diseño que pueden afectar a los atributos de calidad tanto internos como externos, atentando en primera instancia a los atributos de mantenibilidad y de modificabilidad, con lo cual posteriormente se comprometerán las entregas dado que se tiene un código difícil de modificar.

Nuestro estudio se centra en anticiparse a estos problemas que se pueden presentar y prevenirlos con mantenimientos correctivos al software. La anticipación de la que hablamos se enfoca en detectar aquellos módulos del software (archivos) que sean propensos al cambio o propensos al error y que, dentro del desarrollo de este artículo, denominaremos como Hotspots (puntos calientes). Identificar estos Hotspots es de suma importancia para saber en dónde enfocar recursos para reducir la DT y tener un producto de software de mejor calidad y mucho más mantenible.

Algunos estudios en la literatura han abordado la evaluación de los archivos propensos al cambio o al error, varios de ellos utilizando algoritmos de aprendizaje automático (AA) y algunos otros de métodos estadísticos, como regresión logística. En este estudio se propone el uso de métodos cualitativos y la validación de los mismos, considerando métricas tanto internas como externas del código apoyados de algoritmos de clasificación, por su capacidad de aprender de los datos de prueba, para determinar qué combinación de métricas nos ayudan a detectar con mayor certeza los Hotspots.

Cada método se estará validando sobre once proyectos de código abierto desarrollados en lenguaje Java de diferentes tamaños. Además de proponer diferentes normalizaciones a las métricas para ajustar los valores medidos en diferentes escalas respecto a una escala común.

Las métricas son obtenidas de herramientas de análisis de código, disponibles en el mercado, como: Archinaut[1] (ARCH), Designite[2] (DES), Succinct Code Counter[3] (SCC), DV8[4] (DV8) y SonarQube[5] (SQ). Estas herramientas toman como entrada código fuente, código compilado e historial de las aplicaciones generado a través del uso de los sistemas de control de versiones como Git y de sistemas de control de

[1] https://dl.acm.org/doi/pdf/10.1145/3387906.3388633

[2] http://www.designite-tools.com/

[3] https://boyter.org/posts/sloc-cloc-code/

[4] https://www.archdia.net/products-and-services/

[5] https://www.sonarqube.org

incidentes los cuales ofrecen información muy valiosa que se traduce en métricas a utilizar.

De los proyectos que se seleccionaron, se hizo un corte en la versión de cada uno de ellos en la que se consideraron mil commits asociados a incidentes registrados, a lo que llamaremos "bug commit". Como resultado obtuvimos métricas estructurales e históricas, algunas de ellas repetidas entre las diferentes herramientas. De un total de treinta y seis métricas obtenidas, se seleccionaron para nuestro estudio dieciséis de ellas (entre estructurales e históricas) y una más para validación de resultados.

Se aplicaron métodos de clasificación exhaustivos y de AA para buscar en el espacio de soluciones la mejor combinación de métricas que nos clasifique los archivos Hotspot y No Hotspot. Los clasificadores AA se han aplicado con éxito en diferentes dominios de la ingeniería de software, tales como predicción de defectos o la predicción de la capacidad de mantenimiento [6], así que, los tomaremos en este estudio también. El reto es encontrar un conjunto reducido de métricas para detectar Hotspots en las aplicaciones de manera temprana.

Para abordar este reto se realizó el presente estudio empírico buscando dar respuesta a las siguientes preguntas de investigación: P1) ¿Existen combinaciones de métricas que puedan ofrecer mejores resultados en la detección de Hotspots en las aplicaciones de software? P2) ¿Las métricas para detectar Hotspots de una aplicación pueden funcionar para determinar Hotspots en otras aplicaciones? P3) ¿Es posible determinar qué algoritmos resulta mejor para la detección de Hotspots entre un método de clasificación y un método base (exhaustivo)?

El resto del artículo está organizado como sigue: Sección 2 resumen de trabajos previos relacionados. Sección 3 antecedentes de la investigación. Sección 4 explica la metodología de la investigación. Sección 5 presenta el análisis de resultados y se da respuesta a las preguntas de investigación. Sección 6 presenta las conclusiones del estudio y trabajos futuros sugeridos.

2. Trabajos relacionados

Se han desarrollado varios enfoques en la búsqueda de la detección de archivos propensos al cambio o al error en los sistemas utilizando combinaciones de métricas, como los siguientes: Hilton et al. [3] aplican un método que utiliza métricas internas OO y externas como tasa de defectos y tiempo que lleva ejecutar las funciones por archivo, solo que son obtenidas una vez que el sistema es liberado en producción.

Usando el clasificador de Bosques aleatorios y métricas mencionadas obtiene una predicción del 75% de los Hotspots, con un 75% de exactitud para aplicaciones diferentes a las que fue entrenado el algoritmo. Concluyen que entre más datos tenga el clasificador, mejorarán la exactitud del algoritmo. La métrica con mejor rendimiento fue: WMC (Weighted Methods per Class). Bansal et al. [4] plantearon un método de predicción del cambio, "en proyectos" y "entre proyectos" a través de: métricas y características de distribución de conjuntos de métricas para definir reglas de selección de conjuntos de datos para entrenamiento y prueba, utilizando el algoritmo C4.5 (Generador de árboles de decisión) para dicho fin.

Enfatizando que los conjuntos de datos seleccionados son de suma importancia para la clasificación. Para la clasificación de los archivos, utilizaron 3 clasificadores:

AdaBoost, LogitBoost y Bagging, sobre 3 aplicaciones de código abierto, donde Bagging tuvo los mejores resultados con un poco más del 65% de exactitud.

Al-Khiaty et al. [5] plantearon un método que utiliza métricas estructurales y evolutivas, estas últimas las obtienen del historial de versiones de las aplicaciones. El clasificador utilizado para la detección de archivos propensos al cambio fue redes Aductivas: Método grupal de manejo de datos (GMDH). Concluyen que su método mejora la precisión de clasificar que los métodos estadísticos. Hicieron experimentos con: métricas estructurales, evolutivas y combinadas. Los mejores resultados fueron con métricas combinadas.

Las métricas con mejor rendimiento fueron: edad y frecuencia de cambio (BCO), ocurrencia de cambio (FCH, LCH, CHO), tamaño (LCOM). Además de hacer un experimento más en el que tomaban las métricas de salida sugeridas por el clasificador en una siguiente iteración como entradas, alcanzando mejores resultados con un 83.58% de exactitud. Malhotra et al. [6] propusieron una comparación de 10 clasificadores de AA junto con un método de regresión logística para la predicción de archivos propensos al cambio.

Utilizaron tres aplicaciones de código abierto con dos versiones de cada una de ellas junto con sus métricas estructurales OO de cada archivo. Concluyen que el algoritmo con mejor rendimiento en las tres aplicaciones es máquinas de soporte vectorial (SVM) y que hay cinco clasificadores mejor evaluados que el método de regresión logística. Además de que las métricas que representan buenos indicadores de cambio son: CBO (Coupling between Objects), SLOC (Source Line of Code) y WMC (Weighted Methods for Class).

Los clasificadores de machine learning tienen una precisión del 64% al 80%. Bura et al. [9] utilizaron un método óptimo de predicción del cambio, a partir del manejo de control de versiones, Diagramas de Clases y Diagramas UML con los cuales se calculan cinco métricas: duración de ejecución, información de tiempo de ejecución, dependencia de clases, regularidad y popularidad. Las dos primeras se calculan directamente del código, la dependencia de clases con diagramas de clases y UML, la de regularidad con la extracción frecuente de conjuntos de elementos con algoritmo ABC y la de popularidad se calcula a partir de la regularidad tomándola como entrada (con fórmulas).

La clasificación la realiza aplicando algoritmo ID3: árbol de decisión. El método lo aplican sobre dos aplicaciones de código abierto. Y los resultados revelan que las clases con valores de métricas altas, son más propensas a cambios. Yanb et al. [10] proponen un modelo de predicción de clases propensas al cambio. Esto a través de un esquema no etiquetado (sin control de versiones) utilizando un algoritmo de clasificación no supervisado de última generación llamado CLAMI y haciendo el comparativo con una variación del mismo llamado CLAMI+.

El método se probó para catorce aplicaciones de código abierto, mostrando que los métodos no supervisados obtienen buenos resultados comparados con: LogitBoost, Multilayer perceptron, Radial basis function network, Support vector machine y K-means. Utilizan cinco métricas OO: WMC, DIT, NOC, RFC, LCOM. Su método lo compararon: con el método usando datos históricos del mismo proyecto (within a project) y con el método usando datos históricos de otros proyectos (cross a project), concluyendo que los métodos no supervisados dan resultados similares o los mejoran a los supervisados.

Garba [11] Propone un método que usa la entropía como medida efectiva de la degradación, considerándola como la tendencia del software a volverse más complejo y desorganizado por los cambios los que lo vuelve más difícil y costoso de mantener.

Valores bajos de entropía indica madurez del software. Se calcula el índice de madurez del software, el cual es una métrica que proporciona un indicador de la estabilidad software en función de los cambios que se producen para cada versión del producto. Determinando el modelo de costo constructivo (COCOMO) para sugerir la inversión que se tiene que hacer en aquellos archivos que tengan altos niveles de entropía.

3. Antecedentes de la investigación

En este trabajo de investigación entenderemos como Hotspots a la pieza o piezas de software que son propensas al cambio o al error como lo consideraron en [6][7] y [8]. En la mayoría de los desarrollos podemos identificar una pieza de software como un archivo, que puede representar una clase para lenguajes OO o puede representar un archivo cabecera para el lenguaje C. En el desarrollo moderno, es común el uso de dos herramientas que permiten tener disciplina en el desarrollo: los sistemas de control de versiones (SCV) y los sistemas de control de incidentes (SCI).

Generalmente, los cambios a realizar en el sistema se registran en el SCI como un incidente (puede ser una nueva funcionalidad o una corrección de defecto) y una vez aceptados se realizan en el código que es cambiado y que es versionado en el SCV mediante un "commit". Cuando los equipos tienen una buena disciplina de desarrollo, asocian los commits que realizan a los incidentes registrados en el SCI. Esta práctica permite identificar qué commits están asociados a incidentes relacionados con defectos (bugs) y, más específicamente, qué archivos son modificados para corregir bugs.

En este artículo, haremos referencia a estos commits relacionados con corrección de bugs como "bug commits". Cuando un mismo archivo aparece frecuentemente en "bug commits", se considera que es un Hotspot ya que es propenso a cambio y el hecho de que tenga que ser cambiado frecuentemente cuando se corrigen defectos permite pensar que es dentro de él que se originan estos defectos. El problema de la identificación de hotspots mediante esta métrica es que solo puede obtenerse de manera tardía.

En esta investigación buscamos identificar conjuntos de métricas de código que ayuden a detectar Hotspots de manera temprana. Estas métricas son obtenidas a partir de diversas herramientas de análisis de código, las cuales ofrecen por archivo, métricas estructurales tales como: tamaño, complejidad, presencia de "malos olores" de código o de diseño, dependencias, entre otras; todas ellas pueden ser obtenidas desde la primera versión del sistema. Otras herramientas toman como entrada códigos fuente, algunas otras el código compilado y algunas más consideran también información proveniente de los SCV y SCI.

En este estudio utilizamos para generar métricas por archivos las siguientes herramientas: Archinaut (ARCH), Designite (DES), Succinct Code Counter (SCC), DV8 (DV8) y SonarQube (SQ). Esta selección de herramientas es la misma que se realizó en [8] ya que se busca poder comparar nuestros resultados con los de ese estudio. ARCH toma varias fuentes de datos como entrada, incluida la información de dependencia generada por Depends y el historial de SCV de un proyecto. DES toma el

Tabla 1. Métricas seleccionadas para el estudio.

Id Métrica	Clave de Métrica	Descripción
M01	Des_Design Smells	Diseño oloroso
M02	SCC_CLOC	Tamaño en Lineas de Código Fisico (LOC)
M03	SCC_COMPLEXITY	Complejidad
M04	ARCH_Dependent Partners	Número de archivos que dependen de este archivo (fan-in)
M05	ARCH_Depends on Partners	Número de archivos de los que depende este archivo (fan-out)
M06	ARCH_Total Dependendencies	Fan in + Fan out
M07	ARCH_CoChange Partners (h)	Número de archivos que cambiaron al mismo tiempo que el archivo en uno o más "commits"
M08	DV8_TotalIssues	Número total de problemas de DV8 para los archivos
M09	DV8_Clique	Número de componentes fuertemente conectado en las que participa este archivo
M10	DV8_Crossing(h)	Número de cruces en los que participa este archivo
M11	DV8_ModularityViolation(h)	Número de violaciones de modularidad en los que participa este archivo
M12	DV8_PackageCycle	Número de ciclos de paquetes en los que participa este archivo
M13	DV8_UnhealthyInheritance(h)	Número de jerarquías de herencias en mal estado en los que participa este archivo
M14	DV8_UnstableInterface	Número de interfaces inestables en los que participa este archivo
M15	DV8_PresentinIssues	¿En cuántos de 6 problemas este archivo está presente?
M16	SQ_Issues	Número de problemas de SonarQube
M17	DV8_TargetChangeCount	Número de veces que aparece un archivo en un commit asociado con error ("bug commit")

código fuente como entrada e identifica los olores en los niveles de implementación, diseño y arquitectura. SCC es una herramienta para medir el tamaño y la complejidad del código.

DV8 realiza el cálculo del nivel de desacoplamiento, el costo de propagación, la identificación de raíces arquitectónicas y la identificación de anti patrones de diseño. SQ es una herramienta de revisión automática de código de grado industrial ampliamente utilizada y reconocida que define un amplio conjunto de reglas de calidad [8].

Dentro de las diferentes métricas que ofrecen las herramientas descritas previamente, simplificando aquellas que describen la misma información, la que se comprueba que son más fiable y las que se ofrecen a nivel archivo, tenemos la Tabla 1 en la que se listan las métricas utilizadas en nuestra investigación. Las columnas de la Tabla 1 son: Id Métrica, Clave de Métrica (que incluye como prefijo la herramienta de la que se obtuvo) y una descripción.

Además de que se marca el Identificador de la métrica con una "(h)" al final para indicar que esa métrica es obtenida del historial que proporcionan los SCV y SCI, y que pueden ser obtenidas desde los primeros "bug commit". Las primeras dieciséis métricas serán las que utilizaremos para nuestro estudio, las cuales nos ofrecen características por archivo de tipo: estructural como tamaño por número de líneas de código (CLOC) o complejidad (COMPLEXITY) y de ciclos como archivos que dependen de uno (fan-

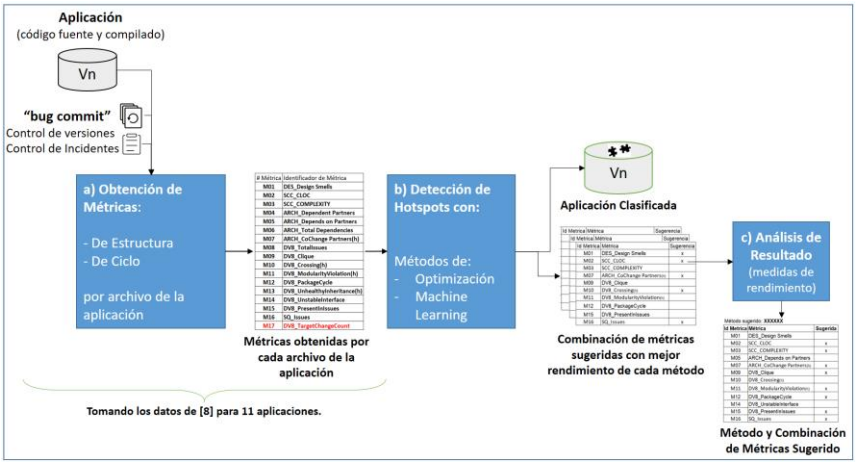

Fig. 1. Diagrama de la metodología aplicada en la detección de Hotspots.

in) o paquetes que dependen de otro (Package Cycle). La métrica DV8 Target ChangeCount (TCC) es especialmente importante ya que nos indica el número de "bug commits" que ha tenido un archivo. Por cada aplicación de las once disponibles se hizo un análisis con el diagrama de cajas y bigotes para la métrica TCC y una vez analizados los datos se determinó seleccionar como Hotspots todos los archivos cuyo valor en esta métrica fuera igual o mayor al Q3 (cuartil 3). El resto de los archivos se consideran como No Hotspots. Como podemos ver en la Tabla 2 se identificaron como Hotspots el 5.3% del total de los archivos que están presentes en nuestra investigación.

4. Metodología de la investigación

La metodología que se aplicó en este trabajo está representada en el flujo de la Figura 1. y cómo podemos observar consta de tres etapas: a) Obtención de métricas. A partir de la versión actual del sistema y de los registros de los "bug commit", se obtienen las métricas de cada una de los archivos que componen el sistema: dieciséis para ser evaluadas y una que sirve de referencia para comprobar la efectividad de la clasificación del método (TCC). Se retoma la extracción de métricas que realizaron [8] para las once aplicaciones que se usaron en esta investigación, por lo que este paso ya está realizado.

b) Detección de Hotspots a través de métodos de clasificación. Esta etapa recibe como entrada las métricas de cada archivo de la aplicación a analizar y entrega como resultado los archivos de la aplicación clasificados como Hotspots y los resultados de la evaluación de cada uno de los métodos de clasificación aplicados, con las variantes en los conjuntos de datos por la normalización de los mismos. c) Análisis de Resultados. Se toma como entrada los resultados de los seis métodos aplicados para la clasificación, se analizan para determinar cuál de ellos es el que ofrece los resultados más altos en las medidas de rendimiento que regularmente se utilizan para evaluar los clasificadores como en [4, 5, 6] y que se explican más adelante, principalmente: sensibilidad,

Tabla 2. Proyectos de software para la investigación.

	Entrenamiento									Prueba		
Aplicación	**tez**	**tajo**	**knox**	**nifi**	**oozie**	**sentry**	**flume**	**deltaspike**	**hbase**	**broke-j**	**amqp-0x**	**Totales**
Achivos Totales	839	1557	863	4112	812	561	441	742	1786	1865	561	**14,139**
Archivos Hotspots (TCC)	90	78	23	123	98	12	58	28	129	92	12	**743**
												5.3%

Combinación	{	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0** }
Núm. de Métrica		M01	M02	M03	M04	M05	M06	M07	M08	M09	M10	M11	M12	M13	M14	M15	M16

Fig. 2. Representación binaria de una combinación de métrica.

especificidad y el área bajo la curva (AUC), las cuales reflejan la capacidad de detectar Hotspots.

4.1. Proyectos de software para entrenamiento y prueba

Para la investigación se consideró un conjunto de once proyectos de software OO con las siguientes características: están desarrollados en Java, son mantenidos por "The Apache Software Foundation", son de código abierto, y los mensajes de commit del SCV están vinculados con incidencias en el SCI.

Se consideraron 1,000 commits asociados a incidencias (TCC). Se dividieron los conjuntos de datos siguiendo la regla que se aplicó en [5] con una variación de 5% para favorecer a los datos de entrenamiento, quedando en 75:25, con nueve aplicaciones para entrenamiento y dos para el entrenamiento (broker-j y ampq-0x) para comprobar el resultado del método sugerido como el mejor predictor de Hotspots.

En cada una de las aplicaciones se tiene definidos cuáles y cuántos son los archivos que se están marcando como Hotspots, en base a la medida TCC, que nos servirán para validar el resultado de la clasificación que se obtenga de cada uno de los métodos. La Tabla 3 muestra la lista de proyectos, así como la cantidad de archivos totales y la cantidad de archivos considerados como Hotspots.

A través de la matriz de confusión de cada experimento obtuvimos las medidas de rendimiento que nos permitieron determinar el método que mejor identifica los Hotspots. Se clasificaron un total de 14,139 archivos, de los cuales 743 (5.3%) son catalogados como Hotspots (con ayuda de la métrica TCC, previamente comentada).

4.1. Normalización de los datos

La normalización de los conjuntos de datos, como ya lo mencionamos, es el ajuste a los valores medidos en diferentes escalas respecto a una escala común. Se utilizó esta técnica estadística de normalización de datos para explorar si es posible encontrar resultados diferentes y más favorable debido a que analizando los valores entre las métricas de la Tabla 2 vemos que son muy diversos. Para buscar la homogeneidad de los datos se optó por aplicar: a) Normalización por Máximo valor en cada métrica. Por métrica de los datos se encuentra el valor máximo y se realiza el cociente entre el valor

de cada archivo en esa métrica y el valor máximo obtenido para la misma métrica. De esta manera tendremos valores entre 0 y 1.

Esto evita que una métrica con un valor muy grande influya más que otras métricas donde sus valores sean más bajos. b) Normalización por Máximo valor de CLOC (M02), Se eligió número de líneas de código (CLOC) dado que es sabido que a archivos grandes mayor riesgo de presentar errores, por lo que al normalizar el resto de las métricas con base en esta se provoca que todas estén a un mismo nivel.

En este caso se busca que los datos estén homogeneizados con respecto al tamaño del archivo más grande de la aplicación que se vaya a evaluar, de esta forma se propone que ahora todos los archivos tienen valores de métricas homogéneos y puedan ser comparables. Esta normalización también remueve las correlaciones que existen entre métricas como complejidad y tamaño.

4.2. Métodos propuestos para clasificar Hotspots

Como recordaremos, una de nuestras preguntas de investigación es encontrar si existe una combinación de métricas que nos ayuden a clasificar los Hotspots de un sistema. Para ello, se consideraron dos métodos para clasificar: Exhaustivo (Exh) el cual nos permitió evaluar todas las combinaciones de métricas combinado con tres tratamientos de los conjuntos de datos (sin normalizar, normalizar por máximo por métrica y normalizar por máximo CLOC) y Bayes Ingenuo (BI) con función de probabilidades condicionadas para datos continuos (Gaussiana) combinado con los mismos tres tratamientos de datos que el Exh, por lo que en total se tuvieron seis métodos a evaluar.

Para cada método: i) Se implementó en lenguaje C, ii) Para cada una de las aplicaciones se evaluó cada una de las combinaciones de métricas posibles para determinar qué combinación nos da las mejores medidas de rendimiento al ser evaluada, iii) Se propone una representación binaria de la combinación que se estará evaluando (Fig. 2), donde un 1 indica que la métrica se utiliza y un 0 que no.

Como podemos ver en el ejemplo de la Fig. 2, se estaría evaluando una combinación de métricas que utiliza las métricas: M01, M03, M06, M08, M09, M11, M12, M14 y M15 presentadas en la Tabla 1, iv) Dado que son 16 métricas tendremos 2^16 (65,635) combinaciones posibles, v) Las medidas de rendimiento (obtenidas a partir de la matriz de confusión) que se utilizaron y que se usan más frecuentemente para la evaluación de los métodos de clasificación son: Exactitud (proporción de predicciones correctas y total de predicciones), Precisión (proporción entre los positivos reales predichos y todos los casos positivos), Sensibilidad (fracción de verdaderos positivos), Especificidad (fracción de verdaderos negativos), Medida-F (valor único ponderado entre precisión y exactitud) y Análisis ROC (área bajo la curva –AUC-, representación gráfica de Sensibilidad vs. 1-Especificidad), vi) Los resultados de cada experimento estarán dados por los elementos de la matriz de confusión: TP (verdaderos positivos –es Hotspots y se clasifica como Hotspot-), TN (verdaderos negativos –No es Hotspots y se clasifica como No Hotspot-), FP (falsos positivos –No es Hotspots y se clasifica como Hotspot-) y FN (falsos negativos –es Hotspots y se clasifica como No Hotspot-) con los cuales se podrán calcular las medidas de rendimiento del experimento i-ésimo.

Con las medidas de rendimiento serán con las que estaremos evaluando el resultado del método de clasificación más apto. A continuación, discutiremos el detalle para cada uno de los dos métodos.

4.2.1. Método exhaustivo (Exh)

Para este método se aplicó: Por cada combinación de métricas se evaluó con la función de evaluación (FE) a todos los archivos a para obtener los Hotspots Propuestos (HPP). Los HPP serán los archivos que resulten con valor más alto en FE descrita en la Ecu. 1. Lo anterior se logra ordenando de manera descendente todos los archivos de la aplicación que han sido evaluados con la *FEa* y se tomará el mismo número de archivos que tenemos establecidos como Hotspots en la Tabla 2 por aplicación.

Con ello se procede a calcular la matriz de confusión y sus medidas de rendimiento. Donde Valor(a, m) es el valor de la métrica m para el archivo a, *CM(m)* es el valor de la representación binaria de la combinación que se está evaluando, teniendo como posibles valores un 1 o un 0. Para este método se realizarán tres experimentos: **a) Exh (sNorm)** sin normalizar, no se le hace nada a los valores de las métricas de cada base de datos, **b) Exh (nMax)**, se Normalizan los datos por valor Máximo en cada métrica. Se homogeniza la base de datos con valores entre 0 y 1, **c) Exh (nCLOC)**, es Normalizando los datos por Máximo Número de Líneas (métrica: M02):

$$FEa = \sum_{m=1}^{16} (Valor(a,m) * CM(m)) \quad (1)$$

.

4.2.2. Método bayes ingenuo (BI)

Para este método y tomando en consideración la recomendación de [10] sobre los clasificadores supervisados, se aplicó: El clasificador supervisado de BI en su versión para datos continuos que utiliza la distribución Gaussiana (Ec. 3) para calcular las probabilidades condicionadas *p(Mn|Cj)* de la Ec. 2, donde: *Cj* son las clases que serán asignadas a cada archivo (*C1*=Hotspots y *C0*=No Hotspot), *Mn* son los valores de las diferentes métricas que se están evaluando del archivo que se está clasificando, μ_{Cj} es la media de la clase *Cj* y σ^2_{Cj} es la desviación estándar de la clase *Cj*. Para cada archivo se realizará la evaluación de la probabilidad condicional por clase: *p(Cj|M1,...,Mn)* y se clasificó como Hotspot cuando *p(C1|M1,...,Mn)* > *p(C0|M1,...,Mn)*, en otro caso se clasificó como No Hotspot:

$$p(Cj|M1,...,Mn) = p(Cj) \prod_{i=1}^{n} p(Mi|Cj) \quad (2)$$

,

$$p(Mi|Cj) = \frac{1}{\sqrt{2\pi\sigma^2_{Cj}}} * e^{-(Mi - \overline{\mu_{Cj}})2 /(2\sigma^2_{Cj})} \quad (3)$$

.

Se evaluó cada combinación de las métricas (2^16) para determinar la combinación que tenga la mejor medida de rendimiento. Se aplicó a los conjuntos de datos de entrenamiento. Para las nueve aplicaciones de entrenamiento originalmente definidas en la Tabla 3, se dividirán para tener la porción de datos de entrenamiento y de pruebas para BI: se toma una de las aplicaciones como de prueba y las otras ocho se utilizan

para entrenamiento y así sucesivamente se va variando la aplicación de prueba, de esta forma nos resultan nueve experimentos los cuales ofrece la variedad que se busca para validar el clasificador de BI. Para este método se realizaron tres experimentos: **a) BI (sin Norm)**, no se normaliza la base de datos. **b) BI (nMAX)**, se normalizan los datos por máximo valor de cada métrica, **c) BI (nCLOC)**, se normalizan los datos por Máximo Número de Líneas (métrica: M02).

5. Análisis de resultados

En esta sección se describen los resultados obtenidos de los dos métodos considerados, así como una comparación de ellos para posteriormente saber cuál es el que mejor identifica los Hotspots y con cuál combinación de métricas. Se recapitulan los elementos que estuvieron presentes en cada uno de los experimentos: a) dieciséis métricas para cualquier base de datos y una más que sirve para clasificar los HPR, b) once aplicaciones, nueve para entrenamiento, dos para validar los resultados y dar respuesta a la P2 de la investigación, c) seis métodos de clasificación a validar y d) seis medidas de rendimiento que se usarán para evaluar los resultados.

5.1. Resumen de resultados

Se considera la medida de rendimiento AUC que se introdujo anteriormente como el indicador más representativo para seleccionar el método de clasificación, dado que esta medida engloba la capacidad del clasificador en detectar los Hotspots como los No Hotspots (ponderación de Sensibilidad y Especificidad). Al ser los conjuntos de datos tan desbalanceados (5.3% Hotspots y 94.7 No Hotspots), ambas clases nos interesan y AUC representa el balance de ambas clasificaciones.

Los valores de AUC van entre 0.0 y 1.0, donde 0.0 es que no clasifica nada y 1.0 es la clasificación perfecta de todos los datos. Los clasificadores son evaluados a través de AUC con los siguientes rangos: [0.0, 0.50) es como lanzar una moneda, [0.50, 0.60) malo, [0.6, 0.75) regular, [0.75, 0.90) bueno, [0.90, 0.97) muy bueno y [0.97, 1.00) excelente.

En la Tabla 3, mostramos el resumen de los resultados obtenidos en cada experimento, identificando la mejor combinación de métricas en cada uno de ellos y podemos comentar lo siguiente: **i)** Los dos métodos que más destacan por sus resultados en las diferentes medidas de rendimiento son: Exh (nMAX) y BI (nMAX), **ii)** Exh (nMAX) obtiene los mejores resultados en cuatro de las seis medidas de rendimiento que se están considerando, pero los resultados en Sensibilidad (capacidad de detectar Hotspots) está en un 53.7%.

iii) BI (nMAX) tiene a su favor dos de las seis medidas de rendimiento que se están considerando, donde en Sensibilidad tiene un 81.6% y una Especificidad de 87.8% lo que sugiere un balance entre ambas clasificaciones (Hotspots y No Hotspots) lo cual se refleja en el AUC con un valor de 84.7%, iv)

El método que utiliza el menor número de métricas es el de BI (nMAX) el cual solo utiliza tres de ellas: SCC_CLOC, DV8_Crossing(h) y DV8_ModularityViolation(h), revisando las métricas tenemos que una es el tamaño de los archivos (CLOC) y las otras dos se basan en el historial de las aplicaciones obtenidas de los SCV; DV8_ Modularity

Tabla 3. Mejores resultados de cada método evaluado con aplicaciones de entrenamiento.

		Medidas de rendimiento					
Método	**Métricas sugeridas**	**Exactitud**	**Precisión**	**Sensibilidad**	**Especificidad**	**Medida-F**	**AUC**
Exh (sNorm)	M01,M08,M09,M10,M11, M13,M14,M15,M16	0.936	0.475	0.476	0.965	0.619	0.721
Exh (nMAX)	M01,M02,M03,M05,M10, M11,M14,M15,M16	0.944	0.535	0.537	0.969	0.670	0.753
Exh (nCLOC)	M02,M08,M09,M11,M12, M15,M16	0.936	0.467	0.467	0.965	0.608	0.716
BI (sNorm)	M01,M02,M03,M07,M09, M10,M11,M12,M15,M16	0.897	0.362	0.584	0.919	0.478	0.751
BI (nMAX)	M02,M10,M11	0.873	0.323	0.816	0.878	0.458	0.847
BI (nCLOC)	M01,M03,M05,M07,M11, M12,M13,M14,M15,M16	0.884	0.292	0.560	0.905	0.410	0.733

Tabla 4. Resultados obtenidos con aplicaciones de prueba.

		Medidas de rendimiento					
Método	**Métricas sugeridas**	**Exactitud**	**Precisión**	**Sensibilidad**	**Especificidad**	**Medida-F**	**AUC**
Exh (sNorm)	M01,M08,M09,M10,M11, M13,M14,M15,M16	0.936	0.475	0.476	0.965	0.619	0.721
Exh (nMAX)	M01,M02,M03,M05,M10, M11,M14,M15,M16	0.944	0.535	0.537	0.969	0.670	0.753
Exh (nCLOC)	M02,M08,M09,M11,M12, M15,M16	0.936	0.467	0.467	0.965	0.608	0.716
BI (sNorm)	M01,M02,M03,M07,M09, M10,M11,M12,M15,M16	0.897	0.362	0.584	0.919	0.478	0.751
BI (nMAX)	M02,M10,M11	0.873	0.323	0.816	0.878	0.458	0.847
BI (nCLOC)	M01,M03,M05,M07,M11, M12,M13,M14,M15,M16	0.884	0.292	0.560	0.905	0.410	0.733

Violation(h) en [8] es una de las métricas que en ese estudio de correlación de métricas se detecta como una de las mejores predictora de Hotspots.

En la Tabla 3 columna Métricas sugeridas, mostramos el resumen de las métricas sugeridas en cada método y de donde podemos comentar lo siguiente: i) La métrica que está presente en todas las combinaciones de métricas de los seis métodos es DV8 Modularity Violation(h) (número de violaciones de modularidad en las que participa este archivo), lo cual reafirma a la conclusión realizada por [8], que esta métrica es una buena predictora de Hotspots.

ii) Dos métricas que participan en cinco de los seis métodos, estás son: DV8_PresentInIssues (en cuantos errores de las 5 herramientas está presente este archivo) y SQ_Issues (se engloban: "malos olores" de código, errores, vulnerabilidades y Hotspots de seguridad) [8]. iii) Dos de las métricas propuestas no fueron utilizadas en ninguna de las seis combinaciones sugeridas por los métodos, estas son: ARCH_DependentPartners (número de archivos que dependen de él –FanIn-) y ARCH_TotalDependencies (FanIn + FanOut). iv) El método que propone la combinación con el menor número de métricas es BI(nMAX), las cuales son: SCC_CLOC, DV8_Crossing(h) y DV8_ModularityViolation(h).

Como lo mostramos en la Tabla 1, se tienen dos aplicaciones de prueba para validar los resultados con los datos de entrenamiento. Se ejecutaron las mejores soluciones de

cada uno de los métodos utilizados y ver el comportamiento con otras aplicaciones que no han participado aún en el proceso de entrenamiento. Los resultados se presentan en la Tabla 4 y obtenemos resultados similares a los obtenidos con los datos de entrenamiento, donde los métodos de Exh (nMAX) y BI (nMAX) nos dan los mejores resultados en AUC con valores de 86.5 y 95.3%.

5.2. Respuesta a las preguntas de investigación

Con los resultados obtenidos daremos paso a dar respuesta a nuestras preguntas de investigación e ir obteniendo conclusiones de nuestra investigación.

P1. ¿Existen combinaciones de métricas que puedan ofrecer mejores resultados en la detección de Hotspots en las aplicaciones de software? **R1**. Con base a los resultados que podemos ver en la Tabla 3, con resultados desde 71.6% y hasta 84.7% en AUC, lo cual indica que tenemos una buena clasificación de Hotspots. Teniendo el mejor resultado de sensibilidad y AUC con el método BI (nMAX) y una combinación de 3 métricas (M02, M10 y M11 mostradas en la Tabla 3 columna Métricas sugeridas), favoreciendo tanto la detección de Hotspots (sensibilidad con un 81.7%) como los No Hotspots (especificidad con un 87.8%) obteniendo una AUC de 84.7%.

P2. ¿Las métricas para detectar Hotspots de una aplicación pueden funcionar para determinar Hotspots en otras? **R2**. Como podemos ver en la Tabla 4, para ambas aplicaciones se obtienen muy buenos resultados en exactitud con cualquiera de los métodos. También podemos ver que ambas aplicaciones de prueba, los dos métodos que más destacan son: Exh (nMAX) y BI (nMAX) lo que va acorde con el resultado que se obtuvo con las aplicaciones de entrenamiento. El AUC que ofrece BI (nMAX) es del 86.5% para broker-j (una aplicación mediana) y del 95.3% para amqp-0x (una aplicación pequeña), por lo que podemos recomendar el método de BI (nMAX) junto con la combinación de métricas sugerida (M02, M10 y M11) para intentar predecir Hotspots en otras aplicaciones.

P3. ¿Es posible determinar qué algoritmos resulta mejor para la detección de Hotspots entre un método de clasificación y un método base (exhaustivo)? **R3**. Con base a los resultados obtenidos en este estudio podemos señalar que el método de Bayes Ingenuo con los diferentes tratamientos de datos casi siempre estuvo por arriba en los resultados del AUC con respecto al método Exhaustivo, por lo que podemos proponerlo como un buen clasificador de Hotspots.

6. Conclusiones y trabajo futuro

Como pudimos darnos cuenta, la clasificación de los archivos de Hotspots en una aplicación es de suma importancia al tratar de disminuir su DT, por lo que proponemos los resultados obtenidos en esta investigación como una opción para poder ayudar en la detección de Hotspots, con una algoritmos de AA, como lo es BI, acompañado de una normalización de datos para homogeneizarlos y el uso de tres métricas que resultaron bastante reveladoras la información que representan y que permite al algoritmo trabajar para clasificar con un porcentaje bastante aceptable del 83.5% y del 95.3% de AUC para aplicaciones totalmente ajenas al proceso previo de entrenamiento. Se podría hacer una detección temprana de Hotspots, si los hay, después de los primeros

"bug commit". Cabe señalar que conforme se vaya dotando al algoritmo de mayor información de entrenamiento de las tres métricas propuestas se podrían mejorar los resultados de clasificación. Los métodos exhaustivos no estuvieron tan lejos de ofrecer también buenos resultados, solo que el conjunto de métricas que ocupan es mayor a la combinación que se encontró con BI, pero estos también podrían ser una opción.

También podemos comentar que los resultados de [4, 5, 6] están por debajo en medidas de rendimiento como la exactitud o el AUC, lo que nos hace pensar que nuestros resultados son prometedores. Al igual que en la mayoría de los estudios previamente realizados y revisados, la métrica que está presente en la mayoría de ellos, al igual que en el nuestro, es el número de líneas de código de cada archivo (CLOC), dado que a mayor número de líneas mayor la probabilidad de incurrir en errores.

El trabajo futuro que se visualiza, es: trabajar en un método sistemático para la obtención de las métricas con mayor facilidad, continuar creciendo la base de entrenamiento del algoritmo para su aprendizaje y una posible mejora de la detección de Hotspots y probar el método con aplicaciones no necesariamente con las mismas características base que se tomaron para esta investigación y validar el comportamiento del método.

Referencias

1. Baresi, L., Colazzo, S., Mainetti, L., Morasca, S.: A modelling notation for complex web applications. In: Mendes, E., Mosley, N. (Eds) Web Engineering, Springer, pp. 335–364 (2006)
2. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-based Web Engineering: An approach based on Standards. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (Eds.): Web engineering: Modelling and implementing web applications. Springer, pp. 157–191 (2008) doi: 10.1007/978-1-84628-923-1_7
3. Sánchez-Santamaria, M., García-García, L. A.: La ingeniería web: Desarrollo de aplicaciones web de alta calidad. HYPATIA (2011)
4. Vera, P., Pons, C., González, C., Giulianelli, D., Zodríguez, R.: Metodología de modelado de aplicaciones web móviles basada en componentes. In: XV Workshop de Investigadores en Ciencias de la Computación, pp. 451–455 (2013)
5. Moroni, N., Señas, P.: Uso de grafos para el modelado de experiencias educativas colaborativas basadas en la web. In: XIII Congreso Argentino de Ciencias de la Computación, pp. 1134–1146 (2007) http://sedici.unlp.edu.ar/handle/10915/22913
6. Bouchrika, I., Ait-Oubelli, L., Rabir, A.: Mockup-based navigational diagram for the development of interactive web applications. In: Proceedings of the International Conference on Information Systems and Design of Communication, pp. 27–32 (2013) doi: 10.1145/2503859.2503864
7. Lipschutz, S.: Sets and basic operations on sets: Set theory and related topics. McGraw-Hill, pp. 5–6 (1998)
8. Lipschutz, S., Lars Lipson, M.: Set Theory Discrete Mathematics. McGraw-Hill, pp. 1–22 (2007)

ISSN 1870-4069

Plataforma digital para la seguridad escolar mediante el control de acceso automatizado basado en el reconocimiento facial

Cynthia B. Perez[1], Jesus R. Villavicencio[1], Jessica Beltran[2], Roberto Limon-Ulloa[1], Perla Duran[1], Samuel Torres[1], Bryan García[1]

[1] Instituto Tecnológico de Sonora (ITSON), México

[2] Centro de Investigación y Desarrollo de Tecnología Digital (CITEDI-IPN), México

{cynthia.perez, roberto.limon, jesus.villavicencio162447, perla.duran204151, samuel.torres202413, bryana.garcia}@potros.itson.edu.mx, jbeltranm@ipn.mx

Resumen. Las condiciones de inseguridad pública ameritan que se desarrollen sistemas orientados a brindar seguridad dentro de instituciones educativas. Se pueden utilizar diferentes enfoques tecnológicos orientados a apoyar la seguridad, entre ellos, el uso de sistemas de monitoreo automático de personas para verificar que solo ingresan a los campus quienes tienen previa autorización. Una forma no intrusiva de abordar este problema es mediante el reconocimiento facial con técnicas de aprendizaje automático. En este trabajo, se describe el desarrollo de una plataforma digital para el control de acceso automático y notificación de alertas a los guardias de seguridad de una institución educativa. Se realizaron experimentos con la base de datos LFW [28] y con imágenes de estudiantes y profesores de una institución educativa para conocer y comparar la efectividad del reconocimiento facial. Así mismo, se utilizaron características extraídas con redes neuronales profundas y algoritmos de clasificación para identificar a la persona. Los resultados indican la viabilidad de implementar esta plataforma *in situ* y proporcionan direcciones para el trabajo futuro.

Palabras clave: Reconocimiento Facial, universidad inteligente, openface, seguridad escolar.

A Digital Platform for School Security through Automated Access Control based on Facial Recognition

Abstract. Public unsafe conditions justify the development of systems that help to provide security inside educational institutions. There can be different technological approaches to support security, including automatic systems to monitor that only authorized people can enter campus. A non intrusive way to address this problem, is through facial recognition with machine learning techniques. In this work, is described the development of a digital platform to control access automatically and to send alerts to security guards from an educational institution. Experiments were conducted using the data set LFW [28] and with images from students and professors from the

educational institute to know and compare the effectiveness of facial recognition. Thus, features extracted with deep learning and classification algorithms to identify persons were used. The results show the viability to implement the platform *in situ* and provide directions for future work.

Keywords: Facial recognition, smart campus, openface, school surveillance system.

1. Introducción

Hoy en día, el tema de seguridad pública urbana es una de las principales preocupaciones que tiene la sociedad alrededor del mundo, principalmente en aquellos países en vías de desarrollo. En México, de acuerdo con el Instituto Nacional de Estadística y Geografía (INEGI), en el año 2020, el 67.8% de la población mayores de 18 años considera que vivir en su ciudad es inseguro [1]. En ese sentido, debido a la situación que se vive en torno a la seguridad en el país, la ocurrencia de incidentes relacionados con la seguridad no es ajena a las instituciones educativas.

Es por ello que particularmente en las universidades, existe un gran interés por incorporar tecnologías de la información para facilitar la administración y seguridad de los campus. Para apoyar estas iniciativas, se han propuesto diferentes sistemas automatizados para el control de acceso en organizaciones públicas y/o privadas basadas en comunicación de campo cercano (Near Field Communication, NFC) [8], identificación por radio frecuencia (Radio-frequency Identification, RFID) [2-4], sistemas basados en iris y sistemas basados en reconocimiento facial [5-7].

El reconocimiento facial es una de las técnicas más populares para incorporar a los sistemas de seguridad por su facilidad de uso, su naturaleza sin contacto y que no es una técnica intrusiva ni agresiva [1,4,6,7]. De acuerdo con Victor Skinner [9], la Universidad de San Francisco hace uso de un sistema comercial basado en reconocimiento facial llamado iOmniscient [1] para administrar el acceso de los estudiantes a los dormitorios de la universidad, donde el objetivo es contar con una forma de controlar el acceso a las residencias universitarias que no sea intrusivo y que no requiera que los estudiantes se detengan en su acceso al campus.

Mediante este sistema de control de acceso basado en reconocimiento facial, la universidad puede tener un historial de los ingresos a las residencias e identificar a los visitantes no autorizados que necesitan registrarse. Por otro lado, la escuela de desarrollo infantil de la Universidad de Seattle y la academia católica St. Therese utilizan un sistema de control de acceso basado en reconocimiento facial llamado SAFR[2] donde el personal debe sonreír a la cámara para que el sistema les permita el acceso abriendo automáticamente la puerta de entrada, de lo contrario, deben pasar a una oficina a registrarse. Así mismo, este sistema se utiliza para monitorear pasillos y áreas comunes permitiendo activar alertas de sonido, envío de mensajes y llamar a la policía en caso que así lo requieran [10].

En China, la Universidad Normal de Beijing hace uso de sistemas de reconocimiento facial en todos los dormitorios para el control de acceso donde los estudiantes tienen tres opciones para activar el sistema de reconocimiento facial como pasar su credencial

[1] https://iomni.ai/
[2] https://es.safr.com/

(ID) del campus, decir su nombre o ingresar los últimos cuatro dígitos de la contraseña de su credencial para que se les permita el acceso al edificio de dormitorios [11]. Así mismo, la Universidad de Pekín en China desarrolló un sistema de reconocimiento facial para identificar tanto a estudiantes como personal académico y administrativo cuando ingresan al campus.

En este caso, la universidad acondicionó el acceso instalando torniquetes electrónicos con lectores de credencial y tabletas con cámaras incorporadas para que el estudiante o el personal tengan la opción de ingresar a la institución por medio de su credencial o por medio del sistema de reconocimiento facial [12]. De esta manera, el control de acceso en instituciones educativas mediante reconocimiento facial se está considerando como una herramienta tecnológica que ofrece una solución eficiente, precisa, de acceso rápido e imparcial ayudando a las instituciones a mejorar la seguridad del campus permitiendo el acceso a solo personas autorizadas.

Sin embargo, implementar de forma eficiente este tipo de sistemas a gran escala es desafiante y representa un costo que muchas veces las instituciones educativas no pueden costear [13]. Es por ello, que este documento describe el desarrollo de una plataforma digital para el control de acceso por medio de reconocimiento facial basado en aprendizaje profundo que permita identificar personas autorizadas y no autorizadas en los diferentes puntos de entrada/salida de una institución educativa utilizando librerías de código abierto como OpenFace [14].

El resto del artículo está organizado como sigue. En la Sección 2, se describe el problema de reconocimiento facial basado en aprendizaje profundo. En la Sección 3 se presenta el diseño e implementación de la plataforma digital. La Sección 4 presenta la experimentación y resultados obtenidos. Finalmente, las conclusiones y trabajo futuro son presentados en la Sección 5.

2. Reconocimiento facial basado en aprendizaje profundo

El reconocimiento facial es un problema clásico de la visión por computadora que hasta la fecha continúa siendo de gran interés para la comunidad científica, así como también para la sociedad a través de aplicaciones comerciales. La Figura 1 presenta el proceso que se sigue para llevar a cabo el reconocimiento facial de forma automatizada de tal forma que primero es necesario que el sistema reciba como entrada una imagen (offline/online); posteriormente, se lleva a cabo el proceso de detección y alineación del rostro para que inicie el proceso de extracción de características y a su vez, el sistema compare mediante el proceso de correspondencia/clasificador, las características extraídas de la imagen de entrada con el conjunto de características de los rostros que se encuentran en la base de datos. Finalmente, se evalúa el rendimiento del algoritmo mediante diferentes métricas y se muestra la imagen con los datos de la persona que ha sido reconocida.

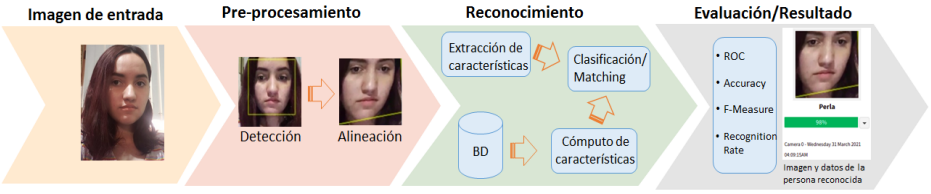

Fig. 1. Esquema tradicional del reconocimiento facial.

Los algoritmos de reconocimiento facial utilizados en aplicaciones reales hacen uso de cámaras donde las imágenes pueden tener variaciones como cambios de iluminación, baja resolución, movimientos del rostro, oclusiones, difuminación en la calidad de la imagen, entre otros. Es por ello que la mayoría de los sistemas de reconocimiento facial trabajan adecuadamente en entornos controlados como el de la iluminación y un posicionamiento frontal del rostro.

En este tipo de variaciones, los algoritmos descriptivos han mostrado ser eficaces. Algunas de las técnicas descriptivas más populares se encuentran SIFT [15], LBP [16] y HOG [17]. Pérez y Olague propusieron operadores descriptivos evolucionados llamados $RDGP_2$ que mejoraron en un 26.7% el rendimiento de los algoritmos de reconocimiento de objetos por imágenes de SIFT, SURF y GLOH [18]. Es por ello, que existe un interés en abordar el reconocimiento facial bajo este tipo de condiciones llamadas no restrictivas en donde el uso de técnicas de aprendizaje profundo ha ganado popularidad ya que ofrece mayor robustez contra las diferentes variaciones que pueden afectar el proceso de reconocimiento [19].

En ese sentido, las técnicas descriptivas han sido utilizadas en algoritmos de aprendizaje profundo como las redes neuronales convolutivas en un entorno de IoT [13,20-21], demostrando recientemente una mejora significativa en la precisión del reconocimiento facial, como por ejemplo el algoritmo DeepFace de Facebook [22], el FaceNet de Google [23], COCOLoss [24] y ArcFace [25]. En este trabajo, se utilizó la versión de código abierto de FaceNet llamada OpenFace [14] para llevar a cabo el proceso de reconocimiento facial bajo un enfoque de aprendizaje profundo.

OpenFace es uno de los algoritmos de reconocimiento facial de código abierto más preciso que combina la técnica Triplet Loss de Deepface [22] con una red neuronal convolucional profunda basada en la red de FaceNet [23]. Openface se entrenó con 500k imágenes combinando dos bases de datos utilizadas para reconocimiento facial, CASIA-WebFace [26] y FaceScrub [27] obteniendo un modelo de red llamado *nn4.small2* con un menor número de parámetros para la base de datos utilizada. Para la etapa de detección del rostro se puede utilizar HOG (Histogram of Oriented Gradient) y SVM (Support Vector Machine) de DLib, o bien, Haar cascade de OpenCV.

De acuerdo con Amos et. al. [14], HOG y SVM obtienen mejor precisión que el detector de OpenCV en la etapa de detección. Una vez que el modelo de red es obtenido y se obtienen las características de los rostros, se utilizan técnicas de clasificación para completar la tarea de reconocimiento. Finalmente, para evaluar el rendimiento de OpenFace, Amos et. al. utilizaron la base de datos LFW [28] con el modelo de red *nn4.small2.v1* obteniendo una precisión de 0.9292± 0.0134.

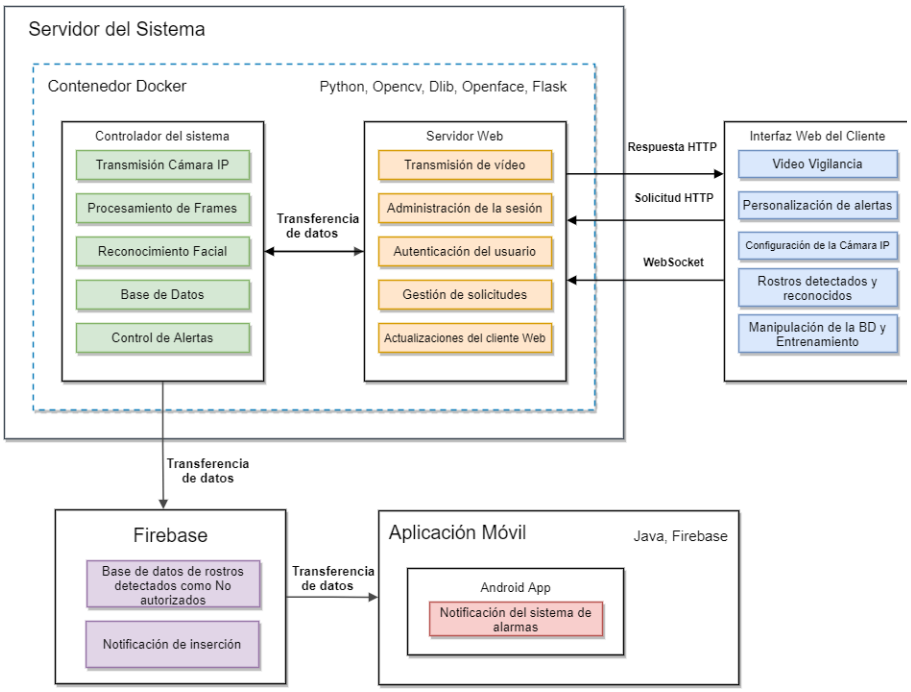

Fig. 2. Diseño de la arquitectura de la plataforma para el control de acceso universitario.

La etapa de clasificación permite que el sistema pueda identificar quién es quién no solamente saber si la persona se encuentra o se parece a alguien almacenada en la base datos. En ese sentido, existen diferentes clasificadores que han sido utilizados en el reconocimiento facial como la máquina de vectores de soporte (SVM, por sus siglas en inglés), Naive Bayes y Árboles de decisión (Decision Trees) por mencionar algunos.

El algoritmo SVM, es un algoritmo de aprendizaje supervisado que se utiliza en problemas de clasificación; éste encuentra los parámetros óptimos de un hiperplano separador que maximiza el margen entre los objetos de entrenamiento. Se utiliza el truco del kernel, que consiste en mapear las entradas, que no son linealmente separables en el espacio de entrada, a un espacio de alta dimensión. El kernel lineal entre dos objetos $\boldsymbol{x}$ y $\boldsymbol{x}'$ se define como $Klineal = \boldsymbol{x} \cdot \boldsymbol{x}'$, mientras que el kernel radial, o Gaussiano, se define como $Kradial = e\text{^}(-(\|x - x\text{^}' \|\text{^}2/(2\sigma\text{^}2))$). Adicionalmente, para incluir regularización, se agrega el parámetro C en el proceso de optimización [30].

El algoritmo probabilístico de clasificación Bayes ingenuo o *Naive Bayes* en inglés, se basa en el teorema de Bayes y supone que las características de los objetos son independientes. En la etapa de entrenamiento, se calculan las probabilidades a *priori* y se encuentran los parámetros que maximizan la verosimilitud dadas las características obtenidas para los objetos de cada clase. Se obtiene la probabilidad *posteriori* de que un objeto evaluado pertenece a una clase con base a sus características [31]. Por otro lado, los modelos de árboles de decisión para clasificación consisten en un conjunto de

reglas para estratificar el espacio de predicción en subgrupos más pequeños y homogéneos [32].

3. Descripción de la plataforma digital

En esta sección, presentamos la descripción de la plataforma digital para el control de acceso en un campus universitario mediante reconocimiento facial. El objetivo es detectar a las personas autorizadas/no autorizadas que desean ingresar al campus de forma automática por medio de reconocimiento facial.

Cuando una persona no autorizada desea entrar a la institución, se manda automáticamente una alerta a una aplicación móvil diseñada para los guardias de seguridad del campus quienes están capacitados para tomar la acción correspondiente; por ejemplo, tomar sus datos y permitirle la entrada temporalmente, registrarlo en el sistema, o bien, no permitirle la entrada.

El diseño de la plataforma se basó en la propuesta de Brandon Joffe [29], en donde considera un servidor dedicado al procesamiento de reconocimiento facial. Para ello, se usa una comunicación basada en web que permite la interacción con el usuario y manipulación de la plataforma. Adicionalmente, se desarrolló una aplicación móvil para el manejo de alertas, la cual se comunica con el servidor central por medio de Firebase[3], obteniendo de esta manera, información sobre el proceso de reconocimiento facial que se enviará como alerta a la aplicación móvil, ver Figura 2.

La plataforma está diseñada para establecer comunicación con cámaras IP por medio de las cuales se obtendrá el rostro de las personas. Una vez que se obtiene la información de la(s) cámara(s) se procede a realizar el proceso de reconocimiento y automáticamente se envía la alerta a la aplicación móvil cuando se identifica una persona no autorizada por la institución, ver Figura 3. En este caso, el objeto IPCamera transmite el video directamente desde una cámara IP para que sea procesado y transmitirlo al cliente web.

Cada cámara tiene su propio objeto MotionDetector y FaceDetector que son utilizados por otros procesos para llevar a cabo el reconocimiento facial. En ese sentido, el proceso de reconocimiento está basado en el algoritmo OpenFace usando el modelo de red neuronal pre-entrenado *nn4.small2.v1.t7* y el clasificador LinearSVM.

4. Experimentación y resultados

Esta sección describe la experimentación llevada a cabo para la etapa de clasificación del proceso de reconocimiento facial descrito en la Sección 2 y se presenta la implementación de una plataforma digital.

3 https://firebase.google.com/

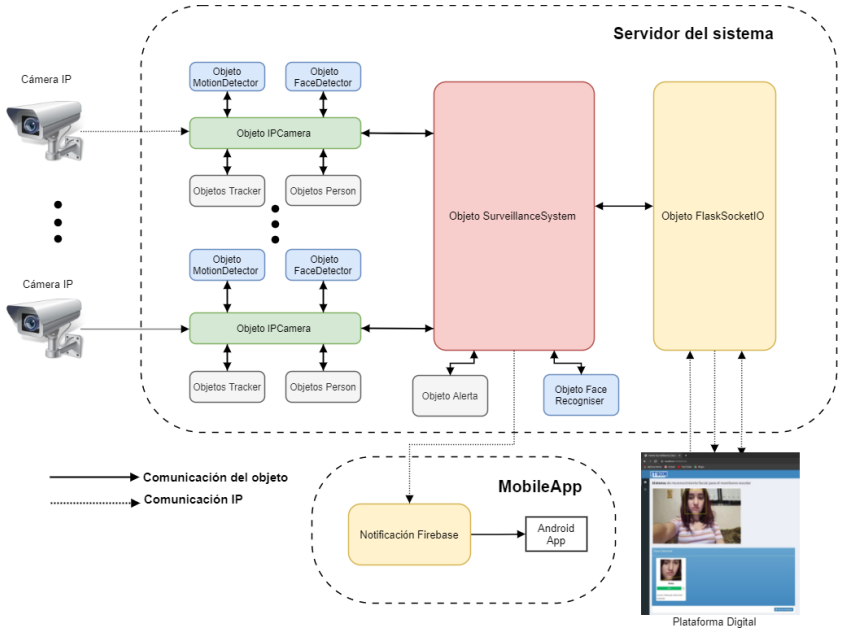

Fig. 3. Funcionamiento de la plataforma para el control de acceso y envío de alertas.

4.1. Clasificación de características para el reconocimiento facial

Para llevar a cabo la comparación de clasificadores se utilizó una computadora Lenovo, Intel Core i5 con 8GB de RAM y sistema operativo Linux Ubuntu kernel 5.3.0-40. En ese sentido, para los experimentos se utilizó la base de datos LFW [28] la cual es ampliamente utilizada en el reconocimiento facial. Esta base de datos cuenta con 5749 clases (personas) y 13233 fotos.

Se observó que la mayoría de las clases cuentan con una sola foto lo cual para este tipo de experimentación es insuficiente. 5591 clases tienen entre 1 y 10 fotos; 96 clases tienen entre 10 y 20 fotos; 28 clases tienen entre 20 y 30 fotos; 15 clases tienen entre 30 y 40 fotos; 7 clases tienen entre 40 y 50; 4 clases tienen entre 50 y 60 fotos y el resto de clases más de 60 fotos por clase.

Considerando esto, se decidió utilizar el grupo de 28 clases que contiene un total de 653 fotos y el grupo de 4 clases que contiene un total de 212 fotos. Los resultados de este experimento se pueden observar en las Figuras 4 y 5.

Para llevar a cabo el entrenamiento y la prueba de cada uno de los clasificadores, se dividió la base de datos en un 80% para entrenamiento y un 20% para pruebas para cada uno de los grupos (28 y 4 clases). Como se puede observar en la Tabla 1, LinearSVM y GaussianNB son mejores que RadialSVM y DecisionTrees en ambos grupos de clases. Es por ello, que se decidió utilizar LinearSVM como clasificador en el proceso de reconocimiento facial.

Adicionalmente, se evaluó el sistema de reconocimiento de la plataforma utilizando una base de datos de la institución con 91 fotos de 3 estudiantes y 1 profesor (4 clases)

Tabla 1. Comparación del rendimiento de 4 clasificadores utilizados en el reconocimiento facial sobre la base de datos LFW.

Clasificador	28 clases		4 clases	
	% Reconocimiento	F-Measure	% Reconocimiento	F-Measure
LinearSVM	96.15	0.9804	100	1
RadialSVM	95.38	0.9764	100	1
DecisionTrees	70.77	0.8288	92.86	0.96295
GaussianNB	96.15	0.9804	100	1

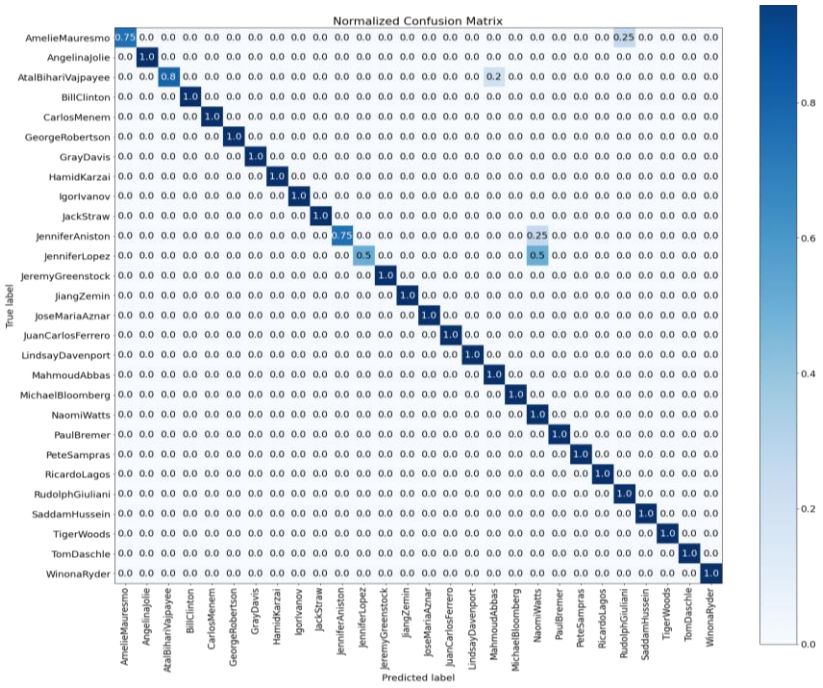

Fig. 4. Matriz de confusión normalizada para 28 clases de la base de datos LFW [28] utilizando Linear SVM.

obteniendo como resultado un porcentaje de reconocimiento del 95.24% y una F-Measure de 0.9978, ver Figura 6.

4.2. Plataforma digital para el control de acceso universitario

La plataforma se implementó en una computadora Lenovo, Intel Core i5 con 8GB de RAM y sistema operativo Linux Ubuntu kernel 5.3.0-40 y se utilizó la aplicación IP WebCam en un teléfono móvil con sistema operativo Android para usar la cámara del

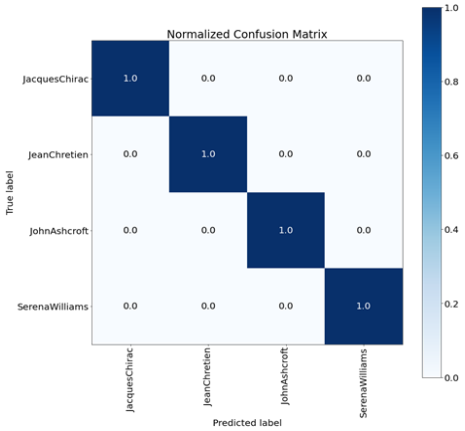

Fig. 5. Matriz de confusión normalizada para 4 clases de la base de datos LFW [28] utilizando LinearSVM.

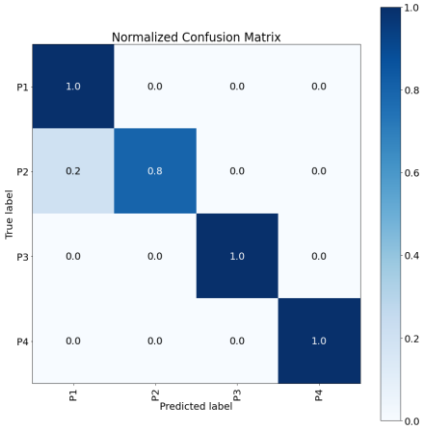

Fig. 6. Matriz de confusión normalizada utilizando la base de datos de la Institución (4 clases).

teléfono como una cámara IP. En el caso de la aplicación móvil, ésta fue desarrollada con Java en Android Studio en un Huawei Nova 3 y probada en un Samsung S9.

Por otro lado, la plataforma cuenta con la opción de agregar personas al sistema en tiempo real. Si esto sucede, se debe de re-entrenar el clasificador en la misma plataforma para que la persona sea reconocida como persona autorizada en el futuro.

En ese sentido, la Figura 7 presenta un ejemplo de la plataforma mostrando una persona autorizada por el sistema, en donde se despliegan datos como nombre de la persona y el porcentaje de confianza indicando con ello, la certeza del reconocimiento

Por otro lado, la Figura 8 presenta un ejemplo de la plataforma cuando una persona no autorizada se detecta, si esto sucede, se envía automáticamente una alerta a la aplicación móvil, ver Figura 9b. De lo contrario, cuando el sistema reconoce a una persona autorizada la aplicación móvil se presenta sin cambios, ver Figura 9a.

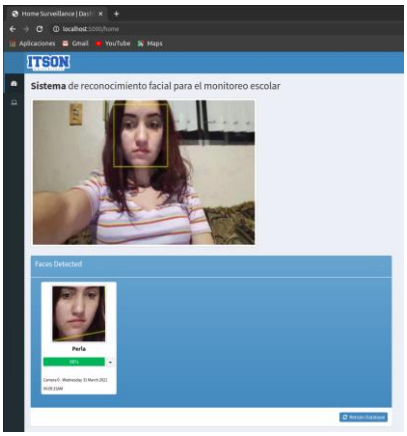

Fig. 7. Ejemplo de la plataforma reconociendo a una persona autorizada por el sistema.

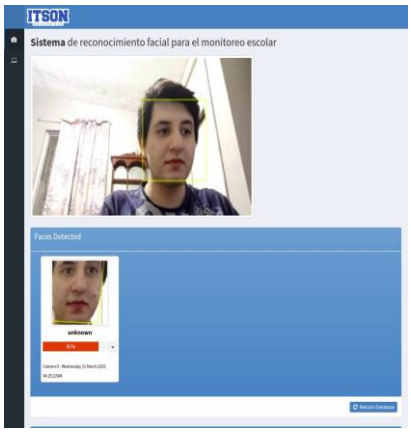

Fig. 8. Ejemplo de la plataforma mostrando a una persona NO autorizada.

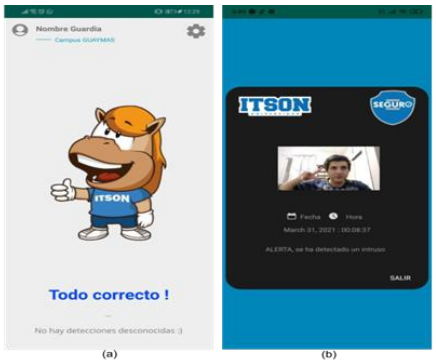

Fig. 9. Alerta enviada de la plataforma a la aplicación móvil. (a) Persona autorizada (b)

5. Conclusiones y trabajo futuro

Este trabajo presenta el diseño e implementación de una plataforma digital para el control de acceso del personal académico, administrativo y estudiantil a un campus universitario mediante reconocimiento facial. La plataforma digital fue implementada en un equipo local, sin embargo, se tiene programado migrarla a un servidor institucional con cuatro cámaras IP HikVision H.265+ instaladas en los accesos de entrada al campus.

Esta plataforma permite ingresar nuevas personas al sistema y cuenta con el envío de alertas a una aplicación móvil de forma automática cuando una persona no autorizada intenta ingresar a la institución. Para la etapa de reconocimiento facial se utilizó el algoritmo de código abierto OpenFace con el modelo de red neuronal pre-entrenado *nn4.small2.v1.t7* y el clasificador LinearSVM.

Para esto, se presenta una comparación de 4 clasificadores utilizando la base de datos LFW con 28 y 4 clases mostrando un rendimiento superior al 95%. Adicionalmente, se evaluó la plataforma con 4 clases utilizando imágenes del personal académico y estudiantil mostrando un porcentaje de reconocimiento del 95.24% y un valor de F-Measure de 0.9978.

En ese sentido, como trabajo futuro se tiene pensado implementar la plataforma digital *in situ*, y para ello será necesario preparar una base de datos institucional y proponer un esquema de reconocimiento a gran escala ya que la institución cuenta con alrededor de 21,818 estudiantes y 2,820 trabajadores, lo que representa alrededor de 24,638 clases para el sistema de reconocimiento facial.

Agradecimientos. Este trabajo fue financiado por el programa para el desarrollo profesional docente (PRODEP) 2019.

Referencias

1. INEGI. Departamento de comunicación social. Encuesta Nacional de Seguridad Pública Urbana (ENSU) México. https://www.inegi.org.mx/contenidos/saladeprensa/boletines/2020/ensu/ ensu2020_10.pdf (2021)
2. Gonzalez-Garcia, C., Meana-Llorián, D., Pelayo G-Bustelo, B. C., Cueva-Lovelle, J. M., Garcia-Fernandez, N.: Midgar Detection of people through computer vision in the internet of things scenarios to improve the security in Smart Cities, Smart Towns, and Smart Homes. Future Generation Computer Systems, Elsevier, vol. 76, pp. 301–313 (2017) doi: 10.1016/j.future.20 16.12.033
3. Dong, X., Kong, X., Zhang, F., Chen, Z., Kang, J.: On campus a mobile platform towards a smart campus. SpringerPlus, pp. 1–9 (2016) doi: 10.1186/s40064-016-2608- 4
4. Kumar, P. M., Gandhi, U., Varatharajan, R., Manogaran, G., Jidhesh, R., Vadivel, T.: Intelligent face recognition and navigation system using neural learning for smart security in internet of things. Cluster Computing, no. 22, pp. 7733–7744 (2019) doi: 10.1007/s10586-017-1323-4
5. Rameswari, R., Kumar, S. N., Aananth, M. A., Deepak, C.: Automated access control system using face recognition. Materials Today Proceedings. vol. 45, pp. 1251–1256 (2020) doi: 10.1016/j.matpr.2020.04.664
6. Lee, H., Park, S. H., Yoo, J. H., Jung, S. H., Huh, J. H.: Face recognition at a distance for a stand-alone access control system. Sensors, vol. 20, pp. 1–18 (2020) doi: 10.3390/s20030785
7. Tisse, C. L., Martin, L., Torres, L., Robert, M.: Person identification technique using human iris recognition. In: Proceedings of. Vision Interface Canadian Image Processing and Pattern Recognition Society (CIPPRS), 15th International Conference on Vision Interface. pp. 294–299 (2002)
8. Bueno-Delgado, M. V., Pavón-Marino, P., De-Gea-Garcia, A., Dolon-Garcia, A.: The smart university experience: An NFC-based ubiquitous environment. In: Proceedings of the IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 799–804 (2012) doi: 10.1109/IMIS.2012.110
9. Skinner, V.: Universitiy of San Francisco using facial recognition to track dorm activity. EAG. News.org (2014). https://www.eagnews.org/2014/11/university-of-san-francisco-using-facial-recognition-to-track-dorm-activity/

10. Mikkelsen, D.: Two Seattle schools among first to use facial recognition software in the US. K5 News (2018). https://www.king5.com/article/news/education/two-seattle-schools-amongfirst-to-use-facial-recognition-software-in-us/281-609937626
11. Xiaofei, Du.: Facial recognition system introduced to Beiging Normal University dorms. People's Daily Online (2017) http://en.people.cn/n3/2017/0516/c90000-9216437.html
12. Gan, N.: Want to get into Peking University? Just show your face. Inkstone news (2018). https://www.inkstonenews.com/tech/peking-university-installs-facial-recognition-system-students-and-staff-campus-gate/article/2152893
13. Oh, S. H., Kim, G. W., Lim, K. S.: Compact deep learned feature-based face recognition for visual internet of things. The Journal of Supercomputing, vol. 74, pp. 6729–6741 (2018) doi: 10.1007/s11227-017-2198-0
14. Amos, B., Ludwiczuk, B., Satyanarayanan, M.: Openface: A general-purpose face recognition library with mobile applications. Technical Report, CMU-CS-16-118, CMU School of Computer Science (2016)
15. Lowe, D.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, vol. 60, pp. 91–110 (2004) doi: 10.1023/B:VISI.0000029664.99615.94
16. Ahonen, T., Hadid, A., Pietikainen, M.: Face description with local binary patterns: Application to face recognition. IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 28, pp. 2037–2041 (2006) doi: 10.1109/TPAMI.2006.244
17. Déniz, O., Bueno, G., Salido, J., De la Torre, F.: Face recognition using histograms of oriented gradients. Pattern Recognition Letters, vol. 32, pp. 1598–1603 (2011) doi: 10.1016/j.patrec.2 011.01.004
18. Perez, C. B., Olague, G.: Learning invariant region descriptor operators with genetic programming and the f-measure. In: Proceedings of the 19th International Conference on Pattern Recognition, pp. 1–4 (2008) doi: 10.1109/ICPR.2008.4761178
19. Guo, G., Zhang, N.: A survey on deep learning based face recognition. Computer vision and image understanding, 189, pp-1–47 (2019) doi:10.1016/j.cviu.2019.102805
20. Yang, S., Luo, P., Loy, C. C., Tang, X.: From facial parts responses to face detection: A deep learning approach. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 3676–3684 (2015)
21. Farfade, S. S., Saberian, M. J., Li, L. J.: Multi-view face detection using deep convolutional neural networks. In: Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, pp. 643–650 (2015) doi: 10.1145/2671188.2749408
22. Parkhi, O. Vedaldi, M., Zisserman A.: Deep face recognition. In British Machine Vision Conference, pp. 1–12 (2015)
23. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 815–823 (2015)
24. Liu, Y., Li, H., Wang, X.: Rethinking feature discrimination and polymerization for large-scale recognition. arXiv preprint arXiv (2017) doi: 10.48550/arXiv.1710.00870
25. Deng, J., Guo, J., Xue, N., Zafeiriou, Arcface S.: Additive angular margin loss for deep face recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4690–4699 (2019)
26. Yi, D., Lei, Z., Liao, S., Li, Stan. Z.: Learning face representation from scratch arXiv preprint arXiv. (2014) doi: 10.48550/arXiv.1411.7923
27. Ng. H. W., Stefan, W.: A data-driven approach to cleaning large face datasets. IEEE International Conference on Image Processing, pp. 343–347 (2014) doi: 10.1109/ICIP.2014.7025068
28. Huang, G., B., Mattar, M., Berg, T., Learned-Miller, E.: Labeled Faces in the Wild A database for studying face recognition in unconstrained environments. Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition, Erik Learned-Miller and Andras Ferencz and Frédéric Jurie (2008)

29. Joffe, B.: Home Surveillance with facial recognition. https://github.com/BrandonJoffe/home_ surveillance#features
30. Muller, K. R., Mika, S., Ratsch, G., Tsuda, K., Scholkopf, B.: An introduction to kernel-based learning algorithms. IEEE transactions on neural networks, pp. 181–201 (2001)
31. Duda, R. O., Hart, P. E., Stork, D. G.: Pattern Classification, 2nd edition, Jhon Wiley & Sons, Inc (2001)
32. James, G., Witten, D., Hastie, T., Tibshirani, R.: An introduction to statistical learning. Ed. Springer, 112, pp. 3–7 (2013) doi: 10.1007/978-1-0716-1418-1.pdf