TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica

Escuela de Computación
Curso: IC6831
Aseguramiento de la Calidad del Software
Grupo 2, II Semestre 2017
Profesor: Saúl Calderón Ramírez

Requerimientos de Calidad

Integrantes:
Brandon Dinarte 2015088894
Armando López 2015125414
Andrey Mendoza 2015082908
Julian Salinas 2015114132

Jueves 12 de Octubre del 2017
ITCR, Sede Central Cartago

1. Introducción

Con el fin de mantener un estándar aceptable de calidad en el proyecto, se incluyen una serie de requerimientos relacionados a la misma, que deben ser cumplidos durante el proceso de desarrollo de todos los ítems de configuración definidos para el sistema.

Se incluyen los detalles de la definición de los ítems de la configuración, los procedimientos para su correspondiente modificación, así como los encargados de su revisión. Además se describen los procesos de utilización de las herramientas de análisis de código y para implementación de las métricas utilizadas.

2. Propósito del Documento

El objetivo de este documento consiste en definir los requerimientos de calidad para el proyecto EigenFaces, además de la revisión de la misma, mediante la implementación y análisis de métricas. Además, se describe los componentes de diseño del sistema junto con los requerimientos funcionales que les dan origen.

3. Validación del Diseño

La implementación de diseño para las diferentes partes de un sistema de software es bastante beneficioso para la construcción y mantenibilidad del mismo. Los diagramas permiten un análisis fácil y rápido sobre la estructura de un sistema, lo que permite traducir los requerimientos a un lenguaje de programación, verificar que el alcance del sistema es el acordada, extender funcionalidades y otros de manera fácil.

Esta sección pretende mostrar el o los requerimientos funcionales o no funcionales del proyecto que se satisfacen mediante los diseños implementados para el sistema de *Eigen-Faces*. Para ello, se citará uno o más requerimientos y se mostrarán ítems de diseño que demuestren el cumplimiento de las funcionalidades.

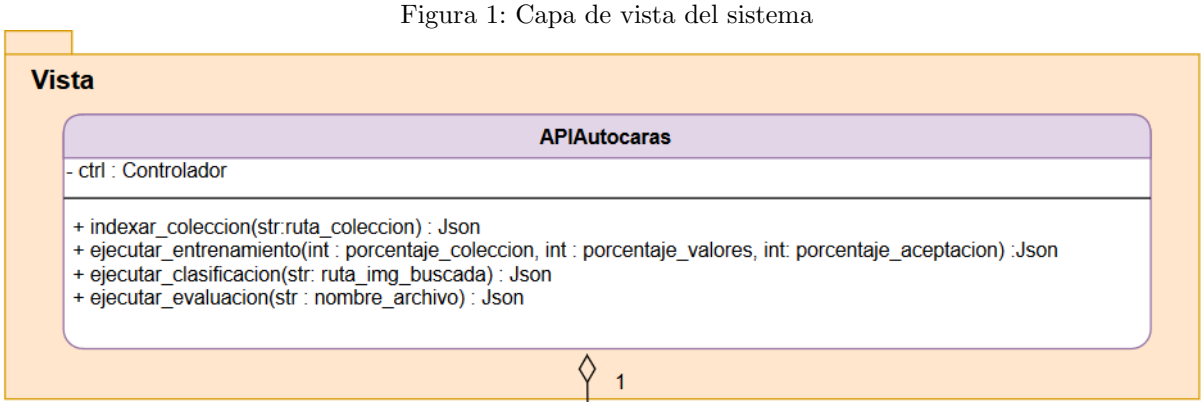
3.1. Modo Entrenamiento

Los requerimientos solicitados por el cliente para el entrenamiento, fueron los siguientes:

- **REQ #1 - Ruta del conjunto de imágenes:** el sistema deberá tomar como parámetro la ruta donde se encuentran almacenadas las imágenes sobre las que se va a realizar el entrenamiento.
- **REQ #2 - Carpetas por cada sujeto:** en la ruta especificada para el entrenamiento, deben existir carpetas por cada sujeto. Dentro de ellas, se encuentran las imágenes de cada sujeto.
- **REQ #4 - Almacenamiento de las auto-caras:** una vez entrenado el sistema, se deben almacenar los datos, esto con el fin de luego utilizar estas auto-caras precalculadas para el modo de identificación.
- **REQ #5 - Parámetros de entrenamiento:** será la cantidad de autovectores a conservar en el nuevo conjunto de vectores base.

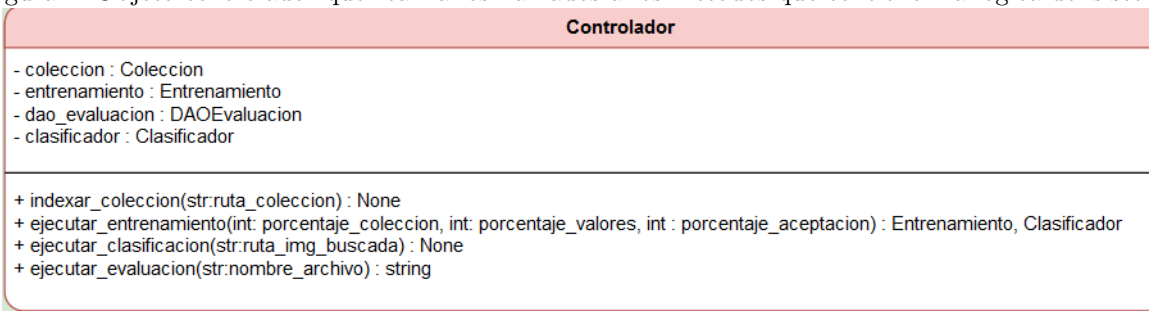
Para cumplir con este modo, se crea una interfaz de usuario que solicita la ruta de la colección con los debidos sujetos como subdirectorios de la misma, como se puede apreciar en la siguiente imagen. El sistema primero indexa la colección, solicitando la ruta como parámetro y luego hace uso de los otros parámetros de porcentaje de autovectores y autovalores para realizar el entrenamiento.

Figura 1: Capa de vista del sistema



Dentro de la lógica interna del programa, se hace uso de un controlador que se encarga de instanciar los parámetros solicitados por el usuario para hacer el llamado al entrenamiento, cumpliendo así con los requerimientos: **#1, #2 y #5**.

Figura 2: Objeto controlador que realiza los llamados a los métodos que contienen la lógica del sistema

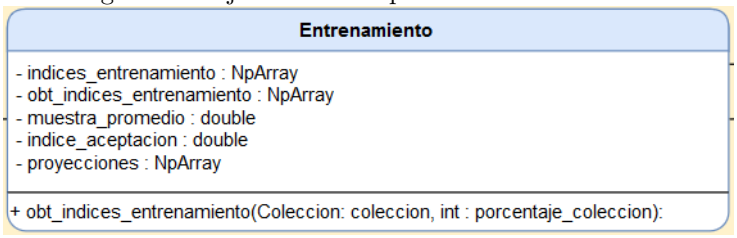


El indexar colección, lo que hace es crear un objeto de configuración utilizando el patrón singleton para que pueda ser accedido por los demás objetos que lo necesiten sin la necesidad de pasar estos datos por parámetros.

Figura 3: Objeto de configuración que utiliza el patrón singleton

Para el cumplimiento del **#4** se hace uso de una biblioteca de almacenamiento persistente del estado actual de los objetos, Pickle. El objetivo, es guardar en persistencia el estado final del objeto entrenamiento y colección para poder cargarlos y hacer una evaluación sin la necesidad de realizar el entrenamiento nuevamente.

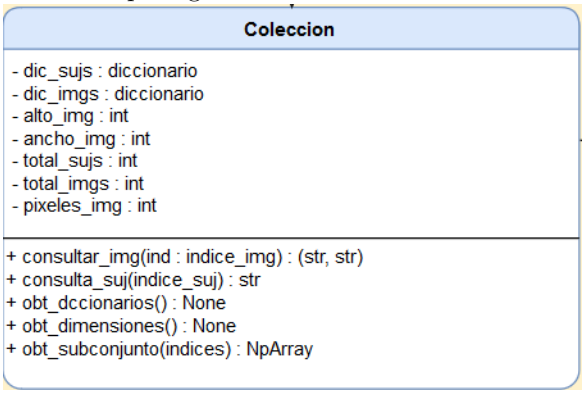
Figura 4: Objeto utilizado para entrenar el sistema



Todo el proceso del algoritmo y guardado en persistencia de los objetos necesarios, se hace en el único método de la clase entrenamiento mostrada en la imagen anterior.

Para la evaluación, también es necesario conocer el nombre del sujeto, la ruta y demás, por lo que también se guarda en persistencia. El contenido de esta clase se muestra en la siguiente imagen:

Figura 5: Objeto utilizado para guardar la información de la colección de imágenes

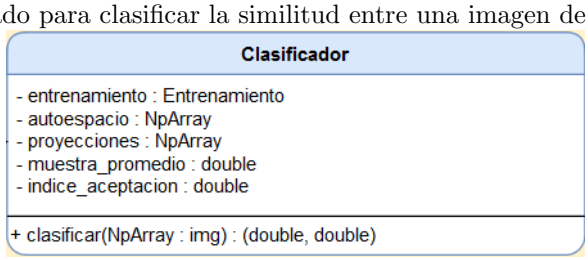


3.2. Modo Evaluación

- **REQ #3 - Generación de Autocaras:** para el entrenamiento del sistema, se utilizará el algoritmo del centroide más cercano.
- **REQ #6 - Carga de una imagen:** para la identificación de un sujeto, el usuario debe poder cargar una imagen de un rostro y luego proceder a la identificación del mismo.
- **REQ #9 - Resultados de la identificación:** una vez identificado el sujeto, se mostrará la etiqueta que tiene el mismo en la base de datos.

El cumplimiento de estos requerimientos se da en el siguiente objeto llamado clasificador, el cual recibe el arreglo de la imagen cargada previamente desde el controlador, la cual a su vez fue indicada por el usuario desde la interfaz de usuario (ver 1). Haciendo uso de la misma figura 1 sobre el servicio web, en el objeto Json que retorna el método de clasificación, se encuentran los resultados de la clasificación.

Figura 6: Objeto utilizado para clasificar la similitud entre una imagen de entrada con la colección



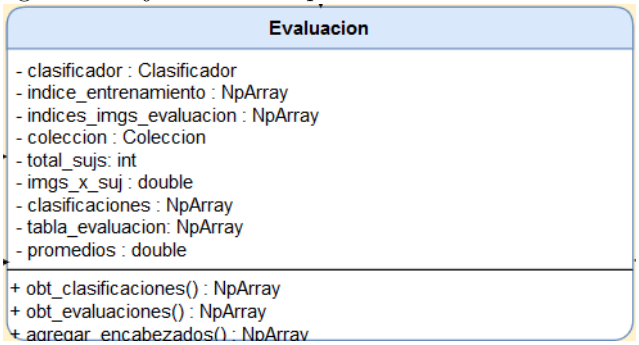
3.3. Modo Evaluación

El cliente solicita la evaluación del sistema con el objetivo de conocer la fiabilidad de los resultados. Para ello, se definieron los siguientes requerimientos:

- **REQ #7 - Métrica de precisión:** para medir la precisión del sistema, deberá cargarse un conjunto de pruebas para y calcular los falsos positivos y negativos, la precisión y el recall. Implementando las métricas vistas en el curso, se generará un informe en un archivo con formato ".csv"que contenga los resultados.
- **REQ #11 - Comparación de entrenamientos:** debe realizarse una comparación entre el entrenamiento mediante el algoritmo del centroide y el otro algoritmo de clasificación supervisado no paramétrico implementado.

Para el cumplimiento de estos requerimientos se decidió crear un objeto llamado Evaluación, el cual tiene la lógica de evaluación del sistema mediante los falsos positivos y negativos para hacer el cálculo de la precisión y la exhaustividad del sistema.

Figura 7: Objeto utilizado para la evaluación del sistema



Una vez finalizada la evaluación del sistema, se debe crear un archivo en formato ".csv" para la legibilidad del mismo. Para ello se hace uso de la biblioteca Pandas.

4. Modo de uso de las herramientas de verificación de código

Existen dos herramientas que fueron usadas para verificar la codificación durante el desarrollo del proyecto. Estas son **SonarQube 6.5**, una aplicación web que analiza *bugs*, vulnerabilidades, olores de software y duplicaciones, y **PyCharm 2017.2**, un IDE especializado para Python que permite realizar análisis estático del código mientras hace sugerencias para en ocasiones donde se hace mal uso de la sintaxis, procedimientos e incluso donde se hace caso omiso a las reglas PEP8 (estándar de codificación usado).

4.1. SonarQube

SonarQube puede ser usado tanto en modo *online* como local, sin embargo, se hará énfasis en este último. Hay que tomar en cuenta que SonarQube usa un modelo *cliente-servidor*, por lo tanto, si se desea usar de modo local, es necesario obtener los dos programas para realizar el análisis, el servidor (que incluye interfaz web) y el cliente.

Para descargar el servidor accedemos a la página oficial de SonarQube `https://www.sonarqube.org/`. Una vez en dicha página, el botón para realizar la descarga es fácil de visualizar. Por otra parte, para obtener uno de los programas cliente es necesario ingresar a la sección de documentación y descargas. En está sección se puede notar que existen distintos tipos de clientes basados en un método de análisis, por ejemplo, SonarQube para Maven, Ant, Jenkins o Gradle. De estos, se hará uso del cliente predeterminado, nombrado como SonarQube Scanner.

Los archivos obtenidos de estas descargas serán los siguientes, **sonarqube-6.5** (servidor) y **sonar-scanner-2.6.1** (escáner cliente). Es necesario que dichos archivos sean descomprimidos en una ruta donde los permisos de lectura no sean problema. Una vez descomprimidos podemos realizar los pasos necesarios para el análisis:

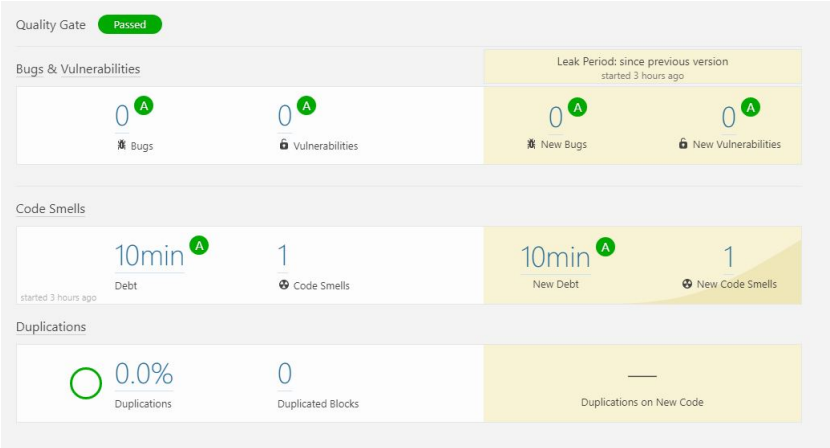
1. El proyectos se desarrolla sobre el sistema operativo Windows 10 de 64 bits, por tanto, para inicializar el servidor se debe acceder a la ruta `/sonarqube-6.5/bin/windows-x86-64`. Es necesario que antes de ejecutar se inhabilite el antivirus ya que según la página oficial, el programa podría tener un comportamiento restringido o indeterminado. Estando en la carpeta mencionada se procede a ejecutar el archivo **StartSonar.bat** usando el *símbolo del sistema*. Una vez ejecutado es necesario esperar aproximadamente 10 segundos a que nos muestre un mensaje de confirmación. En caso de éxito el sistema nos mostrará el mensaje *SonarQube is up!*.

2. Después de inicializar el servidor es necesario ejecutar el escáner para analizar el código. Primeramente debemos copiar el directorio con el código fuente del proyecto dentro del directorio `/sonar-scanner-2.6.1/bin/`. Luego en el directorio `/sonar-scanner-2.6.1/conf/` se debe crear un archivo de configuración con el nombre **sonar-scanner.properties**. Este archivo debe contener las propiedades de nuestro proyecto, como por ejemplo, un id único, el nombre, la versión y el nombre del directorio que se acaba de copiar y que contiene el código fuente. De forma más especifica el archivo de configuración debe tener los argumentos que se ejemplifican a continuación:

```
sonar.projectKey=id_del_proyecto
sonar.projectName=nombre_del_proyecto
sonar.projectVersion=versión_del_proyecto
sonar.sources=carpeta_con_código_fuente
```

3. Dentro de `/sonar-scanner-2.6.1/bin/` ejecutamos el archivo **sonar-scanner.bat** mediante el *símbolo del sistema*. Con esto el proyecto comenzará a ser analizado. Al terminar, el programa nos mostrará un mensaje con el resultado de la operación. Si esta fue correcta nos mostrará una *url* que podemos abrir en el navegador para ver los resultados obtenidos. Está ruta usualmente es `http://127.0.0.1:9000/dashboard?id=id_del_proyecto`. La figura 8 muestra como se verían los resultados después de ejecutar el análisis.

Figura 8: Resultados del análisis con SonarQube



4. Se puede revisar con más detalle los resultados obtenidos accediendo a una de las tres categorías que se muestran en la figura 8. También podemos revisar cada uno de los problemas encontrados junto con una estimación de cuánto tardaría un desarrollador solucionándolos. Un ejemplo de como la aplicación representa los problemas encontrados está dado en la figura 9

Figura 9: Problemas encontrados por medio de SonarQube

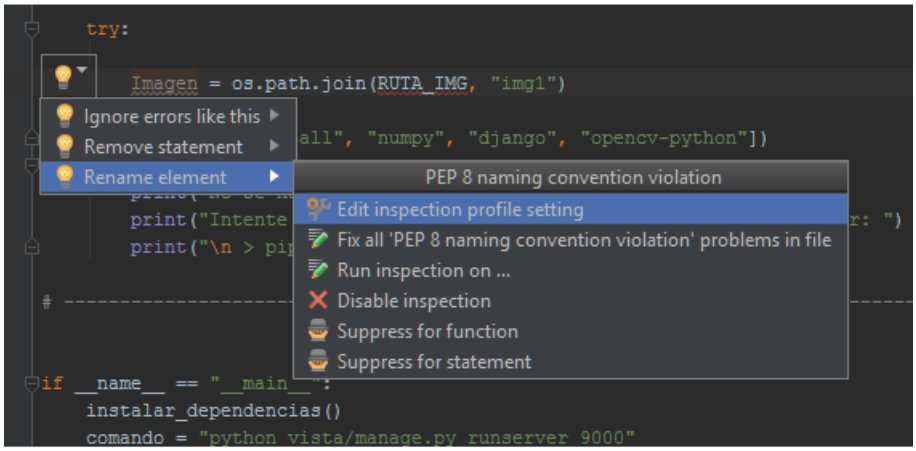
4.2. PyCharm

Este provee un ambiente predispuesto para cumplir con PEP8 y PEP257, estándar de codificación y documentación respectivamente. A su vez, realiza sugerencias para cumplir con PEP 20, o también llamado *The Zen of Python*, el cual reúne una serie de buenas practicas que se deberían de tomar en cuenta a la hora de programar. PyCharm, además de las funciones de autocompletado y sugerencias, también analiza el código fuente de forma estática, haciendo revisiones básicas de módulos, clases, funciones y variables, así como, de documentación interna y formato. Nota: *PEP es la abreviación de Python Enhancement Proposals.*

Específicamente, entre las principales inspecciones que realiza PyCharm se pueden mencionar, el cumplimiento de *PEPs*, indentaciones incorrectas, funciones, clases o módulos obsoletos, código inalcanzable, docstrings incorrectos o incompletos, declaraciones sin efecto, opacidad en los nombres, verificación de tipos al aplicar operadores, entre otros.

PyCharm lo podemos obtener de su página oficial https://www.jetbrains.com/pycharm/. Para hacer uso de las utilidades descritas anteriormente debemos ingresar al menú de opciones y habilitar las inspecciones necesarias. Es decir, ingresar a `File/Settings/Editor/Inspections` dentro del editor y marcar las inspecciones que se desean que el IDE realice. Un ejemplo de una inspección realizada por PyCharm se muestra en la figura 10.

Figura 10: Inspección realizada por *PyCharm*



Para maximizar su utilidad se usará la extensión SolarLint, la cual proporciona de forma instantánea información útil sobre nuevos errores encontrados y problemas de calidad inyectados en el código fuente, lo cual, disminuye el tiempo que corresponde a las revisiones de la calidad del código. Cabe destacar que es una herramienta versátil, ya que puede analizar código en diferentes lenguajes usando como base el server de SonarQube, esto permitirá que distintos módulos del sistema, sin importar el lenguaje usado, puedan ser analizados.

La extensión SolarLint es de gran utilidad, sin embargo, no cubre ciertas características o reglas puesto que su objetivo no es ralentizar el IDE, por tanto, cada vez que se alcance un punto de control del sistema, este será analizado por medio de SonarQube, el cual requiere un poco más de tiempo para inicializar.

5. Métricas implementadas

Como parte del proceso de aseguramiento de calidad se incluyen los resultados de la aplicación de dos métricas sobre el trabajo realizado hasta el momento para el sistema.

5.1. Porcentaje de comentarios por líneas de código

Esta métrica está relacionada al atributo de **analisibilidad** dentro de la categoría de **mantenibilidad**, lo cual resulta de utilidad ante la posibilidad de una posterior continuación del trabajo realizado en el sistema por un diferente grupo de desarrolladores.

Para el análisis de esta métrica se utiliza una herramienta gratuita para el conteo automático de líneas de código según algunos aspectos que pueden resultar de importancia, el nombre de la herramienta es **CLOC**, cuyas iniciales significan **Count Lines of Code**.

CLOC

1. **Datos de la herramienta**
   Licencia: Uso gratuito.
   Versión: 1.74.
   Fuente del repositorio: https://github.com/AlDanial/cloc
   Enlaces de descarga: https://github.com/AlDanial/cloc/releases/tag/1.74

2. **Uso de la herramienta**
   La función específica de CLOC es el conteo de líneas de código bajo diferentes criterios. Para definir los criterios del conteo es necesario iniciar la ejecución de CLOC desde consola con los parámetros requeridos. Para la medición se utilizaron los archivos fuente, localizados en las carpetas de **controlador** y **modelo** que pueden encontrarse dentro de la carpeta **Codigo** del proyecto. A continuación se especifican las instrucciones ejecutadas para realizar las mediciones, las cuales tienen la forma: *ruta del ejecutable _ ruta de la carpeta con los archivos fuente _ parametros*

   a) La primer instrucción ejecutada es para hacer el conteo de los archivos pertenecientes al controlador. El parámetro **−−include−lang=Python** establece que únicamente se tomen en cuenta los archivos **.py**, y el parámetro **−−by−file** define el despliegue de los resultados divididos por archivo.

Instrucción: >cloc-1.74.exe controlador −−include−lang=Python −−by−file
Nota: La consola se encuentra posicionada en la carpeta de **Codigo** junto con el ejecutable de CLOC.

b) La segunda instrucción es igual a la anterior excepto por la carpeta **controlador** que cambia por **modelo**.
Instrucción: >cloc-1.74.exe modelo −−include−lang=Python −−by−file

3. Resultados obtenidos

A continuación se muestran las salidas desplegadas por la herramienta utilizada, con base a dichas salidas se establece un análisis de la métrica del porcentaje de comentarios por líneas de código.

```
C:\Users\Brandon\Documents\TEC\ACS\GIT\Autocaras\Codigo>cloc-1.74.exe controlador --include-lang=Python --by-file
       4 text files.
       4 unique files.
       2 files ignored.

github.com/AlDanial/cloc v 1.74  T=0.50 s (8.0 files/s, 632.0 lines/s)
-------------------------------------------------------------------------------
File                                     blank        comment           code
-------------------------------------------------------------------------------
controlador\controlador.py                  38             41             48
controlador\api_autocaras.py                39             31             47
controlador\dao_evaluacion.py               13             12             22
controlador\configuracion.py                 8              4             13
-------------------------------------------------------------------------------
SUM:                                        98             88            130
-------------------------------------------------------------------------------
```

Figura 11: Resultados de la métrica en los archivos de controlador.

Según la figura anterior se definen los siguientes datos de importancia para los archivos de controlador.

- Cantidad de comentarios por documento: $[41, 31, 12, 4]$
- Cantidad de líneas por documento: $[89, 78, 34, 17]$
- Total de líneas escritas: $88 + 130 = 218$
- Total de líneas de comentario: 88

Según los datos anteriores se puede llegar a los siguientes resultados.

- Porcentaje comentario/código: $[46\,\%, 40\,\%, 35\,\%, 23\,\%]$
- Porcentaje promedio: $36\,\%$
- Porcentaje de código del controlador: $40\,\%$

```
C:\Users\Brandon\Documents\TEC\ACS\GIT\Autocaras\Codigo>cloc-1.74.exe modelo --include-lang=Python --by-file
      10 text files.
      10 unique files.
       6 files ignored.

github.com/AlDanial/cloc v 1.74  T=0.50 s (20.0 files/s, 1468.0 lines/s)
-------------------------------------------------------------------------------
File                                     blank        comment           code
-------------------------------------------------------------------------------
modelo\coleccion.py                         47             47             58
modelo\evaluacion.py                        45             42             54
modelo\pruebas\test_entrenamiento.py        30             23             41
modelo\pruebas\test_ejemplo.py              14              1             35
modelo\entrenamiento.py                     22             26             28
modelo\pruebas\test_coleccion.py            23             17             24
modelo\clasificador.py                      20             26             21
modelo\pruebas\test_clasificador.py         15             11             21
modelo\utilitarios\conversor.py             10              3             16
modelo\utilitarios\fuentes.py                3              2              9
-------------------------------------------------------------------------------
SUM:                                       229            198            307
-------------------------------------------------------------------------------
```

Figura 12: Resultados de la métrica en los archivos de modelo.

Para los archivos que componen la capa de modelo se realiza un análisis similar al anterior. Según la figura anterior se definen los siguientes datos de importancia para los archivos de modelo.

- Total de líneas escritas: $198 + 307 = 505$
- Total de líneas de comentario: 198

Según los datos anteriores se puede llegar a los siguientes resultados.

- Porcentaje de comentarios en el código de modelo: 39 %

Conclusión general

Según los análisis realizados y la herramienta utilizada, se puede afirmar que la cantidad de comentarios con respecto a la totalidad de líneas escritas ronda el 40 % para el código fuente de la capa del controlador y la capa del modelo.

5.2. Métricas de exactitud

Como métrica para determinar la exactitud del sistema se utilizan los resultados almacenados en el archivo **eval.inform.csv**, en el cual se especifican los valores de verdaderos positivos, falsos positivos y falsos negativos por cada sujeto, así como el cálculo de los valores de exhaustividad y precisión. En la tabla siguiente se destacan los valores más importantes encontrados al promediar los resultados para todos los sujetos.

Métrica	**Porcentaje promedio obtenido**
Exhaustividad	91.22 %
Precisión	90.5 %

Cuadro 1: Promedios de exhaustividad y precisión.

Conclusiones

Con base en los resultados obtenidos, puede observarse que se cumplen las métricas de exactitud con valores superiores al 90 % de los casos, margen que es aceptable considerando la cantidad de sujetos utilizados como base muestral.

5.3. Dependencia modular del sistema

Como métrica para determinar la complejidad de instalación del sistema se definió anteriormente que el mismo no debía contar con más de diez dependencias sobre bibliotecas externas del lenguaje. Se muestra en la tabla siguiente la información de las mismas. Nótese que únicamente se toman en cuenta las bibliotecas que no son parte por defecto de las estándar para Python 3.6.

Dependencia	**Versión**	**Uso de la dependencia**	**Archivos dependientes**
numpy	1.13.3	Manejo de objetos multidimensionales y funcionalidades para operaciones de álgebra lineal.	3
opencv-python	3.3.0	Funcionalidades para procesamiento de imágenes de diferentes formatos.	2
django	1.11.6	Web framework de alto nivel diseñado para trabajar con Python.	6
pandas	0.20.3	Herramientas de análisis de datos de alto rendimiento.	1

Cuadro 2: Información de dependencias.

Conclusiones

Con base en la cantidad de dependencias necesarias para el sistema se puede observar que la métrica de complejidad de instalación es cumplida según lo anteriormente establecido.

6. Manual de administración de la configuración del software

6.1. Definición de los ítems de configuración de software

Los documentos y productos finales o parciales realizados por el equipo del proyecto que están sujetos a la posibilidad de sufrir eventuales cambios durante las diferentes etapas de la finalización de dicho proyecto, son considerados como ítems de la configuración del software. Cada uno de estos elementos posee un procedimiento definido con el bien de asegurar la calidad del mismo posterior a la concreción del cambio en cuestión, en caso de ser necesario. Según el progreso actual del proyecto se incluyen como ítems factibles para cambios los siguientes elementos.

1. **Tipo de ítem:** ítem de análisis de requerimientos.
   Detalle: documento que contiene las especificaciones brindadas por el usuario, que debe poseer el sistema en su versión final para la entrega. Detalla las funcionalidades solicitadas, así como los aspectos esperados implícitos en la solicitud del cliente, con el fin de documentar el acuerdo concretado en el momento de aceptación del escrito por parte del cliente.
   Nombre: Especificación de requerimientos del sistema.
   Herramienta editora: Overleaf Real Time Writing and Publishing Tool.
   Volatilidad: Se considera baja en magnitud de los cambios y baja en la frecuencia de cambio. En términos de la magnitud de los cambios que puedan ser realizados, se esperan que estos sean producto de pequeñas correcciones por motivo de errores en la escritura del documento y no en los contenidos e ideas generales del mismo.

2. **Tipo de ítem:** ítem de análisis de requerimientos.
   Detalle: define los estándares establecidos para la generación del código para los archivos fuente que componen el sistema según lo solicitado por el usuario.
   Nombre: Estándar de codificación.
   Herramienta editora: Overleaf Real Time Writing and Publishing Tool.
   Volatilidad: Baja, el estándar fue establecido al inicio del proyecto y no será modificado a menos que se presente un motivo de fuerza mayor principalmente orientado a la mejora de la calidad del código.

3. **Tipo de ítem:** ítem de análisis de requerimientos.
   Detalle: el documento presenta las características y procedimientos definidos con respecto a la calidad del producto a realizarse durante la duración del proyecto actual.
   Nombre: Requerimientos de calidad.
   Herramienta editora: Overleaf Real Time Writing and Publishing Tool.
   Volatilidad: baja.

4. **Tipo de ítem:** ítem de diseño.
   Detalle: especifica los objetos presentes en el diseño del producto, así como sus respectivas interacciones y comportamientos.
   Nombre: Diagrama de clases del sistema.
   Herramienta editora: Draw.io Flowchart Maker & Online Diagram Software.
   Volatilidad: en la etapa actual del proyecto se espera una volatilidad baja para este ítem.

5. **Tipo de ítem:** ítem de diseño.
   Detalle: especifica las tecnologías y plataformas a utilizarse para la implementación y despliegue del sistema.
   Nombre: Diagrama de despliegue del sistema.
   Herramienta editora: Draw.io Flowchart Maker & Online Diagram Software.
   Volatilidad: en la etapa actual del proyecto se espera una volatilidad prácticamente nula para este ítem, pues todas las tecnologías se consideran estables como para no requerir cambio alguno.

6. **Tipo de ítem:** ítem de diseño.
   Detalle: especif ica las acciones disponibles en la navegación dentro del sistema por parte del usuario, así como una generalidad de las funcionalidades presentes en el mismo.
   Nombre: Diagrama de casos de uso.
   Herramienta editora: Draw.io Flowchart Maker & Online Diagram Software.

Volatilidad: prácticamente nula.

7. **Tipo de ítem:** ítem de implementación.
   Detalle: todos los archivos necesarios para la ejecución del sistema, así como su proceso de inicialización en caso de poseerlo.
   Nombre: Archivos de código fuente.
   Herramienta editora: PyCharm: Python IDE for Professional Developers, v2017.2
   Volatilidad: alta, el desarrollo del código es constante durante la realización del proyecto.

8. **Tipo de ítem:** ítem de pruebas.
   Detalle: consta de las pruebas de las unidades simples del código fuente. Su edición o cambio está sujeto a reglas similares a aquellas del código.
   Nombre: Pruebas unitarias.
   Herramienta editora: PyCharm: Python IDE for Professional Developers, v2017.2
   Volatilidad: media, las pruebas unitarias no suelen variar, pero ante cambios importantes en el código pueden llegar a necesitar cambios radicales.

9. **Tipo de ítem:** ítem de pruebas.
   Detalle: consiste en la muestra de sujetos que está siendo utilizada para medir la eficacia y eficiencia del sistema en el ambiente de desarrollo y pruebas.
   Nombre: Base muestral de pruebas.
   Herramienta editora: sin herramienta.
   Volatilidad: Baja, para no variar el marco de comparación entre diferentes versiones del sistema, la base muestral se mantiene constante durante el proceso de pruebas.

6.2. Herramientas de control de versiones

6.2.1. GitHub

GitHub es un servicio de almacenamiento en línea y un repositorio de control de versiones para diferentes tipos de archivos. Está basado en Git por lo que provee funcionalidades de administración de código fuente y control de versiones, además de otros aspectos que resultan útil para las intenciones del proyecto.

6.3. Procedimientos de cambio

6.3.1. Ítems de análisis de requerimientos

Identificación y documentación de la necesidad de cambio

Información de cambio	**Ítem de análisis de requerimientos**
Nombre del ítem:	Identificador distintivo del documento a modificar.
Motivo(s):	Lista de razones por las que se solicita o realiza el cambio.
Tipo de cambio:	Cambio en requerimiento, corrección de error en redacción.
Autor de cambio:	Encargado del cambio.
Consideraciones:	Información relevante acerca del requerimiento.

Cuadro 3: Plantilla de cambio para ítem de análisis de requerimiento.

Evaluación y aprobación del cambio

Ante un eventual cambio en un ítem de análisis de requerimientos se define el siguiente procedimiento para su evaluación y posterior aceptación en caso de cumplir con los estándares que se hayan especificado para el tipo de ítem.

1. Debe completarse el documento o plantilla de "información de cambio" para cada uno de los ítems en cuestión. Los datos contenidos dentro de dicho documento deben permitir a los encargados de la aceptación del cambio, el comprender las circunstancias y consecuencias del cambio propuesto.

2. La solicitud de cambio es entregada o notificada a los encargados del control de versiones. Por la naturaleza del proyecto el rol de encargado de aceptación para cambios sobre este tipo de ítem, recae en el scrum master o administrador del proyecto actual según sea el caso.

3. Se hace disponible el cambio a los miembros del equipo de aceptación de cambios, para su revisión.

4. Los encargados de la aceptación deben revisar que los cambios especificados sean correctos y cumplan con el formato del documento y estándares del mismo.

5. En caso de rechazarse la solicitud de cambio, se notifica al miembro solicitante junto con la información pertinente a los motivos del rechazo.

Implementación de los cambios

En caso que el cambio sea aprobado se define el procedimiento a seguir para su implementación.

1. En caso de ser un cambio en los requerimientos del sistema, todo el equipo de desarrollo debe ser notificado sobre el mismo.

2. Se procede a hacer oficial el cambio y poner a disponibilidad de todo el equipo el ítem actualizado.

6.3.2. Ítems de diseño

Identificación y documentación de la necesidad de cambio

Información de cambio	**Ítem de diseño**
Nombre del ítem:	Identificador distintivo del diagrama a modificar.
Motivo(s):	Lista de razones por las que se solicita o realiza el cambio.
Aspectos a cambiar:	Detalle del cambio que requiere el diagrama.
Tipo de cambio:	Corrección de errores, rediseño de la solución.
Autor de cambio:	Responsables de llevar a cabo el cambio en cuestión.
Consideraciones:	Implicaciones o consecuencias de la aplicación del cambio.

Cuadro 4: Plantilla de cambio para ítem de diseño.

Evaluación y aprobación del cambio

Ante un eventual cambio en un ítem de análisis de diseño se define el siguiente procedimiento para su evaluación y posterior aceptación en caso de cumplir con los estándares que se hayan especificado para el tipo de ítem.

1. Debe completarse el documento o plantilla de "información de cambio" para cada uno de los ítems en cuestión. Los datos contenidos dentro de dicho documento deben permitir a los encargados de la aceptación del cambio, el comprender las circunstancias y consecuencias del cambio propuesto.

2. La solicitud de cambio es entregada o notificada a los encargados del control de versiones. Los cambios sobre este tipo de ítem tienen como encargados de revisión al arquitecto de software correspondiente a la sección de diseño específica a modificarse, así como el scrum master o administrador del proyecto según sea el caso para evitar incoherencias entre criterios.

3. Se hace disponible el cambio a los miembros del equipo de aceptación de cambios, para su revisión así como al equipo de desarrollo para confirmar la capacidad del mismo de llevar a implementación el rediseño especificado.

4. Los encargados de la aceptación deben revisar que los cambios realizados sean congruentes con los requerimientos del sistema.

5. Los cambios al diseño deben ser revisados para constatar que se ajusten a los principios de buen diseño especificados para el proyecto.

6. En caso de rechazarse la solicitud de cambio, se notifica al miembro solicitante junto con la información pertinente a los motivos del rechazo.

7. Ante la eventualidad de la necesidad de cambio del diseño bajo una propuesta pobre de rediseño se procede a definir una reunión con el arquitecto de software para llevar a cabo un análisis sobre la situación.

Implementación de los cambios

En caso que el cambio sea aprobado se define el procedimiento a seguir para su implementación.

1. El arquitecto de software aplica las correcciones o adaptaciones que considere necesarias a la propuesta de cambio en caso de no ser él mismo el origen de dicha solicitud.
2. Se adecúa la propuesta a los estándares de calidad de diseño definidos para el sistema.
3. Se actualizan los ítems de configuración correspondientes y se hacen disponibles a todo el equipo mediante la herramienta de control de versiones definida.

6.3.3. Ítems de implementación

Identificación y documentación de la necesidad de cambio

Información de cambio	**Ítem de implementación**
Nombre del ítem:	Identificador distintivo de los archivos por modificar.
Tipo de cambios:	Adición de funcionalidad, corrección de errores, optimización.
Autor de cambio:	Responsables de llevar a cabo el cambio en cuestión.
Archivos afectados:	Listado de los archivos fuente afectados de forma indirecta.
Consideraciones:	Implicaciones o consecuencias de la aplicación del cambio.

Cuadro 5: Plantilla de cambio para ítem de implementación.

Evaluación y aprobación del cambio

Ante un eventual cambio en un ítem de implementación se define el siguiente procedimiento para su evaluación y posterior aceptación en caso de cumplir con los estándares que se hayan especificado para el tipo de ítem.

1. Debe completarse el documento o plantilla de "información de cambio" para cada uno de los ítems en cuestión. Los datos contenidos dentro de dicho documento deben permitir a los encargados de la aceptación del cambio, el comprender las circunstancias y consecuencias del cambio propuesto.
2. Por la naturaleza del tipo de ítem, es posible que se requieran cambios en ítems de pruebas relacionados a la implementación cambiaba, en ese caso es necesario también seguir el procedimiento de control de cambios para dichas pruebas.
3. La solicitud de cambio es entregada o notificada a los encargados del control de versiones. Los cambios sobre este tipo de ítem tienen como encargados de revisión al scrum master o administrador del proyecto, además se notifica a todos los desarrolladores sobre el cambio ya sea que afecte su área de forma directa o indirecta, aunque estos no participan en el proceso de aprobación.
4. Se hace disponible el cambio a los miembros del equipo de aceptación de cambios para su revisión.
5. Deben hacerse disponibles las pruebas y sus resultados correspondientes para demostrar el correcto funcionamiento del código luego de haber realizado los posibles cambios.
6. El equipo de aceptación revisa las pruebas y resultados proporcionados, junto con el documento de información del cambio para determinar si es aceptable bajo las circunstancias presentadas.
7. En caso de rechazarse el cambio, se notifica al encargado del mismo, junto con las razones detrás de la decisión.

Implementación de los cambios

En caso que el cambio sea aprobado se define el procedimiento a seguir para su implementación.

1. Los cambios en el código se adecúan según los estándares establecidos para la codificación al inicio del proyecto.
2. El encargado de los cambios debe proceder a implementarlos y agregar las pruebas correspondientes a disposición de todos los miembros del equipo.
3. Habiendo siendo ejecutadas las pruebas con resultados exitosos luego de la integración se define el sistema modificado como la última versión del mismo, según el estándar de versionamiento que se ha definido para el proyecto.

6.3.4. Ítems de pruebas

Identificación y documentación de la necesidad de cambio

Información de cambio	**Ítem de implementación**
Nombre del ítem:	Identificador distintivo de las pruebas por modificar.
Motivo(s):	Lista de razones por las que se solicita o realiza el cambio.
Aspectos a cambiar:	Detalle de los módulos que serán cambiados.
Autor de cambio:	Responsables de llevar a cabo el cambio en cuestión.

Cuadro 6: Plantilla de cambio para ítem de pruebas.

Evaluación y aprobación del cambio

Ante un eventual cambio en un ítem de pruebas se define el siguiente procedimiento para su evaluación y posterior aceptación en caso de cumplir con los estándares que se hayan especificado para el tipo de ítem.

1. Debe completarse el documento o plantilla de "información de cambio" para cada uno de los ítems en cuestión. Los datos contenidos dentro de dicho documento deben permitir a los encargados de la aceptación del cambio, el comprender las circunstancias y consecuencias del cambio propuesto.
2. La solicitud de cambio es entregada y notificada a los encargados del control de versiones. Los cambios sobre este tipo de ítem tienen como encargados de revisión al equipo de desarrollo de los que se esperan sus criterios propios.
3. Se hace disponible el cambio a los miembros del equipo encargado de la revisión de cambios.
4. Se ejecutan las pruebas resultantes del cambio para comprobar su correctitud, dichos resultados son analizados por los encargados de desarrollo en el área de la prueba.
5. En caso de rechazo de los cambios, se descartan las pruebas y se notifica al generador de la solicitud de la necesidad de replantear dichos cambios.

Implementación de los cambios

En caso que el cambio sea aprobado se define el procedimiento a seguir para su implementación.

1. Si el cambio en el ítem de pruebas es aceptado, se estandariza según las especificaciones definidas para la codificación en caso de necesitarlo.
2. Se agregan los cambios implementados a la línea base del sistema para disponibilidad de todos los miembros del equipo.

7. Especificación de pruebas unitarias

En este apartado se definen las pruebas unitarias implementadas hasta el momento para el sistema, se incluyen las entradas esperadas y atípicas de las unidades probadas, así como los correspondientes resultados esperados para cada una de ellas.

7.1. Pruebas de clasificación

Unidad probada	**Entradas esperadas**	**Entradas atípicas**	**Resultados esperados**
clasificar *test-clasificar*	-Subconjunto usado para entrenamiento -Colección de sujetos identificables -Imagen de sujeto desconocido en escala de grises	-Imagen de sujeto a color -Imagen de sujeto con tamaño diferente a colección -Subconjunto diferente al usado para entrenamiento	-Clase del sujeto más similar al buscado.

Cuadro 7: Pruebas de clasificación.

7.2. Pruebas sobre colección

Unidad probada	Entradas esperadas	Entradas atípicas	Resultados esperados
obtener subconjunto *test-obt-subconjunto*	-Número de sujetos -Número de imágenes por sujeto	-Número de sujetos mayor al disponible -Número de imágenes mayor al disponible	-El subconjunto con la cantidad de imágenes solicitadas

Cuadro 8: Pruebas sobre colección.

7.3. Pruebas de entrenamiento

Unidad probada	Entradas esperadas	Entradas atípicas	Resultados esperados
promedio muestras *test-obt-promedio-muestras*	-Matriz de muestras	-Matriz de muestras incompleta	-Al restar el promedio a la matriz, la sumatoria de todas las columnas debe ser 0
matriz covarianza *test-mat-covarianza*	-Matriz de muestras	-Matriz de muestras incompleta	-Una matriz de covarianza igual a la calculada mediante otros sistemas.
autovectores *test-autovects*	-Matriz de covarianza	-Matriz de covarianza errónea o incompleta	-Los autovectores/ valores deben coincidir con los resultados tomados con la herramienta http://www.arndt-ruenner.de/mathe/scripts/engl-eigenwert2.htm
autoespacio *test-autoespacio*	-Matriz de muestras y los autovectores	-Matriz de muestras errónea -Autovectores mal calculados	-El autoespacio conformado por los autovectores debe coincidir con el obtenido manualmente

Cuadro 9: Pruebas de entrenamiento.

7.4. Pruebas de evaluación

Unidad probada	Entradas esperadas	Entradas atípicas	Resultados esperados
evaluacion *test-evaluacion*	-Entrenamiento previo del sistema -Directorio para salvar los informes generados	-Entrenamiento incorrecto o incompleto -Directorio inválido o sin permisos de escritura	-Dos archivos .csv con los resultados de la evaluación de precisión y recall

Cuadro 10: Pruebas de evaluación.