Introduccion: ¿Qué son los logs de testing?

Vamos a empezar diciendo que son y cuál es su proposito

- Son artefactos que contienen información de una ejecución de un test. Proveen un resumen detallado del test indicando si ha sido exitoso o no.
  Nota: ¿Que es un test? Un test es una ejecución de un programa que prueba las funcionalidades de otro programa
- Contiene información de los distintos procesos ejecutados y fallos que haya podido encontrar
- Detallan información como: cuando se ha ejecutado el test, los distintos eventos ocurridos, anomalías y errores encontrados, etc.

Introduccion: ¿Para qué sirven los logs de testing?

- Comprueban que las funcionalidades modificadas se mantienen tras cambios en el código
- Informan de los test exitosos y los que no, proporcionando información específica de los errores encontrados en cada test individual

Introduccion: ¿Por qué clasificarlos automáticamente?

- Encontrar un error puede llegar a ser difícil cuando el log es muy largo
- Mayor contexto, la clasificación automática toma todo lo aprendido y lo aplica a cada log, una persona no puede tener en cuenta todo.
- El coste efectivo de manejar un gran numero de logs, un clasificador automático puede manejar miles de logs mucho más rápido que una persona

Objetivos

Objetivo principal: Desarrollar un sistema que sea capaz de crear, entrenar y probar un modelo predictivo que clasifique logs de testing en categorías de errores.

Otros objetivos

- Capaz de diferenciar si un log contiene una ejecución exitosa o fallida
- Cada lenguaje de programación nombra errores similares con distinto nombre y se detallan de distinto modo, es conveniente que el modelo sea independiente del lenguaje de programación de los logs
- Conocer la librería elegida para desarrollar el proyecto, Weka.

Objetivos del dataset:

Para entrenar el modelo es necesario un dataset, un conjunto de datos, de logs, que se adecue al problema

- Dataset grande con muchos ejemplos para asegurar la fiabilidad de los resultados
- Dataset equilibrado. Buscamos un dataset que contenga distintos errores, pero también buscamos que estos errores estén equilibrados y haya un mínimo por cada tipo
- Lenguajes variados. Como buscamos la independencia del lenguaje de los logs queremos que haya varios lenguajes con varios tipos de errores.

Dataset

Búsqueda del dataset

El dataset seleccionado cumplía la mayoría de nuestros objetivos marcado. Era grande y contenía un cierto numero de tipos de errores adecuado. También tenía un mínimo de logs por cada tipo de error. Pero todos los logs eran de ejecución de programas escritos en Java. El objetivo de independencia del lenguaje no se cumple al tener solo un lenguaje con el que probar.

El dataset también requiere de ciertas cambios y modificaciones:

- El dataset seleccionado contenía logs de muchos repositorios, pero no todos los repositorios estaban clasificados. Algunos repositorios contenían logs sin clasificar y clasificarlos manualmente requeriría mucho tiempo, por lo que se descartaron.
- Correcciones en las clasificaciones de errores. Por ejemplo, en un mismo repositorio había dos categorías con el mismo nombre, se decidió cambiar la segunda y añadirle un ‘2’
- Finalmente, el cambio mas importante, anteriormente se vio que los logs estaban divididos por repositorio, pero esto no nos sirve. Tenemos la información de que error hay en cada categoría. Necesitamos por tanto reordenar los logs en base a los errores.
  Este proceso se ha automatizado y es una parte fundamental.

  Además, como se ve el numero de errores de cada categoría no esta equilibrado, para ello limitamos el numero de logs en cada categoría a 70. Por desgracia la categoría ‘lint’ se queda por debajo de ese límite, pero reducirlo mucho mas para incluir la clase podría poner en duda los resultados

Funcionamiento del sistema

Esquema

El sistema se divide en 3 etapas: preprocesado, entrenamiento y evaluación

Preprocesado

Una vez proporcionados los logs al sistema debemos transformarlos a un formato que la librería permita. Los textos contienen palabras clave que indican más o menos información sobre la clase a la que pertenece el log.

- El primer objetivo es transformar esas palabras en valores numéricos con los que trabajar. Esto se lleva acabo con un tokenizador, un artefacto que separa las palabras o grupos de palabras en tokens. Después utilizando el filtro StringToWordVector asigna valores numéricos a cada token en función de su frecuencia de aparición en cada log.
- El segundo objetivo es evaluar que tokens nos son útiles y cuales no aportan valor. Para ello se utiliza el filtro AttributeSelection que utilizando un ranking ordena de mas a menos valor los atributo y elimina aquellos que aporte poca o ninguna información acerca de la clase a la que pertenecen
- Después hacemos un filtrado rápido de aquellos atributos que no cumplan un regex indicado. Este regex limita a tributos que contengan solo caracteres ascii, de este modo eliminamos atributos que no nos proporcionan información importante.
- Por último, debemos dividir todo para crear 2 subconjuntos un de entrenamiento y otro de pruebas. Se concede 2/3 del total a entrenamiento y 1/3 para pruebas.

Entrenamiento

La fase de entrenamiento se ocupa de crear el modelo y entrenarlo con el subconjunto de entrenamiento previo

- Se empieza creando un clasificador base, en este caso se utiliza un clasificador que hace uso del algoritmo “vecino más próximo”. Este algoritmo no genera un modelo muy preciso, pero si lo suficiente para trabajar con él, en cambio es rápido que es lo que buscamos.
- A continuación empezamos con la técnica boostin baggign stacking. Se trata de una técnica que combina una serie de algoritmos que, aunque no potentes por si solos, generan un modelo fuerte al combinarlos. La primera parte, boosting, consiste en revisar las instancias en las que el clasificador base se equivocó. Utilizando el clasificador anterior, se aprende de los fallos anterior y se itera un numero de veces para mejorarlo. Ayuda a reducir la varianza general del modelo
- La segunda técnica, bagging, es un método que crea muestra separadas de todo el conjunto y crea un clasificador para cada parte. Los resultados son después combinados.

- La ultima técnica, stacking o blending, incrementa la fuerza predictiva del modelo general. Esta técnica utiliza distintos algoritmos como por ejemplo NaiveBayes, RandomTree o J48, para aprender de los puntos fuertes de cada uno.
- Finalmente, todo lo aprendido se vota utilizando un meta algoritmo llamado Voting. Que vota por peso todo para generar un modelo robusto.

Evaluación: medidas de evaluación

Weka nos proporciona distintas medidas de evaluación

- La matriz de confusión es un concepto muy sencillo para ver el desempeño de nuestro modelo. Se trata de una matriz NxN siendo N el numero de clases o categorías. Sirve para mostrar cuando una clase es confundida por otra.
  Por ejemplo, si nuestro modelo intenta predecir una instancia de clase A y predice que es de A es que ha predecido correctamente. De igual modo con el resto de las clases.

- Weka también nos proporciona una serie de medidas calculadas en base a los distintos ratios positivas y negativos sacados de la matriz de confusión. Entre ellos el que mas nos interesa es el área bajo la curva ROC o AUC.
  Es una curva que representa la sensibilidad frente a la especificidad en un sistema. La mejor predicción se sitúa en la esquina superior izquierda (0,1) mientras que la diagonal central representa una clasificación totalmente aleatoria y en la esquina inferior derecha representa fallo total.
- Por último, he de mencionar que se proporcionan resultados simplificados para entender, sin mucho conocimiento, como de bien han id las predicciones

Evaluación: resultados

Para evaluar este modelo hemos utilizado el subconjunto anteriormente dividido y weka ha realizado predicciones a él, simulando un entorno real.

- Empezando por la matrix de confusión. Podemos ver como la mayoría de las predicciones se encuentran en la diagonal, marcadas en verde, indicando que han sido correctas. Mientras que los pocos fallos en rojo. Ya tan solo con esta vista se puede ver que el modelo predice bien por lo general, pero podría darse algún caso, alguna clase, que tenga peores resultados. Se destacan tres clases, lint, androidsdk y license. Lint seria una clase a eliminar o buscar mas logs puesto que solo tiene 1 caso. Androidsdk indica un error en el dataset pero que no afecta a los resultados. Por ultimo la clase license tiene un numero por debajo de lo normal y serie prudente equilibrarlo
- Al calcularlos los resultados avanzados podemos ver mucha más información de cada clase. Nos interesa ver la columna “ROC área” es de nuestro mayor interés. Como sea ha visto antes tener curvas que se aproximen a 1 es lo ideal y en nuestro caso prácticamente todas estan por encima del 0,99

- Por último, ver los resultados simplificados, que nos confirman que mas de un 97% de los logs de pruebas han sido clasificados correctamente