Práctica 1 IAA

Daniel Ranchal Parrado, Francisco Vera Herencia

19 de febrero de 2019

32 horas de trabajo

Índice general

Índice de figuras

Capítulo 1

Primer ejercicio

1.1. ¿Cuántos atributos caracterizan los datos de esta base de datos?

Los datos se caracterizan por 5 atributos, como se puede apreciar en la figura 1.1.

No.		Name
1	☐	sepallength
2	☐	sepalwidth
3	☐	petallength
4	☐	petalwidth
5	☐	class

Figura 1.1: atributos

1.2. ¿Se trata de regresión o clasificación?

Se trata de una clasificación, ya que a partir de unos atributos se puede decir que una instancia pertenece a una clase u a otra según los valores de estos atributos.

1.3. ¿Cuál es el rango de valores del atributo petalwidth?¿Y su media? ¿y su desviación típica?

Respecto al rango de valores del atributo petalwidth (anchura del pétalo), vendrá dado por los valores mínimo y máximo registrado, por lo que, según la figura

1.2 este rango será [0'1,2'5]. En la misma imagen también se indica la media y su desviación típica, cuyos valores son 1'199 y 0'763, respectivamente.

Statistic	Value
Minimum	0.1
Maximum	2.5
Mean	1.199
StdDev	0.763

Figura 1.2: Rango, media y desviación típica de los datos del atributo petalwidth

1.4. Determinar que atributo permite discriminar linealmente entre la clase iris-setosa y las otras dos clases

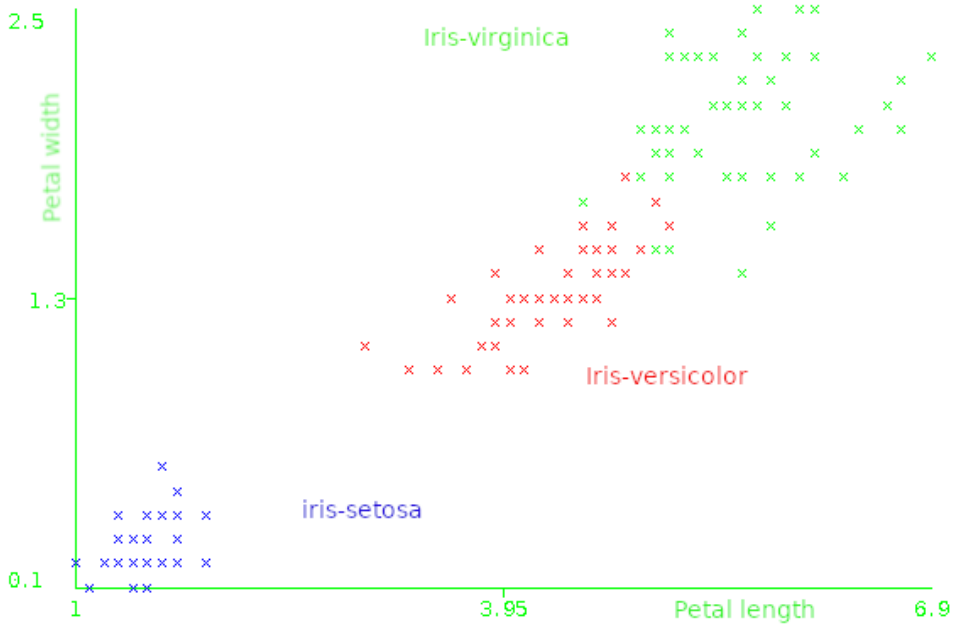


Figura 1.3: Gráfico en el que se diferencian las diferentes instancias según el ancho y la longitud del pétalo

Como se puede observer en la figura 1.3, se puede diferenciar con bastante exactitud la clase iris-setosa y el resto de la clases. Para poder diferenciarlas de esta manera se puede utilizar los atributos petallenght o petalwidth.

1.5. ¿Es posible separar linealmente la clase iris-versicolor de la clase iris-virginica?

Como se puede ver en la figura 1.3, la clases iris-versicolo y la clase iris-virginica se pueden separar linealmente utilizando los atributos petallenght y petalwidth.

1.6. ¿Con qué dos atributos te quedarías para discriminar entre las tres clases del problema?

Las tres clases se pueden discriminar utilizando los atributos petallenght y petalwidth tal y como se puede observar en la figura 1.3.

1.7. ¿Que diferencia hay entre instancias Distinct y Unique?

Distinct determina el número de diferentes valores que se pueden dar en un atributo, mientras que Unique determina el número de instancias que tienen atributos con valores únicos que otras instancias no tienen.

Capítulo 2

Segundo ejercicio

2.1. Aplique el filtro filters/unsupervised/attribute/NominalToBinary y describa como quedan ahora los atributos.

No.		Name
1		age_gt_60
2		air
3		airBoneGap
4		ar_c
5		ar_u
6		bone
7		boneAbnormal
8		bser
9		history_buzzing
10		history_dizziness
11		class

Figura 2.1: Atributos antes de aplicar el filtro

Después de aplicar el filtro NominalToBinary, habrá los siguiente atributos:

No.	Name
1	age_gt_60=f
2	age_gt_60=t
3	air=mild
4	air=moderate
5	air=normal
6	air=profound
7	air=severe
8	airBoneGap=f
9	airBoneGap=t
10	ar_c=absent
11	ar_c=elevated
12	ar_c=normal
13	ar_u=absent
14	ar_u=elevated
15	ar_u=normal
16	bone=mild
17	bone=moderate
18	bone=normal
19	bone=unmeasured
20	boneAbnormal=f
21	boneAbnormal=t
22	bser=degraded
23	bser=normal
24	history_buzzing=f
25	history_buzzing=t
26	history_dizziness=f
27	history_dizziness=t
28	class

Figura 2.2: Atributos después de aplicar el filtro

2.2. ¿Podría saber con antelación el número de atributos finales al aplicar este filtro?

Cada atributo tiene una serie de etiquetas. Como estos atributos son de tipo nominal, cada etiqueta, al aplicar el filtro, se convertirán en un atributo. Por lo que si el primer atributo tiene dos etiquetas, esta se convertirá en dos. El segundo atributo se convertirá en 5 atributos, mientras que el tercer atributo se convertirá en dos atributos, y así en el resto de atributos. Al final habrá un número de 28 atributos después de aplicar el filtro, como se puede apreciar en la figura 2.2

Capítulo 3

Tercer ejercicio

3.1. Particione su base de datos Criaturas tenebrosas usando filters/supervised/instance/StratifiedRemoveFolds

reobiegrfjbirepgbebgjjkbsfddddsjbkgfjbgffgbj

Capítulo 4

Cuarto ejercicio

4.1. Construya a partir del fichero dataset371.csv un fichero .arff para Weka. Ponga nombres de atributos descriptivos y use las herramientas que considere necesarias

Para poder construir un fichero .arff a partir de un .csv, primero hay que cargar este último en weka. Para ello, le daremos a la pestaña de open y buscaremos los archivos de esta extensión y escogeremos el archivo dataset371.csv, como se puede observar en la figura 4.1.

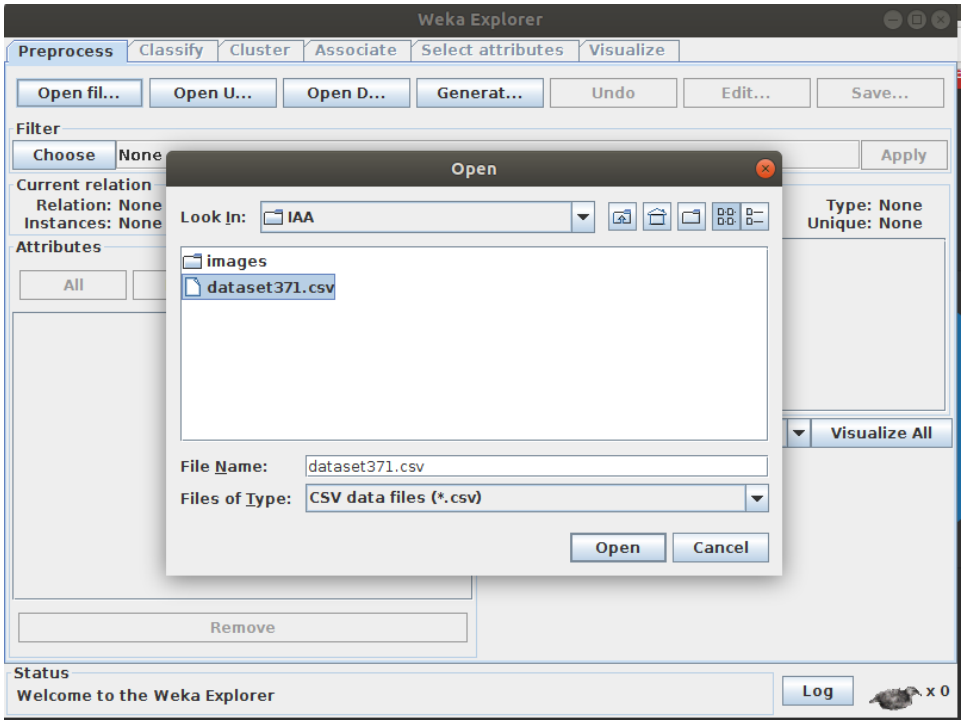


Figura 4.1: Abriendo el archivo dataset371.csv

Una vez que tenemos cargado el fichero .csv en weka, solo hace falta guardarlo con formato .arrf. Para ello solo hace falta irse a la pestaña Save, donde guadaremos los datos con la extensión .arff. Este proceso se puede ver en la figura 4.2.

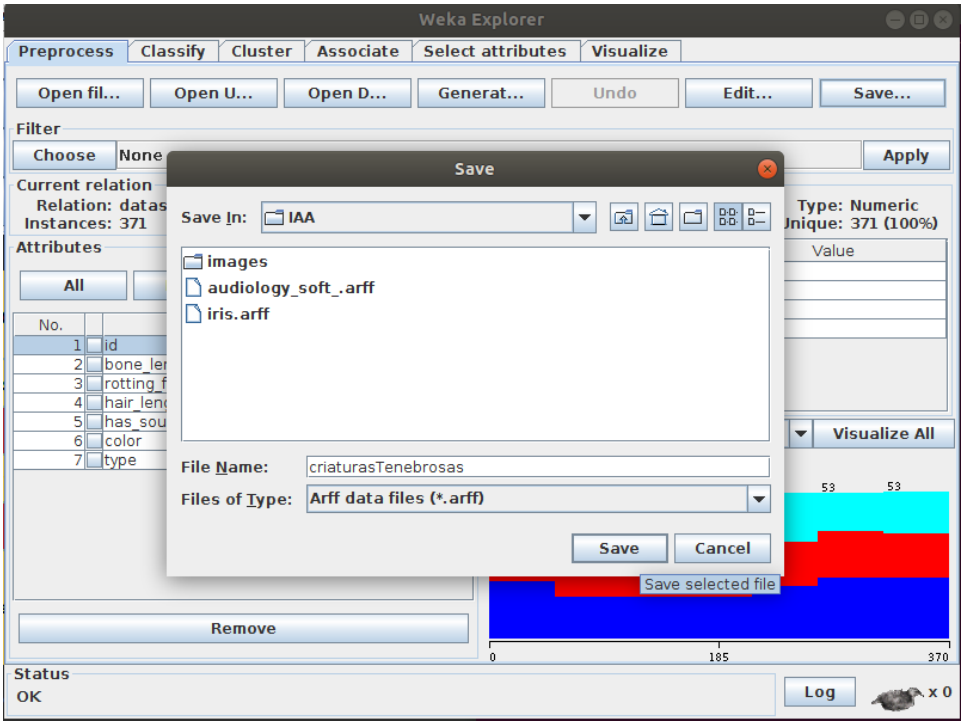
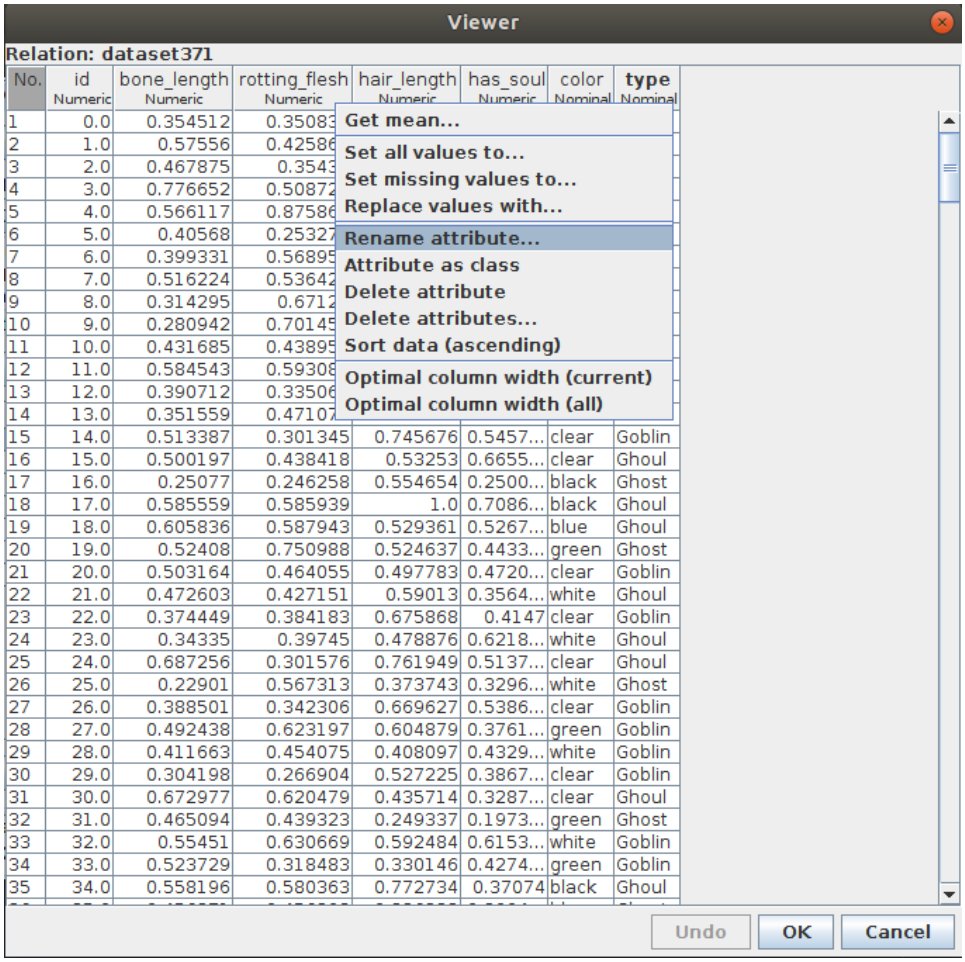


Figura 4.2: Guardando el archivo en formato .arff

Para poder editar el nombre de los distintos atributos, solamente hay que irse a la pestaña Edit, tal y como se puede ver en la figura 4.3. En este caso no se va a modificar ningún nombre de los atributos ya que cada uno de ellos expresa de una manera concisa su significado.



No.	id Numeric	bone_length Numeric	rotting_flesh Numeric	hair_length Numeric	has_soul Numeric	color Nominal	type Nominal
1	0.0	0.354512	0.35083				
2	1.0	0.57556	0.42586				
3	2.0	0.467875	0.3543				
4	3.0	0.776652	0.50872				
5	4.0	0.566117	0.87586				
6	5.0	0.40568	0.25327				
7	6.0	0.399331	0.56895				
8	7.0	0.516224	0.53642				
9	8.0	0.314295	0.6712				
10	9.0	0.280942	0.70145				
11	10.0	0.431685	0.43895				
12	11.0	0.584543	0.59308				
13	12.0	0.390712	0.33506				
14	13.0	0.351559	0.47107				
15	14.0	0.513387	0.301345	0.745676	0.5457...	clear	Goblin
16	15.0	0.500197	0.438418	0.53253	0.6655...	clear	Ghoul
17	16.0	0.25077	0.246258	0.554654	0.2500...	black	Ghost
18	17.0	0.585559	0.585939	1.0	0.7086...	black	Ghoul
19	18.0	0.605836	0.587943	0.529361	0.5267...	blue	Ghoul
20	19.0	0.52408	0.750988	0.524637	0.4433...	green	Ghost
21	20.0	0.503164	0.464055	0.497783	0.4720...	clear	Goblin
22	21.0	0.472603	0.427151	0.59013	0.3564...	white	Ghoul
23	22.0	0.374449	0.384183	0.675868	0.4147	clear	Goblin
24	23.0	0.34335	0.39745	0.478876	0.6218...	white	Ghoul
25	24.0	0.687256	0.301576	0.761949	0.5137...	clear	Ghoul
26	25.0	0.22901	0.567313	0.373743	0.3296...	white	Ghost
27	26.0	0.388501	0.342306	0.669627	0.5386...	clear	Goblin
28	27.0	0.492438	0.623197	0.604879	0.3761...	green	Goblin
29	28.0	0.411663	0.454075	0.408097	0.4329...	white	Goblin
30	29.0	0.304198	0.266904	0.527225	0.3867...	clear	Goblin
31	30.0	0.672977	0.620479	0.435714	0.3287...	clear	Ghoul
32	31.0	0.465094	0.439323	0.249337	0.1973...	green	Ghost
33	32.0	0.55451	0.630669	0.592484	0.6153...	white	Goblin
34	33.0	0.523729	0.318483	0.330146	0.4274...	green	Goblin
35	34.0	0.558196	0.580363	0.772734	0.37074	black	Ghoul

Figura 4.3: Cambiando el nombre de un atributo

4.2. Describa de forma concienzuda tanto los atributos como las clases de la base de datos.

No.	Name
1	id Numeric
2	bone_length Numeric
3	rotting_flesh Numeric
4	hair_length Numeric
5	has_soul Numeric
6	color Nominal
7	type Nominal

Figura 4.4: Atributos y su tipo de la base de datos dataset371

Como se puede ver en la figura 4.4, se distiguen 7 atributos, de los cuales 5 de ellos son numéricos y el resto son nominales. El atributo id no tiene en esta base de datos no tiene ningún significado, ya que solo enumera cada una de las instancias que hay. El segundo atributo cuyo nombre es bone_lenght y el cuál es númerico, determina la longitud del hueso de la criatura, la cuál está normalizada entre 0 y 1. El tercer atributo, de tipo numérico, que está denominado como rotting_flesh representa el porcentaje de este en un rango de 0 a 1. El atributo hair_lenght, de tipo numérico, representa la longitud del pelo, cuyos datos están normalizados entre 0 y 1. El atributo has_soul es de tipo numérico y representa un porcentaje que está normalizado entre 0 y 1. El atribito color, como su propio nombre indica, determina el color de la criatura. Es un atributo nominal y puede tomar los valores whhite, black, clear, blue, green y blood. El último atributo, llamado type, determina el tipo de la criatura, es decir, nos indica a que clase pertenece. Es de tipo nominal y puede tomar los valores Ghost, Goblin y Ghoul.

Hemos hecho uso de tres cargas distintas de ab. Cada carga se ha realizado con una concurrencia y una cantidad de peticiones distintas, además de que cada una de estas se ha realizado en cinco ocasiones en tres index: uno con fotos grandes, otro con fotos medianas y otro con fotos pequeñas, para comprobar si existen diferencias entre las tres versiones. Las cargas son:

- **ab -c 5 -n 20**: Pretende medir la capacidad del programa a la hora de tener una concurrencia media y una cantidad de peticiones moderada/alta.

- **ab -c 1 -n 10**: Pretende probar el servidor con una cantidad de peticiones más baja que la anterior prueba, pero sin concurrencia, para comprobar la diferencia entre una prueba con concurrencia o sin ella.

- **ab -c 10 -t 40 -s 40**: Pretende probar el servidor con una concurrencia alta en un tiempo, dándonos, por ejemplo, el número de peticiones que puede realizar el servidor en x segundos.

4.3. Diseño de Experimentos

Para realizar estos experimentos he creado un script que realiza en cinco ocasiones las pruebas anteriormente nombradas para cada tamaño de imagen. A su vez, el monitor sar está corriendo en el servidor y guardándolo en un fichero. Para mostrar gráficamente los resultados de estas pruebas se hace uso de gnuplot.

4.4. Análisis de los resultados

Tras realizar ab y a los tres index con las distintas cargas podemos proceder a mostrar sus datos. Los primeros datos que nos resultarían interesantes serían los del propio Apache benchmark. Los primeros datos que podemos mirar son los que nos da el propio apache benchmark. Estos datos, obtenidos de la propia salida del comando "ab" son los siguientes. De la primera prueba se extrae lo siguiente:

	Pequeños			Medianos			Grandes		
Ejecución	Pet/s	ms/Petición	Total(s)	Pet/s	ms/Petición	Total(s)	Pet/s	ms/Petición	Tot
1	0.17	28875.345	115.501	0.16	31396.260	125.585	0.18	27800.182	111
2	0.17	29610.067	118.440	0.17	30268.598	121.074	0.18	27508.504	110
3	0.17	29360.455	117.442	0.17	30065.704	120.263	0.18	27296.960	109
4	0.17	29445.476	117.782	0.16	32097.834	128.391	0.18	27261.818	109
5	0.16	28917.908	115.672	0.17	29742.525	118.970	0.18	26643.844	106
Media	0.168	29241.8502	116.9674	0.166	30714.1842	122.8566	0.188	27302.2616	109.

Es curioso el hecho de que la variación del tiempo no dependa tanto del tamaño como de la capacidad de la red en ese momento. Se puede ver que aunque en los valores medianos sube el tiempo, tanto por petición como el total, con respecto a los valores pequeños; en los grandes esta sube debido a que la red no está siendo utilizada.

Tras la segunda prueba nos da los siguientes resultados:

	Pequeños			Medianos			Grandes		
Ejecución	Pet/s	ms/Petición	Total(s)	Pet/s	ms/Petición	Total(s)	Pet/s	ms/Petición	Total(s)
1	0.17	5897.409	58.974	0.16	6207.914	62.079	0.17	5732.952	57.330
2	0.17	5952.298	59.523	0.16	6166.424	61.664	0.17	5792.583	57.926
3	0.17	5968.026	59.680	0.16	6159.262	61.593	0.17	5720.497	57.205
4	0.16	6904.65	60.946	0.16	6291.626	62.916	0.17	5761.08	57.611
5	0.17	6033.483	60.335	0.16	6239.134	62.391	0.17	5817.991	58.180
Media	0.168	5989.1642	59.8916	0.16	6212.8708	62.1286	0.17	5765.0206	57.6504

Se puede comprobar que el tiempo de respuesta en este caso, sin ninguna concurrencia, es sustancialmente menor al tiempo con concurrencia (de media, unas 4.853981 veces menor), debido a que con más de dos peticiones a la vez el servidor se satura debido a su baja capacidad de cómputo.

Por último, en el último test, siendo los dos únicos datos que nos interesan la media de peticiones que son capaces de resolver en ese tiempo y el tiempo de respuesta por petición.

	Pequeños		Medianos		Grandes	
Ejecución	Pet. hechas	ms/Petición	Pet. hechas	ms/Petición	Pet. hechas	ms/Petición
1	4	88795.308	4	110129.467	2	218473.645
2	3	107076.853	3	130129.422	3	157925.583
3	4	86692.71	4	108129.467	4	110129.562
4	5	66522.661	2	160291.622	3	142129.465
5	3	108506.811	4	110129.467	4	100345.234
Media	3.8	75914.1808	3.4	123761.889	3.2	145800.6978

Aqui si se puede notar una clara diferencia entre la performance entre la versión pequeña, la mediana y la grande, así como la del tiempo de respuesta entre las distintas ejecuciones. La principal diferencia de esta prueba con respecto a las otras dos es que, además de ser de tiempo, también tiene una concurrencia alta, sobre todo teniendo en cuenta, como veremos posteriormente, que la cpu a partir de dos concurrencias llega casi al 100 % de utilización.

A continuación procedemos a sacar con sar -u los datos de la CPU que se han sacado durante las distintas ejecuciones de ab. Con gnuplot los ponemos de manera visual, siendo la abscisa Y el porcentaje de uso de cpu y la abscisa X el tiempo.

Lo primero que se puede notar es que para una concurrencia mayor que uno, el servidor utiliza la cpu casi al 100 %, siendo el único momento donde este uso baja, cuando llega el momento de la prueba 2, donde no hay concurrencia y solo utiliza un 80 % del disco.

En cuanto a las pruebas de red, he utilizado sar -n DEV para comprobar todos los paquetes que se han enviado. Además de esto, he usado sar -n EDEV para comprobar aquellos paquetes que no han sido recibidos, dando como resultado que en ninguna de las pruebas ha habido paquetes fallidos. Para mostrar los resultados, he vuelto a hacer uso de gnuplot, siendo la abscisa Y los kbs, tanto transmitidos como recibidos, y la abscisa X el tiempo.

Se puede notar como el ritmo de transferencia da muchas subidas y bajadas. Aun así los datos son más o menos estables, siendo el punto en el que es más baja la transferencia durante el tercer test, mientras que el punto en el que alcanza sus picos es en el segundo test.

Por último, haciendo uso del monitor de azure y sacando media de los datos durante las ejecuciones de los benchmark, podemos sacar los datos de utilización de disco.

Para mostrar estos datos, se ha hecho uso del excel, mostrando en ese caso la media por ejecución.

Se pueden ver como las transferencias son muy parecidas, tanto entre benchmark como entre los distintos index.

Capítulo 5

Conclusiones y discusión

En primer lugar, se aprecia un fuerte problema en el hardware de este servidor debido al cuello de botella que aparece al ejecutar los bencharmarks en la CPU, esto es fácil de comprender, debido a que la concurrencia toma protagonismo en estos tests, con algunos fijándola en valores como 10 peticiones concurrentes. Esta teoría es afirmada por las gráficas mostradas, donde vemos fácilmente como el procesador llega a funcionar a niveles cercanos al 100 % de utilidad durante la duración de las pruebas efectuadas.

Además de esto, otro factor determinante en este caso es el bajo número de transferencias por segundo que el servidor es capaz de aguantar y que está fuertemente relacionado con el almacenamiento del mismo en el cual se almacenan las fotografías que sirven como cargas de prueba, el cual en ninguna prueba es capaz de llegar ni tan siquiera a la unidad, y eso es algo que los encargados de testear su funcionamiento hemos conocido de primera mano al tener que esperar grandes cantidades de tiempo a que todos los tests fueran completados varias veces con éxito. En cuanto a las peticiones por segundo, casi nunca superan el cuarto de unidad, hecho que no hace más que corroborar las afirmaciones anteriores.

Si contemplamos el gráfico pipo, nos damos cuenta de que el número de peticiones por segundo, al ser un parámetro íntimamente ligado al hardware sobre el que se esté ejecutando el banco de pruebas escogido, apenas varía, puesto que la CPU solo es capaz de ejecutar un número determinado de ellas, siendo todas iguales. Lo interesante de este gráfico, es ver como el tiempo empleado en servir un grupo de peticiones concurrentes decae a lo largo de las cinco ejecuciones del test. Tras indagar en los motivos de este aumento del rendimiento, nos damos

cuenta de que Azure es un servicio de computación en la nube que alardea de proveer de una caché inteligente a sus máquinas virtuales que estén corriendo en sus servidores, como es nuestro caso, eso por ello por lo que se aprecia esta mejora en el rendimiento, debido a que gran parte de los datos que son necesarios para la ejecución de la prueba, ya están cargados en la caché del servidor y son accesibles de manera mucho más rápida para la siguiente ejecución.

diferencia entre ellas en cuanto al parámetro de medición, el número de transferencias por segundo del disco del servidor. Para analizar las gráficas, recordemos que estos datos han sido tomados con el monitor sar, mientras que ab era ejecutado, tras ello, se ha realizado la media aritmética de los valores del parámetro interesado que sar tomaba cada 10 segundos durante la ejecución del test. Entonces, podemos ver como apenas hay diferencias significativas normalmente el test de menor concurrencia y menos repeticiones (línea amarilla), tarda normalmente menos tiempo que el test con más concurrencia y el doble de repeticiones (línea verde), aunque veamos ejecuciones donde estas posiciones se inviertan, esto se debe más al margen de resultados del mosaico de imágenes pequeñas, observamos como la línea azul, la correspondiente con el uso de https, lo cual es curioso debido a que en las dos últimas pruebas se habían mantenido por debajo, sin embargo, no es de preocupación, ya que no se aprecian diferencias muy significativas en la mayoría de las ejecuciones, sin embargo, vemos como esa línea sube hasta una transferencia por segundo en la segunda ejecución, lo cual es muy significativo. La imagen general de estas pruebas es que mientras más grandes sean las imágenes del banco de pruebas, menos transferencias por segundo se llevarán a cabo, lo cual es lógico, puesto que son archivos más grandes.
Si analizamos ahora el uso de la red del servidor, aquí no tenemos ningún problema de cuello de botella, Azure nos proporciona una velocidad de conexión veloz y estable y no es algo que un administrador de un servidor pequeño como este tenga que preocuparse, por supuesto, mientras más exigente sea el test con la CPU, lo será con el uso de red que éste requerirá, puesto que estamos hablando de un servidor web.

En conclusión, hemos detectado que nuestro cuello de botella es una CPU que a pesar de ser muy potente en el conjunto de todos sus núcleos, apenas aguantaría un servidor web pequeño como este ante un importante número de peticiones al servidor, esto es debido a que la cuenta de estudiante de Microsoft Azure que

estamos empleando para este trabajo solo ofrece un núcleo de procesador, potente, si, pero que flaquea en tareas que involucren una alta concurrencia de peticiones. Y es que ese es el argumento por el cual los procesadores empleados para su uso en servidores de cualquier tipo suelen contar con un gran número de núcleos tanto físicos como lógicos y poder procesar rápidamente todas las peticiones que reciba. En cuanto a la referenciación, sabemos que los resultados aquí mostrados corresponden con los de un servidor de especificaciones limitadas, útil para este propósito, pero de limitado uso, y es que, tras haber ejecutado numerosos bancos de pruebas centrados en multitud de aspectos de un sistema informático, entre ellos Apache Benchmark, hemos encontrado que fácilmente, un ordenador doméstico con todos sus núcleos activados puede sobrepasar fácilmente el rendimiento de este procesador, organizaciones como SPEC tienen precisamente esa función, la de examinar este tipo de equipos informáticos para poder realizar comparativas, lo cual nos es muy útil a la hora de sacar conclusiones sobre el desempeño de este hardware. Por último, hemos de decir, que Microsoft Azure es una útil herramienta para alojar cualquier tipo de servidor o aplicación web y que nos ha sido muy útil para la realización de este trabajo y que, previo pago de la capacidad necesaria, se puede obtener muchísima potencia de cómputo de ella.

5.1. Cuestiones

- **¿Qué factores pueden hacer variar el tiempo de respuesta a la hora de realizar una petición? ¿Como puede el servidor mejorar este?**
  El tiempo de respuesta puede variar por factores, tanto del lado del cliente como del del servidor. Del lado del cliente puede variar por la lejanía al centro del servidor o el estado de la conexión a internet. Del lado del servidor, la cantidad de llamadas en un instante y el hardware son las causas de esta variación. El servidor puede mejorar el tiempo de respuesta variando el hardware, viendo cual es el cuello de botella y mejorando sus componentes, de tal manera que pueda encontrar un punto en el que sea capaz de encontrar una capacidad de cómputo suficiente para sus peticiones.

- **¿Por qué repetimos varias veces el test de ab y con varias configuraciones?**
  Al ejecutar las distintas configuraciones de ab varias veces estamos logrando dos cosas: la primera, bajo una misma configuración de prueba tomamos varias medidas en cada una de las cinco ejecuciones del test y aplicamos la

media artimética sobre cada una de ellas con el fin de obtener la máxima fidelidad de los datos a una hipotética situación real de trabajo del servidor. El segundo logro corresponde con el comprobamiento del funcionamiento del servidor bajo diferentes condiciones de trabajo, por ejemplo variando la concurrencia de trabajos que ha de procesar. Combinando estos dos procedimientos, nos hacemos una idea de la potencia que tiene el servidor y su desempeño en una situación real.

- **Teniendo en cuenta todos los datos extraidos del servidor web, ¿Es rentable y de utilidad Tomcat?**
  Respecto a la utilidad, Tomcat restringe o más bien, imposibilita aquellas aplicaciones web que no sean hechas con Java y el lenguaje de marcado html, y por lo tanto, deja atrás lenguajes útiles para este campo de la informática como PHP. Respecto a la correcta configuración de Tomcat, es muy fácil y nos proporciona herramientas ya mencionadas en la memoria. Pero el mayor problema para utilizar Tomcat en un sistema basado en Linux es la instalación de Java Virtual Machine, es bastante complicado de instalar y por ello, a lo mejor, Tomcat no es tán utilizado como nginx, apache2 o httpd. Teniendo en cuenta las características del servidor que sirve Tomcat, es lo suficientemente bueno para soportar la funcionalidad de los archivos jsp.

Bibliografía

[1] Michael Kerrisk. *The Linux man-pages project*.

[2] Aula de Software Libre de la Universidad de Córdoba. *Taller de Docker*, 2018.

[3] Microsoft Azure. *Máquinas virtuales Linux*, 2018.