**Pràctica 3 - Spark**

universitat de barcelona | Semestre de tardor

bases de dades avançades

pau sanchez i guillem rabionet

2018

**1. Neteja del text del Quijote i reconstrucció dels paràgrafs**

El primer que cal fer és netejar el text del Quijote i eliminar els caràcters estranys i símbols del text, així com substituir aquelles vocals accentuades per elles mateixes sense accentuar.

Per dur a terme aquesta tasca s’ha creat un programa en llenguatge Scala. Aquest rep el fitxer del Quijote sense cap mena de tractament i retorna un fitxer amb les següents característiques:

- Els accents han estat eliminats (substituïts per la vocal sense accent)

- Cada línia de text és un paràgraf sencer (s’han reconstruït els paràgrafs)

- S’han filtrat les StopWords (aquelles paraules que no aporten valor)

Nota: Tot i que l’enunciat no ho demanava, s’ha cregut que no tenia sentit comparar paràgrafs donant valor a articles, pronoms febles, determinants, etc. Aquestes s’anomenen stop words i han estat eliminades del text per a poder fer un anàlisi i comparació de paràgrafs més rigorós.

Aquest programa de neteja és l’encarregat de generar l’Output que serà absorbit per DataBricks i amb ell es crearà la taula de dades per a fer l’entrenament del model i la corresponent vectorització dels paràgrafs.

**2. Com executar el programa de neteja**

El programa està fet en Scala i per executar-lo cal tenir Scala instal·lat, si el lector no té instal·lat Scala pot fer-ho molt fàcilment amb la següent instrucció (Linux):

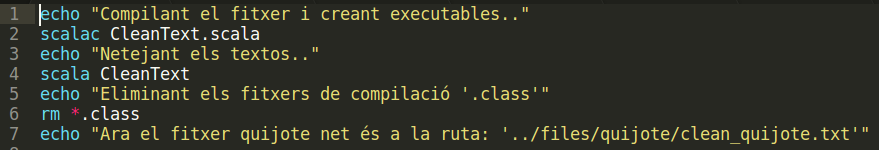
>> sudo apt-get update

>> sudo apt-get install scala

Una vegada instal·lat scala només cal executar l’script.sh que s’ha preparat. Ell mateix fa la compilació, l’execució i després elimina els fitxers ‘.class’ de compilació d’Scala. L’script es troba a la següent ruta relativa al projecte:

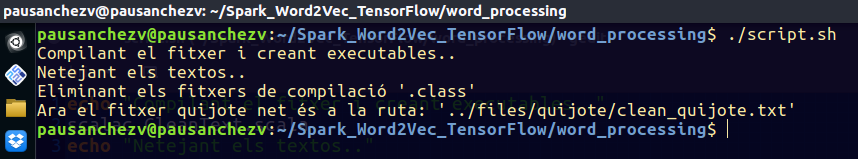
>> /word\_processing/script.sh (Executar des de dins la carpeta, llegeix rutes relatives!)

Vegem doncs aquest script al detall



Com es pot veure el programa que s’ha executat es diu CleanText.scala. Aquest és el resultat:

(Executar des de dins la carpeta ‘word\_processing’, el programa llegeix rutes relatives!)



El programa sencer es pot veure fàcilment accedint al fitxer ‘/word\_processing/CleanText.scala’, amb qualsevol editor com per exemple ‘kate’ o ‘sublime text’. Consta d’una classe de tipus Object d’Scala, que intenta emular un Singleton de qualsevol altre llenguatge orientat a objectes. El seu mètode principal es diu ‘main’ i s’executa de manera automàtica. A partir d’aquesta crida es desencadena la neteja dels textos i posterior generació del fitxer. Ho fa de la següent manera:

- Llegeix el fitxer: ‘../files/quijote/raw\_quijote.txt’

- Fa el processat

- Genera el nou fitxer: ‘../files/quijote/clean\_quijote.txt’

Per executar les tasques anteriors la classe disposa de 4 funcions bàsiques:

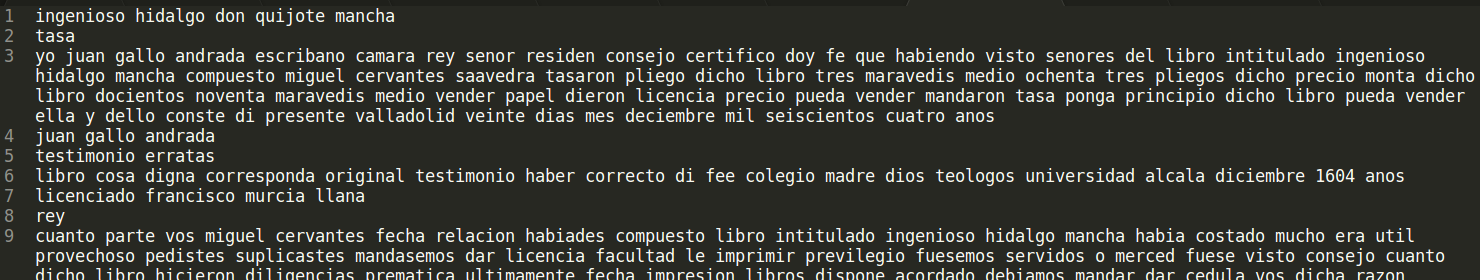
- filterStopWords:: per excloure les paraules buides

- cleanStrings:: per netejar les línies de caràcters estranys i accents

- constructParagraphs:: genera els paràgrafs a partir de les línies de text

- main:: distribueix les tasques repartint-les a les funcions anteriors i genera l’output

Vegem l’aspecte que prenen els textos una vegada processats i havent aplicat el programa de neteja (cada número de línia representa un paràgraf):



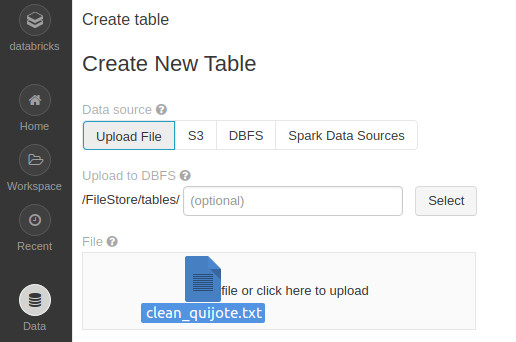
A la pràctica es dóna el fitxer generat, però per comprovar que tot funciona perfectament, el lector pot esborrat el fitxer ’../files/quijote/clean\_quijote.txt’, tornar a executar l’script.sh i comprovar que el fitxer s’ha generat una altra vegada.

*Per dur a terme la pràctica s’ha preferit fer abans la neteja dels textos de manera que a DataBricks entri un fitxer de text que ja estigui preparat, processat i net.*

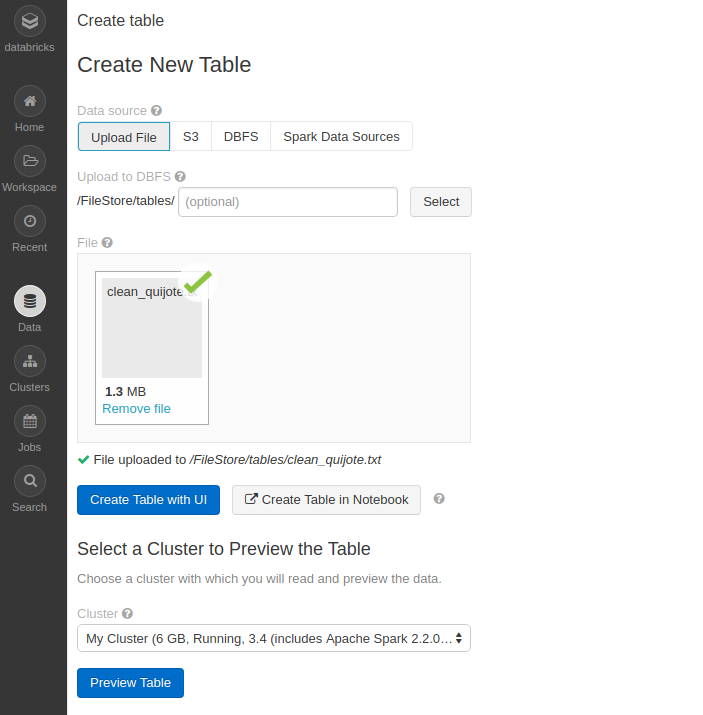
**3. Importació del Quijote a DataBricks**

Una vegada obtingut el Quijote processat, net de símbols i construït per paràgrafs cal importar-lo a DataBricks per construir la taula que permetrà fer la lectura del fitxer des del notebook.

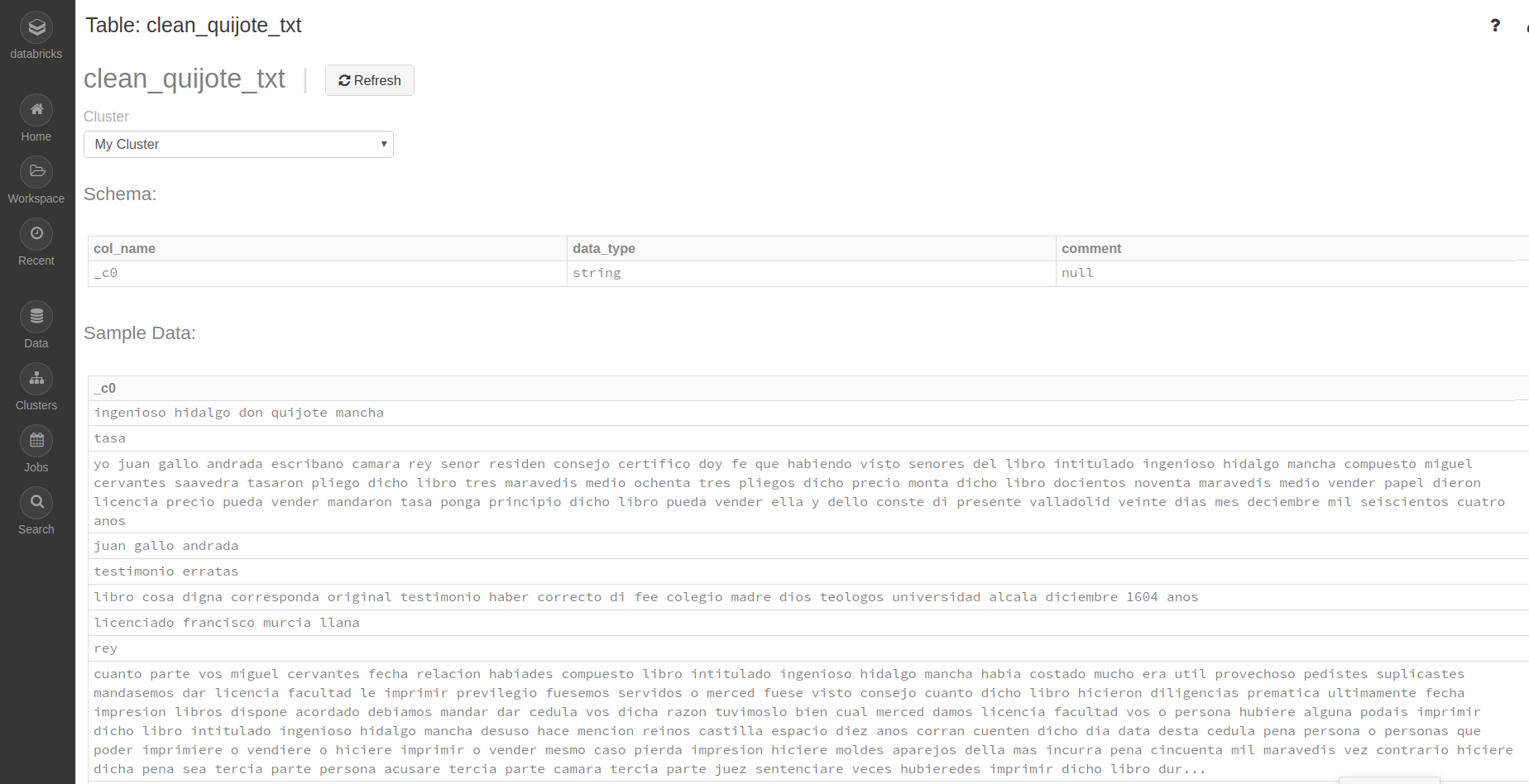
El primer que cal fer és dirigir-se a l’apartat de creació d’una nova taula i arrossegar el fitxer ‘/files/quijote/clean\_quijote.txt’ obtingut al pas anterior:



Una vegada s’ha carregat el fitxer cal prémer el botó de creació de taula mitjançant UI i acte seguit seleccionar el cluster de 6gb que DataBricks ofereix a l’usuari.



Una vegada s’ha creat la taula, aquesta pren el següent aspecte:



Aquest és el pas intermig que cal fer per importar un fitxer del disc local i poder-lo utilitzar a DataBricks, en un format que aquest pugui entendre.

Ara el nostre fitxer ‘clean\_quijote.txt’ es troba també a la ruta relativa al projecte de DataBricks: ‘/FileStore/tables/clean\_quijote.txt’, a més d’estar a la ruta del nostre projecte local a disc. Per tant per llegir el fitxer des del notebook l’única cosa que cal fer és invocar la següent crida:

sc.textFile("/FileStore/tables/clean\_quijote.txt")

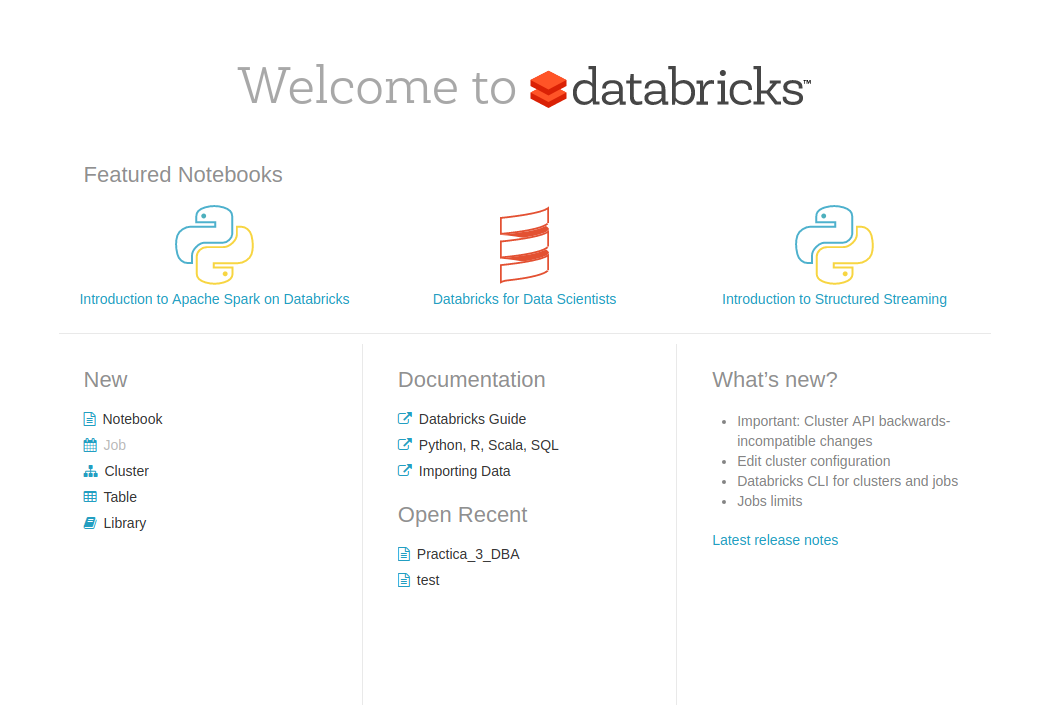
**4. Llenguatge de desenvolupament**

Una vegada més, el llenguatge triat pel desenvolupament de la part de la pràctica que s’explica ha continuació, ha estat Scala, ja que aquest és el llenguatge és el llenguatge mestre per combinar amb Spark. Val a dir que Scala és a Spark allò que Java és a Android. Altrament també s’hagués pogut desenvolupar amb Python, que disposa del Framewrok PySpark per tractar Spark de manera igualment òptima, o també Java.

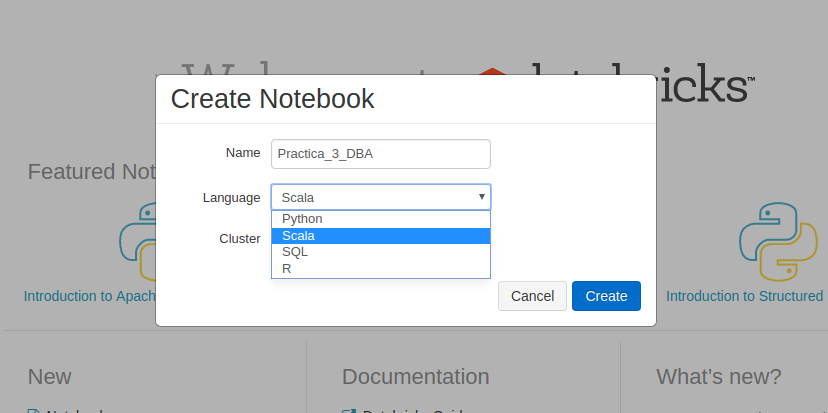
Scala es caracteritza per ser un llenguatge compilat, funcional i orientat a objectes (python té les mateixes característiques excepte que és interpretat en lloc de compilat), i a la pràctica s’ha utilitzat tant com s’ha pogut el poder dels llenguatges funcionals (maps, filters, etc..) així com les característiques dels llenguatges orientats a objectes.

**5. Creació i execució del notebook a DataBricks**

Per crear el notebook de treball a DataBricks cal dirigir-se a la pàgina de DataBricks Community i logar-se al sistema, una vegada fet això, trobem les següents opcions de treball:



Com no podia ser d’altra manera, cal triar l’opció ‘New → Notebook’ de la part esquerra de la pantalla. Aquest acte ens obre el següent ventall de possibilitats:



Després de posar nom a la pràctica, cal triar el llenguatge Scala i clicar el botó ‘create’. Això portarà al lector directament al nou notebook anomenat ‘Practica\_3\_DBA’. Però aquest notebook està buit! El que cal fer ara és dirigir-se a la ruta relativa al projecte ‘/files/notebook\_databricks/notebook.txt’ i copiar tot el contingut del fitxer per enganxar-lo tot a la primera cel·la del notebook. (El notebook només farà servir la primera cel·la, doncs n’hi ha prou!). Per tant els passos a seguir són:

- Accedir a la ruta relativa ‘/files/notebook\_databricks/notebook.txt’ i obrir el fitxer ‘.txt’, amb un editor qualsevol, per exemple, sublime, kate, etc..

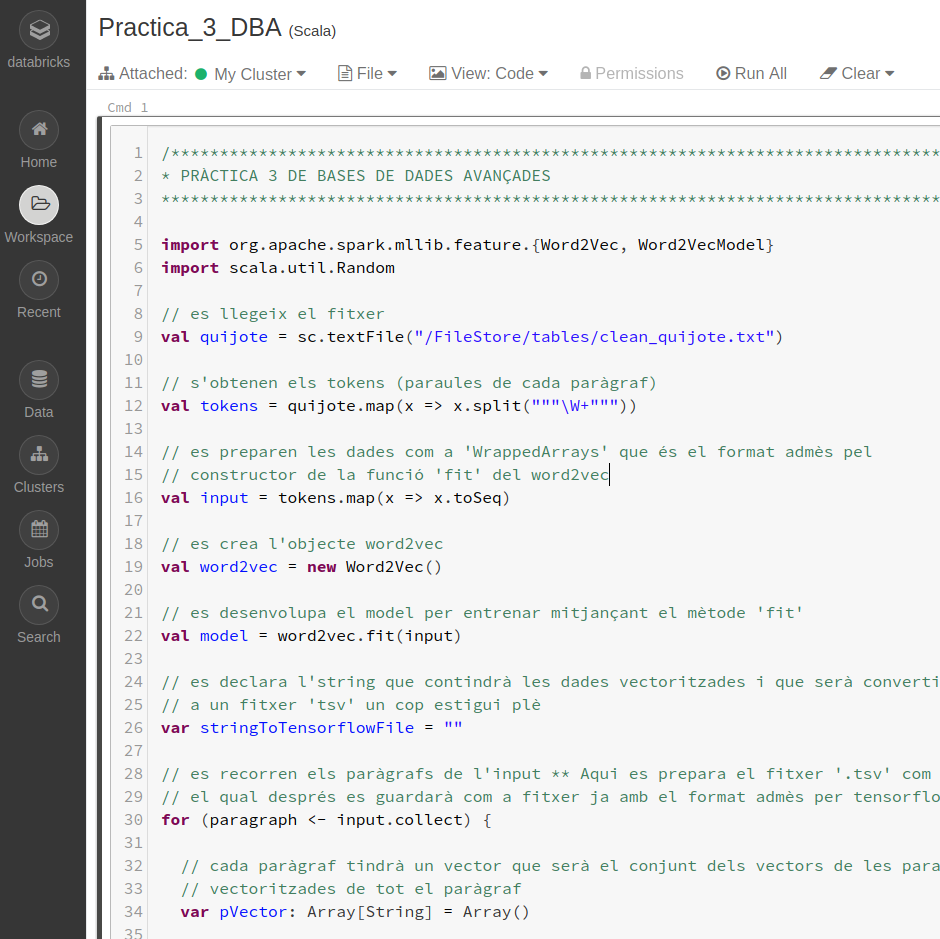
- Copiar tot el contingut del fitxer: ‘ctrl + a’ per seleccionar tot el text i ‘ctrl + c’ per a copiar-lo.

- Dirgir-se a la primera cel·la del notebook que s’ha creat al pas anterior

- Enganxar (ctrl + v) el contingut del fitxer dins la cel·la

- Executar la cel·la (ctrl + intro) o botó ‘play’ de la interfície gràfica

*Si el professor ho prefereix, a la carpeta /files/notebook\_databricks/exports\_databricks s’hi troba el notebook exportat de totes les maneres que permet DataBricks*. El resultat dels passos anteriors hauria de ser com el següent, i s’hauria de visualitzar el notebook d’aquesta manera després de copiar el ‘.txt’ o importar un dels fitxers de databricks:



Una bona manera de donar un primer cop d’ull al notebook és executant el fitxer: /files/notebook\_databricks/exports\_databricks/Practica\_3\_DBA.html

Aquest fitxer obre una pàgina web local on es pot veure d’una ullada tot el notebook.

Nota: Això no crea el notebook per executar, només serveix per a visualitzar-lo!

**6. Detall del programa (Part 1: Com fer l’execució)**

Abans d’entrar en el detall del programa cal explicar un aspecte important al lector:

L’objectiu del programa és generar un output en format ‘.tsv’ que és l’admès per TensorFlow a la darrera part de la pràctica. Aquest output ja es dóna generat amb l’entrega per tal d’estalviar feina al lector, es troba a la ruta relativa:

‘/files/quijote/quijote\_to\_tensorflow.tsv’

Ara bé, aquest output, que es genera executant el notebook de databricks de l’entrega, es pot obtenir de la següent manera:

* A l’executar el notebook el fitxer de sortida ‘quijote\_to\_tensorflow.tsv’ que s’ha generat ha anat a parar al sistema de fitxers de databricks. Per obtenir-lo cal executar el notebook per generar el fitxer, i una vegada fet això cal obrir una nova pestanya al navegador i executar la següent URL:

https://community.cloud.databricks.com/files/tables/quijote\_to\_tensorflow.tsv?o=1519882038357773

*Alerta, però, amb el paràmetre ‘o=1519882038357773’ ja que pertany a la sessió de Databricks de Pau Sanchez. Cal que el lector el canviï pel seu paràmetre que es troba a la URL del seu notebook. Tot això s’explica també dins el mateix notebook.*

Una vegada s’ha executat la URL el fitxer anirà a parar al directori local de descàrregues del l’usuari que ha fet la descarrega com si es descarregués qualsevol altra cosa.

A part d’això i per curar-se en salut, els autors proposen una segona manera de generar el fitxer ‘.tsv’ per si hi hagués algun problema amb l’altre sistema.

- S’ha creat un projecte de Intellij Idea que s’adjunta a la pràctica amb exactament el mateix codi que conté el notebook que el que fa el que fa és gravar el fitxer a disc, i ho fa a la següent ruta relativa al projecte: ‘/files/quijote/quijote\_to\_tensorflow.tsv’

Un aspecte important és que aquest fitxer ‘.tsv’ ja s’entrega generat a la pràctica, ara bé, si el lector vol comprovar el bon funcionament del programa només cal que executi el projecte d’Intellij, el qual també s’ha adjuntat degudament.

Un altre detall important és que per poder visualitzar correctament les dades a TensorFlow (més endavant se’n parla amb detall), s’ha preferit generar el fitxer d’output pels 1000 primers paràgrafs del text (el quijote en té al voltant de 5000), doncs la visualització de tots els paràgrafs a TensorFlow quedava ‘massa plena’, es barrejava i se solapava la informació i no s’entenia res de res a la visualització. Per aquest motiu s’ha decidit ‘capar’ l’output ‘.tsv’ a 1000 línies on cada línia conté el text del paràgraf i un vector de característiques que el representa.

Nota: Si el lector té el desig comprovar la visualització amb tots els paràgrafs enlloc dels primers 1000, només ha de canviar la instrucció ‘input.take(1000)’ per ‘input.collect’ i re-executar per generar el nou output que es trobarà a la mateixa ruta havent sobreescrit l’anterior. Ara bé, s’avisa que la visualització de 5000 paràgrafs és absolutament nefasta. Molt millor amb 1000.

Una vegada entès això, es pot passar al desenvolupament de l’entrenament i la vectorització del text, al següent apartat.

**7. Detall del programa (Part 2: Desenvolupament)**

El programa, que s’explica a continuació es pot trobar tant al notebook proposat com a la següent ruta relativa:

>> /Intellij\_Word2vec/Word2vec\_P3\_DBA/src/main/scala/Main.scala

S’aconsella tenir-lo obert per seguir l’explicació i el detall del programa, que s’explica a continuació:

- Es llegeix el fitxer del quijote net de símbols i ordenat per paràgrafs mitjançant el context d’Spark (sc)

- De cada paràgraf s’obtenen els tokens, és a dir, ara cada paràgraf és un vector de paraules (recordar que s’han filtrat les stop-words al programa de neteja).

- Es transformen els vectors en ‘WrappedArrays’ que és el format de vector admès per la funció ‘fit’ de Word2Vec.

- Es crea l’objecte word2vec i es crida al mètode ‘fit’ que rep com a argument l’input generat de ‘WrappedArrays’. Aquesta funció desenvolupa i entrena el nostre model, que més endavant es farà servir per dur a terme la vectorització de les paraules.

Aquí cal fer un incís i explicar com es farà la vectorització abans de seguir amb l’execució del programa:

*Quan es vectoritza una paraula s’obté la paraula seguida d’un vector de característiques que la representen. Aquestes característiques són valors numèrics de tipus flotant, i per defecte se’n generen 100 per cada paraula. D’això se n’encarrega el mètode transform que es crida amb l’objecte del model. Ara cada paraula vectoritzada té una forma com la següent:*

*A [float, float, float, float ........ float] → On A és la paraula i cada float és un valor numèric en coma flotant que representa una de les 100 característiques de la paraula.*

*El que cal, però, aconseguir, és un vector representatiu d’un paràgraf ja que el que es pretén comparar són paràgrafs sencers i no paraules! Aquest acabarà tenint la següent forma:*

*ABCDEFG [float, float, float, float ........ float] → On ABCDEFG són les paraules del paràgraf i cada float és una característica, atenció, que representa una de les 100 característiques d’entre les característiques de les paraules de tot el paràgraf.*

*Per generar el vector de cada paràgraf el primer que s’ha fet és generar un vector gegant per cada paràgraf, amb les característiques de totes les paraules que el composen. Per exemple un paràgraf de 30 paraules, de moment, tindrà 3000 característiques. Ara és quan arriba el problema: TensorFlow no admet vectors de característiques de diferent mida, sinó que només reconeix fitxers de vectors de característiques simètrics (vectors de mida idèntica). Com que les característiques de cada paraula són 100, això fa que que s’hagin de ‘capar’ les característiques del paràgraf a 100, per tractar el cas en que un paràgraf només està composat per una paraula (es donen uns quants casos).*

*Ara cal triar quina és l’estratègia per, un cop vectoritzades les paraules, crear el vector de característiques representatiu del paràgraf. El que s’ha fet ha estat seleccionar aleatòriament 100 característiques del gran vector de característiques del paràgraf i així s’obté el vector reduït de característiques que representen totes les paraules del paràgraf. Aquesta és una manera de fer-ho però de ben segur que n’hi ha d’altres.*

*Una vegada s’ha entès això, es pot seguir amb l’execució de l’algorisme allà on s’havia deixat:*

- Es crea un string buit on s’aniran afegit les línies: text del paràgraf + vector de caracterśitiques correctament tabulat i alineat. Després només caldrà desar aquest string en un fitxer d’exensió .tsv per ser executat a TensorFlow.

- Es recorren les paraules de cada paràgraf

- Es vectoritza cada paraula ‘model.transform(word)’

- Es va construint el gran vector de característiques del paràgraf

- Se seleccionen 100 característiques de manera aleatòria i es concatenen

- Quan s’obté tot l’string concatenat i tabulat es copia al fitxer ‘.tsv’

Visualtizació del fitxer .tsv que s’ha generat:



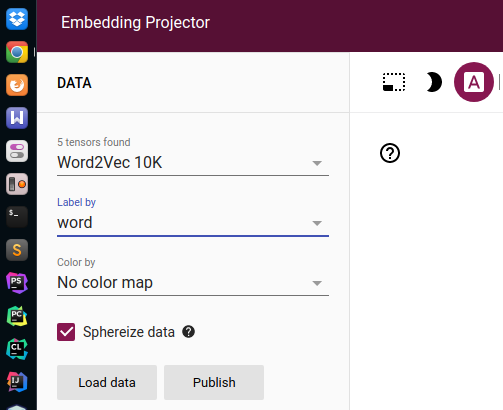
S’aprecia que cada paràgraf va seguit del seu vector de característiques. Això és el TSV.

**8. Visualització a TensorFlow**

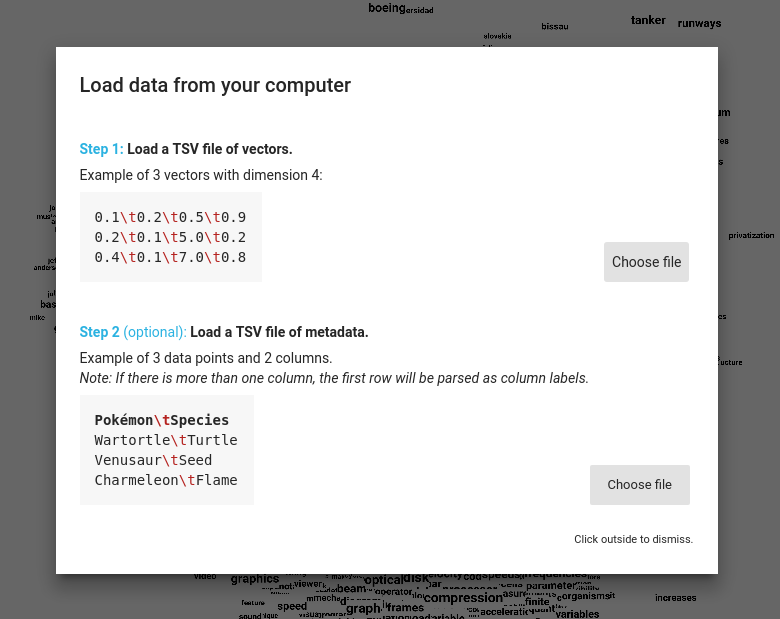
La darrera part de la pràctica consisteix en Visualitzar l’embedding amb l’eina TensorBoard de la suite tensorflow de google. Per fer-ho ha calgut adaptar al format requerit per tensorboard l’output de la fase anterior de la pràctica i carregar-lo al visualitzador de tensorboard.

El primer que cal fer és accedir a la web ‘projector.tensorflow.org’. És una aplicació molt intuïtiva la qual només entrar ja ofereix al visitant una visualització de mostra. Concretament un word-count o comptador de paraules. Vegem però, què cal fer per carregar el nostre fitxer ‘.tsv’ generat a l’apartat anterior per tal de visualitzar les distàncies entre els paràgrafs del Quijote.

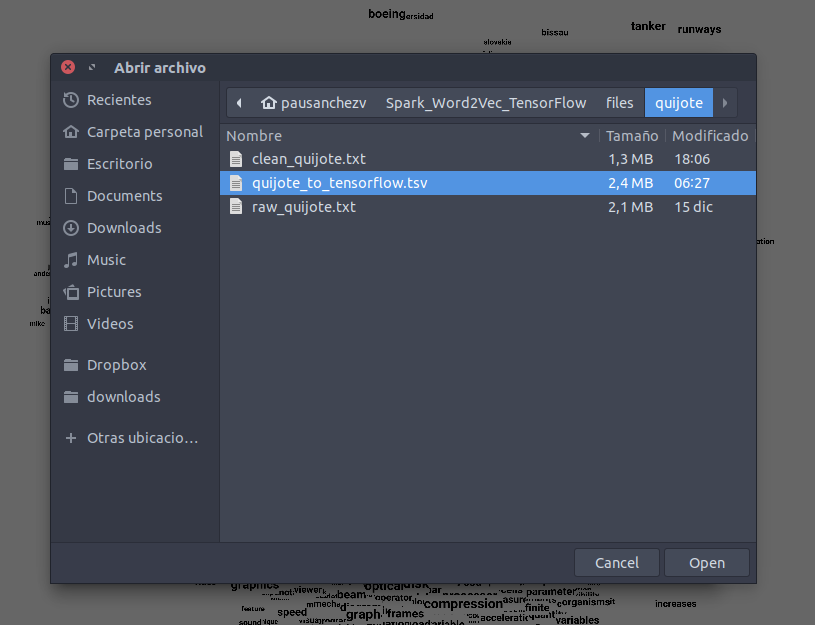
Una vegada dins la web, a la part esquerra trobem un botó ‘Load Data’, sobre el qual fem clic:



Aquest botó obre una finestra que permet dos tipus d’importació: el primer permet carregar un fitxer ‘.tsv’ de vectors, que és justament el que hem generat a l’apartat anterior amb Spark. La segona opció, que no farem servir, permet importar fitxers de metadata.



Seleccionem el fitxer ‘.tsv’ que ha quedat gravat a la següent ruta relativa al projecte: ‘/files/quijote/quijote\_to\_tensorflow.tsv’, i l’obrim.

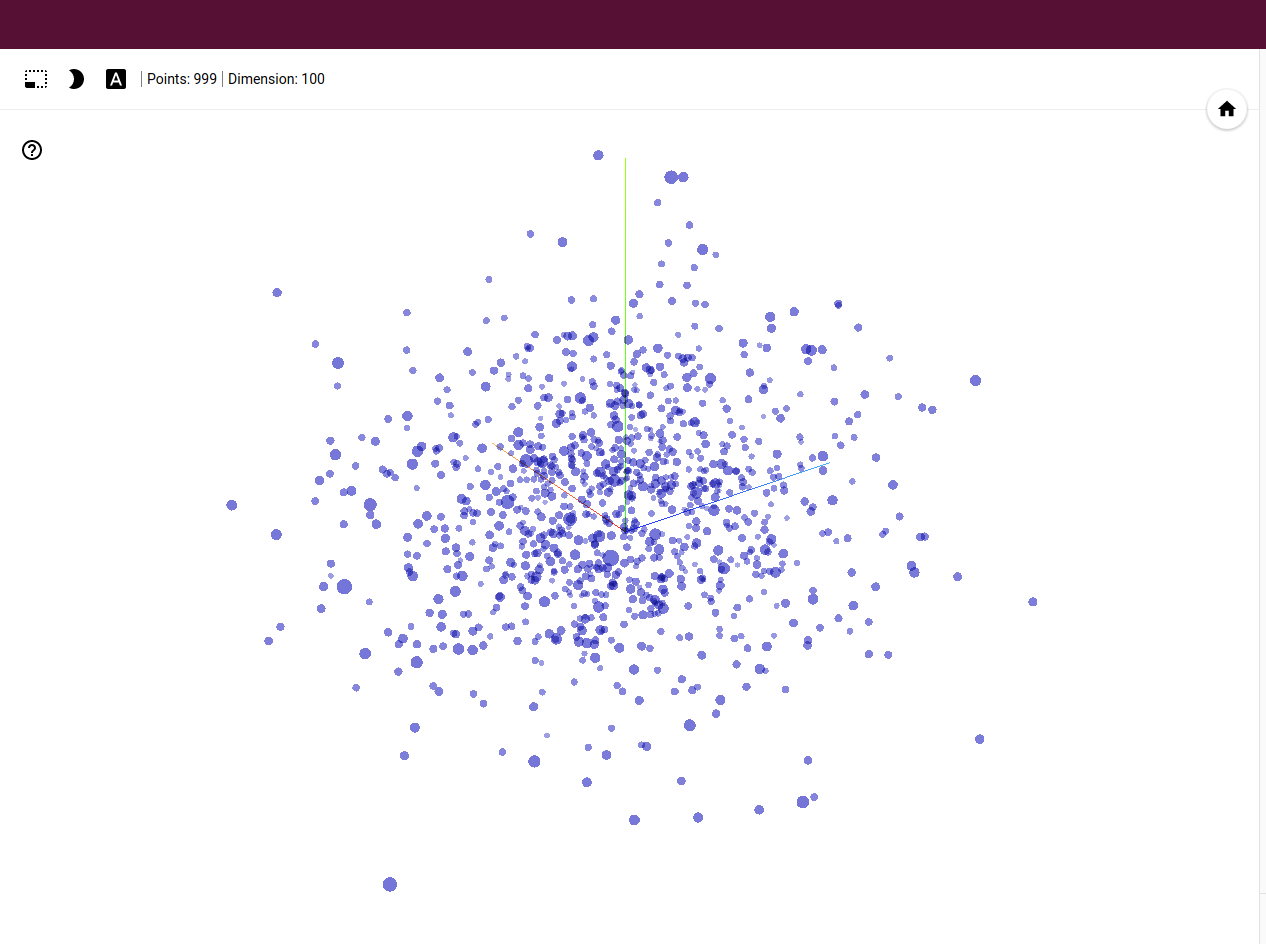


Aquest és el fitxer ‘.tsv’que ja es dóna amb l’entrega i que es pot regenerar executant el projecte d’intellij.

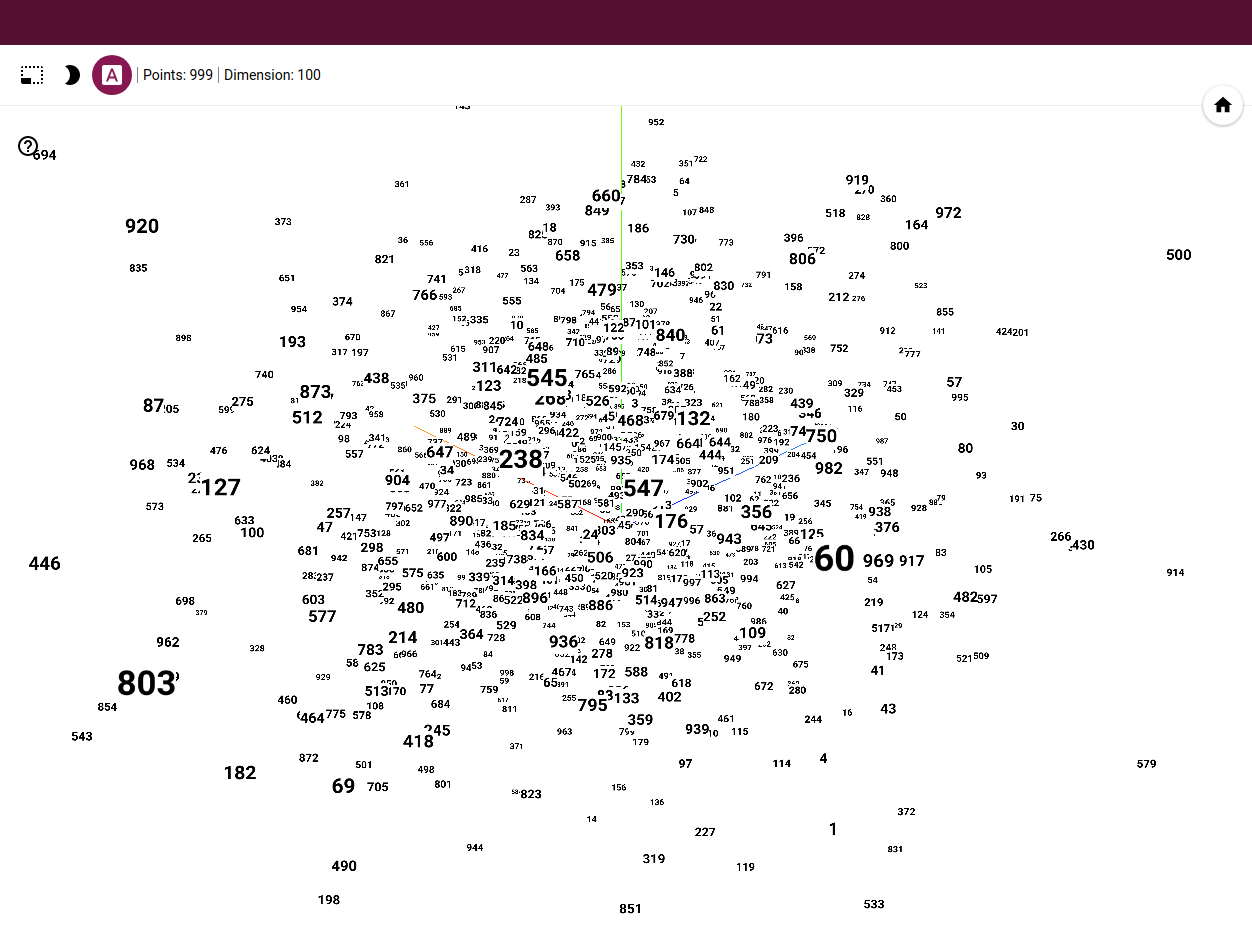
Si el lector vol executar el ‘.tsv’ generat amb el notebook, ha d’haver seguit els passos d’execució i descarregat el fitxer que hauria de trobar-se a la carpeta de descàrregues del seu PC.

El primer que ofereix TensorFlow és una visualització de nodes on cada node de la visualització representa un paràgraf del quijote, en el cas que ens ocupa. Cal recordar com s’ha explicat prèviament que per millorar la visualització s’han agafat només els 1000

primers paràgrafs del llibre. Llavors els nodes esperats haurien de ser 1000 nodes amb dimensió 100 cadascun d’ells, ja que cada paràgraf està representat amb un vector de 100 característiques. A la següent visualització el lector pot comprovar que això és correcte si mira a la part superior on indica que hi ha 1000 nodes, de dimissió 100.



El problema de la visualització de nodes rau en el fet que l’usuari no sap a quin paràgraf correspon cada node. Per millorar la visualització cal prémer el botó de canvi de nodes, aleshores l’usuari pot saber quin numero de paràgraf s’assembla més a quin, i acabar corroborant la similitud mirant els números de línia al fitxer de text.



A la imatge anterior s’aprecia que, per exemple, el paràgraf 60 del 803 estan molt lluny l’un de l’altre, distància que indica que els paràgrafs contenen paraules molt diferents, és a dir, que no s’assemblen. Per contra el lector pot fixar-se, per exemple, en el paràgraf número 100 i 633, estan molt a la vora, cosa que indica similitud entre ells!

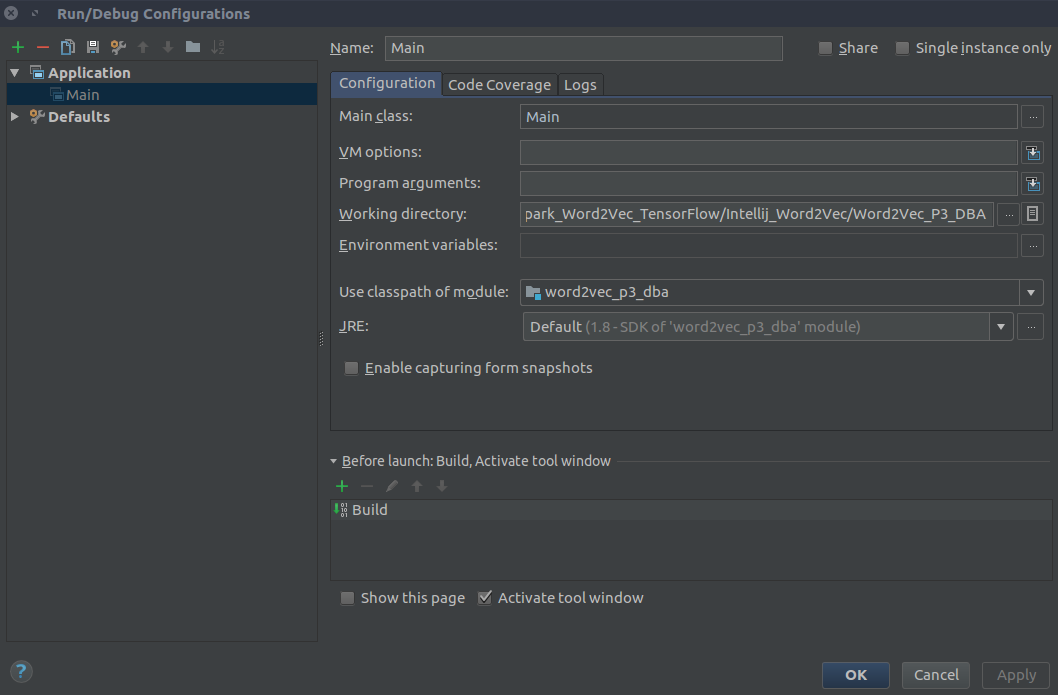
Enlloc del nombre de paràgraf es podria haver tret per pantalla el text del paràgraf sencer, però no s’ha cregut convenient fer-ho ja que la visualització és molt dolenta, solapant-se uns textos amb uns altres i acabant no entenent-se res del que s’està visualitzant. De manera que s’ha considerat mostra la visualització neta amb el número de paràgraf com a representant del mateix.

**Annex 1. Configuració de Intellij Idea**

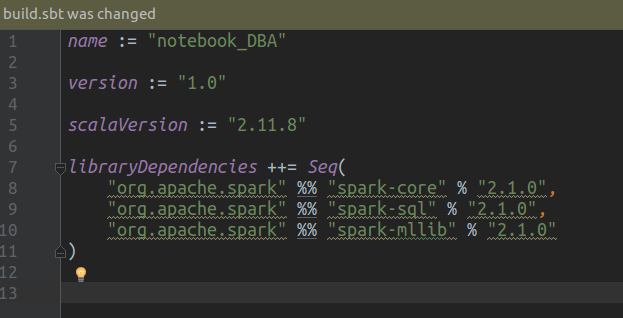
Una vegada acabada l’explicació de la pràctica cal mostrar la configuració que cal fer a Intellij Idea per tal de poder executar el projecte, en cas que el lector vulgui regenerar el fitxer ‘.tsv’ per comprovar que funciona.

El projecte ja està configurat, de manera que n’hi hauria d’haver prou executant-lo directament ja que la configuració és a nivell de projecte i no d’IDE, no obstant a continuació s’expliquen uns petits trucs per si l’execució es negués a funcionar de primeres.

La pestanya ‘Run/Debug Configurations’ ha d’estar configurada de la següent manera, especialment els apartats ‘Working Directory’ i ‘Use class path of module’:



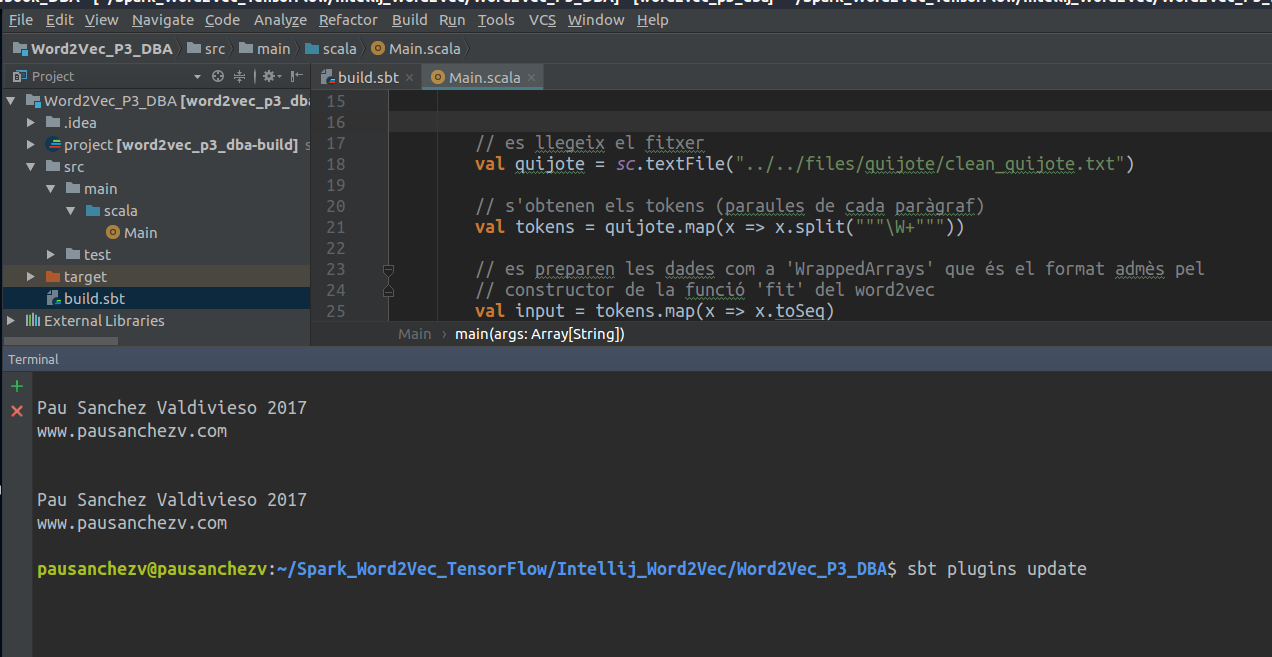
L’arxiu build.sbt és qui fa la importació i adaptació de les llibreries d’Spark a Intellij per a ser utilitzades amb Scala. Ha de tenir la següent configuració (després de modificar-lo cal refrescar el projecte per aplicar els canvis):

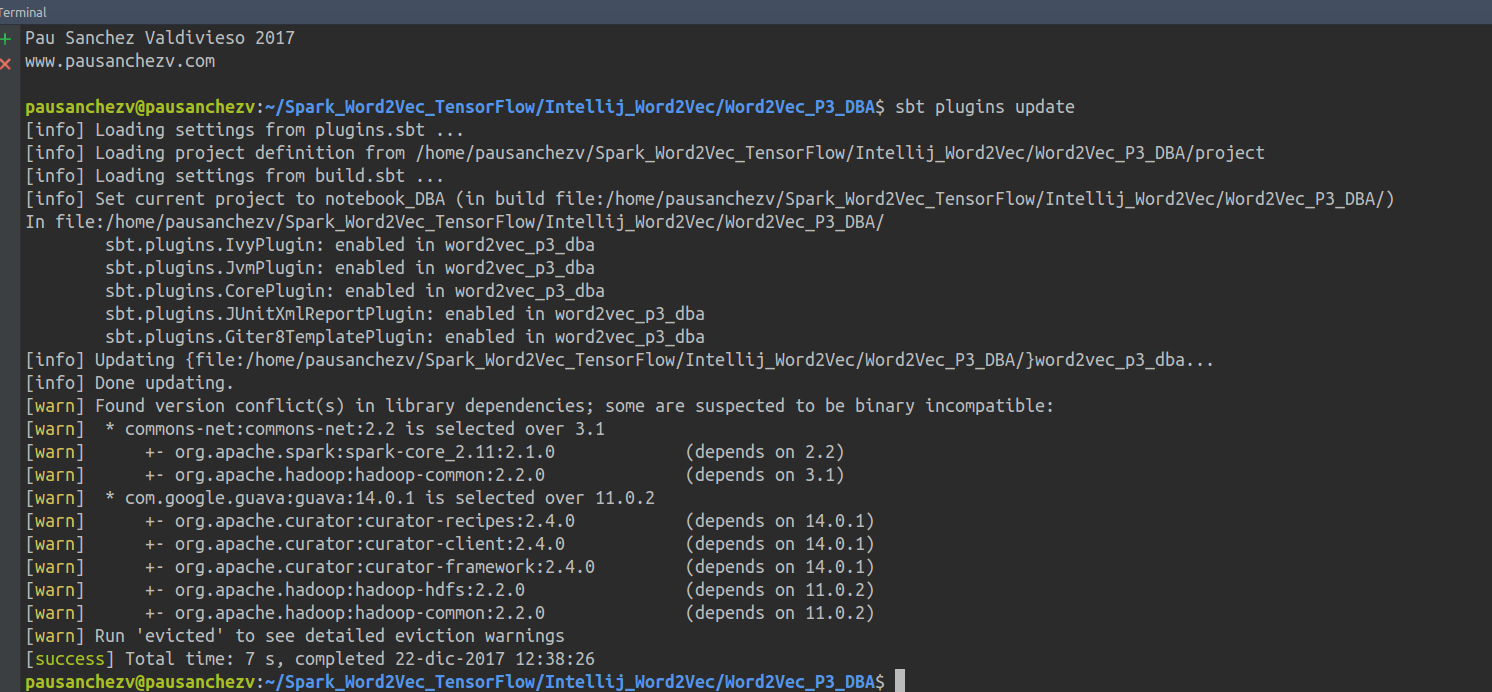


Ara cal fer un últim pas per actualitzar els plugins d’SBT:

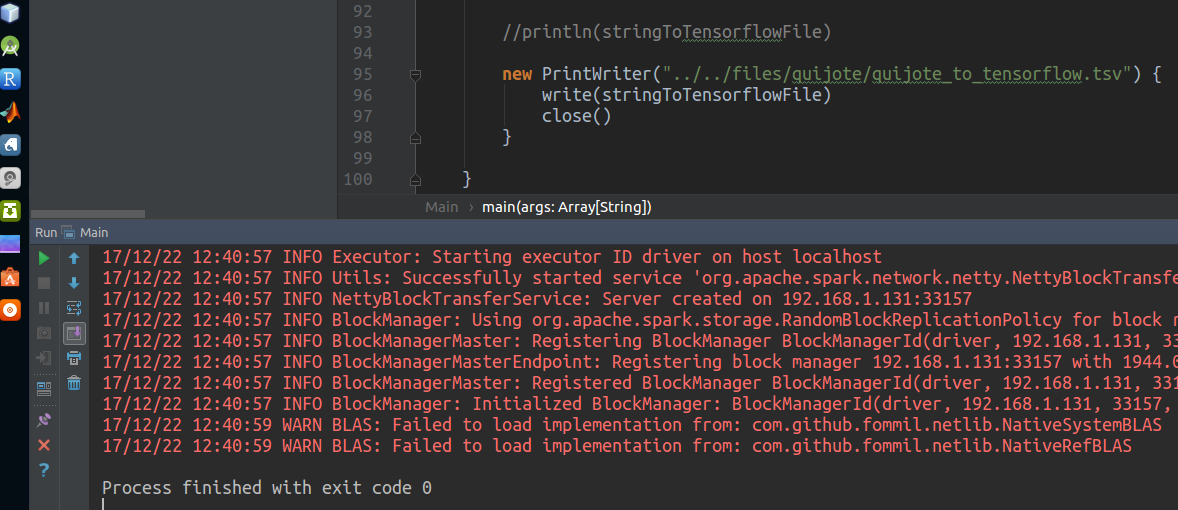
- Clicant alt + F12 obrim una consola de linux dins els mateix IDE i redireccionat a la carpeta on es troba el projecte d’Intellij. Allà mateix escrivim la següent instrucció i cliquem intro:

>> sbt plugins Update





Ara de ben segur que el projecte està apunt per fer clic sobre el botó ‘play’ i executar-lo sense problemes. Comprovem-ho!



Es comprova a la imatge anterior que el projecte ha executat correctament amb la frase ‘Process finished with code 0’ (Els missatges en vermell és informació purament d’Spark, no són pas errors).

Autors: **Pau Sanchez i Guillem Rabionet**

Sota la tutela de: **Enric Biosca**

Universitat de Barcelona, semestre de tardor

4 de gener de 2018