Universitat Oberta de Catalunya

Projecte de Final de Carrera

Àrea d'Intel·ligència Artifical

Consultor: David Isern Alarcón

Aplicació de predicció de risc d'una cartera

Autor:
Jesús Corrius Llavina

Data:
15 de juny de 2013

Índex

Índex de figures

1 Introducció

La motivació principal d'aquest projecte és la d'unir dues de les meves grans passions: les finances i la programació informàtica; tot intentant que el resultat obtingut i l'experiència acumulada sigui útil no només a mi, sinó també a altres persones. La base d'aquest treball és totalment experimental, però al mateix, la meva intenció és la de crear una aplicació que sigui útil per a un perfil d'usuari molt concret.

Crec que és interessant veure, en primer lloc, quina és la meva experiència com a informàtic, per tal d'entendre les meves motivacions a l'hora de realitzar aquest projecte. Sóc llicenciat en Comunicació Audiovisual i programador informàtic amb més de 15 anys d'experiència en C, C++ i Python. He treballat en diferents entorns com són web, Windows, Linux, Mac, etc. Tinc molta experiència en entorns multiplataforma i de programari lliure (sóc fundador i membre del consell d'administració del projecte LibreOffice, abans OpenOffice.org) i de programació de sistemes i multimèdia.

Últimament he estat treballant com a freelance i m'he trobat amb una gran demanda d'aplicacions per a smartphones, especialment per a l'iPhone, que és un mercat molt interessant des d'un punt de vista professional. Com a resultat vaig aprendre a fer aplicacions per iPhone i vaig estar treballant per algunes empreses que necessitaven una aplicació per a aquest entorn. La majoria d'aplicacions eren molt simples, però algunes requerien uns coneixements a baix nivell de la plataforma que em van permetre conèixer bé com funciona internament.

Un cop he conegut com funciona l'iPhone, i com crear aplicacions en aquesta plataforma, he pensat que seria molt interessant també aprendre com funciona la plataforma Android. Es tracta principalment d'un sistema operatiu per a dispositius mòbils, abanderat per Google i la Open Handset Alliance (OHA), que és el darrer membre de l'escena de la telefonia cel·lular i també dels PDA/smartphones. El seu continu creixement a ritme vertiginós trimestre rere trimestre ha superat totes les expectatives i, tot i ser un nouvingut, ja ha superat a tots els seus competidors directes: l'iPhone d'Apple, la plataforma Blackberry, Symbian o el Windows Mobile de la totpoderosa Microsoft.

Però aquesta és només la capa de presentació a l'usuari. El cor de l'aplicació és un servei web que és l'encarregat de fer els càlculs i processar la informació tot utilitzant tècniques d'Intel·ligència Artifical i una base de dades històrica amb dades financeres. Les dades s'introduiran en el telèfon intel·ligent, s'enviaran el servidor que realitzarà les operacions i els resultats tornaran al dispositiu per mostrar-los a l'usuari.

La motivació per utilitzar la Intel·ligència Artificial per tal d'analitzar dades financeres ve donada pel fet de la gran importància que està adquirint aquesta àrea de la informàtica en el sector financer i les excel·lents oportunitats laborals que hi ha en aquest sector ara mateix. Aquest treball s'adreça una part petita però especialment important com és la gestió de risc i les seves possibilitats d'optimització tot utilitzant tècniques de l'IA com son les Support Vector Machines.

Crec que és important assenyalar, per acabar, que l'objectiu de la meva aplicació és reduïr el risc de les inversions. En cap moment s'intenta predir la direcció que pendrà un determinat valor del mercat o optimitzar el retorn d'una inversió. No és una aplicació màgica per fer-se ric tot "jugant a la borsa".

1.1 Objectius

L'objectiu principal del projecte és la creació d'una aplicació per a telèfons intel·ligents que intenti predir la volatilitat no atribuïble al mercat per tal de permetre a l'usuari crear portfolios òptims utilitzant tècniques d'Intel·ligència Artificial com són les Support Vector Machines (SVM). Una vegada s'hagi predit aquesta volatilitat és crearà un portfolio òptim amb el pes adequat de cada un dels valors, per tal d'obtenir una inversió amb el mínim risc possible.

Per tal que el projecte sigui possible sense una inversió en maquinari alta, i la resposta es produeixi en un temps raonable, necessàriament haurem de limitar el nombre d'empreses que l'usuari podrà escollir. Hem de tenir en compte que algunes operacions matemàtiques que hem de fer per optimitzar el portfolio son molt costoses des del punt de vista de temps i potència de càlcul. En aquest treball ens limitarem a les empreses que formen part de l'índex IBEX 35 del mercat de renda variable espanyol. Això ens donarà un temps de resposta prou petit com per poder tornar les dades processades en un temps raonable a l'usuari de la nostra aplicació.

Els valors d'entrada que s'utilitzaran per fer la predicció no seran valors tècnics (per exemple: la mitjana del preu de l'acció en el mercat dels últims 200 dies), sinó fonamentals (com pot ser la mida de l'empresa, el valor comptable, etc.). Això ens donaran una idea més acurada de com l'economia real afecta el risc de les inversions en el mercat de valors. Normalment aquests valors són més difícils d'obtenir que no pas els tècnics, però existeixen pàgines web que te'ls ofereixen de manera gratuïta o bé de pagament. Si es volgués utilitzar aquesta aplicació per prendre decisions reals sobre inversions seria necessari utilitzar fonts de dades de pagament ja que les dades que t'ofereixen son molt més fiables, però pel nostre treball les dades

gratuïtes són suficients. Tot i que s'haurà de fer un tractament prèvia les dades per tal de detectar problemes com poden ser valors absents per tal que aquests no interfereixin en els resultats.

A continuació es descriurà l'experiència de l'usuari amb l'aplicació així com el seu funcionament intern per tal de fer més clar com funciona tot plegat:

1. En primer lloc l'usuari selecciona una sèrie de valors en l'aplicació per al sistema operatiu Android del seu telèfon intel·ligent. Aquests elements representen les accions del mercat de valors que l'usuari vol comprar. És important assenyalar aquí que hi ha aspectes molt importants per a la inversió, com és una correcta diversificació dels recursos, que l'aplicació no ho té en compte.
2. Els valors seleccionats són enviats al servidor a través d'una API HTTP en REST.
3. En el servidor s'utilitzen les dades històriques que hi ha a la base de dades per tal de calcular la volatitat esperada tot utilitzant una predicció a través de Suport Vector Machines.
4. Una vegada tenim la volatilitat de cada un dels valors, es calcula el portfolio òptim, tot assignant els pesos corresponents a cada un dels valors.
5. El portfòlio òptim es retorna al telèfon intel·ligent a través de l'API HTTP en REST.
6. L'aplicació mostra les dades a l'usuari.

1.2 Descripció de les tasques

El projecte es divideix en diferents tasques que, de manera lògica, s'hauran de realitzar una darrera l'altra. En el següent apartat veurem la planificació d'aquestes tasques i aquí ens centrarem a explicar amb cert detall en què consisteixen cada una d'elles. S'ha de tenir en compte que un programa informàtic és una peça d'enginyeria prou complicada, especialment en aquest cas on tenim en realitat dues parts ben diferenciades com son la part de l'aplicació per a Android i la part del servidor, i requereix una bona planificació i també una bona cerca de documentació i investigació sobre allò que es vol realitzar.

1. *Cerca de treball relacionat*: en aquesta tasca es buscaran treballs o aplicacions relacionades amb els objectius del nostre projecte. Es tindran especialment en compte literatura relacionada amb l'optimització

de portfolios i sobre Intel·ligència Artificial aplicada a les finances. Al mateix temps es buscarà a l'Android Market aplicacions que facin operacions similars. També es buscaran idees per a d'interfície d'usuari.

2. *Documentació*: en aquesta tasca es buscarà i llegirà tota la informació necessària per tal de fer l'aplicació, especialment la del Android SDK que proporciona Google, tot i que també existeixen moltes altres pàgines web que ofereixen informació i recursos sobre la plataforma Android. A la part del servidor es buscarà informació sobre com realitzar anàlisi de dades en Python y els diferents frameworks que podem utilitzar. Un cop s'haurà acomplert aquesta tasca, estarem en condicions de passar a la tasca següent.

3. *Disseny*: en aquesta tasca es farà el disseny de l'aplicació. La tasca anterior ens haurà donat tota la informació necessària per tal d'acomplir aquesta tasca satisfactòriament. Es farà el disseny de l'aplicació des del punt de vista d'enginyeria (classes, APIs que farem servir, la comunicació entre el client i el servidor, etc.) i també la interfície d'usuari i les diferents pantalles que tindrà l'aplicació per a telèfons intel·ligents.

4. *Instal·lació de l'entorn de treball*: en aquesta tasca s'instal·larà l'entorn de treball i totes les eines per fer l'aplicació, en aquest cas serà l'entorn Eclipse i, evidentment, l'Android SDK per una banda. Per altra banda també s'instal·larà el servidor i es deixarà operatiu.

5. *Implementació*: en aquesta tasca es durà a terme la programació pròpiament dita de l'aplicació, això és, la creació de les interfícies d'usuari de l'aplicació i també les classes i les crides a l'API del sistema que ens permetran tenir una aplicació perfectament funcional i llesta per poder-la provar. A la part del servidor s'implementarà la part de lògica i es posaran les dades necessàries per fer els càlculs a la base de dades.

6. *Avaluació de la implementació*: en aquesta tasca es provarà l'aplicació que s'haurà creat en la tasca anterior. Si no es disposa de cap dispositiu Android per fer la prova, serà provat principalment en l'emulador que ens ofereix l'Android SDK. Un cop tinguem totes les proves fetes, documentarem els problemes que hem trobat i les millores que s'han de fer a l'aplicació de cara a l'entrega final, tant a la part del telèfon com a la del servidor.

7. *Millores finals sobre la implementació*: en aquesta tasca s'utilitzarà tota la informació obtinguda en l'apartat anterior per tal de modificar el comportament de l'aplicació o bé resoldre els problemes trobats.

Aquesta serà l'última tasca que, una vegada realitzada, permetrà tenir l'aplicació definitiva i realitzar l'entrega final.

1.3 Planificació de les tasques

En l'apartat anterior s'han vist les tasques i ara es farà una estimació temporal de les mateixes. Farem el càlculs tenint en compte que el curs són unes 15 setmanes i així doncs es distribuiran les tasques en aquest context. Es pot afinar una mica més, però s'ha decidit utilitzar les setmanes com a unitat ja que aquesta unitat de temps és suficientment granular i es representativa de la manera d'organitzar el temps a la UOC. Tenint en compte els criteris anteriors, s'ha distribuït el temps de les tasques de la manera següent:

- Cerca de treball relacionat: 1 setmana
- Documentació: 1 setmana
- Disseny: 3 setmanes
- Instal·lació de l'entorn de treball: 0,5 setmanes
- Implementació: 4 setmanes
- Avaluació de la implementació: 2 setmanes
- Millores finals sobre la implementació: 3 setmanes

Si bé és cert que algunes d'aquestes tasques es poden fer en paral·lel, especialment les primeres de la llista, la gran majoria de tasques depenen les unes de les altres. O sigui que el diagrama de Gantt amb la planificació tindrà l'aspecte següent:

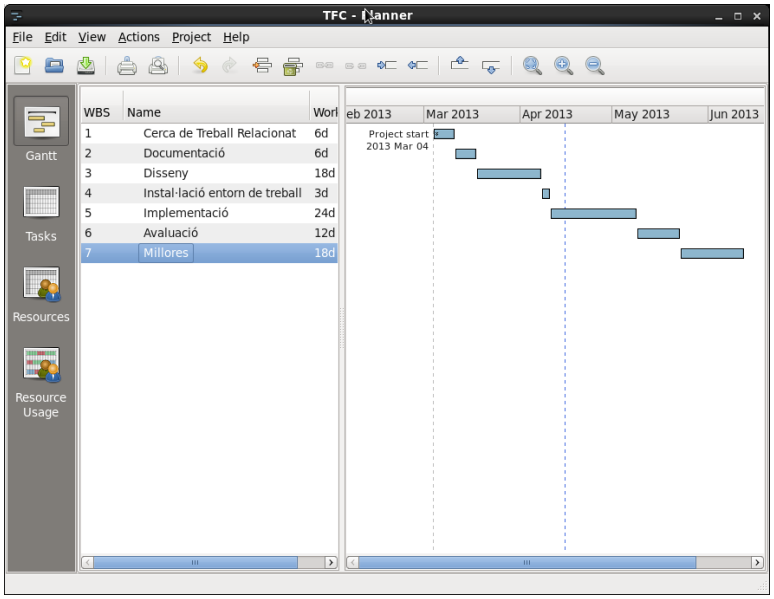

Figura 1: Diagrama de Gantt amb la planificació. *Font: Elaboració pròpia*

I a la taula següent podem trobar amb més detall les dates d'inici i de finalització de cada tasca:

Tasca	**Data d'inici**	**Data de finalització**
Cerca de treball relacionat	04/03/13	10/03/13
Documentació	11/03/13	17/03/13
Disseny	18/03/13	07/04/13
Instal·lació de l'entorn de treball	08/04/13	10/04/13
Implementació	11/04/13	08/05/13
Avaluació	09/05/13	22/05/13
Millores	23/05/13	12/06/13
Total projecte	04/03/13	12/06/13

En quant a l'estimació econòmica del projecte, estem parlant d'un desenvolupament de 15 setmanes de feina pensada per a un sol desenvolupador. Tot i això, hem de tenir en compte que la planificació és setmanal i no implica treballar vuit hores al dia en el projecte. Fent una estimació més realista, el projecte es pot fer perfectament en un mes de feina per un programador. Si posem un sou de 30.000 euros a l'any, el cost econòmic de l'aplicació seria:

30.000 euros / 12 = 2500 euros bruts.

2 La plataforma Android

En aquest apartat es veurà una introducció a la plataforma Android que ens portarà a través de la seva història i, molt especialment, la tecnologia sobre la que està construïda. S'aprofitarà per veure també la difusió de la plataforma que, com es veurà, és molt important i majoritària i això ens serveix com a bon justificant de la nostra elecció tecnològica.

2.1 Història

La plataforma Android, desenvolupada actualment per Google i els membres de l'Open Handset Alliance, és un conjunt de solucions per a dispositius mòbils que ofereix un sistema operatiu, programari intermediari i les aplicacions més importants. El sistema operatiu està basat en el kernel de Linux i en un conjunt de llibreries estàndards de programari lliure, que són el nucli sobre les quals es desenvolupa la plataforma. El projecte utilitza la llicència Apache [1] per a la major part del codi, la qual cosa fa que que també sigui un projecte de programari lliure o de codi font obert [2].

El desenvolupador original del projecte va ser l'empresa Android, Inc., que Google va comprar al 2005 [3]. Dos anys després de l'adquisició, el 25 de novembre del 2007, Google va anunciar la creació de l'Android Open Source Project (AOSP) que és l'organització que s'encarrega del manteniment i el desenvolupament de la plataforma [2].

Al mateix temps també es va anunciar la creació de l'Open Handset Alliance [4], que és un consorci de més 80 empreses de maquinari, programari i telefonia d'arreu del món que té com a objectiu la creació, el manteniment i la difusió d'un sistema de codi font obert per a telefonia mòbil. Els membres inicials de l'Open Handset Alliance van ser: Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile i Texas Instruments [4]. Més tard és van afegir altres empreses com ARM Holdings, Atheros Communications, Asustek Computer Inc, Garmin Ltd, PacketVideo, Softbank, Sony Ericsson, Toshiba Corp, and Vodafone Group Plc [5].

La primera versió pública de la plataforma, la versió 1.0, es alliberar el 21 d'octubre del 2008. A l'any 2011 van aparèixer dues versions diferents de la plataforma: la 2.3 (Gingerbread) i la 3.0 (Honeycomb). La primera versió està pensada per a telèfons mòbils[6] i la segona versió per a tablets [7]. El fet de tenir dues versions diferents per a cada una d'aquestes plataformes va ser només durant un temps i al final es va tornar a una sola versió per a tots els dispositius. En el moment d'escriure aquest treball l'última versió disponible és la 4.2.2 (Jelly Bean) [8] que funciona tan a telèfons mòbils com a tablets.

2.2 Difusió

El nombre d'usuaris de dispositius que utilitzen la plataforma Android ha tingut un creixement espectacular en molt poc temps. Durant el segon trimestre del 2009 la quota de mercat era del 2.8% [9] i al 2010 havia crescut fins al 33% [10].

El febrer del 2010 comScore va afirmar que Android tenia el 9% del mercat de telèfons intel·ligents dels Estats Units, partint del 5.2% del novembre del 2009. Al final del 2010, la quota de mercat havia pujat fins al 21.4% [11].

Podem veure aquesta evolució en el gràfic següent:

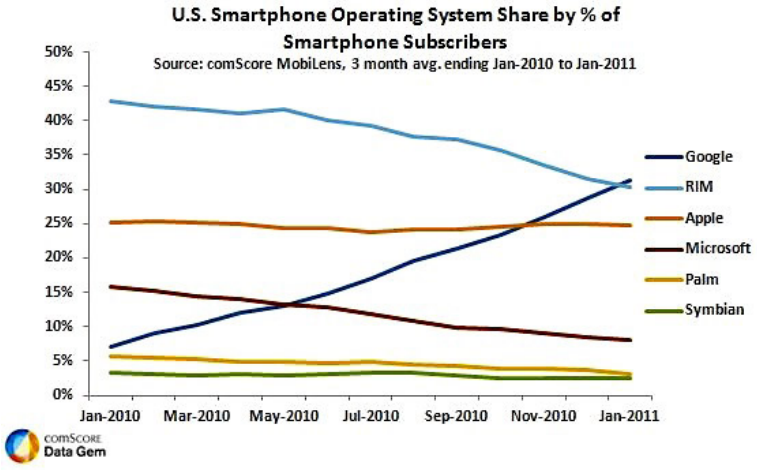

Figura 2: Evolució de la quota de mercat d'Android. *Font: ComScore*

Segons el NPD Group, les unitats venudes de telèfons intel·ligents amb Android el situen en el primer lloc als Estats Units, en el segon i tercer trimestres del 2010 amb una quota de mercat del 43,6% en el tercer trimestre [12].

Al febrer de 2011, durant el Mobile World Congress 2011 a Barcelona, Eric Schmidt, llavors CEO de Google, va anunciar que la plataforma Android ha arribat a 350.000 activacions per dia [13].

Per fer-nos a la idea de la seva gran popularitat, segons les dades publicades per l'empresa americana Nielsen a principis de març del 2011 [14], Android va esdevinir el sistema operatiu per a smartphones amb més demanda per part dels usuaris dels Estats Units.

En el gràfic següent podem veure quin és l'últim dispositiu mòbil adquirit pels usuaris que van participar en l'estudi:

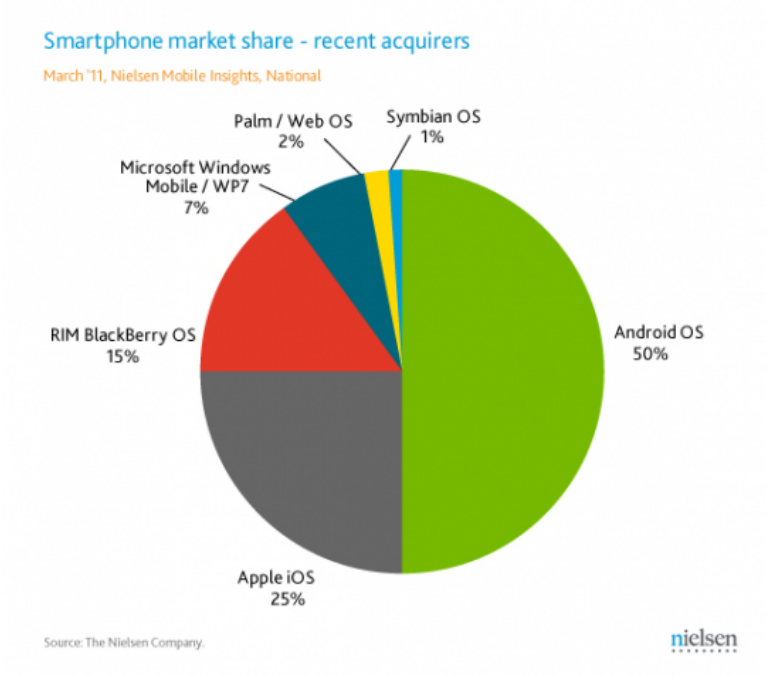

Figura 3: Quota de mercat dels smartphones, març del 2011. *Font: Nielsen*

El 10 de maig del 2011, Google va anunciar que 400.000 nous dispositius s'activen cada dia i que, en total, se n'havien activat ja més de 100 milions de dispositius Android [15].

Aquesta tendència va seguir creixent durant el 2012 i, durant el tercer quart del 2012, la quota de mercat dels Android era del 75% [16], amb 750 millons de dispositius activats i un total d'un milió i mig d'activacions per dia [17].

Durant el Maig del 2013 es calcula que hi ha un total de 900 milions de dispositius Android activats [18].

2.3 Tecnologia

En aquest apartat s'estudiarà la plataforma Android des d'un punt de vista estrictament tecnològic. En primer lloc, es veurà una introducció general a la seva arquitectura i després ens centrarem en un aspecte clau per a aquest projecte, la creació d'aplicacions tot utilitzant l'Android SDK.

2.3.1 Arquitectura

En el seu nivell més baix, el sistema operatiu està basat en un nucli de Linux de la versió 3.1 que actua com a capa d'abstracció entre el maquinari i les

capes de programari superiors. Inclou els controladors bàsics per als dispositius disponibles al terminal: pantalla, so, xarxa sense fils, gestió d'energia, etc.

Per altra banda, a les capes superiors, el conjunt de solucions Android consisteix en una sèrie d'aplicacions escrites en el llenguatge de programació Java que, al mateix temps, s'executen sobre un framework de llibreries principals escrites també en Java. Tot aquest codi s'executa sobre una màquina virtual que s'anomena Dalvik [19].

Però no només hi ha Java en aquest conjunt de solucions, sinó que podem trobar també llibreries i programes escrits en altres llenguatges de programació. Les llibreries escrites en C inclouen el gestor de superfícies, l'OpenGL ES 2.0 3D, el WebKit, el motor gràfic SGL, l'SSL, i la Bionic libc. El sistema operatiu Android, inclòs el nucli de Linux, es compon aproximadament de12 milions de línies de codi, incloent 3 milions de línies de XML, 2,8 milions de línies de C, 2.1 milions de línies de Java, i 1,75 milions de línies de C++ [20].

Tot i que la majoria de desenvolupament sobre la plataforma es fa amb Java, podem utilitzar també altres llenguatges de programació. De fet, gràcies al Android Native Development Kit, podem crear codi en C i altres llenguatges de programació que es puguin compilar amb el gcc a codi ARM natiu i instal·lar- los al dispositiu [21]. Després aquest codi es pot executar des del Java com si fos una llibreria nativa escrita en aquest llenguatge de programació.

En el gràfic següent podem veure d'una manera molt clara quina és l'arquitectura del sistema:

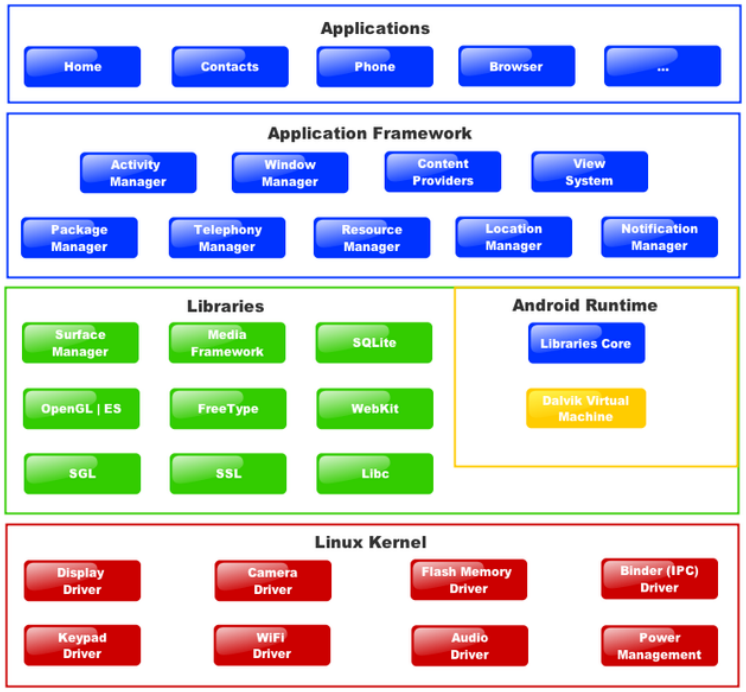

Figura 4: Arquitectura del sistema Android. *Font: Google*

Com es pot veure en el gràfic, l'arquitectura té diferents nivells o capes, amb el nucli de Linux a sota de tot i les aplicacions en el punt més alt. Una de les maneres de separar les diferents capes és veure què s'executa amb codi natiu, i quines parts utilitzen Java.

Codi natiu

A baix nivell tenim el nucli de Linux que s'encarrega de carregar els controladors del maquinari i controlar la gestió de processos, memòria, entrada i sortida, etc. Per sobre el nucli, tenim les llibreries del sistema, com la llibreria estàndard de C, i les altres llibreries que ens ofereixen el codi bàsic sobre el qual es construeix la resta del sistema. L'últim element que utilitza codi natiu és la màquina virtual de Java Dalvik, que és la que executa tot el codi Java que trobem al nivell que veurem a continuació.

Codi en Java

La màquina virtual Dalvik, vista a l'apartat anterior, executa tot el codi Java de la plataforma, però no tot el codi Java es troba en el mateix nivell. En primer lloc tenim les llibreries bàsiques sobre les quals es construeix tota la resta, aquestes inclouen, per exemple, l'accés des de Java del codi natiu

en C o C++ de les llibreries que hem vist en l'apartat anterior. Sobre aquestes, tenim el framework que ens proporciona l'Android SDK propiament dit, sobre el qual es construeixen les aplicacions, i que ens ofereix totes les abstraccions típiques d'un sistema per a construir aplicacions, com pot ser una "classe finestra" o bé l'accés d'alt nivell sobre les propietats de la càmera dels dispositius. És sobre aquesta API que es desenvolupen les aplicacions que interactuen amb l'usuari final, tant les que venen de sèrie amb el sistema com les que els desenvolupadors externs poden crear.

2.3.2 Desenvolupament d'aplicacions

Les aplicacions per a la plataforma Android es desenvolupa en Java utilitzant l'Android Software Development Kit, la primera versió del qual es va publicar alguns mesos després de la presentació pública del projecte. L'Android SDK inclou un conjunt de llibreries, un depurador i un emulador que permet executar el codi sense necessitat d'un dispositiu real. També, com ja hem comentat en l'apartat anterior, existeix un kit de desenvolupament en codi natiu conegut com a NDK (Native Development Kit) que permet escriure i compilar programari per a executar directament per a processadors de l'arquitectura ARM en comptes d'utilitzar la màquina virtual. Aquesta aproximació al desenvolupament proporciona beneficis en rendiment i versatilitat, però és només adequat per a aplicacions que tenen unes necessitats molt específiques en aquest sentit. Tot i que hem de realitzar algunes activitats de baix nivell, per crear l'aplicació per a aquest projecte s'utilitzarà el mètode de desenvolupament estàndard per a la plataforma que és el llenguatge de programació Java conjuntament amb l'Android SDK. A més a més de la plataforma en sí, l'entorn de desenvolupament que farem servir serà un IDE (o sigui: Integrated Development Environment o Entorn de Desenvolupament Integrat) que si bé no és en absolut necessari i seria possible utilitzar un editor de text i les eines de línia d'ordres a través d'un terminal per a escriure, compilar, depurar i executar un programa, la feina és molt més fàcil i ràpida de realitzar a través d'un entorn integrat. L'entorn oficialment suportat i recomanat per al desenvolupament en Android és Eclipse (versió 3.4 o superior) i el connector específic d'aquest per a Android, anomenat ADT plugin (Android Development Tools) [19].

Per tal de configurar l'entorn de desenvolupament es necessiten els següents elements:

- JDK (Java Development Kit) versió 6 o 7: Es pot descarregar des de la pàgina web següent: http://www.oracle.com/technetwork/java/javase/overview/index.html. És important remarcar que es necessita el JDK, ja que

amb el JRE (Java Runtime Environment o Entorn d'Execució de Java), que és el que es troba normalment instal·lat en els nostres sistemes operatius, no n'hi ha prou. La versió utilitzada en aquest projecte és la JDK versió 6 Update 45, que és l'ultima versió estable suportada per Oracle del Java 6.

- Eclipse IDE: Es pot descarregar a la pàgina web: http://www.eclipse.org/downloads/. Per a aquest projecte es va instal·lar la versió clàssica 4.2.2 (Helios) en la seva edició per a arquitectures de 64 bits.
- Android SDK: Aquest component essencial es troba disponible a l'adreça web següent: http://developer.android.com/sdk/index.html. Pel projecte s'ha utilitzat la versió 17 de l'API, que era l'última versió disponible en aquell moment.
- ADT Plugin: Aquest és el component que integra l'Android SDK amb l'entorn integrat de desenvolupament Eclipse. Es pot descarregar des de la pàgina web següent: http://developer.android.com/sdk/eclipse-adt.html. La versió que s'ha utilitzat en el projecte és la 22.0.0.

A la figura següent es pot veure l'entorn de desenvolupament integrat Eclipse configurat per a treballar amb la plataforma Android:

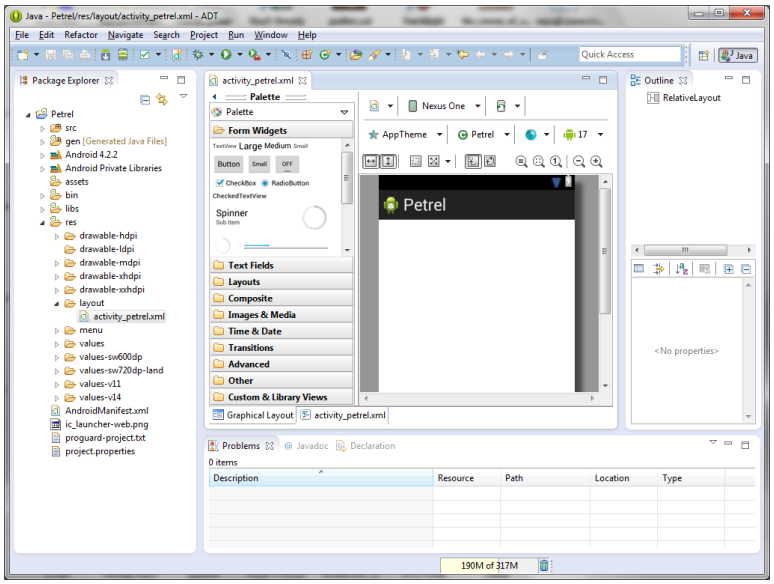

Figura 5: Entorn de desenvolupament de l'Android SDK per a Eclipse. *Font: elaboració pròpia*

Per a depurar el programari s'utilitza l'emulador incorporat al SDK. Per tal de crear un nou emulador cal engegar el "Android SDK and AVD Manager"

(per accedir a ell es pot utilitzar la icona de la barra superior o bé es pot utilitzar la icona corresponent del menú 'Window' que trobem a la part superior de la finestra de l'Eclipse) i crear-la amb el botó 'New' tot escollint els paràmetres que volem personalitzar. S'ha de tenir en compte que si el directori d'usuari no es troba en la seva ubicació per defecte, hem de modificar la variable d'entorn ANDROID_SDK_HOME perquè indiqui el el lloc corresponent en el nostre sistema de fitxers.

En la següent captura de pantalla, es pot veure l'emulador de l'Android en funcionament. La versió de l'emulador utilitzada és la 4.2.2 (Jelly Bean):

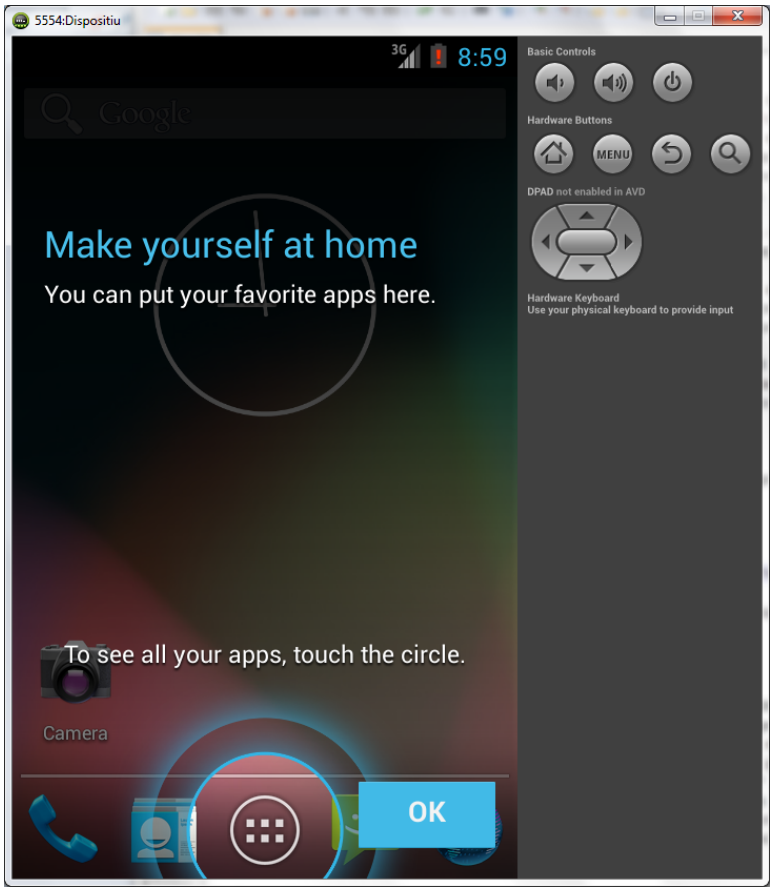

Figura 6: Emulador de dispositiu Android. *Font: elaboració pròpia*

Amb aquestes eines i aquest entorn de desenvolupament que hem vist en aquesta secció es desenvoluparà la nostra aplicació.

3 Anàlisi del risc

En aquesta secció es descriu l'anàlisi del risc a nivell formal, tal i com s'implementarà en l'aplicació. A la pràctica, en la majoria dels casos, utilitzarem llibreries que ja estan fetes, que son utilitzades per molts professionals de tot el món i que ens proporcionen aquesta funcionalitat de manera eficient. Tot i això és important tenir un marc teòric per entendre què és el què fan aquestes llibreries internament.

Aquesta secció està dividida en dos grans apartats que descriuen el risc del mercat en general i el risc d'una cartera de valors respectivament. En el nostre programa s'utilitzaran tots dos tipus.

3.1 Anàlisi del risc del mercat

3.1.1 Introducció

En aquesta secció veurem com es fa una anàlisi del risc del mercat a través de sèries temporals de dades, normalment el preu de tancament del dia d'una acció al llarg d'un període determinat de temps, i com podem utilitzar aquestes dades per tal de fer una anàlisi de la volatilitat d'aquest mercat en general.

Aquesta secció està basada especialment en la introducció del llibre *An Introduction to high-frequency finance*[41] de Dacorogna, Gencay, Muller, Olsen i Pictet, publicat a l'any 2001.´El contingut del llibre és d'un nivell molt més avançat del què tractarem en aquesta secció, o en la resta del treball.

3.1.2 Característiques de les sèries de dades

Podem considerar el retorn del mercat durant un període determinat de temps com una sèrie de valors ordenats cronològicament que representen el preu de tancament del dia d'un valor determinat. Aquest retorn es pot calcular tot utilitzant la fórmula següent:

$$r_i = ln(X_i/Xi - 1) \tag{1}$$

On i és l'índex de la seqüència (el preu del valor al final del dia en el nostre cas en particular) i X_i és el preu de l'acció en el temps t_i.

Per altra banda, la definició de volatilitat realitzada és molt més complicada perquè depén de l'ús que se'n vulgui fer i les propietats de les sèrie temporal de dades que ens interessi. Normalment es defineix com la *desviació estàndard* dels retorns d'un interval determinat, que és la més simple però també és molt utilitzada en tota mena de càlculs financers.

Algunes característiques interessants de les sèries de dades financeres són les següents:

- Els valors de retorn tenen només una correlació amb el temps amb un interval molt petit, d'uns pocs minuts, això vol dir que hi ha una clara absència de correlació lineal en les a llarg termini com les que utilitzarem en aquest treball.
- Per altra banda, la volatilitat té una memòria molt més llarga, normalment diversos mesos. La funció d'autocorrelació en aquest cas exhibeix un comportament hiperbòlic.
- La funció de densitat de la probabilitat dels retorns té una cua llarga en temps petits (de minuts a dies) i és gausiana amb interval de temps més grans que aquests que hem mencionat.
- Les volatilitats també estan correlacionades negativament amb els retorns corresponents. Aquesta és una característica pròpia dels mercats de valors.

Aquestes diferencies segons si estem parlant a curt o a llarg termini fa que segons el tipus d'inversió que vulguis realitzar o que estiguis analitzant, t'interessa utilitzar diferents mesures a l'hora de calcular la voltilitat que tinguin en compte aquestes característiques. En aquest treball només es veuran les que fan referència a més llarg termini, tenint en compte les dades que utilitzarem i el seu interval.

3.1.3 Models de volatilitat

En primer lloc és molt important reconèixer i tenir en compte que no hi ha actualment cap model de volatilitat que expliqui completament totes les característiques de la volatilitat dels mercats. Això vol dir que es fan servir models diferents i que cada un d'ells s'ajusta millor a una característica concreta. No hi ha un model universal que expliqui la volatilitat en tots els casos possibles.

Els dos models de volatilitat més utilitzats són el model deterministic i el model estocàstic. Els models deterministics consideren que la volatilitat (o

variància condicional) és una funció determinística dels valors passats que estan descrits per alguna mena de procés estocàstic. Per altra banda, els models estocàstics descriuen la volatilitat des de dins del procés estocàstic matiex i són molt més flexibles que els models deterministics.

Per la seva facilitat d'ús, en aquest treball només veurem el model deterministic, el més famós del qual és el GARCH que forma part de la família ARCH i que defineix la volatilitat de la manera següent:

$$\sigma_t^2 = \alpha_0 + \sum_{i=j}^{p} \alpha_i r_{t-1}^2 + \sum_{i=j}^{q} \beta_i \sigma_{t-1}^2 \tag{2}$$

On el retorn del procés es defineix com a:

$$r_t = \sigma_t \epsilon_t \tag{3}$$

3.2 Anàlisi del risc d'una cartera

3.2.1 Introducció

En aquesta secció parlarem del marc teòric de l'anàlisi de la volatilitat d'una cartera d'accions en relacio amb un mercat en concret. Aquest marc teòric està basat en la Teoria Moderna de Carteres (*Modern portfolio theory*) desenvolupada per Harry Markowitz a partir del 1952 i recollida en forma de llibre a *Portfolio Selection: Efficient Diversification of Investments* [22] el 1959. Per aquesta teoria, entre altres, Markowitz va aconseguir el Premi Nobel d'Economia el 1990.

La Teoria Moderna de Carteres ens permet decidir on es volen invertir els diners segons dos principis fonamentals i excloents:

1. Es vol maximitzar el retorn esperat.
2. Es vol minimitzar el risc, que es definirà en aquesta secció com la desviació estàndard (volatilitat) del retorn.

Normalment es dóna per suposat que els actius amb més risc són els que obtenen millors retorns, ja que els inversors demanen una recompensa més alta per tal de compensar el risc de la inversió. Tot els inversors comparen aquest risc amb el retorn d'actius que es consideren sense risc, com per exemple els dipòsits bancaris. Així, per exemple, si un banc et paga un 2%

d'interés pels teus diners en un dipòsit, no té sentit invertir en renda variable en un valor que té un retorn esperat similar perquè suposa un risc molt més elevat pel mateix retorn.

Per altra banda, i aquesta és la base de l'aplicació, és possible trobar punts de compromís òptims entre el retorn esperat i el risc. Per exemple, és possible maximitzar el retorn esperat amb un límit de risc o bé minimitzar el risc segons un retorn màxim preestablert. La clau de tot això és ajustar la diversificació de la cartera per tal d'aconseguir els resultats que volem.

No és la intenció d'aquest treball d'introduir tots els conceptes relacionats amb aquesta teoria, sinó només els que s'utilitzaran en l'aplicació que volem construir. S'enumeraran tots els conceptes de manera exhaustiva, tot i que no dedicarem moltes línies a explicar els conceptes més trivials que es poden entendre fàcilment només llegint la seva fòrmula matemàtica.

Les fòrmules matemàtiques d'aquesta secció estan extretes del llibre de David Ruppert *Statistics and Data Analysis for Financial Engineering* [23] tot i que no seguim l'estructura del llibre per presentar-les, ja que es tracta d'un llibre d'un nivell molt més avançat que el que es requereix en aquest treball.

3.2.2 Fòrmules bàsiques de retorn i de risc

En aquesta secció enumerarem breument les fòrmules bàsiques generals que utilitzarem en els següents apartats. Se suposa al lector una maduresa matemàtica suficient com per entendre-les sense explicacions addicionals a les donades aquí.

- El retorn esperat d'una cartera:

$$E(R_p) = \sum_i w_i E(R_i) \tag{4}$$

  On R_p és el retorn de la cartera, R_i és el retorn de l'actiu i i w_i és la proporció de l'actiu i a la cartera.

- La variància del retorn esperat d'una cartera:

$$\sigma_p^2 = \sum_i w_i^2 \sigma_i^2 + \sum_i \sum_{j \neq i} w_i w_j \sigma_i \sigma_j \rho_{ij} \tag{5}$$

  On ρ_{ij} és el coeficient de correlació entre els retorns dels actius i i j. Normalment s'utilitza una versió simplificada d'aquesta fòrmula per fer els càlculs com veurem més endavant.

- La volatilitat del retorn esperat d'una cartera (desviació estàndard):

$$\sigma_p = \sqrt{\sigma_p^2} \tag{6}$$

Un punt importantíssim que cal assenyalar aquí és que la Teoria Moderna de Carteres utilitza la desviació estàndard com a mesura del risc de la cartera. Aquest punt serà vàlid si els retorns segueixen una distribució normal, com veurem que és el cas [23].

La implicació més important d'aquestes fòrmules és que un inversor pot reduir el risc d'una cartera pel simple fet de tenir actius que no estiguin correlacionats de manera positiva perfecta ($\rho_{ij} \neq 1$) i això s'aconsegueix mitjançant la diversificació de la cartera.

3.2.3 Combinació d'un actiu de risc i un actiu sense risc

De manera general si el retorn esperat del actiu de risc i l'actiu sense risc són respectivament μ_1 i μ_f amb la desviació estàndard del valor amb risc σ_1 llavors el valor de retorn de la cartera serà $w\mu_1 + (1-w)\mu_f$ i la desviació estàndard del retorn serà $|w|\sigma_1$.

L'exemple més simple que podem trobar és el de combinar un actiu de risc i un actiu sense risc, que és l'exemple clàssic d'una cartera d'un fons d'inversió.

Suposem que l'actiu de risc té un retorn esperat del 15% i una desviació estàndard de 0.25.

També suposem que l'actiu sense risc té un retorn esperat del 6%. Per definició, com que no té risc, la desviació estàndard en aquest cas és 0.

Suposem també que w és la inversió que es fa en l'actiu de risc i $1-w$ és la part de la inversió que es fa en l'actiu sense risc.

Així doncs, utilitzant les fòrmules de l'apartat anterior, el retorn esperat serà:

$$E(R_p) = w(0.15) + (1-w)(0.6) = 0.06 + 0.09w \tag{7}$$

La variància del retorn esperat:

$$\sigma_p^2 = w^2(0.25)^2 + (1-w)^2(0)^2 = w^2(0.25)^2 \tag{8}$$

I, finalment, la desviació estàndard del retorn esperat:

$$\sigma_p = 0.25w \tag{9}$$

Per tal de decidir la proporció w s'ha d'escollir o bé el retorn esperat: $E(R_p)$ o bé la quantitat de risc: σ_p.

3.2.4 Combinació de dos actius de risc

Les matemàtiques que hi ha darrera la combinació d'actius de risc es poden entendre fàcilment només quan hi ha dos elements. Quan hi ha més de dos elements, els càlculs es compliquen molt i normalment s'han de fer amb un ordinador.

Si tenim dos actius de risc amb retorns esperats R_1 i R_2 combinats amb proporció w i $1-w$ llavors el retorn de la cartera serà $R_p = wR_1 + (1-w)R_2$ i el valor esperat de retorn serà $E(R_p) = w\mu_1 + (1-w)\mu_2$.

Si suposem que la correlació entre els retorns és ρ_{ij} llavors la variància del retorn serà:

$$\sigma_R^2 = w^2\sigma_1^2 + (1-w)^2\sigma_2^2 + 2w(1-w)\rho_{12}\sigma_1\sigma_2 \tag{10}$$

Com a exemple, utilitzem una cartera amb dos actius de risc que tenen les característiques següents:

$\mu_1 = 0.14$, $\mu_2 = 0.08$, $\sigma_1 = 0.2$, $\sigma_2 = 0.15$, $rho_{12} = 0$.

El resultat serà el següent:

$$E(R_p) = 0.08 + 0.06w \tag{11}$$

$$\sigma_{R_p}^2 = (0.2)^2w^2 + (0.15)^2(1-w)^2 \tag{12}$$

A partir d'aquest moment podem utilitzar les tècniques de càlcul diferencial per tal d'obtenir de manera fàcil la cartera amb el risc més baix, que en aquest cas és quan $w = 0.36$. Per aquesta cartera el retorn seria $E(R_p) = 0.08 + 0.06(0.36) = 0.1016$ i la desviació estàndard $\sigma_{R_p} = \sqrt{(0.2)^2(0.36)^2 + (0.15)^2(0.64)^2} = 0.12$.

Si dibuixem la corba que hi ha en els punts de $(\sigma_R, E(R))$ quan w està entre 0 i 1 tindrem un resultat molt similar al següent:

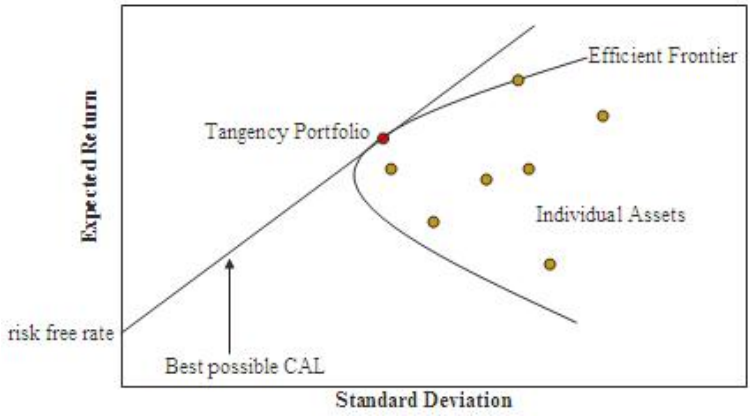

Figura 7: Frontera eficient amb els actius de l'exercici. *Font: Wikipedia*

3.2.5 Combinació de dos actius de risc amb un actiu sense risc

L'objectiu final és el de combinar N actius amb un actiu sense risc. El fet que hi hagi un actiu sense risc és important perquè marca el límit en el qual la inversió en un mercat de valors no té sentit (perquè podem obtenir el mateix retorn sense teòricament cap mena de risc). El cas més simple és quan tenim només una combinació de 2 actius amb risc amb un actiu sense risc.

Tal i com s'ha vist a la secció anterior, cada punt de la frontera eficient és el resultat de $(\sigma_R, E(R))$ per a un valor entre 0 i 1 i si fixem w tindrem una cartera en concret. Si es combina amb un actiu sense risc, el resultat serà gràficament una línia que connecta l'eix ordenades amb un punt en concret de la frontera eficient.

El pendent d'aquesta línia es coneix com a *ràtio de Sharpe* en honor a William Sharpe, el seu descobridor. Si suposem que $E(R_p)$ és el retorn esperat d'una cartera, σ_{R_p} és la desviació estàndard del retorn esperat i μ_f és el retorn de l'actiu sense risc, llavors podem calcular el ràtio de Sharpe utilitzant la formula següent:

$$\frac{E(R_p) - \mu_f}{\sigma_{R_p}} \tag{13}$$

Un pendent de línia més pronunciat ens dóna un millor retorn esperat per a un particular nivell de retorn, per tant, com més gran sigui el valor del ràtio

de Sharpe millor, independentment del nivell de risc. El punt en el qual la línia del ràtio de Sharp toca de manera tangencial la frontera eficient, s'anomena la *cartera tangencial*.

Un cop tenim la cartera tangencial trobada, ja es pot crear una cartera eficient. Entem com a cartera eficient la combinació de la cartera tangencial amb l'actiu sense risc. Es tracta en aquest cas doncs de decidir quina proporció dels diners es posen a la cartera tangencial i quina va a parar a l'actiu sense risc.

Cada cartera eficient té les següents dues propietats:

1. Té el retorn esperat més gran que qualsevol altra cartera del mateix risc.
2. Té el risc més petit que qualsevol altra cartera amb el mateix retorn esperat

Si suposem una altra vegada que tenim dos actius de risc amb els retorns esperats μ_1 i μ_2 i un actiu sense risc amb retorn esperat μ_f, que tenen com a desviacions estàndard del retorn esperat σ_1 i σ_2 (en el cas de l'actiu sense risc la desviació estàndard és obviament 0) i amb la correlació dels retorn de risc essent ρ_{12}.

Definim també $V_1 = \mu_1 - \mu_f$ i $V_2 = \mu_2 - \mu_f$ com l'excés dels retorn esperats.

Ara per tal de trobar la cartera tangencial podem aplicar la següent fòrmula:

$$w_t = \frac{V_1\sigma_2^2 - V_2\rho_{12}\sigma_1\sigma_2}{V_1\sigma_2^2 + V_2\sigma_1^2 - (V_1 + V_2)\rho_{12}\sigma_1\sigma_2} \tag{14}$$

Un cop hem trobat w_t utilitzant la fòrmula anterior, podem obtenir fàcilment el retorn esperat de la cartera tangencial i la seva desviació estàndard amb aquestes fòrmules:

$$E_{R_t} = w_t\mu_1 + (1 - w)w_t\mu_2 \tag{15}$$

$$\sigma_t = \sqrt{w_t^2\sigma_1^2 + (1 - w_t)^2\sigma_2^2 + 2w_t(1 - w_t)\rho_{12}\sigma_1\sigma_2} \tag{16}$$

Ara podem utilitzar aquestes fòrmules amb l'exemple anterior per demostrar el seu funcionament. Els valors de l'exemple eren els següents: $\mu_1 = 0.14$,

$\mu_2 = 0.08$, $\sigma_1 = 0.2$, $\sigma_2 = 0.15$, $rho_{12} = 0$. Si suposem que $\mu_f = 0.06$ llavors podem calcular $V_1 = 0.14 - 0.06 = 0.08$ i també $V_2 = 0.08 - 0.06 = 0.02$.

$$w_t = \frac{(0.08)(0.15)^2 - (0.02)(0)(0.2)(0.15)}{(0.08)(0.15)^2 + (0.02)(0.2)^2 - (0.08 + 0.02)(0)(0.2)(0.15)} = 0.693 \tag{17}$$

$$E_{R_t} = (0.693)(0.14) + (0.307)(0.08) = 0.122 \tag{18}$$

$$\sigma_t = \sqrt{(0.693)^2(0.2)^2 + (0.307)^2(0.15)^2} = 0.146 \tag{19}$$

Si R_p és el retorn esperat de la cartera que posa la fracció w de la inversió a la cartera tangent i el $1 - w$ a l'actiu sense risc. Llavors:

$$E_{R_p} = \mu_f + w(E_(R_t - \mu_f)) \tag{20}$$

$$\sigma_{R_p} = w\sigma_t \tag{21}$$

3.2.6 Carteres eficients amb N actius

El què hem vist a l'apartat anterior es pot aplicar a les carteres que tenen un número arbitrari d'actius. Per fer-ho s'han d'aplicar les tècniques de la programació quadràtica (l'exemple més conegut és segurament el component Solver del Microsoft Excel) i el càcul de matrius. Aquest és el procediment que seguirà el nostre programa.

Si tenim N actius de risc i el retorn de l'actiu del risc i és R_i i el valor esperat és μ_i podem construir un vector de retorns de la manera següent:

$$E(R) = \mu = \begin{pmatrix} \mu_1 \\ .. \\ .. \\ \mu_N \end{pmatrix} \tag{22}$$

I anomenarem Σ la matriu de covariàncies de R.

Llavors si definim w com el vector del pesos de la cartera:

$$w = \begin{pmatrix} 1 \\ .. \\ .. \\ N \end{pmatrix}$$

(23)

de manera que $w_1 + ... + w_N = 1^T w = 1$, on 1 es un vector columa de N 1 especificat de la manera següent:

$$1 = \begin{pmatrix} 1 \\ .. \\ .. \\ 1 \end{pmatrix}$$

(24)

El retorn esperat de la cartera el podrem calcular amb la fòrmula següent:

$$\sum_{i=1}^{N} w_1 \mu_i = w^T \mu \tag{25}$$

Si volem un valor esperat de retorn de la cartera μ_P, quan $N = 2$ el valor esperat només serà possible amb una cartera determinada que serà la solució de l'equació $\mu_P = w_1\mu_1 + w_2\mu_2 = \mu_2 + w_1(\mu_1 - \mu_2)$ resolta per a $w_1 - valor$. En el cas que N sigui igual o més gran que 3, llavors hi ha un número infinit de carteres que donen com a resultat μ_P. De totes elles, la cartera que tingui una variància més petita és la que anomenarem *cartera eficient*.

La variància del retorn d'una cartera amb pesos w és la que ens dona la fórmula següent:

$$w^T \Sigma \mu \tag{26}$$

Per tant, donat un objectiu μ_P la cartera eficient es minimitza l'equació anterior amb les dues restriccions següents:

$$w^T \mu = \mu_P \tag{27}$$

$$w^T 1 = 1 \tag{28}$$

La *programació quadràtica* és la tècnica que s'utilitza per a minimitzar aquesta funció que està subjecta a les restriccions lineals que hem vist. Veurem en detall com aplicar aquesta tècnica més endavant en aquest treball.

4 La Màquina de Vector de Suport

En aquest apartat veurem breument què són les màquines de vector de suport o, en anglès, *Support Vector Machines* que utilitzarem en el servidor de la nostra aplicació. Es tracta d'una breu introducció a nivell teòric amb la informació que ens serà útil després per tal d'implementar el nostre programa a posteriori.

4.1 Introducció

Una màquina de vector de suport és una eina conceptual formada per un conjunt d'algorismes que son capaços de analitzar dades i reconèixer patrons mitjançant l'ús de mètodes d'aprenentatge supervisat. La màquina agafa com a entrada un set de dades i prediu, per a cadascuna d'aquestes entrades, a quina de les possibles classes pertany.

Una màquina de vector de suport estableix un model que separa les dues classes entrants mitjançant l'entrenament (on agafa dades que ja estan classificades). Aquest model estableix una frontera entre els tipus en el punt en què la diferència entre les classes sigui el més gran possible i el marge d'error sigui el més petit possible.

El nom de la màquina ve dels vectors de suport, que són els punts que conformen les dues línies paral·leles al hiperplà on la distància (o marge) és el més gran possible. Les màquines de vectors de suport van ser inventades per Vladimir Vapnik.

4.2 Classificació

La màquina de vector de suport és coneix també com el classificador del marge màxim i la idea prové de la intuició que de tots els límits de decisió possibles utilitzats per a les classificacions entre classes, el millor és que té els marges més grans. El marge es defineix com el doble de la distància mínima entre el límit de decisió i qualsevol de les instàncies de les dades d'entrenament.

Pel fet de maximitar el marge és possible obtenir una millor generalització i és possible provar utilitzant la dimensió VC, introduïda pel mateix Vapnik i Chervonenkis, de tal manera que són les sigles dels seus noms respectius, que redueix de manera substancial la complexitat del model.

Podem veure el concepte de marge de manera gràfica en la figura següent:

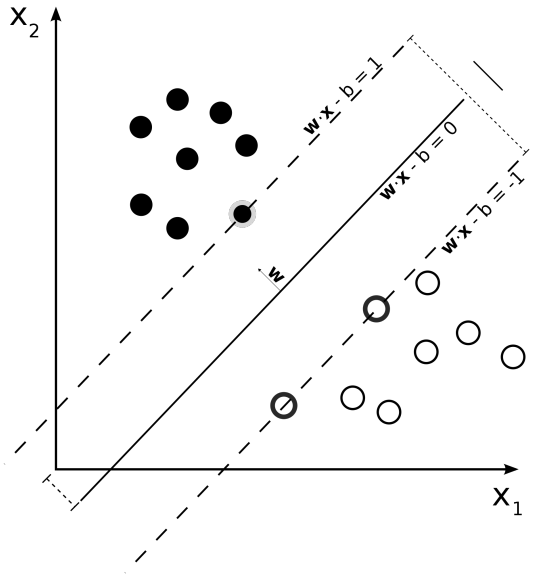

Figura 8: Representació del concepte de marge de les SVM

Hi ha dos tipus diferents de màquines de vector de suport, la original introduïda per Vapnik és coneix com la versió de *marge fort*. El marge fort vol dir que els marges que estan definits a la màquina de vector de suport no es poden violar i els marges seran maximitzats sota aquestes condicions estrictes.

Una versió més nova de les SVM va ser introduïda per Corinna Cortes i es coneix amb el nom de màquina de vector de suport de *marge suau*. En aquest cas, la condició que impedeix que es violin els marges s'ha eliminat i les violacions es penalitzen en la funció de cost. El grau de penalització es controla amb el $\mathcal{C}$ *paràmetre de marge suau* del model. I que ve representat de la manera següent:

$$\min_{\theta,\ b} \ \frac{1}{2}||\theta||^2 \qquad s.t.\ y_i(\theta^T x_i + b) \geq 1 \tag{29}$$

On θ són els pesos de la SVM, x_i i y_i són les característiques i la classe del element i del conjunt de dades (i la condició s'aplica a tots els elements que hi ha en ell). Aquesta forma es coneix com la forma primitiva de les màquines de vector de suport del cas de marge fort. Si bé, utilitzant els multiplicadors de Lagrange es pot obtenir una altra forma que és millor per poder fer optimitzacions.

En canvi, si es considera el cas del marge tou, s'afegeix ξ_i que representa la violació del marge de cada element i llavors la suma de tots els elements

es minimitza, ponderat pel paràmetre de marge tou $\mathcal{C}$ que s'ha mencionat abans. Tenint en compte aquestes consideracions, la forma primitiva del marge tou és la següent:

$$\min_{\theta,\ b,\ \xi} \ \frac{1}{2}||\theta||^2 + \mathcal{C}\sum_{i=1}^{m} \xi_i s.t.\ y_i(\theta^T x_i + b) \geq 1 - \xi_i,\ \xi_i \geq 0 \tag{30}$$

El límit de decisió que s'obté és el que maximitza la distància de les instàncies de cada classe. Això, al mateix temps, minimitza la distància del conjunt d'instàncies que estan properes al marge. Eliminant una d'aquestes instàncies, permet que el marge es pugui moure, per tal de poder crear una distància més gran amb la resta de punts propers. Però, el què encara és més important, tot eliminant una instància que és més lluny, no afecta el marge.

Això fa que el límit de decisió depengui només d'un conjunt en concret d'instàncies. Aquestes instàncies s'anomenen els *vectors de suport*. Aquesta dependència a un part del conjunt de dades té una relació directa al fet que sigui un model amb una complexitat més baixa.

4.3 Tipus de funcions de nucli

La màquina de vector de suport que hem definit crea un classificador òptim basat en concepte de marge màxim, però sempre com a classificador lineal. Tot utilitzant funcions de nucli (*kernels*) es produeixen transformacions en les dades de manera que la classificació linel es produeix en l'espai transformat mentre que en l'espai original hi ha un classificador lineal.

Hi ha diferents tipus de funcions de nucli que serveixen per produir diferents transformacions, però hi ha una característica que s'ha d'explicar abans d'explicar-les. Aquesta característica és que quan es transformen les dades, el resultat pot ser un conjunt de dades en un altre espai, on la seves dimensions són les noves característiques que es calculen.

Algunes de les funcions del nucli produeixen transformacions que poden produir dimensions molt gran o, fins i tot, infinites. Però es pot demostrar que quan s'utilitzen funcions de nucli, les transformacions sempre són part d'un producte escalar entre elles. Per tant, la transformació no es produeix mai de manera explícita sinó que el resultat del producte escalar es calcula directament. El resultat d'aquest producte escalar no depen de la dimensionalitat de l'espai on es produeix la transformació, sinó en el número d'elements del conjunt de dades.

Els tipus de funcions de nucli d'una màquina de vector de suport són:

Lineal Aquest tipus de funció de nucli és equivalent al cas on no s'utilitza cap funció de transformació. Però, en el moment en què es defineix, el model de la màquina de vector de suport es converteix en un cas especial. En aquest cas no es produeix cap transformació i les dades es discriminen en la seva dimensió original. Essent un tipus de funció lineal, en el cas que dues classes tinguin la mateixa matriu de covariàncies, una SVM amb una funció de nucli lineal, hauria de ser suficienment complex com per discriminar les dades de manera òptima.

Quadràtic Aquest tipus és un cas especial del cas polinòmic quan el grau del polinomi és 2. Ja que el model quadràtic és el de complexitat màxima a l'hora de discriminar dues classes amb distribució normal, no es considerarà aquí cap altre cas. La complexitat és suficient per a classifcar les classes generades per una distribució normal en el cas de covariàncies de matrius diferents, com és el cas dels retorns esperats de les carteres.

Aquest tipus quadràtic de funció de nucli es defineix de la manera següent:

$$k(x_i, x_j) = \left(x_i^T x_j + 1\right)^2 \tag{31}$$

Funció de base radial (RBF) Aquest tipus també s'anomena la funció guassiana i fa que la SVM classifiqui els elements en un espai dimensional infinit. La representació d'aquest espai original fa que cada instància del conjunt de dades creïi una distribució normal amb la mitjana en la seva posició i una variància especificada pel paràmetre sigma (o gamma). Les distribucions normals estan en direccions oposades en cas que les classes siguin diferents. Es pot pensar que aquest mètode el què fa és empènyer l'element perquè es classifiqui cap a la seva direcció. Aquest efecte en l'espai el produeixen els vectors de suport.

El tipus de funció de nucli RBF es defineix de la següent manera:

$$k(x_i, x_j) = e^{\left(-\frac{1}{2}\frac{||x_i - x_j||^2}{\sigma^2}\right)} \tag{32}$$

5 Descripció de la plataforma

En aquesta secció es farà una descripció detallada de la plataforma a nivell formal. Aquesta secció es divideix en dos gran apartats que parlaran de la part de client (implementada en Java i que correrà sobre un telefòn intel·ligent Android) i una part de servidor (que serà implementada en Python, tot utilitzant el framework Django). Veurem més detalls de tot això més endavant en aquesta mateixa secció.

Tot i que l'usuari percebrà que es tracta d'una sola aplicació, la realitat és que es tracta de dues aplicacions molt diferents que no té sentit analitzar conjuntament. De fet, el cor de l'aplicació, i la part que implementa tota la funcionalitat interessant, és el servidor. El client Android només s'encarrega d'enviar les dades i mostrar els resultats. Es podria utilitzar qualsevol client que implementi el protocol de comunicació (en aquest cas, una API REST) per tal d'enviar i rebre les dades del servidor com podria ser una pàgina web fent crides AJAX o un programa d'escriptori de Windows escrit en C++ o en qualsevol dels llenguatges suportats a la plataforma .NET. Això ens permet que, per exemple, si vulguéssim crear una aplicació per a iPhone, només s'hauria de tornar a implementar la part del client (segurament en Objective-C) i fer les crides pertinents al servidor de Python a través de la seva API REST.

5.1 El client Android

en aquest apartat es veurà el disseny de l'aplicació de la part del dispostiu Android. es farà, en primer lloc, una enumeració de les funcionalitats i la seva especificació sense oblidar, evidentment, l'arquitectura del sistema, que és la part més important.

és important tenir en compte que el client Android també està organtizat internament en una estructura de tipus client-servidor, per tal que les operacions de xarxa es facin de la manera més eficient possible (i no es quedi bloquejada l'aplicació mentre s'espera una resposta del servidor). aquest comportament és normal en totes les aplicacions android que realitzen activitats semblants a les que fa la nostra aplicació.

per tant, cal tenir en compte que en aquesta secció quan estem parlant del client i del servei, ens estem referint a la implementació només de la part Android i no a la implementació del servidor django, que s'explicarà en una secció a part. quan em refereixi a aquest servidor web en aquesta secció, l'anomenaré de manera explícita "servidor REST".

5.1.1 Descripció de les funcionalitats

En aquest punt farem una descripció funcional de l'aplicació des del punt de vista de l'usuari, sense entrar en detalls tècnics sobre la mateixa. Aquesta secció servirà, doncs, d'introducció a la resta de seccions que aniran aprofundint en cada una de les funcionalitats de manera detallada.

1. En primer lloc l'aplicació et permet crear una cartera de valors de manera totalment automàtica tot assignant els pesos a cada un dels valors. Per tal de fer això, l'usuari només ha de seleccionar quina és la volatilitat màxima (que nosaltres identifiquem amb el risc) que vol asumir. L'aplicació farà una petició al servidor que retornarà els resultats segons les especificacions.

2. En un nivell una mica més avançat, l'aplicació disposa de l'opció de completar una cartera segons uns criteris determinats però amb certes condicions. El cas més habitual és quan l'usuari escull un o més valors i els hi assigna un pes fix. L'aplicació llavors el què farà serà completar automàticament la resta de valors per tal d'aconseguir una cartera òptima però sempre complint amb les restriccions imposades per l'usuari.

3. L'aplicació també et permet guardar i recuperar una cartera de la base de dades. La cartera es guardarà en la base de dades SQLite que incorpora el programa. També es possible eliminar de la base de dades les carteres que ja no ens interessen.

4. El servei que s'executa en segon pla i que és l'encarregat de realitzar la conexió amb el servidor REST també s'encarrega de baixar-se i, si s'escau, actualitzar la llista de valors disponibles de la base de dades que es manté en local, d'aquesta manera els tickers dels valors estan sempre sincronitzats sense que l'usuari hagi d'actualitzar l'aplicació si canvia algun valor (els tickers poden canviar o fins i tot les empreses que hi ha al darrere poden arribar a desaparèixer)

Un aspecte especialment destacable de la nostra aplicació serà les seves grans possibilitats de configuració i personalització. Si bé, hi haurà uns valors per defecte raonables i que haurien de ser vàlids per a la majoria dels usuaris, els paràmetres es podran ajustar segons les necessitats de cadascú. Serà possible, fins i tot, crear perfils que no seran més que una agrupació de paràmetres de diferents categories. Aquests perfils podrien ser, per exemple: "maximitzar retorn esperat" o bé "minimitzar risc" amb diferents valors intermedis adaptats a les necessitats de cadascú.

5.1.2 Requeriments

El programa està dividit entre un client i un servei, tot seguint l'arquitectura més adequada a les necessitats de l'aplicació i, al mateix temps a les restriccions que ofereix la plataforma i l'SDK de l'Android, ja que estem treballant a un nivell més baix que la majoria de les altres aplicacions d'usuari i, per tant, hem de seguir les guies i directrius que ens proporciona la documentació del sistema Android per a casos com aquest.

El client és l'encarregat de controlar la interfície d'usuari, gestionar i reaccionar als events (com pot ser que l'usuari fa clic a un botó) i mostrar les dades a la pantalla, tot això s'ha de fer sense que l'aplicació es quedi bloquejada i respongui en tot moment. L'Android té un sistema automàtic que s'encarrega de vigilar que les aplicacions no estiguin bloquejant el sistema si no responen en un temps predetermitat i, si és el cas, mostrar un missatge a la pantalla que convida a l'usuari a eliminar el procés.

Com que la nostra aplicació s'ha de comunicar amb un servidor i no podem controlar el temps que pot trigar, una crida directa des del fil d'execució de la interfície d'usuari és una molt mala idea, és per això que disposem d'un servei que és l'encarregat de realitzar les peticions al servidor i gestionar la obtenció i l'enviament d'informació de manera asíncrona per evitar el bloqueig de l'aplicació. En les aplicacions per a telèfons intel·ligents fer que la teva aplicació respongui als esdeveniments tota l'estona i no es quedi bloquejada és molt important i, com podem veure, el sistema Android disposa d'un mecanisme per obligar a les aplicacions que es comportin de la manera esperada.

Les premisses generals de funcionament de l'aplicació són les següents:

- Control sobre les accions del servei.
- Control de seguretat per permetre que només el client pugui accedir al servei i a les seves notificacions.
- Control d'accés a la base de dades tant del client com del servei.

Tal i com ja s'ha comentat en la descripció de les funcionalitats, el servei ha de ser el més lleuger possible perquè estarà funcionant en el dispositiu en tot moment, mentre que el client serà l'encarregat de realitzar totes les tasques importants a més a més de proporcionar la interfície d'usuari perquè els actors del sistema es puguin relacionar amb l'aplicació. Per aquest motiu s'han dividit els requeriments en aquests dos programes i seran tractats per separat:

Requeriments del client

- Autenticació: el client s'ha d'autenticar al servei per tal de poder-hi accedir.
- El client també ha de poder accedir a la base de dades on tant el servei i el client guarden informació de configuració sense que hi hagi conflictes entre tots dos programes accedint a les mateixes dades.

Requeriments del servei

- Autenticació: qualsevol client s'ha d'autenticar mitjançant un sistema de sessions que controli l'autorització i el nivell d'accés a les dades.
- La comunicació amb el client es fa a través de d'alertes i missatges del sistema.
- El servei llegeix i escriu valors en una base de dades que externament només han de ser accessibles al client identificat.

5.1.3 Especificació

L'especificació es farà en UML amb la creació d'actors i de casos d'ús que veurem a continuació. El cas dels actors es podria resoldre amb un sol actor, que és l'usuari que instal·la i fa servir l'aplicació en el seu telèfon intel·ligent. Però és també interessant dividir les tasques entre un usuari normal i un usuari més avançat ja que són realment perfils d'usuaris diferents tot i que una mateixa persona pot intercanviar-se els rols segons les seves necessitats en moments determinats.

Actors. A l'hora de definir els actors, s'ha especialitzat l'usuari en dos rols diferents. No es tracta d'un problema de permisos, ja que els dos tipus d'usuaris poden fer de tot, sinó que ens serveix per fer una classificació de les tasques que poden fer els usuaris segons el seu nivell de coneixements o necessitats d'ús en un moment determinat.

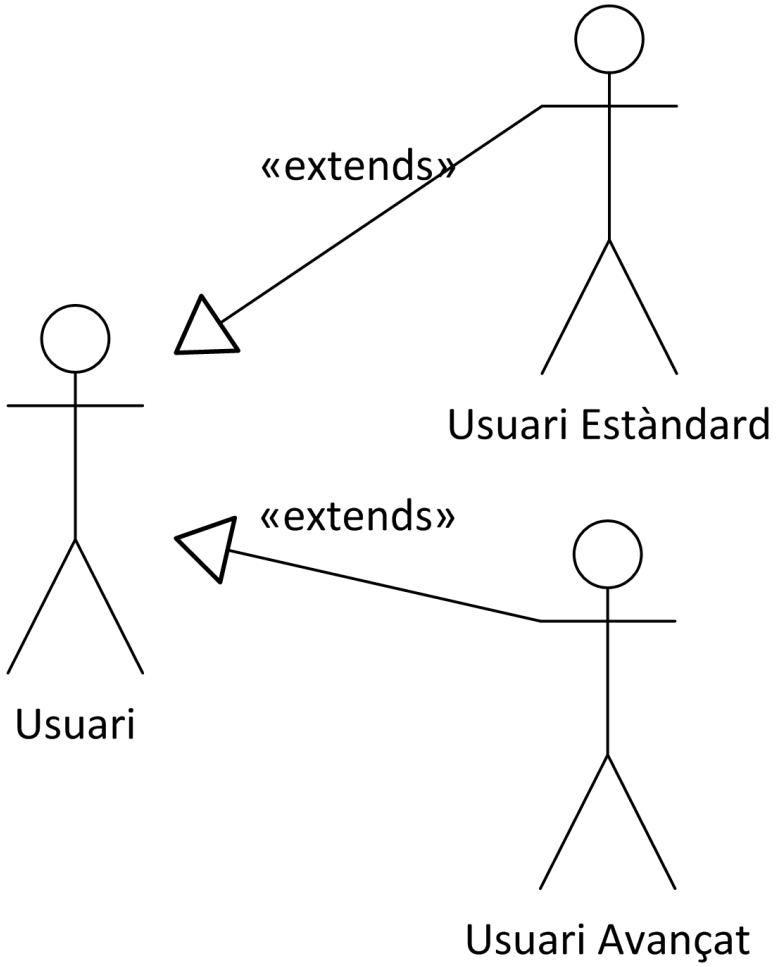

Figura 9: Diagrama dels actors del dispositiu Android. *Font: elaboració pròpia*

Les tasques que faria conceptualment cada un d'aquests usuaris serien les següents:

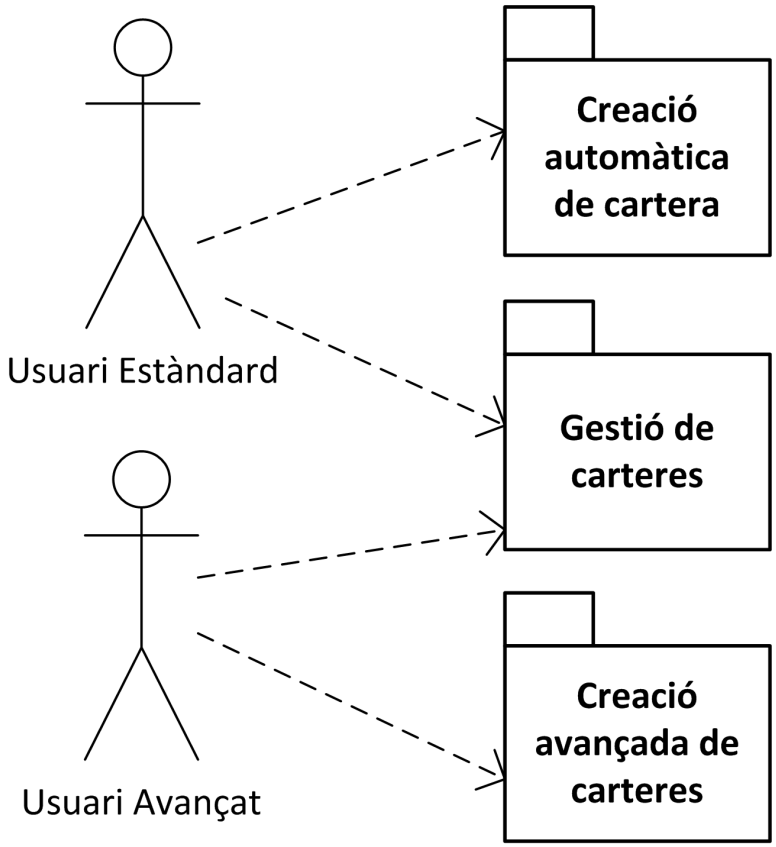

Figura 10: Diagrama de classificació dels casos d'ús del dispositiu Android. *Font: elaboració pròpia*

Diagrama de casos d'ús. En els següents diagrames representarem els casos d'ús de la nostra aplicació:

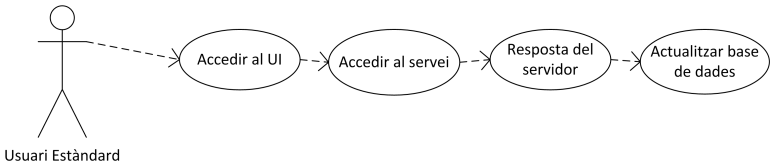

Figura 11: Diagrama de creació automàtica de cartera. *Font: elaboració pròpia*

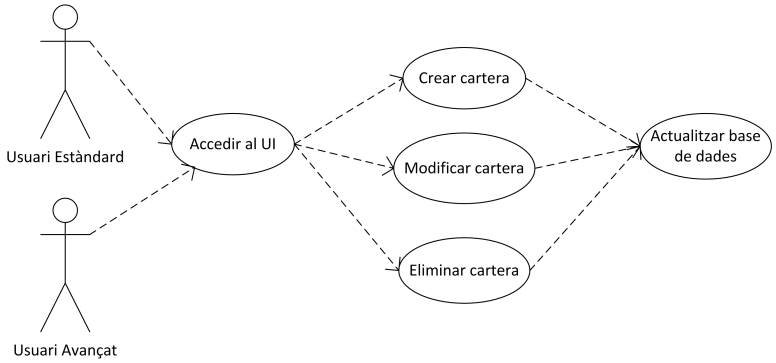

Figura 12: Diagrama de gestió de carteres. *Font: elaboració pròpia*

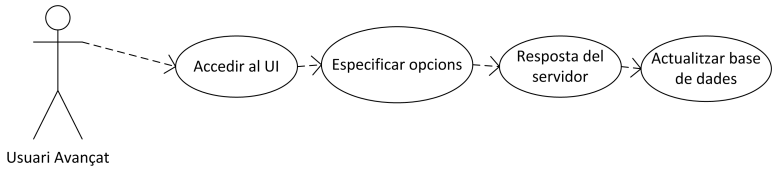

Figura 13: Diagrama de creació avançada d'una cartera. *Font: elaboració pròpia*

Especificació dels casos d'ús. En aquest apartat especificarem els casos d'ús de l'aplicació dividits entre els dos tipus d'usuaris que hem mencionat en els apartats anteriors.

Accedir al UI

Resum de la funcionalitat:	Inici de l'accés al sistema.
Actors:	Usuari estàndard, Usuari avançat.
Casos d'ús relacionats:	Accedir a la base de dades, Accedir al servei.
Precondició:	L'aplicació ha d'estar instal·lada.
Resum:	Inicia l'interacció de l'usuari amb l'aplicació.

Accedir al servei

Resum de la funcionalitat:	Accedir al servei que es connecta al servidor REST remot. Per tal de demanar una cartera optimitzada o bé o bé obtenir valors dels tickers per a la base de dades.
Actors:	Usuari estàndard, Usuari avançat.
Casos d'ús relacionats:	Resposta del servidor.
Precondició:	L'usuari ha de fer una petició que requereixi accés al servidor remot.
Resum:	Comunicació entre el client Android i el servidor remot.

Resposta del servidor

Resum de la funcionalitat:	El servei rep la resposta del servidor.
Actors:	Usuari estàndard, Usuari avançat.
Casos d'ús relacionats:	Accedir a la base de dades, Accedir al servei.
Precondició:	L'aplicació ha d'haver iniciat una petició al servidor i aquest ha d'haver respost.
Resum:	La resposta del servidor és processada i enviada a la base de dades interna i/o mostrada al usuari.

Actualitzar la base de dades

Resum de la funcionalitat:	Actualització de la base de dades.
Actors:	Usuari estàndard, Usuari avançat.
Casos d'ús relacionats:	Resposta del servidor, Crear cartera, Modificar Cartera, Eliminar Cartera.
Precondició:	S'inicia el procés de modificació de les dades locals en qualsevol dels casos d'ús especificats.
Resum:	S'actualitza la base de dades amb la nova informació.

Especificar paràmetres

Resum de la funcionalitat:	Especificar paràmetres com la variància màxima o el retorn esperat en una cartera.
Actors:	Usuari avançat.
Casos d'ús relacionats:	Crear Cartera, Resposta del servidor.
Precondició:	L'usuari està en el procés de crear una cartera.
Resum:	L'usuari avançat vol crear una cartera nova amb uns paràmetres predeterminats o amb restriccions.

Crear cartera

Resum de la funcionalitat:	Creació d'una cartera nova a petició de l'usuari
Actors:	Usuari estàndard, Usuari avançat.
Casos d'ús relacionats:	Accedir al UI, Accedir al servei. Accedir a la base de dades
Precondició:	L'usuari ha d'haver accedit al UI.
Resum:	Creació d'una cartera de valors nova a la base de dades.

Modificar cartera

Resum de la funcionalitat:	Modificació d'una cartera existent a petició de l'usuari
Actors:	Usuari estàndard, Usuari avançat.
Casos d'ús relacionats:	Accedir al UI, Accedir al servei. Accedir a la base de dades
Precondició:	L'usuari ha d'haver creat una cartera.
Resum:	Modificació d'una cartera de valors de la base de dades.

Eliminar cartera

Resum de la funcionalitat:	Eliminació d'una cartera existent a petició de l'usuari
Actors:	Usuari estàndard, Usuari avançat.
Casos d'ús relacionats:	Accedir al UI, Accedir al servei. Accedir a la base de dades
Precondició:	L'usuari ha d'haver creat una cartera.
Resum:	Eliminació d'una cartera de valors nova a la base de dades.

5.1.4 Arquitectura

L'aplicació utilitza, a nivell general, dos components perfectament diferenciats: un servei i un client. Aquesta decisió està clarament influenciada també per les possibilitats i les restriccions de la plataforma Android a l'hora de realitzar aplicacions d'aquesta mena.

Servei

En primer lloc, el servei s'executa en segon pla tota l'estona i s'encarrega de monitoritzar el fil d'interfície d'usuari a l'espera de notificacions. Un requeriment d'aquest servei és que, per les seves característiques, ha de consumir el menor nombre de recursos possible. Això implica que el servei ha de realitzar una funció molt bàsica i delegar la resta de funcionalitats al client,

que només s'executa quan l'usuari ho demana directament o bé quan rep una notificació del servei. El servei, doncs, no té interfície d'usuari ni cap element que pugui interactuar amb l'usuari. El servei funciona en segon pla en tot moment i l'usuari no té cap control sobre ell.

El servei és en realitat un bucle que s'encarrega d'esperar ordres d'enviar informació al servidor REST o de notificar al fil d'interfície d'usuari que ja hi ha dades disponibles. Aquestes dades es poden enviar en temps real al client, si aquest s'està executant. Per fer-ho, i utilitzant la terminologia pròpia de l'Android, el client s'ha de subscriure com a Listener. També, en el cas que es produeixin unes condicions determinades prèviament configurades (per exemple, avís que s'han rebut noves dades processades), el servei envia una notificació d'aquest canvi d'estat al client.

Totes les dades sobre les quals treballa l'aplicació es llegeixen d'una base de dades i es carreguen en memòria. El servei ha de rebre una crida especial del client que li indica que ha de rellegir aquestes dades. Si aquesta crida no es produeix, el servei continua amb les dades anteriors ja que no les actualitza de manera automàtica. El servei només realitza algunes funcions determinades quan el client li demana explícitament, per estalviar recursos del dispositiu i contribuir el menys possible a l'exhauriment de la bateria.

Client

El segon component de l'arquitectura és el client. El client té la doble funció de ser la interfície gràfica sobre la que actua l'usuari (tot mostrant dades i permetent la configuració de certs paràmetres) i també, al mateix temps, el responsable de dur a terme les accions corresponents quan el servei li comunica algun canvi d'estat, per exemple que ha rebut noves dades processades pel servidor REST.

Base de dades

Un tercer component del sistema és la base de dades. Tant el client i el servei hi accedeixen per tal de llegir o escriure les dades configuració. Android té suport per a bases de dades SQLite i aquesta serà la que farem servir per a la nostra aplicació. A la base de dades hi guardarem informació sobre els valors del mercat (que es poden actualitzar remotament) i les carteres generades a partir de la informació obtinguda del nostre servidor REST.

Diagrames de seqüència

En aquest apartat especificarem alguns diagrames de seqüència de les funcionalitats del client Android que hem detectat en els apartats anteriors, per tal d'il·lustrar encara amb més detall el funcionament de l'aplicació.

Accedir al UI

En aquest primer diagrama podem veure com es produeix l'accés al UI. Tot i que l'usuari no s'ha d'identificar en el UI per tal d'accedir al servei, es produeix una identificació interna a través del sistema que ens proporciona la plataforma Android per defecte per tal d'assegurar que només el client es pot connectar en el servidor.

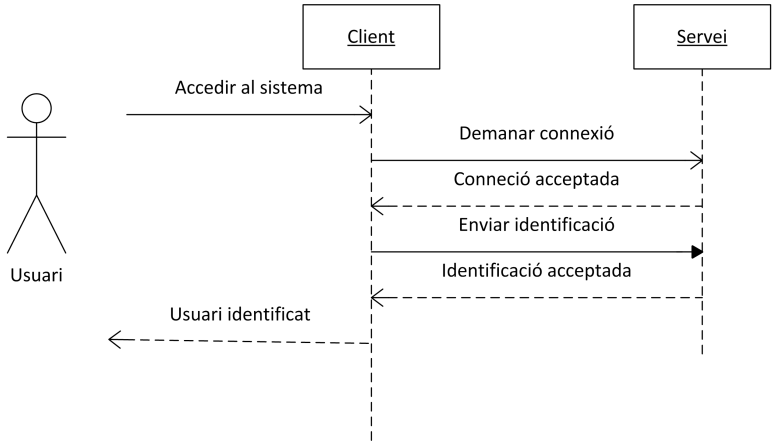

Figura 14: Diagrama de seqüències d'accés al UI. *Font: elaboració pròpia*

Creació automàtica de cartera

En aquest cas es pot observar la seqüència per tal de crear una cartera de manera totalment automàtica. En primer lloc l'usuari fa la petició per demanar la creació de la cartera a l'aplicatiu i es crea una cartera buida a la base de dades amb un identificador concret.

Aquest identificador es torna al client que fa una petició al servei per obtenir una cartera optimitzada nova. Com que és una creació totalment automàtica d'una cartera, no s'introdueix cap mena de restriccions en la crida. El servei s'encarrega de fer la petició de manera asíncrona i rebre el resultat, que s'envia un altre cop al client.

Un cop tenim el resultat, el client envia les dades noves de la cartera a la base de dades utilitzant l'identificador que ja tenia guardat prèviament. Una vegada les dades estan introduïdes a la base de dades, es pot notificar a l'usuari que la cartera està llesta i es pot visualitzar.

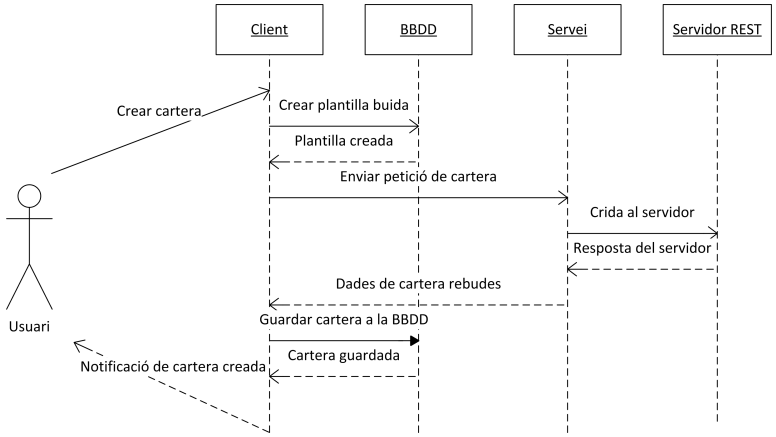

Figura 15: Diagrama de seqüències de creació automàtica d'una cartera. *Font: elaboració pròpia*

Eliminar una cartera

Com a exemple de la gestió de les carteres que es produeix en la part del client Android, es mostrarà el cas d'eliminar una cartera existent de la base de dades. El mecanisme per realitzar les altres operacions és molt similar i, per tant, ens centrarem en aquest cas en concret.

En el diagrama podem veure que la gestió de carteres és molt simple en la nostra aplicació i simplement el fil de interfície d'usuari fa una crida amb l'identificador de la cartera que es vol eliminar i el sistema de control de la base de dades s'encarrega de la resta i de notificar si el procés s'ha executat correctament.

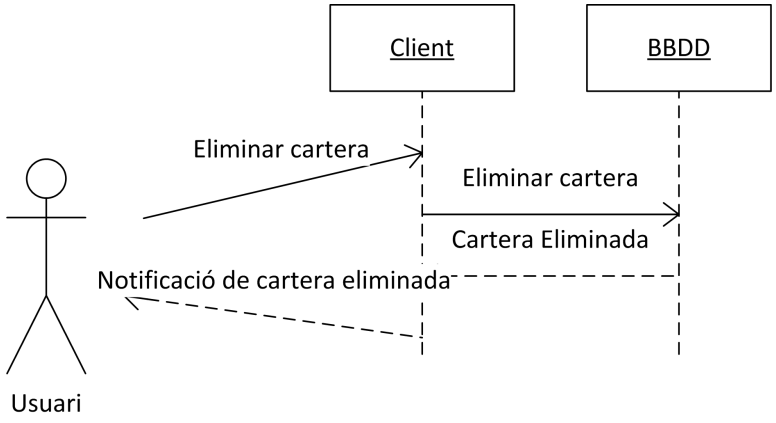

Figura 16: Diagrama de seqüències d'eliminar una cartera. *Font: elaboració pròpia*

Actualització dels tickers

Un últim diagrama de seqüències que pot resultar interessant és el de l'actualització automàtica dels valors disponibles en el dispositiu Android. Aquesta actualització és automàtica i l'usuari simplement pot especificar cada quan es comprova si hi ha nous tickers en el servidor REST.

En el diagrama podem veure que el servei és l'encarregat d'anar al servidor REST i objtenir els tickers actualitzats per posar-los a la base de dades. L'usuari, des del client, simplement pot modificar el valor que indica cada quant de temps el servei ha de preguntar al servidor si hi ha tickers actualitzats.

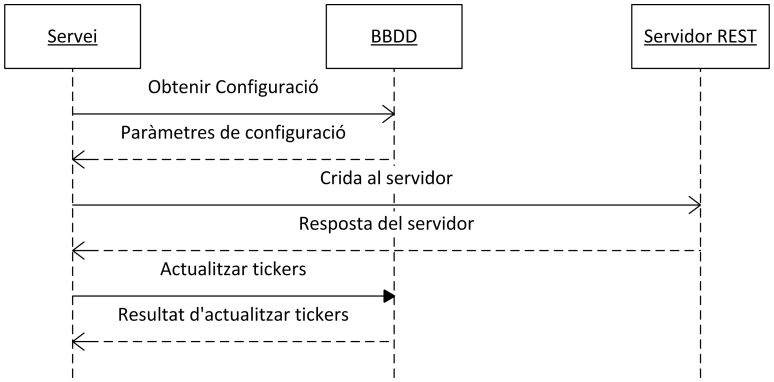

Figura 17: Diagrama de seqüències d'actualització dels tickers. *Font: elaboració pròpia*

Classes

La part del client Android està implementada en Java tot utilitzant el paradigma de la programació orientada a objectes. En aquest apartat veurem quines són les classes que formen el projecte, com estan relacionades entre elles i també una breu descripció funcional dels seus mètodes. Això permetrà fer-se una idea molt clara de quin és el funcionament de l'aplicació i com es relacionen les classes entre elles per tal de construir el programa i de quina manera s'han implementat.

Els noms de les classes totes comencen per Petrel que és el nom del programa. En el framework d'Android és interessant fer-ho d'aquesta manera perquè has de fer subclasses tota l'estona de classes ja existents i necessites posar-hi un nom únic. És per això que utilitzar el nom del programa en totes les classes té molt de sentit.

En primer lloc, es veurà una descripció a molt alt nivell de les classes de l'aplicació, i les seves relacions, representades en el diagrama següent:

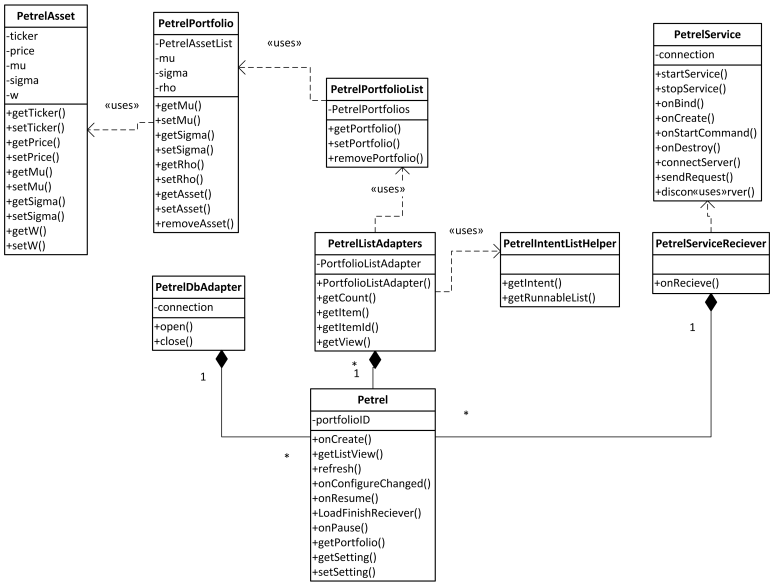

Figura 18: Diagrama de classes del client Android. *Font: elaboració pròpia*

Un cop s'han vist les classes que formen l'aplicació, es veurà quins són els membres que implementen cada una d'elles:

Classe **Petrel**

Aquesta és la classe principal i el punt d'entrada de l'aplicació. S'encarrega d'inicialitzar el UI i els diferents serveis.

onCreateBundle()	Punt d'entrada de l'aplicació
getListView()	Obtenir els ListViews del programa
refresh()	Es crida quan l'aplicació ha d'actualitzar el UI
onConfigurationChange(Configuration)	Callback que es crida quan hi ha canvis en la configuració del programa
onResume()	El sistema posa l'aplicació en mode continuar
LoadFinishReciever()	Callback que es crida després de rebre un Intent
onPause()	El sistema posa l'aplicació en mode pausa
getPorfolio()	Crida per a obtenir un portfolio determinat
getSetting()	Obtenir el valor d'un paràmetre de configuració
setSetting()	Establir un valor d'un paràmetre de configuració

Classe **PetrelDbAdapter**

Aquesta classe gestiona l'accés a la base de dades.

PetrelDbAdapter	Inicialitza l'accés a les dades de la base de dades
open()	Obre la connexió amb la base de dades
close()	Tanca la conneixió amb la base de dades

Classe **PetrelListAdapters**

Aquesta classe implementa els diferents adaptadors de llista que ens serveixen per implementar el UI de l'aplicació.

PortfolioListAdapter()	Constructor de la classe
getCount()	Retorna el nombre d'elements
getItem()	Retorna l'element
getItemId()	Retorna l'identificador de l'element
getView()	Retorna la vista

Classe **PetrelIntentListHelper**

Aquesta classe implementa el menú de context quan es fa clic sobre algun dels elements de les llistes del UI.

getIntent()	Obté la classe Intent de l'opció determinada
getRunnableList()	Obté la llista de UI

Classe **PetrelServiceReceiver**

Aquesta classe rep les notificacions que envia el servei a l'aplicació.

onRecieve()	Callback que obté les crides que envia el servidor

Classe **PetrelService**

La classe que implementa el servei que s'executa en segon pla i que es connectar al servidor REST asíncronament per tal de realitzar les operacions.

startService()	Mètode que inicia el servei
stopService()	Mètode que atura el servei
onBind()	Callback que es crida quan el servei està incialitzat
onCreate()	Callback que es el punt d'entrada del servei
onStartCommand()	Callback que es crida quan el servei rep una petició d'actualització
OnDestroy()	Callback que es crida quan el servei està a punt de ser eliminat
connectServer()	Connecta amb el servidor REST
sendRequest()	Envia una petició al servidor REST
disconnectServer()	Desconnecta del servidor

Classe **PetrelAsset**

Aquesta classe és molt similar a una estructura de dades que descriu un valor dins d'una cartera.

getTicker()	Obté el nom del ticker del valor
setTicker()	Estableix el nom del ticker del valor
getPrice()	Obté el preu del valor
setPrice()	Estableix el preu del valor
getMu()	Obté μ del valor
setMu()	Estableix μ del valor
getSigma()	Obté σ del valor
setSigma()	Estableix σ del valor
getW()	Obté w del valor
setW()	Estableix w del valor

Classe **PetrelPortfolio**

Aquesta classe representa una cartera.

getMu()	Obté μ de la cartera
setMu()	Estableix μ de la cartera
getSigma()	Obté σ de la cartera
setSigma()	Estableix σ de la cartera
getRho()	Obté ρ de la cartera
setRho()	Estableix ρ de la cartera
getAsset()	Obté un valor determinat
setAsset()	Crea o edita un valor determinat
removeAsset()	Elimina un valor determinat

Classe **PetrelPortfolioList**

La classe que representa un llistat de carteres.

getPortfolio()	Obté una cartera determinada
setPortfolio()	Crea o edita una cartera
removePortfolio()	Elimina una cartera

5.2 El servidor REST

En aquest apartat es veurà el disseny de l'aplicació des de la part del servidor web *Django*[24]. Com en el cas anterior del dispositiu Android, es farà, en primer lloc, una enumeració de les funcionalitats i la seva especificació per després centrar-nos en l'arquitectura del sistema. En aquest aspecte en particular, es tracta d'un disseny força més simple que en el cas del client que hem vist abans, ja que l'ús d'un framework que implementa la major part de la funcionalitat ens fa la vida més fàcil. Tot i això, també s'ha de tenir en compte que en aquesta capa és on trobem la part més complexa de tot el codi de l'aplicació.

El servidor incorpora la lògica, les dades bursàtils i el codi relacionat amb la part d'Intel·ligència Artificial de l'aplicació i és el que s'encarrega de realitzar les operacions que es mostren en el client Android. De fet la seva API REST permet que qualsevol client pugui connectar-se i obtenir la informació a través d'una crida.

El servidor està escrit en el llenguatge de programació Python i utilitza el *Django Piston*[25] per automatitzar la comunicació de la manera més eficient possible, ja que la comunicació entre el el client i el servidor no és la part més important ni interessant de l'aplicació i és per això que és millor resoldre-la de la manera més ràpida possible sense dedicar-hi molt de temps ni esforços.

El fet, però, de seleccionar un framework en Python té la raó de ser principalment per l'interés de realitzar la part d'Intel·ligència Artifical en Python i per això s'ha optat per utilitzar un servidor web fet amb el mateix llenguatge de programació per evitar problemes que puguéssin sortir més endavant i, en definitiva, que la implementació d'aquesta part fos el més ràpid i fàcil possible.

La implementació de la part de càlcul de portfolios òptims es farà amb les eines de l'anomenat "Python científic"[26] com són les llibreries *numpy* [27] i *pandas*[28], mentre que la màquina de vectors de suport s'implementarà utilitzat el la llibreria *scikit-learn*[29] que ens ofereix tot el què necessitem per tal de fer els càlculs de la volatilitat.

Com es combinen tots aquests elements una mica dispars per tal de crear l'aplicació és el què veurem en les seccions següents. El què fa la nostra aplicació, en realitat, és incorporar totes aquestes funcionalitats dins d'un projecte de Django. I és per això, que per tal de facilitat aquesta integració, s'ha decidit utilitzar el llenguatge de programació Python en tot moment a l'hora de desenvolupar les funcionalitat d'aquesta capa.

5.2.1 Descripció de les funcionalitats

En aquest punt farem una descripció funcional de l'aplicació des del punt de vista del client que es connecta a ella a través de la seva API, sense entrar en detalls tècnics sobre la mateixa. Aquesta secció servirà, doncs, d'introducció a la resta de seccions que aniran aprofundint en cada una de les funcionalitats de manera molt més detallada.

Separarem la descripció de funcionalitats en dos tipus: les externes, que es poden accedir des de fora del servidor (la seva API) i les internes, que donen suport a les funcions externes però que no es poden cridar directament i per tant no són accessibles des de fora.

Així doncs, des del punt de vista del client, la funcionalitat del servidor és la següent:

1. El client pot fer una crida per obtenir el llistat actualitzat dels tickers soportats pel servidor. D'aquesta manera el client pot actualitzar periòdicament els tickers que es poden utilitzar. Normalment el client guardarà aquests símbols en una base de dades en remot com a memòria cau, per no haver d'anar carregant-los cada vegada, tot i que res li impedeix fer-ho si això ho vol.
2. El client fa una crida per demanar una cartera eficient amb l'API proporcionada i el servidor respondrà amb un JSON[30] que conté la cartera completada. La cartera de resultat inclou el valor i el pes dins de la cartera de cada un dels elements. A la crida se li pot passar una cartera completament buida i en aquest cas el servidor l'omplirà amb el número de valors especificats o bé pot incloure ja algun valor i, opcionalment, pes d'aquests valors.

Per altra banda, a nivell de funcionalitat interna, aquestes són les funcionalitats que també fa el servidor:

1. El servidor per la seva banda es connecta també a la base de dades de Yahoo Finance per tal d'obtenir els valors de la borsa que interessent. Aquests valors es guarden a una base de dades interna i es poden recuperar i actualitzar. Aquests valors són els que s'utilitzaran per tal de fer els càlculs necessaris per obtenir la volatilitat esperada.
2. Calcular la volatilitat esperada de cada valor del mercat que hi ha guardat prèviament a la base de dades. Aquestes dades s'utilitzaran posteriorment en la creació de la cartera òptima. Aquests càlculs es realitzen de manera periòdica per tal de tenir els valors a disposició del client de la manera més ràpida possible a l'hora de retornar els resultats.

3. Calcular la cartera òptima amb les restriccions de retorn esperat o de volatilitat que imposa el client (si s'escau).

Més endavant en aquesta secció s'especificaran les crides del client amb el màxim de detall possible. I també les relacions entre les funcionalitats internes i les externes del servidor.

5.2.2 Requeriments

En aquesta secció hem de parlar de tres tipus de requeriments diferents, i que s'enumeraran per separat explícitament, que formen part de la mateixa entitat lògica, que és el servidor, tot i que realitzen funcions ben diferents. Els tres tipus de funcionalitats són les que podem veure seguidament:

1. En primer lloc tenim la part de comunicació, que és el servidor REST en sí. La seva funció és el pas d'informació entre les operacions que realitza el servidor i el client Android.
2. Després tenim la part que s'encarrega de recuperar les dades del Yahoo Finance (tot i que podem fer servir qualsevol altre servei que ens ofereixi dades financeres) i guardar-les en local en una base de dades per poder-les processar a posteriori.
3. En tercer lloc tenim les operacions que s'encarreguen de fer la predicció de la volatilitat del mercat i de generar les carteres òptimes. Tot i que són dues funcions bastant diferents, a nivell de requeriments les podem combinar en una de sola perquè els requeriments de les dues són exactament els mateixos.

A continuació, com ja hem esmentat abans, es veuran els tres tipus de requeriments per separat:

Servidor REST

Els requeriments mínims del servidor REST en els proporciona ja el framework de Django sobre el que treballarem. Això vol dir que tota la part de comunicació a través de REST està ja propiament resolta amb un sistema sòlid i segur. El fet de triar un sistema que ja ens resol el tema del servidor i de la comunicació és, sobretot, degut al fet de no haver de perdre temps implementant una funcionalitat que no és realment del nostre interés en aquest treball.

Els requeriments mínims del servidor REST que executa l'aplicació són els següents:

- El servidor ha de ser multitasca per tal de poder acceptar diverses connexions de manera concurrent. Aquesta part és important perquè no pot ser que el servidor es quedi bloquejat tot calculant una cartera òptima. Hi ha d'haver un sistema de tasques que permetin fer tots aquesta càlculs en paral·lel i al mateix temps. S'ha de pensat tot el sistema perquè això sigui possible.

- Des del punt de vista de l'autenticació, el client s'ha d'autenticar amb el servidor per poder accedir a comunicar-se amb ell. Aquesta restricció és podria relaxar una mica, però al tractar-se de dades financeres és important que la comunicació sigui autenticada.

- Relacionat amb el punt anterior, les connexions entre el client i el servidor han de ser xifrades utilitzant el protocol HTTPS i un certificat digital. Aquest és un requeriment totalment obligatori.

- Totes les accions del servidor es guardaran en un fitxer de bitàcola. El motiu és la sensibilitat de les dades que el servidor gestiona. S'ha de tenir un registre de totes les peticions per si de cas i analitzar aquest registre de manera periòdica.

- La configuració del servidor es troba en un fitxer dins del propi servidor. També seria possible utilitzar la base de dades per a guardar certs valors com poden ser, per exemple, la periodicitat en què s'ha de connectar per tal d'obtenir dades bursàtils actualitzades.

- És necessari una validació dels missatges que es passen entre el client i el servidor per assegurar-se que les dades son correctes. Aixo va més enllà que l'autenticació o el xifrat. El format de les dades ha de ser correcte perquè el client, en principi, no farà cap validació del format de les dades que rep. Tot i que es podria arribar a implementar aquesta validació en la part del client en el futur.

Servei d'obtenció de dades financeres

El servei d'obtenció de dades financeres s'encarrega de connectar-se de manera periòdica a un servidor que ofereixi dades financeres (en en nostre cas el *Yahoo Finance*) i obtenir les dades que es necessiten per poder realitzar els càlculs de la volatilitat de les carteres. La seva funció es limita a obtenir aquestes dades financeres i posar-les a la base de dades de manera el més eficient possible.

Un aspecte interessant és la obtenció dels símbols del mercat. La llista es podria aconseguir de manera automàtica, però, de moment, s'editarà al servidor de manera manual. Això es fa d'aquesta manera perquè a l'API de *Yahoo Finance*[31] se li ha de passar sempre el símbol que vols i no li pots

dir, per exemple, dona'm tots els valors que formen part de l'IBEX35. I la composició de l'IBEX35 va canviant de tant en tant.

Els requeriments pel servei d'obtenció de dades financeres són els següents:

- El servei ha de disposar d'un llistat dels símbols que ha d'actualitzar en en un fitxer de configuració o a la base de dades mateix.
- El servei s'ha de connectar a l'API de *Yahoo Finance* de manera respectuosa per tal de no afectar ni col·lapsar el servei. Les dades s'obtindran a través de peticions regulars separades per un mínim d'un segon entre elles.
- La base de dades del servidor ha d'estar pensada per poder treballar amb series de dades temporals. Hi ha bases de dades que són més eficients per treballar amb aquestes dades que les bases de dades relacionals normals i normalment s'anomenen bases de dades *NOSQL*[32] com a exemples d'aquestes bases de dades podem trobar el *Apache Cassandra*[33] o el *CouchDB*[34].

Predicció de volatilitat i creació de carteres òptimes

El sistema de predicció de volatilitat s'executa cada vegada que s'actualitzen les dades financeres, això és un cop al dia després del tancament dels mercats, tot i que es podria canviar la periodicitat. Per tant, s'executa després del servei d'obtenció de dades financeres que hem vist al punt anterior.

Per altra banda, el sistema de creació de carteres òptimes s'executa sota demanda del client Android cada vegada que aquest fa una petició d'aquest servei. I, de fet, és l'únic servei que està exportat a l'exterior i que proporciona el servidor a través de l'API REST.

Els requeriments pel sistema de predicció de volatilitat i creació de carteres òptimes són el següents:

- Un servidor prou potent per tal de fer els càlculs necesaris, aquest punt està molt relacionat amb el volum de dades financeres que ha de tractar. En general, és interessant que disposi d'una GPU per poder fer el càlcul de matrius de manera el més eficient possible[35]. És important que la resposta a l'usuari sigui la més ràpida possible. Aquests requeriments també impliquen una bona quantitat de memòria RAM i discos amb un bon temps d'accés.

- Com en el cas anterior, es necessari disposar d'una base de dades NOSQL amb un bon suport per a les series temporals de dades. La lectura d'aquestes dades de la base de dades ha de ser el més ràpid possible.
- Si els recursos del servidor ho permeten és interessant disposar de les dades financeres precalculades en una memòria cau, per tal de poder fer els càlculs de manera gairebé instantània.

5.2.3 Especificació

Igual com s'ha fet en el cas del client per a Android, l'especificació es farà en UML tot creant actors i casos d'ús que veurem durant aquest apartat. Al contrari del cas del client, però, els actors no seran els usuaris humans de l'aplicació, sinó que seran la representació del client Android així com altres serveis que estan en el servidor. Els casos d'ús, també, estaran adaptats a aquests actors peculiars.

Actors En aquest cas els actors seran el client Android i el servidor Django, que són els que interactuen amb les diferents funcionalitats d'aquesta part de l'aplicació. El fet d'utilitzar aquests dos actors en comptes de només el client és per intentar definir completament tota la funcionalitat del servidor i no centrar-nos només en la part de comunicació, tot i que evidentment té una importància cabdal en el projecte.

Les tasques que farien conceptualment cada un d'aquests dos actors serien les següents:

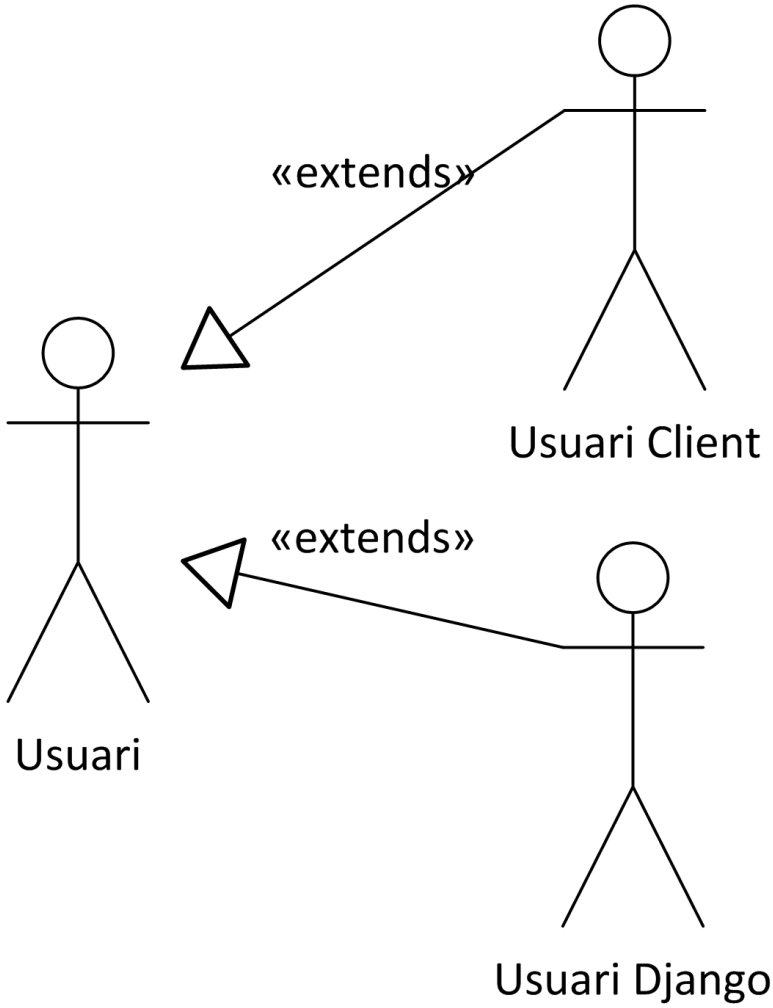

Figura 19: Diagrama dels actors del servidor. *Font: elaboració pròpia*

I les tasques que farien conceptualment cada un d'aquests usuaris serien les següents:

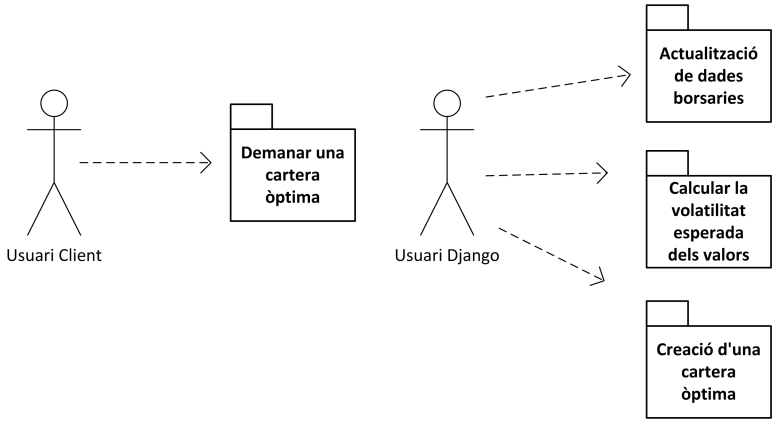

Figura 20: Diagrama de classificació dels casos d'ús del servidor. *Font: elaboració pròpia*

Diagrama de casos d'ús. En els següents diagrames representarem els casos d'ús de la nostra aplicació:

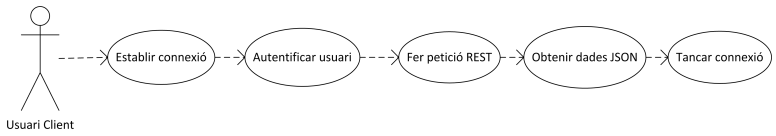

Figura 21: Diagrama de demanar una cartera òptima. *Font: elaboració pròpia*

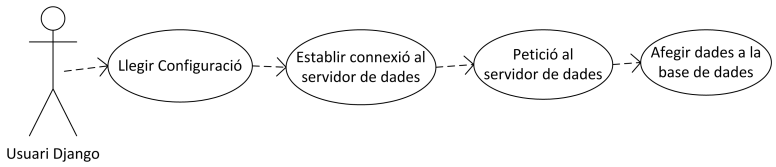

Figura 22: Diagrama d'actualització de dades borsàries. *Font: elaboració pròpia*

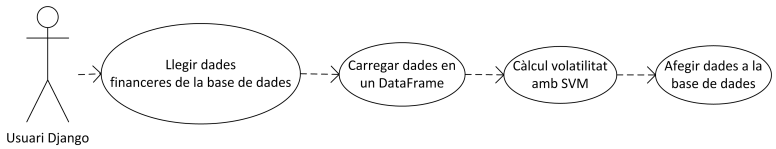

Figura 23: Diagrama de calcular la volatilitat dels valors. *Font: elaboració pròpia*

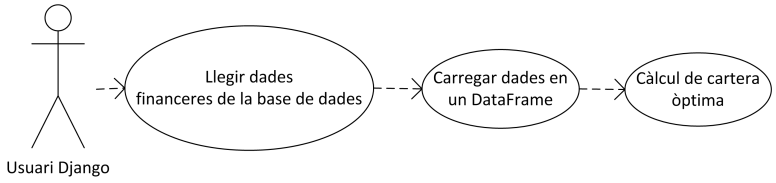

Figura 24: Diagrama de creació d'una cartera òptima. *Font: elaboració pròpia*

Especificació dels casos d'ús. En aquest apartat hi haurà l'especificació dels casos d'ús de del servidor REST dividits entre els usuaris Client i Django que hem vist en l'apartat anterior. L'especificació es mostra en la taula següent:

Establir connexió

Resum de la funcionalitat:	Inicia la comunicació amb el servidor i obté els recursos necesaris per a realitzar una petició
Actors:	Usuari client
Casos d'ús relacionats:	Tancar connexió
Precondició:	El servidor ha d'estar engegat
Resum:	Inicia la comunicació amb el servidor

Autenticar usuari

Resum de la funcionalitat:	Comprova l'autenticitat de l'usuari
Actors:	Usuari client
Casos d'ús relacionats:	Establir connexió, Tancar connexió
Precondició:	Establir connexió
Resum:	Verifica l'usuari i li permet realitzar peticions o talla la connexió

Fer petició REST

Resum de la funcionalitat:	Fa una petició REST al servidor
Actors:	Usuari client
Casos d'ús relacionats:	Obtenir dades en format JSON
Precondició:	Establir connexió, Autenticar usuari
Resum:	Passa els paràmetres opcionals al servidor que poden restringuir la cartera creada

Obtenir dades en format JSON

Resum de la funcionalitat:	Obté les dades de la cartera creada
Actors:	Usuari client
Casos d'ús relacionats:	Fer petició REST
Precondició:	Fer petició REST
Resum:	La resposta amb les dades en format JSON de la cartera creada amb les restriccions demanades per l'usuari

Tancar connexió

Resum de la funcionalitat:	Tanca la connexió amb el servidor i allibera els recursos utilitzats en la mateixa
Actors:	Usuari client
Casos d'ús relacionats:	Establir connexió
Precondició:	Establir connexió
Resum:	Tanca la connexió amb el servidor

Tancar connexió

Resum de la funcionalitat:	Tanca la connexió amb el servidor i allibera els recursos utilitzats en la mateixa
Actors:	Usuari client
Casos d'ús relacionats:	Establir connexió
Precondició:	Establir connexió
Resum:	Tanca la connexió amb el servidor

Llegir configuració

Resum de la funcionalitat:	Llegeix un valor de configuració al fitxer de configuració de l'aplicació
Actors:	Usuari Django
Casos d'ús relacionats:	Usuari Django
Precondició:	El Django ha d'estar en funcionament
Resum:	Llegeix la freqüència d'actualització a la configuració

Establir connexió al servidor de dades

Resum de la funcionalitat:	Estableix una connexió al Yahoo Finance
Actors:	Usuari Django
Casos d'ús relacionats:	Petició al servidor de dades
Precondició:	Llegir configuració
Resum:	Inicia la connexió amb el servidor de dades i la tanca quan s'ha acabat la petició de dades

Petició al servidor de dades

Resum de la funcionalitat:	Fa una petició d'informació al Yahoo Finance
Actors:	Usuari Django
Casos d'ús relacionats:	Establir connexió al servidor de dades, Actualitzar dades a la base de dades
Precondició:	Establir connexió al servidor de dades
Resum:	Fa una petició de dades borsàries

Actualitzar dades a la base de dades

Resum de la funcionalitat:	Actualitza les dades financeres de la base de dades
Actors:	Usuari Django
Casos d'ús relacionats:	Petició al servidor de dades, Càlcul de la volatilitat amb SVM
Precondició:	Establir connexió
Resum:	Desa les noves dades financeres a la base de dades

Llegir dades financeres de la base de dades

Resum de la funcionalitat:	Llegeix les dades financeres de la base de dades
Actors:	Usuari Django
Casos d'ús relacionats:	Carregar dades en un DataFrame
Precondició:	Petició al servidor de dades
Resum:	Llegeix dades financeres de la base de dades

Carregar dades en un DataFrame

Resum de la funcionalitat:	Posa les dades en un DataFrame per poder- hi treballar
Actors:	Usuari Django
Casos d'ús relacionats:	Llegir dades financeres de la base de dades, Càlcul de la volatilitat amb SVM
Precondició:	Llegir dades financeres de la base de dades
Resum:	Les dades financeres es carreguen a memòria en un DataFrame

Càlcul de la volatilitat amb SVM

Resum de la funcionalitat:	Calcula la volatitat esperada dels valors del mercat
Actors:	Usuari Django
Casos d'ús relacionats:	Carregar dades en un DataFrame
Precondició:	Carregar dades en un DataFrame Petició al servidor de dades
Resum:	Operació de calcular la volatilitat esperada dels valors del mercat

Càlcul de cartera òptima

Resum de la funcionalitat:	Calcula la cartera òptima
Actors:	Usuari Django
Casos d'ús relacionats:	Carregar dades en un DataFrame
Precondició:	Carregar dades en un DataFrame, Càlcul de la volatilitat amb SVM
Resum:	Utilitza els valors de la base dades per calcular la cartera òptima

5.2.4 Arquitectura

El servidor REST, que utilitza la infraestructura de Django, disposa de tres funcionalitats perfectament diferenciades entre elles a nivell lògic, tot i que totes formen part d'aquesta infraestructura i estan implementades seguint les seves directrius. Tot i així, es tracta d'un codi que és prou independent perquè es pugui traslladar amb facilitat a qualsevol altra aplicació amb un petit esforç d'adaptació.

L'arquitectura d'aquests quatre components que ja hem mencionat diverses vegades en els apartats anteriors és la que es mostrarà a continuació.

Servidor REST

El servidor de Django utilitza el protocol REST per a comunicar-se amb el client Android. El REST va ser desenvolupat pel W3C Technical Architecture Group en paral·lel amb el HTTP/1.1 i com aquest està basat en el disseny del HTTP/1.0 [38].

El client Android inicia la petició al servidor i el servidor Django retorna les respostes adequades a les peticions, o bé un missatge d'estat d'errir si per algun motiu no pot processar la petició. Els missatges d'error poden estar relacionats amb les dades que s'han passat (el format no és correcte o el valor no està entre el rang esperat) o bé amb l'estat general del servidor (la

petició no es pot processar perquè no hi ha memòria suficient per a executar les operacions necessaries per a realitzar la operació).

Les peticions i les responses estan construïdes al voltant de la transmissió de la representació de recursos. En aquest cas, els recursos amb els quals treballen tant el client com el servidor son, per exemple, les carteres o els actius que estan representats amb un format JSON que tant el client com el servidor poden entendre.

En aquest sentit, el client envia al servidor la representació d'una cartera que té uns quants valors opcionals buits (com poden ser w o bé σ). El servidor fa els càlculs per a aquests valors i els posa en aquesta representació. Llavors els valors es tornen a enviar al client que llegirà aquesta representació i la posarà a la seva base de dades.

La URL del API REST del servidor és la següent:

http://api.corrius.org/services/rest/

I els mètodes que es poden utilitzar els podem veure en la taula següent:

Mètode	*Paràmetres*
petrel.get.stocks	
petrel.optimize.portfolio	id
	name
	w
	mu
	sigma
	rho
	assets

Les crides que implementen aquesta API podrien ser, per exemple:

$http://api.corrius.org/services/rest/?method = petrel.get.stocks$

$http://api.corrius.org/services/rest/?method = petrel.optimize.portfolio \&id = 1\&name = test\&w = 0\&mu = 0\&sigma = 0\&rho = 0\&assets = SAN; ABE; POP$

Servei d'obtenció de dades financeres

El Servei d'obtenció de dades financeres utilitza el mateix paradigma de l'API REST que hem vist en el cas anterior, però ara el Servidor Django fa de client per al servidor de Yahoo Finance, amb la qual cosa s'inverteixen els papers que hem acabat de veure.

Les dades s'obtenen del servidor de Yahoo Finance a través d'una API REST tot i que el valors en venen en format CSV ja que és un format molt més còmode per a representar series temporals de dades. Per tant, el què hem de fer és passar a l'API el símbol de l'actiu sobre el que volem treballar i un rang de dades que va des de la data d'inici de les dades fins a la data final.

Un exemple d'aquesta crida seria:

$http://ichart.yahoo.com/table.csv?s=SAN\&$
$d=01\&$
$e=01\&$
$f=2005\&$
$g=d\&$
$a=31\&$
$b=12\&$
$c=2010\&$
$ignore=.csv$

Un cop tenim les dades en memòria en format CSV les hem de parsejar i posar-les a la nostra base de dades. Les dades que podem obtenir de manera gratuïta són dades diàries.

Les dades bursaries es desen en una base de dades optimitzada per treballar amb series de dades, com per exemple Cassandra. En el nostre cas, com que el volum de dades no és molt important, es pot obviar i utilitzar una base de dades relacional normal o bé, treballar amb fitxers de text directament o bé objectes serialitzats a disc (en el llenguatge de programació Python el més normal és treballar amb pickles[39]).

Predicció de volatilitat

La predicció de la volatilitat està relacionada amb el cas anterior ja que un cop s'obté, o s'actualitzen les sèries de dades, es prediu la volatilitat esperada dels valors amb els nous valors. El fet d'afegir nous valors a la sèrie de dades implica que s'han de tornar a fer les operacions.

De fet, això es fa per tal de millorar l'eficiència del programa, ja que d'aquesta manera ja tenim els valors precalculats i no fa falta fer-ho cada vegada que se'ns demana. Com que les dades que obtenim son només diàries, només fa falta realitzar aquesta operació un cop al dia quan s'actualitzen les dades. Si treballéssim, per exemple, amb minuts o segons, segurament el més eficient seria calcular aquesta volatilitat només quan se'ns demana un portfolio que inclou aquest valor.

Com veurem més endavant, amb molt de detall, el càlcul de la volatilitat es fa a través del *scikit-learn*, que és una llibreria de Python que permet fer operacions amb Support Vector Machines, o màquines de vectors de suport.

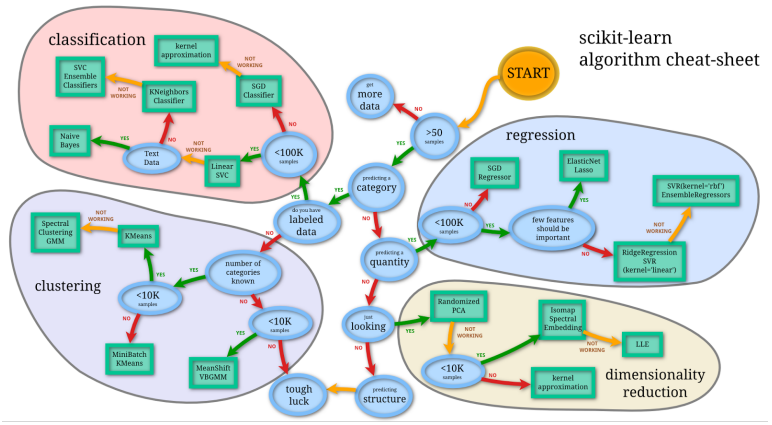

Figura 25: Machine Learning Cheat Sheet pel scikit-learn. *Font: Andreas Mueller*

Creació de carteres òptimes

La creació d'una cartera òptima és simplement calcular quin percentatge de diners es posa a cada un dels actius de la cartera per tal de, per exemple, maximitzar el retorn o minimitzar el risc. En el nostre cas, utilitzarem la minimització del risc per defecte ja que és el què té el nostre interés i ens proporciona uns resultats prou raonables.

La creació d'una cartera òptima es fa en el moment en què el client Android fa la petició REST. Es llegeixen els valors que l'usuari té a la cartera i s'assigna un percentatge a cada un d'ells segons els càlculs de predicció de la volatilitat que s'han fet amb les màquines de vector de suport. Aquests valors ja estan precalculats per tal que la petició sigui la més ràpida possible.

L'optimització de la cartera es fa utilitzant les llibreries de Python *pandas* i *numpy* sobretot, ja que el treball és sobretot operacions amb matrius i solucionadors de problemes lineals. S'utilitzen aquestes llibreries per tal de simplificar l'implementació de la solució.

Diagrames de seqüència

En aquest apartat s'especificaran alguns diagrames de seqüència de les funcionalitats del servidor REST que s'han detectat en els apartats anteriors, per tal d'il·lustrar encara amb més detall el funcionament de l'aplicació. Ens centrarem en les funcionalitats clau i/o més interessants i deixarem de banda aquelles funcionalitats que són comunes o trivials.

Els dos casos que són més interessants són els de l'actualització de dades borsàries (on s'inclourà el càlcul de la volatilitat) i la realització d'una optimització d'una cartera. Són aquests casos els que veurem a continuació.

Actualitzar dades borsàries

En aquest primer diagrama de seqüències veurem el cas en què s'actualitzen les dades de borsa tot connectant-se al servidor de *Yahoo Finance*. El diagrama també inclou el càlcul de la volatilitat si bé, conceptualment, és un cas a part.

El servidor Django fa una petició a la base de dades per llegir la freqüència d'actualització de les dades per tal d'actualitzar les dades quan es necesari. Després fa una petició REST al servidor de Yahoo Finance per a obtenir les dades en format CSV que parseja i verifica. Durant aquest procés també es fa el càlcul actualitzat de la predicció de la volatilitat. Un cop tenim tots els càlculs, es posen les dades a la base de dades.

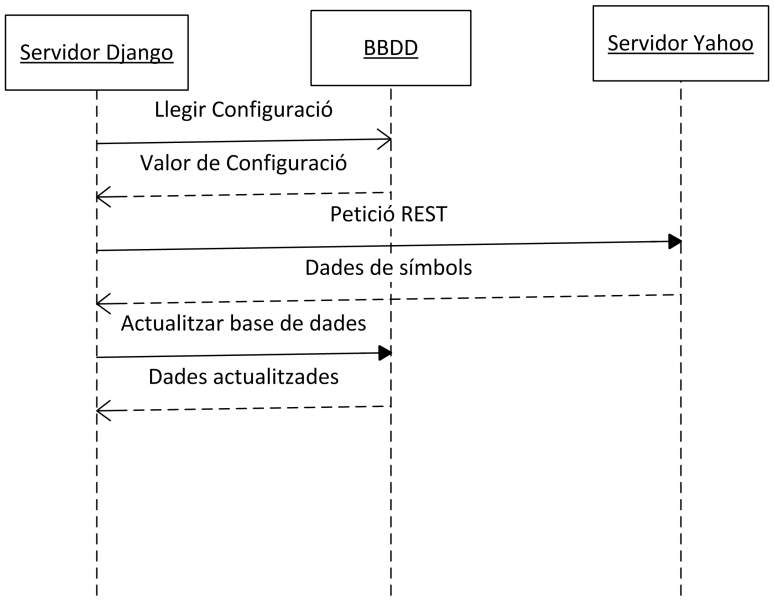

Figura 26: Diagrama de seqüències d'actualitzar dades borsàries. *Font: elaboració pròpia*

Optimització d'una cartera

En aquest segon diagrama de seqüències es pot veure tot el procés d'optimització d'una cartera en la part del servidor de Django. En aquest cas es veu el procés des de la petició REST que surt del client Android.

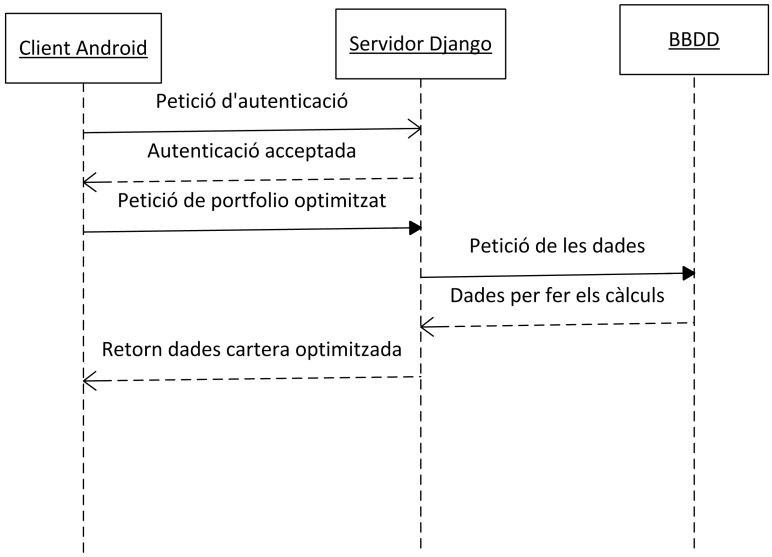

Figura 27: Diagrama de seqüències d'optimització d'una cartera. *Font: elaboració pròpia*

6 Detalls de la implementació

En aquesta secció es veurà amb molt de detall com s'han implementat en codi les funcionalitats dels apartats anteriors. La secció està dividida en dues parts que representen la implementació de les funcionalitats de client Android i la implementació en el servidor Django.

Per l'interès d'aquest treball, la part més important i descrita amb més cura serà la part del servidor, i molt especialment, la relacionada més directament amb la intel·ligència Artificial de l'aplicació. Tot i això, veurem amb detall les parts més interessants de tot el conjunt.

6.1 El Client Android

El codi del Client Android s'ha escrit de manaera natural en el llenguatge de programació Java i per sobre de la plataforma l'Android SDK, que ens ofereix una sèrie d'ajudes molt importants, però també algunes restriccions, en la manera de fer les coses.

En aquesta implementació s'ha intentat seguir en tot moment les bones pràctiques de disseny i les recomanacions de la documentació, i dels exemples, de la plataforma. D'aquesta manera ens podem assegurar, per una banda, un bon funcionament de l'aplicació i també que aquesta serà compatible en versions posteriors a la versió 17 del SDK que és la versió sobre la qual hem fet la nostra implementació.

6.1.1 Implementació del servidor

El servidor funciona en segon pla durant tota la vida de l'aplicació i s'encarrega bàsicament de la comunicació amb el servidor de Django. S'ha de tenir en compte que el servidor està funcionant en tot moment i que, per tant, és important que faci el mínim de coses possibles i que tota la funcionalitat estigui implementada en el client.

El codi de la implementació del servidor és la següent:

```
public class PetrelService extends Service
{
    private Timer timer = new Timer();
    private void startService()
    {
        timer.scheduleAtFixedRate( new TimerTask()
        {
```

```
            public void run()
            {
                Log.v("Petrel", "PetrelService: running");
            }
        }, 0, 100);
    }
    private void stopService()
    {
        if (timer != null)
            timer.cancel();
    }
    public IBinder onBind(Intent intent)
    {
        return null;
    }

    @Override
    public void onCreate()
    {
        super.onCreate();
        startService();
        Log.v("Petrel", "PetrelService Created");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        Log.v("Petrel", "PetrelService onStartCommand");
        return START_STICKY;
    }

    @Override
    public void onDestroy()
    {
        stopService();
        super.onDestroy();
        Log.v("Petrel", "PetrelService Destroyed");
    }
```

6.1.2 Accés a la base de dades

L'aplicació desa les seves dades de configuració en una base de dades SQLite que és la que tenen disponible totes les aplicacions Android. S'utilitza aque-

sta perquè ve per defecte amb tots els clients Android i no necessita cap mena de configuració addicional per fer-se servir.

Les dades de l'aplicació es guarden a quatre taules diferents. Tres taules que representen els elements cartera, actiu i valor. I una quarta taula que ens serveix per relacionar els actius i les carteres sense haver d'utilitzar clau foranes. Això ens simplifica bastant la gestió de la base de dades, tot i que, en principi, no hi ha cap problema en utilitzar claus foranes en una base de dades SQLite.

L'especificació de la taula de carteres és la següent:

Portfolio

_id	integer primary key autoincrement
name	text not null
mu	real
sigma	real
rho	real

L'especificació de la taula de valors és la següent:

Asset

_id	integer primary key autoincrement
name	text not null
price	real
mu	real
sigma	real

L'especificació de la taula de símbols és la següent:

Ticker

_id	integer primary key autoincrement
name	text not null

Finalment la descripció de taula que relaciona les carteres amb els actius és la següent:

Ticker

_id	integer primary key autoincrement
name	text not null

S'han creat també unes classes auxiliars anomenades Portfolio, Asset i Ticker que ens serveixen per representar les dades en la interfície d'usuari de manera molt més còmoda i eficient. Aquestes classes simplement representen els objectes de la base de dades en Java i ens ofereixen uns accessors que podem utilitzar per fer la feina més fàcil.

Podem veure l'exemple de la classe Ticker a continuació. Les altres classes són similars i no aporten més informació que la que hi ha a la taules de la base de dades:

```
package org.uoc.petrel;

public class Ticker {
    private long id;
    private String name;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }
```

Aquestes dades són accessibles tan per part del client com del servidor a través de la classe PetrelDbAdapter que és la que s'encarrega d'encapsular l'accés a més baix nivell i proporcionar uns mètodes públics per tal d'accedir a les dades (per a llegir-es i, si s'escau, per a modificar-les). Així doncs, totes les classes que han d'accedir a la base dades, necessiten una instància de PetrelDbAdapter.

Internament la classe té una implementació de la classe de l'Android SDK anomenada SQLiteOpenHelper, que és la que s'encarrega de gestionar la

creació i l'accés a la base de dades. Per tant, la creació i l'accés a la base de dades (exportats públicament a través dels mètodes "open" i "close) es fan a través de la mateixa.

Aquest és el fragment de codi que ho implementa:

```
package org.uoc.petrel;

import java.util.ArrayList;
import java.util.List;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class PetrelDBAdapter {
    private static class DatabaseHelper extends SQLiteOpenHelper {

        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {

            db.execSQL(TABLE_PORTFOLIO_CREATE);
            db.execSQL(TABLE_ASSET_CREATE);
            db.execSQL(TABLE_TICKER_CREATE);
            db.execSQL(TABLE_PORTFOLIO_TO_ASSET_CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion,
            int newVersion) {
            Log.w(TAG, "Upgrading database from version " +
                oldVersion + " to "
                + newVersion + ", which will destroy all old data");
            db.execSQL("DROP TABLE IF EXISTS portfolio");
            db.execSQL("DROP TABLE IF EXISTS asset");
            db.execSQL("DROP TABLE IF EXISTS ticker");
            onCreate(db);
```

```
    }
}

// Table Names
public static final String TABLE_PORTFOLIO = "portfolio";
public static final String TABLE_ASSET = "asset";
public static final String TABLE_TICKER = "ticker";
public static final String TABLE_PORTFOLIO_TO_ASSET =
    "portfolio_to_asset";

// Key Names
public static final String KEY_PORTFOLIO_ID = "_id";
public static final String KEY_PORTFOLIO_NAME = "name";
public static final String KEY_PORTFOLIO_MU = "mu";
public static final String KEY_PORTFOLIO_SIGMA = "sigma";
public static final String KEY_PORTFOLIO_RHO = "rho";

private String[] PortfolioAllColumns = { KEY_PORTFOLIO_ID,
        KEY_PORTFOLIO_NAME, KEY_PORTFOLIO_MU, KEY_PORTFOLIO_SIGMA,
        KEY_PORTFOLIO_RHO };

public static final String KEY_ASSET_ID = "_id";
public static final String KEY_ASSET_NAME = "name";
public static final String KEY_ASSET_W = "w";
public static final String KEY_ASSET_MU = "mu";
public static final String KEY_ASSET_SIGMA = "sigma";

private String[] AssetAllColumns = { KEY_ASSET_ID, KEY_ASSET_NAME,
        KEY_ASSET_W, KEY_ASSET_MU, KEY_ASSET_SIGMA };

public static final String KEY_TICKER_ID = "_id";
public static final String KEY_TICKER_NAME = "name";
public static final String KEY_TICKER_COMPANY = "company";

public String[] TickerAllColumns = { KEY_TICKER_ID, KEY_TICKER_NAME,
        KEY_TICKER_COMPANY };

public static final String KEY_PORTFOLIO_TO_ASSET_PORTFOLIO = "portfo
public static final String KEY_PORTFOLIO_TO_ASSET_ASSET = "asset";

private String[] PortfolioToAssetAllColumns = {
        KEY_PORTFOLIO_TO_ASSET_PORTFOLIO, KEY_PORTFOLIO_TO_ASSET_ASSE

// Database creation
```

```
public static final String TABLE_TICKER_CREATE =
        "create table ticker ( _id integer primary key autoincrement , "
        + "name text not null , company text not null );";
public static final String TABLE_ASSET_CREATE =
"create table asset ( _id integer primary key autoincrement , "
        + "name text not null , w double , mu double , sigma double );";
public static final String TABLE_PORTFOLIO_CREATE =
    "create table portfolio ( _id integer primary key autoincrement , "
        + "name text not null , mu real , sigma , real , rho real );";
public static final String TABLE_PORTFOLIO_TO_ASSET_CREATE =
"create table portfolio_to_asset ( _id integer primary key autoincreme
        + "portfolio integer reference portfolio , asset integer refer

// Other constants
private static final String DATABASE_NAME = "Petrel";
private static final int DATABASE_VERSION = 2;
private static final String TAG = "ErelisDbAdapter";

// Members
private DatabaseHelper mDbHelper;
private SQLiteDatabase mDb;
private final Context mCtx;

public PetrelDBAdapter(Context ctx) {
    this.mCtx = ctx;
}

public PetrelDBAdapter open() throws SQLException {
    mDbHelper = new DatabaseHelper(mCtx);
    mDb = mDbHelper.getWritableDatabase();
    return this;
}

public void close() {
    mDbHelper.close();
}

//
// Ticker methods
//
public Ticker createTicker(String name, String company) {
    ContentValues values = new ContentValues();
    values.put(KEY_TICKER_NAME, name);
    values.put(KEY_TICKER_COMPANY, company);
```

```
        long insertId = mDb.insert(TABLE_TICKER, null, values);

        Cursor cursor = mDb.query(TABLE_TICKER, TickerAllColumns,
        KEY_TICKER_ID + " = " + insertId, null, null, null, null);
        cursor.moveToFirst();
        Ticker newComment = cursorToTicker(cursor);
        cursor.close();
        return newComment;
    }

    public void deleteTicker(Ticker ticker) {
        long id = ticker.getId();
        mDb.delete(TABLE_TICKER, KEY_TICKER_ID + " = " + id, null);
    }

    public List<Ticker> getAllTickers() {
        List<Ticker> tickers = new ArrayList<Ticker>();

        Cursor cursor = mDb.query(TABLE_TICKER, TickerAllColumns,
        null, null, null, null, null);

        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            Ticker ticker = cursorToTicker(cursor);
            tickers.add(ticker);
            cursor.moveToNext();
        }
        cursor.close();
        return tickers;
    }

    public Cursor getCursorToTickers() {
        Cursor cursor = mDb.query(TABLE_TICKER, TickerAllColumns,
        null, null, null, null, null);

        if (cursor != null) {
            cursor.moveToFirst();
        }
        return cursor;
    }

    public void populateDefaultTickers()
    {
        createTicker("ABG.P", "Abengoa");
```

```
        createTicker("ABE", "Abertis");
        createTicker("ANA", "Acciona");
        createTicker("ACX", "Acerinox");
        createTicker("ACS",
            "Actividades de Construcci[U+FFFD]n y Servicios");
        createTicker("AMS", "Amadeus");
        createTicker("MTS", "Arcelor Mittal");
        createTicker("POP", "Banco Popular Espa[U+FFFD]");
        createTicker("SAB", "Banco de Sabadell");
        createTicker("SAN", "Banco Santander");
        createTicker("BKT", "Bankinter");
        createTicker("BBVA", "Banco Bilbao Vizcaya Argentaria");
        createTicker("BME", "Bolsas y Mercados Espa[U+FFFD]ols");
        createTicker("CABK", "CaixaBank");
        createTicker("DIA",
            "Distribuidora Internacional de Alimentaci[U+FFFD]n");
        createTicker("ENG", "Enag[U+FFFD]s");
        createTicker("ELE", "Endesa");
        createTicker("FCC",
            "Fomento de Construcciones y Contratas");
        createTicker("FER", "Ferrovial");
        createTicker("GAS", "Gas Natural");
        createTicker("GRF", "Grifols");
        createTicker("IAG", "International Airlines Group");
        createTicker("IBE", "Iberdrola");
        createTicker("ITX", "Industria de Dise[U+FFFD] Textil");
        createTicker("IDR", "Indra Sistemas");
        createTicker("JAZ", "Jazztel");
        createTicker("MAP", "MAPFRE");
        createTicker("TL5", "Mediaset Espa[U+FFFD] Comunicaci[U+FFFD]n");
        createTicker("OHL", "Obras[U+FFFD] Huarte Lain");
        createTicker("REE", "Red El[U+FFFD]ctrica Corporaci[U+FFFD]n");
        createTicker("REP", "Repsol");
        createTicker("SYV", "Sacyr Vallehermoso");
        createTicker("TRE", "T[U+FFFD]cnicas Reunidas");
        createTicker("TEF", "Telef[U+FFFD]nica");
        createTicker("VIS", "Viscofan");
    }

    private Ticker cursorToTicker(Cursor cursor) {
        Ticker ticker = new Ticker();
        ticker.setId(cursor.getLong(0));
        ticker.setName(cursor.getString(1));
        ticker.setCompany(cursor.getString(2));
```

```
        return ticker;
    }

    //
    // Asset methods
    //
    public Asset createAsset(String name) {
        ContentValues values = new ContentValues();
        values.put(KEY_ASSET_NAME, name);
        values.put(KEY_ASSET_W, 0);
        values.put(KEY_ASSET_MU, 0);
        values.put(KEY_ASSET_SIGMA, 0);

        long insertId = mDb.insert(TABLE_ASSET, null, values);

        Cursor cursor = mDb.query(TABLE_ASSET,
            AssetAllColumns, KEY_ASSET_ID + " = " + insertId,
            null, null, null, null);
        cursor.moveToFirst();
        Asset newAsset = cursorToAsset(cursor);
        cursor.close();
        return newAsset;
    }

    public void deleteAsset(Asset asset) {
        long id = asset.getId();
        mDb.delete(TABLE_ASSET, KEY_ASSET_ID + " = "
        + id, null);
    }

    public Asset updateAsset(Asset asset, double w,
        double mu, double sigma) {
        ContentValues args = new ContentValues();
        long id = asset.getId();
        args.put(KEY_ASSET_ID, id);
        args.put(KEY_ASSET_NAME, asset.getName());
        args.put(KEY_ASSET_W, w);
        args.put(KEY_ASSET_MU, mu);
        args.put(KEY_ASSET_SIGMA, sigma);

        if (mDb.update(TABLE_ASSET, args, KEY_ASSET_ID +
            "=" + id, null) > 0) {
            Cursor cursor = mDb.query(TABLE_ASSET,
                AssetAllColumns, KEY_ASSET_ID + " = " + id,
```

```
                null, null, null, null);
            cursor.moveToFirst();
            asset = cursorToAsset(cursor);
        }
        return asset;
    }

    private Asset cursorToAsset(Cursor cursor) {
        Asset asset = new Asset();
        if (!cursor.isNull(0))
            asset.setId(cursor.getLong(0));
        else
            asset.setId(0);

        if (!cursor.isNull(1))
            asset.setName(cursor.getString(1));
        else
            asset.setName("None");

        if (!cursor.isNull(2))
            asset.setW(cursor.getDouble(2));
        else
            asset.setW(0);

        if (!cursor.isNull(3))
            asset.setMu(cursor.getDouble(3));
        else
            asset.setMu(0);

        if (!cursor.isNull(4))
            asset.setSigma(cursor.getDouble(4));
        else
            asset.setSigma(0);
        return asset;
    }

    //
    // Portfolio methods
    //
    public Portfolio createPortfolio(String name) {
        ContentValues values = new ContentValues();
        values.put(KEY_PORTFOLIO_NAME, name);
        long insertId = mDb.insert(TABLE_PORTFOLIO, null, values);
```

```
        Cursor cursor = mDb.query(TABLE_PORTFOLIO, PortfolioAllColumns,
                KEY_PORTFOLIO_ID + " = " + insertId,
                null, null, null, null);
        cursor.moveToFirst();
        Portfolio newPortfolio = cursorToPortfolio(cursor);
        cursor.close();
        return newPortfolio;
    }

    public Portfolio updatePortfolio(Portfolio portfolio, double mu,
            double sigma, double rho) {
        ContentValues args = new ContentValues();
        long id = portfolio.getId();
        args.put(KEY_PORTFOLIO_ID, id);
        args.put(KEY_PORTFOLIO_NAME, portfolio.getName());
        args.put(KEY_PORTFOLIO_MU, mu);
        args.put(KEY_PORTFOLIO_SIGMA, sigma);
        args.put(KEY_PORTFOLIO_RHO, rho);

        if (mDb.update(TABLE_PORTFOLIO, args,
            KEY_PORTFOLIO_ID + "=" + id, null) > 0) {
            Cursor cursor = mDb.query(TABLE_PORTFOLIO, PortfolioAllColumn
                    KEY_PORTFOLIO_ID + " = " + id, null, null, null, null
            cursor.moveToFirst();
            portfolio = cursorToPortfolio(cursor);
        }
        return portfolio;
    }

    public void deletePortfolio(Portfolio portfolio) {
        long id = portfolio.getId();
        mDb.delete(TABLE_PORTFOLIO_TO_ASSET,
            KEY_PORTFOLIO_TO_ASSET_PORTFOLIO + " = " + id, null);
        mDb.delete(TABLE_PORTFOLIO, KEY_PORTFOLIO_ID + " = " + id, null);
    }

    public Portfolio addAssetToPortfolio(Asset asset, Portfolio portfolio
        long portfolioId = portfolio.getId();
        long assetId = asset.getId();

        ContentValues values = new ContentValues();
        values.put(KEY_PORTFOLIO_TO_ASSET_PORTFOLIO, portfolioId);
        values.put(KEY_PORTFOLIO_TO_ASSET_ASSET, assetId);
```

```
        if (mDb.insert(TABLE_PORTFOLIO_TO_ASSET, null, values) > 0) {
            portfolio.addAsset(asset);
        }
        return portfolio;
    }

    public Portfolio removeAssetFromPortfolio(Asset asset,
        Portfolio portfolio) {
        long portfolioId = portfolio.getId();
        long assetId = asset.getId();

        mDb.delete(TABLE_PORTFOLIO_TO_ASSET, KEY_PORTFOLIO_TO_ASSET_PORTF
                + " = " + portfolioId + "and " + KEY_PORTFOLIO_TO_ASSET_A
                + " = " + assetId, null);
        portfolio.removeAsset(asset);
        return portfolio;
    }

    public List<Portfolio> getAllPortfolios() {
        List<Portfolio> portfolios = new ArrayList<Portfolio>();

        Cursor cursor = mDb.query(TABLE_PORTFOLIO,
            PortfolioAllColumns, null, null,
                null, null, null);

        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            Portfolio portfolio = cursorToPortfolio(cursor);
            portfolios.add(portfolio);
            cursor.moveToNext();
        }
        cursor.close();
        return portfolios;
    }

    private Portfolio cursorToPortfolio(Cursor cursor) {
        Portfolio portfolio = new Portfolio();
        portfolio.setId(cursor.getLong(0));
        portfolio.setName(cursor.getString(1));
        portfolio.setMu(cursor.getDouble(2));
        portfolio.setSigma(cursor.getDouble(3));
        portfolio.setRho(cursor.getDouble(4));

        Cursor cursorAssets = mDb.query(TABLE_PORTFOLIO_TO_ASSET,
```

```
                PortfolioToAssetAllColumns, KEY_PORTFOLIO_ID + " = "
                        + portfolio.getId(), null, null, null, null);

        cursorAssets.moveToFirst();
        while (!cursorAssets.isAfterLast()) {
            Asset asset = cursorToAsset(cursorAssets);
            portfolio.addAsset(asset);
            cursorAssets.moveToNext();
        }

        cursorAssets.close();
        return portfolio;
    }
}
```

Com podem veure en el codi, es reimplementen dues funcions de la classe SQLiteOpenHelper per tal d'adaptar-la a la base de dades que són "OnCreate" i "OnUpgrade", això es fa només perquè necessitem crear les nostres taules si aquestes no existeixen. Al OnCreate li passem una constant amb la cadena de texta de creació dels camps de la base de dades. La resta de mètodes són utilitzats en la seva implementació per defecte.

6.2 El servidor Django

El codi del client Django està tot escrit en el llenguatge de programació Python i utilitza diferents llibreries per fer les seves tasques, que ja hem mencionat anteriorament.

El servidor de Django està muntat sobre una Ubuntu 12.04 LTS[36] en el servei de cloud computing Amazon EC2[37] que que té les característiques seqüents:

- Kernel de linux versió 3.2
- Apache HTTP versió 2.2.22
- Django versió 1.3
- Python versió 2.7.3
- PostgreSQL versió 8.4

6.2.1 Instal·lació del servidor REST

En aquesta secció veurem el procés d'instal·lació del servidor REST en una Ubuntu 12.04 LTS. El procés hauria de ser semblant en una altra distribució Linux, però agafarem aquesta versió com a referència, ja que és la que hem utilitzat en el nostre projecte.

En primer lloc s'instal·la el *Django Piston* que és el framework REST de Django que utilitzarem en el nostre servidor. Com a dependència s'instal·larà automàticament el *Django* també en executar l'ordre següent:

$ sudo apt-get install python-django-piston

I ara el mòdul de PostgreSQL per a Python.

$ sudo apt-get install python-psycopg2

I, evidentment, s'ha d'instal·lar un servidor de PostgreSQL que és on guardarem les dades. Per a realitzar aquest treball en tenim de sobres, però en un entorn professional, s'utilitzaria una base de dades més pensada per a guardar series de dades. Utilitzem el PostgreSQL perquè és la recomanada per Django i està plenament soportada. Ara mateix el Django no suporta cap base de dades NOSQL però això podria canviar en el futur:

$ sudo apt-get install postgresql-8.4

Ara s'haurà de crear l'usuari i se'ns demanarà una contrasenya. En el nostre cas l'usuari és dirà "django":

$ sudo passwd postgres
$ sudo -u postgres createuser -P django

En aquest punt, hem de crear la taula on el Django posarà les seves dades. La taula també s'anomena django en la nostra configuració:

$ su postgres
$ psql template1
template1= CREATE DATABASE django OWNER django ENCODING 'UTF8';
$ exit

Ara s'ha d'editar el fitxer de configuració per permetre l'accés del sercidor Django a la base de dades que hem acabat de crear:

$ sudo vi /etc/postgresql/8.4/main/pg_hba.conf

I s'afegirà la línia següent al final del fitxer:

local django django md5

Finalment es reiniciarà el servei per tal d'agafar la nova configuració:

$ sudo /etc/init.d/postgresql restart

En aquest punt, ja està tot llest per a iniciar el servidor, simplement s'ha de copiar el codi del servidor en la carpeta on volem que s'executi i sincronitzar la taula "django"de la base de dades. Es farà amb la següent ordre:

$ python manage.py syncdb

Un cop hem fet aquesta instal·lació inicial, el servidor REST ja està a punt per funcionar. Podem comprovar que funciona correctament tot escrivint l'ordre que veurem a continuació. Si el servidor s'inicia sense cap error, és que tot està funcionant correctament:

$ python manage.py runserver

6.2.2 Implementació de la recuperació de dades bancàries

El codi següent implementa la recuperació de dades bancàries des del servidor de Yahoo Finance:

```
def get_historical_data_for_ticker(symbol, start_date, end_date):
  import urllib
  url = 'http://ichart.yahoo.com/table.csv?s=%s&' % symbol + \
    'd=%s&' % str(int(end_date[4:6]) - 1) + \
    'e=%s&' % str(int(end_date[6:8])) + \
    'f=%s&' % str(int(end_date[0:4])) + \
    'g=d&' + \
    'a=%s&' % str(int(start_date[4:6]) - 1) + \
    'b=%s&' % str(int(start_date[6:8])) + \
    'c=%s&' % str(int(start_date[0:4])) + \
    'ignore=.csv'
  days = urllib.urlopen(url).readlines()
```

```
  data = [day[:-2].split(',') for day in days]
  return data

start_date = "20120101"
end_date = "20120131"

for symbol in tickers:
  get_historical_data_for_ticker(symbol, start_date, end_date)
```

6.2.3 Implementació del càlcul de la volatilitat utilitzant SVM

Per implementar el càlcul de la volatilitat utilitzem la llibreria sklern, veurem com funciona en cada un dels exemples següents: **Entrenament**

L'entrenament serveix per generar el model.

```
from sklearn.crossvalidation import train_test_split

# Divisio dels conjunt de dades
X, Xtest, y , ytest = train_test_split (
  finance.data,
  finance.target
  testsize=0.2)
```

Validació

La validació permet seleccionar el millor model.

```
from sklearn.crossvalidation import StratifiedKFold
cv = StratifiedKFold (y, k=10)

def train_classifier(x, y, param):
  from sklearn.svm import SVR
  clf = SVR(kernel='rbf')
  clf.fit(x, y)
  return clf

for param in [0.1, 1., 10.] :
  for train , val in cv:
    clf = train_classifier(
      x = x[train],
      y = y[train],
      param = param)
```

```
best_param = max(
  accs.iteritems(),
  key=lambda x: sum(x[1]),
  )[0]
```

Verificació

La verificació permet comprovar el resultat que ens ha donat el model.

```
clf = train_classifier(x, y, best_param)
accuracy = clf.score(x_test, y_test)

print('%.2f' % accuracy)
```

Predicció

Ara podem fer prediccions amb el model que hem trobat.

```
prediction = clf.predict(noves_dades)
```

6.2.4 Implementació del càlcul del portfòlio òptim

El codi següent implementa el càlcul del protfolio òptim:

```
def step_port_return(rt, lower_bound_weight,
                     upper_bound_weight):

  mubuy = -1E200
  musell = 1E200

  p1 = self
  if getattr(p1, "port_opt", None) is None:
      self.port_opt = p2 = copy.deepcopy(p1)
  else:
      p2 = self.port_opt

  weights = np.array([p2.weights[symbol] for symbol in p2.symbols])
  x = np.mat(weights).T

  mu = p2.calc_marginal_utility(rt)
```

```
for i in range(len(p2.symbols)):
    if x[i] < upper_bound_weight:
        if mu[i,0] > mubuy:
            mubuy = mu[i,0]
            ibuy = i

    if x[i] > lower_bound_weight:
        if mu[i,0] < musell:
            musell = mu[i,0]
            isell = i

if (mubuy-musell)<=0.0001:
    ibuy=isell=0

s = np.zeros(mu.shape)
s[isell]=-1.0
s[ibuy]=1.0
s = np.mat(s)

rates = np.array([p2.stocks[symbol].ratearray['rate']
  for symbol in p2.symbols])
C = np.mat(np.cov(rates))

k0 = s.T*mu
k1 = (s.T*C*s)/rt

amat = k0/(2*k1)
a = amat[0,0]

cu = k0*a - k1*(a**2)

symb_sell = p2.symbols[isell]
symb_buy = p2.symbols[ibuy]
if symb_sell == symb_buy:
    a = 0.0

if p2.weights[symb_sell]-a < lower_bound_weight:
    a = p2.weights[symb_sell]-lower_bound_weight
if p2.weights[symb_buy]+a >upper_bound_weight:
    a = upper_bound_weight-p2.weights[symb_buy]

p2.weights[symb_sell] = p2.weights[symb_sell]-a
p2.weights[symb_buy] = p2.weights[symb_buy]+a
```

```
  return a

def calc_port_rates():

  port_ratearray = None

  for symbol in self.symbols:
      srate = self.stocks[symbol].ratearray
      weight = self.weights[symbol]
      if port_ratearray is None:
          dts = srate['date']
          port_ratearray =  np.array(zip(dts, np.zeros(srate.shape)),
              dtype=srate.dtype)
      port_ratearray['rate'] += srate['rate'] * weight

  self.port_ratearray = port_ratearray
  return port_ratearray


rt=0.10
lower_bound_weight=-0.50
upper_bound_weight=1.5

a = 1.0
count = 0

while a>0.00001:
  a = step_port_return(rt, lower_bound_weight,
                       upper_bound_weight)
  count+=1

result = port_opt.evaluate_holdings()
variance = round(result[0],3)
ret = round(result[1]*100.,3)
```

7 Anàlisi dels resultats

En aquesta secció veurem quin és el resultat que obtenim tot analitzant les dades dels valors tot utilitzant les màquines de vectors de suport o SVM.

En primer lloc veurem quines són les assumpcions que fem i quines són les característiques generals de les dades dels mercats borsaris que ens interessen a l'hora de crear la nostra màquina de vectors de suport. Després veurem quin ha estat el procediment que hem seguit per tal d'obtenir els resultats i, finalment analitzarem aquests resultats per verificar quin és el gran d'encert que hem tingut tot seguint aquest procediment.

No entrarem en detalls de la part teòrica que ja hem vist en altres apartats anteriors, que se suposa coneguda, tot i que tornarem a mencionar alguns dels conceptes que ja s'han vist per tal que la secció estigui continguda en ella mateixa i no s'hagi d'anar mirant altres seccions.

7.1 Dades seleccionades

Per tal de treballar amb un període tancat i ben definit, en tota aquesta secció es treballarà sobre les dades que hem utilitzat en la part teòria del nostre treball, que són els valors de l'IBEX35 que van del 1 de gener del 2012 fins al 31 de gener del 2012.

No cal recordar aquí que es tracta d'un període bastant complicat de la borsa espanyola amb una gran volatilitat. Això vol dir que els resultats obtinguts en poden servir segurament per predir la volatilitat en un entorn de crisi econòmica com el que hi ha ara, però que amb altres circumstàncies i amb una economia més estable, els resultats segurament no serien els esperats, ja que el sistema faria unes perdiccions de volatilitat més altes del normal.

En el cas de l'índex s'ha fet una petita trampa que és utilitzar el valors que composen l'índex actualment en comptes de tenir en compte l'entrada i la sortida de valors que es van produir al llarg del 2012. Així doncs, les accions que s'han analitzat són les següents (independentment si eren a l'IBEX35 durant el 2012 o no) i s'han deixat a fora altres com Bànkia que ara mateix no hi formen part, tot i que estaven a l'índex durant el 2012.

La taula dels valors que s'han utilitzat és la següent:

- ABG.P - Abengoa
- ABE - Abertis

- ANA - Acciona
- ACX - Acerinox
- ACS - Actividades de Construcción y Servicios
- AMS - Amadeus
- MTS - Arcelor Mittal
- POP - Banco Popular Español
- SAB - Banco de Sabadell
- SAN - Banco Santander
- BKT - Bankinter
- BBVA - Banco Bilbao Vizcaya Argentaria
- BME - Bolsas y Mercados Españoles
- CABK - CaixaBank
- DIA - Distribuidora Internacional de Alimentación
- ENG - Enagás
- ELE - Endesa
- FCC - Fomento de Construcciones y Contratas
- FER - Ferrovial
- GAS - Gas Natural
- GRF - Grifols
- IAG - International Airlines Group
- IBE - Iberdrola
- ITX - Industria de Diseño Textil
- IDR - Indra Sistemas
- JAZ - Jazztel
- MAP - MAPFRE
- TL5 - Mediaset España Comunicación
- OHL - Obrascón Huarte Lain

- REE - Red Eléctrica Corporación
- REP - Repsol
- SYV - Sacyr Vallehermoso
- TRE - Técnicas Reunidas
- TEF - Telefónica
- VIS - Viscofan

I les dades son els dies laborales en els quals van cotitzar durant el 2012 segons les dades proporcionades per la base de dades del Yahoo Finance.

7.2 Característiques de les dades financeres

En aquesta secció es definiran les principals mesures que s'utilitzen per a treballar amb sèries de dades financeres i les seves propietats. Per tal d'il·lustrar els conceptes amb propietat, utilitzarem les dades que tenim del Banc Santander i les representarem de manera gràfica per facilitar-ne la comprensió.

En aquesta primera figura podem veure el valor del Banc Santander durant el 2012:

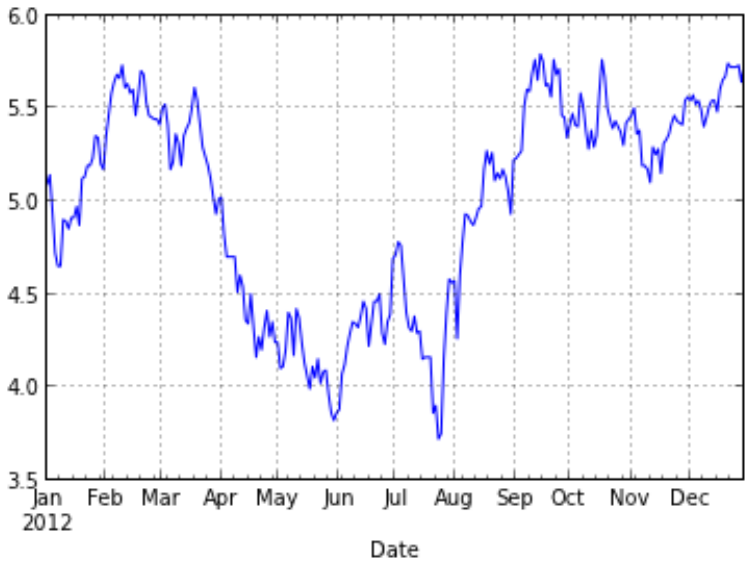

Figura 28: El valor del Banc Santander durant el 2012. *Font: Elaboració pròpia*

Per tal de calcular el retorn diari, s'utilitza la fòrmula següent:

$$r_i = ln(X_i/Xi - 1) \tag{33}$$

On i és l'índex de la seqüència (el preu del valor al final del dia) i X_i és el preu de l'acció en el temps t_i.

En la segona figura podem veure el retorn diari ajustat del Banc Santander durant el 2012:

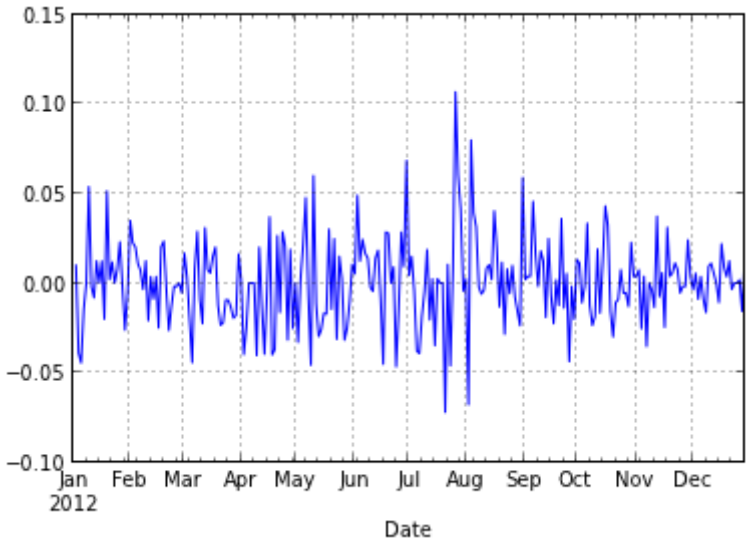

Figura 29: Retorn diari del Banc Santander durant el 2012. *Font: Elaboració pròpia*

En la figura podem veure com durant el mes de juliol i agost del 2012 hi va haver dies on el Banc Santander va tenir valors de retorn diaris del 10% o bé del -5%. Hem d'entendre que aquestes dades representen un període amb alta volatilitat en els mercats.

També podem veure com la mitjana dels valors de retorn és estacionaria i s'hauria d'acostar molt a zero.

Un concepte que és molt important i que ens interessa de manera molt especial és el de la volatilitat que, en termes generals, es descriu com la desviació estàndard dels retorns en un interval concret. Pel nostre propòsit, la volatilitat la definirem com a $v_i = |r_i|$ o $v_i = r_i^2$.

Per tant, el gràfic de la volatilitat diària del Banc Santader durant el 2012 seria el següent:

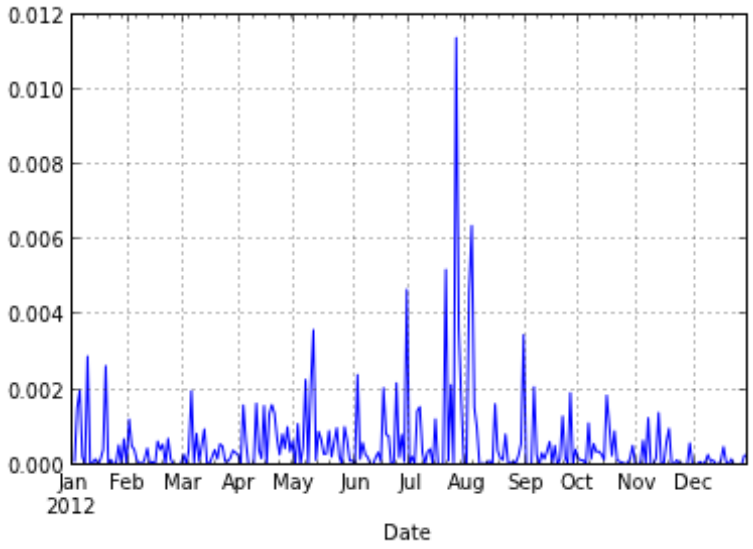

Figura 30: Volatilitat diària del Banc Santander durant el 2012. *Font: Elaboració pròpia*

Si comparem aquesta figura amb l'anterior veurem com el període de juliol i agost és quan hi ha més volatilitat en el valor, que és el què ja s'havia vist en l'altra figura.

7.3 Ús de la màquines de vectors de suport

Podem utilitzar les màquines de vectors de suport per tal d'analitzar la volatilitat de qualsevol acció de l'IBEX35 i predir els seus resultats, però en aquest cas el què farem és intentar predir l'IBEX35 mateix ja que és el què utilitzarem com a valor sense risc per tal de calcular la cartera eficient.

Això vol dir que fem una mica de trampa perquè en realitat no estem calculant la cartera eficient relacionant-la amb un actiu que no té cap mena de risc, sinó que el què estem fent és comparar-la amb l'IBEX35 això vol dir que el risc és en relació als mercats de valors, no en relació al que seria, per exemple, un dipòstit bancari. Que sí que podriem considerar relament sense cap mena de risc ja que, en teoria estan garantits per l'estat.

Utilitzarem la màquina de suport de vectors en tres passos que veurem a continuació de manera detallada: divisió del conjunt de dades, entrenament del model, validació per triar el model i evaluació del model.

7.3.1 Divisió del conjunt de dades

Per tal de poder validar el model i buscar la predicció òptima, el què primer farem primer es dividir les dades per tal d'utilitzar algunes dades per entrenar el model i la resta per validació. Com que utilitzem teòricament les dades de la volatilitat al llarg del 2012 per fer les prediccions, utilitzarem els últims tres mesos del 2012: octubre, novembre i desembre per fer la validació i la resta de dades (de gener a setembre) per a entrenar el model.

Això ens dóna un ús del 25

7.3.2 Entrenament del model

L'entrenament de les dades pel model el realitza automàticament el *sklearn* els paràmetres que es passen són els següents:

```
clf = SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                 epsilon=0.1, gamma=0.0, kernel='rbf', max_iter=-1,
                 probability=False, shrinking=True, tol=0.001,
                 verbose=False)
```

[language=Python]

Com es pot veure es fa servir el kernel RBF perquè és el què ens dóna millors resultats a l'hora de fer prediccions la qual cosa és consistent amb el fet que sigui més avançat.

7.3.3 Validació

En la part de validació s'avalua el model que ofereix un millor resultat tot realitzant una correlació amb el resultat donat i l'IBEX35 per veure quin és el que prediu un millor resultat durant aquest període. Es poden anar tunejant els diferents paràmetres de la SVM fins a obtenir un resultat el millor possible. La validació es fa de la manera següent:

```
from pandas.io.data import DataReader
from datetime import datetime
from pandas.stats.moments import rolling_std
import matplotlib.pyplot as plt

ibex35 = DataReader("^IBEX", "yahoo", datetime(2012,1,1),
    datetime(2012,12,31))
ibex35 = ibex35["Adj_Close"]
```

```
ibex35_returns = (ibex35/ibex35.shift(1) - 1)
ibex35_std = rolling_std(ibex35_returns, 10)

print ibex35.corr(model)
```

7.3.4 Selecció del model

Amb les dades obtingudes se selecciona el model que ens dóna un resultat millor i l'apliquem per a predir la volatilitat de l'IBEX35 en durant el 2012 per tal de generar una cartera òptima a partir dels altres paràmetres. S'ajusten els pesos dels valors en relació amb aquesta volatilitat que s'ha predit.

7.3.5 Evaluació

L'evaluació del model depen en gran part del període que utilitzem a l'hora de fer les prediccions. Si apliquem les prediccions sobre l'IBEX durant tot el 2012 obtenim el resultat següent:

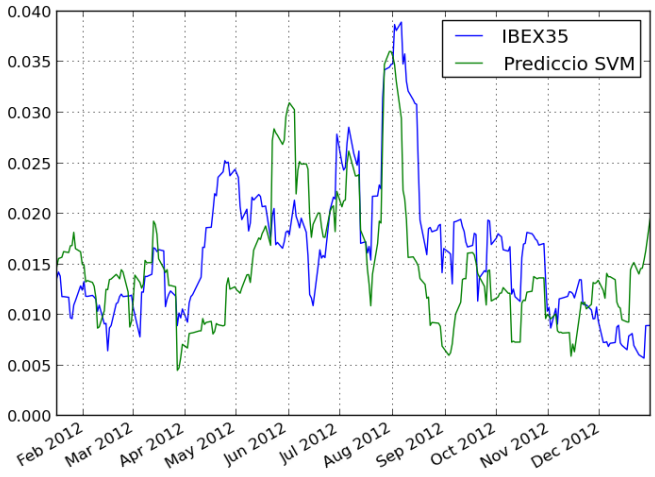

Figura 31: Predicció de la volatilitat de l'IBEX35 durant el 2012. *Font: Elaboració pròpia*

La correlació durant aquest període es d'un 0.57, però si escollim períodes més curts podem arribar a una correlació de gairebé un 0.97. Un fet curiós

és que cap a finals d'any sembla predir la recuperació de principis del 2013 amb la qual cosa seleccionant un període diferent de temps, els resultats milloren bastant.

En tot cas l'anàlisi és del 2012 i aquests són els resultats que s'han obtingut.

8 Conclusions

El treball aquest ha sigut una bona experiència per aprendre moltes coses noves i com es poden utilitzar per tal de crear un programa per a la plataforma Android que serveixi per una cosa útil que no és en cap cas trivial. El contingut teòric en aquest treball és molt important ja el tema de l'optimització de carteres té al darrere molta teoria financera i estadística que s'ha de comprendre bé abans de poder realitzar tota la feina pràctica que hi ha al darrera.

Així doncs, aquest treball tenia dos reptes: una forta base teòrica i intentar convertir tota aquesta base teòrica en una aplicació en particular on part d'aquesta complexitat també s'ha de programar.

Una de les coses negatives és que he perdut molt temps amb tota aquesta arquitectura de client-servidor en comptes de fer-ho tot en l'aplicació Android. Tot i així, no ho considero del tot un error ja que el fet d'utilitzar un servidor m'ha permés implementar tota la funcionalitat principal en Python, que té unes llibreries fantàstiques i fàcils d'utilitzar per fer anàlisi de dades: *numpy* i *pandas* són especialment bones en la seva feina. La programació d'aquesta part en Java hauria sigut segurament un repte o hauria hagut d'utilitzar llibreries que no estan pensades en funcionar en una plataforma com és Android sinó en grans servidors.

El fet de treballar en l'anàlisi i la predicció de la volatilitat és pel fet que hi ha diferents models que ens donen uns resultats bastant raonables sense utilitzar Intel·ligència Artificial, com és el *GARCH*, això vol dir que hi ha mètodes matemàtics que ens permeten obtenir uns resultats relativament bons si ho apliquem a casos concrets.

Aquest no és el cas de totes les prediccions de la borsa. El mercat bursàtil en general té molt soroll i és molt difícil predir res de res. És especialment complicat en el cas del retorns, ja que les correlacions acostumen a ser de pocs minuts (o fins i tot segons) i no segueixen cap model estadístic que sigui fàcil d'analitzar.

La volatilitat és un cas a part en aquest àmbit ja que té m̈emòriaa̋de mesos i és molt més fàcil de traçar. A més a més, si utilitzem un índex ponderat en comptes d'un valor en sí, hi ha molta menys volatilitat i és molt més fàcil de crear un model ben ajustat.

El fet d'utilitzar un any tant volàtil com el 2012 i una borsa com l'espanyola on el seu índex està sobrecarregat de bancs (el Banc Santander té un 17l'índex) fa que els resultats no haguin estat tan bons com si hagués utilitzat, per exemple, la borsa americana i el SP500 de l'any 2004 o 2005. En tot cas, volia fer un exemple real i no intentar el més fàcil.

Un fet bastant curiós es que el meu model sembla predir la volatilitat a l'avançada en alguns punts. És possible que només sigui casualitat, però seria interessant d'investigar.

En tot cas, la predicció de la volatilitat, al contrari dels preus i dels retorns, no suposa directament cap benefici econòmic que es pugui aprofitar per tal de fer diners, sinó que simplement es pot aprofitar per a intentar reduir el risc d'inversions existents. La meva aplicació no és una fòrmula màgica que preten fer millonari a ningú, sinó una simple ajuda als inversors si busquen una eina per tal de reduir el risc de les seves carteres de valors.

Referències

[1] Open Handset Alliance (2013), *Android Overview.* http://www.openhandsetalliance.com/android_overview.html. Data d'accés: 21/03/2013.

[2] Android Open Source Project (2013), *About the Android Open Source Project.* http://source.android.com/source/index.html. Data d'accés: 21/03/2013.

[3] Ben Englin (2005), *Google Buys Android for Its Mobile Arsenal.* http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm. Data d'accés: 22/03/2013.

[4] Handset Alliance (2007), *Industry Leaders Announce Open Platform for Mobile Devices.* http://www.openhandsetalliance.com/press_110507.html. Data d'accés: 22/03/2013.

[5] Jennifer Martinez (2008), *More mobile phone makers back Google's Android.* http://www.reuters.com/article/2008/12/10/openhandsetidUSN0928595620081210. Data d'accés: 22/03/2013.

[6] Android Developers (2010), *Android 2.3 Platform Highlights.* http://developer.android.com/sdk/android-2.3-highlights.html. Data d'accés: 22/03/2013.

[7] Mithun Chandrasekhar (2011), *Google's Android Event Analysis.* http://www.anandtech.com/show/4150/googles-android-event-analysis/2. Data d'accés: 22/03/2013.

[8] Liam Spradlin (2013), *Breaking: Android 4.2.2 Update Rolling Out To GSM Galaxy Nexus, Nexus 7, Nexus 10.* http://www.androidpolice.com/2013/02/11/breaking-android-4-2-2-build-jdq39-update-rolling-out-to-gsm-galaxy-nexus-nexus-7-nexus-10/. Data d'accés: 23/03/2013.

[9] Prince McLean (2009), *Canalys: iPhone outsold all Windows Mobile phones in Q2 2009.* http://www.appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html. Data d'accés: 23/03/2013.

[10] Tarmo Virki i Sinead Carew (2011), *Google topples Nokia from smartphones top spot.* http://uk.reuters.com/article/2011/01/31/oukin-uk-googlenokia-idUKTRE70U1YT20110131. Data d'accés: 23/03/2013.

[11] comScore (2010), *comScore Reports September 2010 U.S. Mobile Subscriber Market Share.* http://www.comscore.com/Press_Events/Press_Releases/2010/11/comScore_Reports_September_2010_U.S._Mobile_Subscriber_Market_Share. Data d'accés: 24/03/2013.

[12] Lance Whitney (2010), *Android hits top spot in U.S. smartphone market.* http://news.cnet.com/8301-1035_3-20012627-94.html. Data d'accés: 24/03/2013.

[13] Quentyn Kennemer (2011), *350,000 Activations Per Day, Says Schmidt.* http://phandroid.com/2011/02/15/350000-activations-per-day-saysschmidt. Data d'accés: 24/03/2013.

[14] James Deruvo (2011), *Nielsen – Users want more Android.* http://androidcommunity.com/nielsen-users-want-more-android-20110426. Data d'accés: 24/03/2013.

[15] Official Google Blog (2011), *Android: momentum, mobile and more at Google I/O.* http://googleblog.blogspot.com/2011/05/androidmomentum-mobile-and-more-at.html. Data d'accés: 24/03/2013.

[16] Gartner, Inc. (2012), *Android Marks Fourth Anniversary Since Launch with 75.0Third Quarter, According to IDC.* http://www.idc.com/getdoc.jsp?containerId=prUS23771812. Data d'accés: 24/03/2013.

[17] Donald Melanson (2013), *Eric Schmidt: Google now at 1.5 million Android activations per day.* http://www.engadget.com/2013/04/16/eric-schmidt-google-now-at-1-5-million-android-activations-per/. Data d'accés: 24/03/2013.

[18] Gartner, Inc. (2013), *Gartner Says Asia/Pacific Led Worldwide Mobile Phone Sales to Growth in First Quarter of 2013.* http://www.gartner.com/newsroom/id/2482816. Data d'accés: 24/03/2013.

[19] Android Open Source Project (2013), *Android, the world's most popular mobile platform.* http://developer.android.com/about/index.html. Data d'accés: 25/03/2013.

[20] Gubatron.com (2010), *How many lines of code does it take to create the Android OS?.* http://www.gubatron.com/blog/2010/05/23/how-manylines-

of-code-does-it-take-to-create-the-android-os/. Data d'accés: 24/03/2013.

[21] Ben Leslie (2007), *Native C applications for Android.* http://benno.id.au/blog/2007/11/13/android-native-apps. Data d'accés: 24/03/2013.

[22] Markowitz, H.M. (1959). Portfolio Selection: Efficient Diversification of Investments. New York: John Wiley & Sons.

[23] Ruppert, David. (2011). Statistics and Data Analysis for Financial Engineering. New York: Springer.

[24] Django Software Foundation (2013), *The Django Project.* https://www.djangoproject.com/. Data d'accés: 18/05/2013.

[25] Jesper Noehr (2013), *Django Piston.* https://bitbucket.org/jespern/django-piston/wiki/Home. Data d'accés: 18/05/2013.

[26] Enthought, Inc. (2013), *Scientific Tools for Python.* http://www.scipy.org/. Data d'accés: 18/05/2013.

[27] The Numpy developers (2013), *The Numpy Project.* http://www.numpy.org/. Data d'accés: 18/05/2013.

[28] Wes McKinney (2013), *Python Data Analysis Library.* http://pandas.pydata.org/. Data d'accés: 18/05/2013.

[29] scikit-learn developers (2013), *scikit-learn: machine learning in Python.* http://scikit-learn.org/stable/. Data d'accés: 18/05/2013.

[30] The JSON Project (2013), *Introducing JSON.* http://www.json.org/. Data d'accés: 18/05/2013.

[31] Yahoo! Inc (2013), *Yahoo! Finance Ticker-Based Dynamic RSS Feeds.* http://developer.yahoo.com/finance/. Data d'accés: 18/05/2013.

[32] The Wikipedia Authors (2013), *NoSQL.* https://en.wikipedia.org/wiki/NoSQL. Data d'accés: 18/05/2013.

[33] The Apache Software Foundation (2013), *The Apache Cassandra Project.* http://cassandra.apache.org/. Data d'accés: 18/05/2013.

[34] The Apache Software Foundation (2013), *A Database for the Web.* http://couchdb.apache.org/. Data d'accés: 18/05/2013.

[35] NVIDIA Corporation (2013), *Computational Finance.* http://www.nvidia.com/object/computational_finance.html. Data d'accés: 19/05/2013.

[36] The Ubuntu Project (2013), *Ubuntu 12.04.2 LTS (Precise Pangolin)*. `http://releases.ubuntu.com/precise/`. Data d'accés: 30/05/2013.

[37] Amazon Web Services, Inc. (2013), *Amazon Elastic Compute Cloud (Amazon EC2)*. `http://aws.amazon.com/ec2/`. Data d'accés: 30/05/2013.

[38] The Wikipedia Authors (2013), *Representational state transfer*. `http://en.wikipedia.org/wiki/Representational`$_s tate_t ransfer. Data d' accés$: 02/06/2013.

[39] The Python Project (2013), *Python Object Serialization*. `http://docs.python.org/2/library/pickle.html`. Data d'accés: 02/06/2013.

[40] The Wikipedia Authors (2013), *Model–view–controller (MVC)*. `http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller`. Data d'accés: 05/06/2013.

[41] Dacorogna MM, Gencay R., Muller U, Olsen RB, Pictet OV (2001). An Introduction to high-frequency finance. San Diego: Academic Press.

9 Apèndix A: Manual d'usuari

En aquest apèndix veurem la funcionalitat de l'aplicació des del punt de vista de l'usuari final. Això és, aquell que utilitza l'Aplicació Android per crear la seva cartera. Serà doncs un petit manual d'instruccions que serveix per mostrar com funciona l'aplicació.

9.1 Pantalla Principal

El primer que veiem quan fem clic sobre l'aplicació és la pantalla principal que ens ofereix dues opcions: la gestió de carteres i la gestió de símbols.

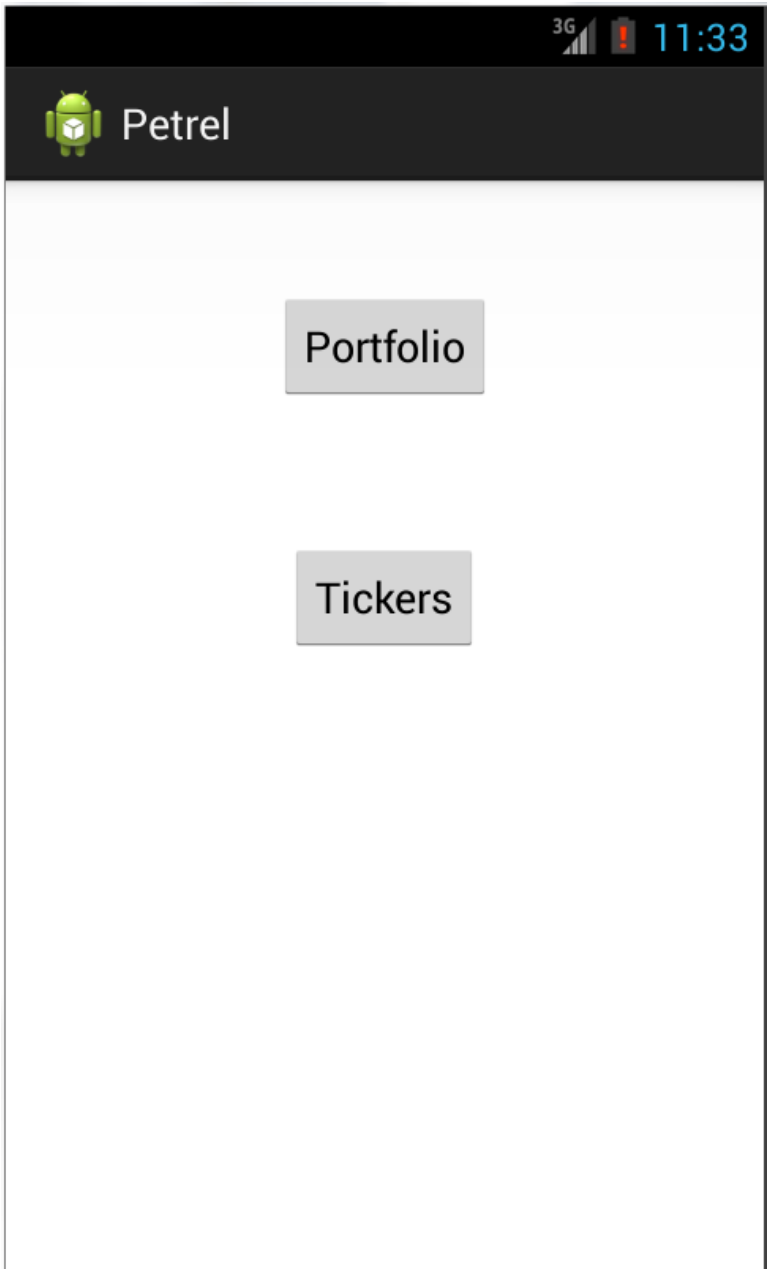

Figura 32: Pantalla principal de l'aplicació. *Font: elaboració pròpia*

9.2 Gestió de carteres

La pantalla de gestió de carteres ens ofereix la funcionalitat de crear, editar i optimitzar automàticament les nostres carteres. El primer que fa l'aplicació és comprovar si hi ha alguna cartera creada. En cas que no n'hi hagi cap, com

serà quan utilitzem l'aplicació per primera vegada, ens sortirà directament la pantalla de creació de carteres que es la que podem veure a continuació:

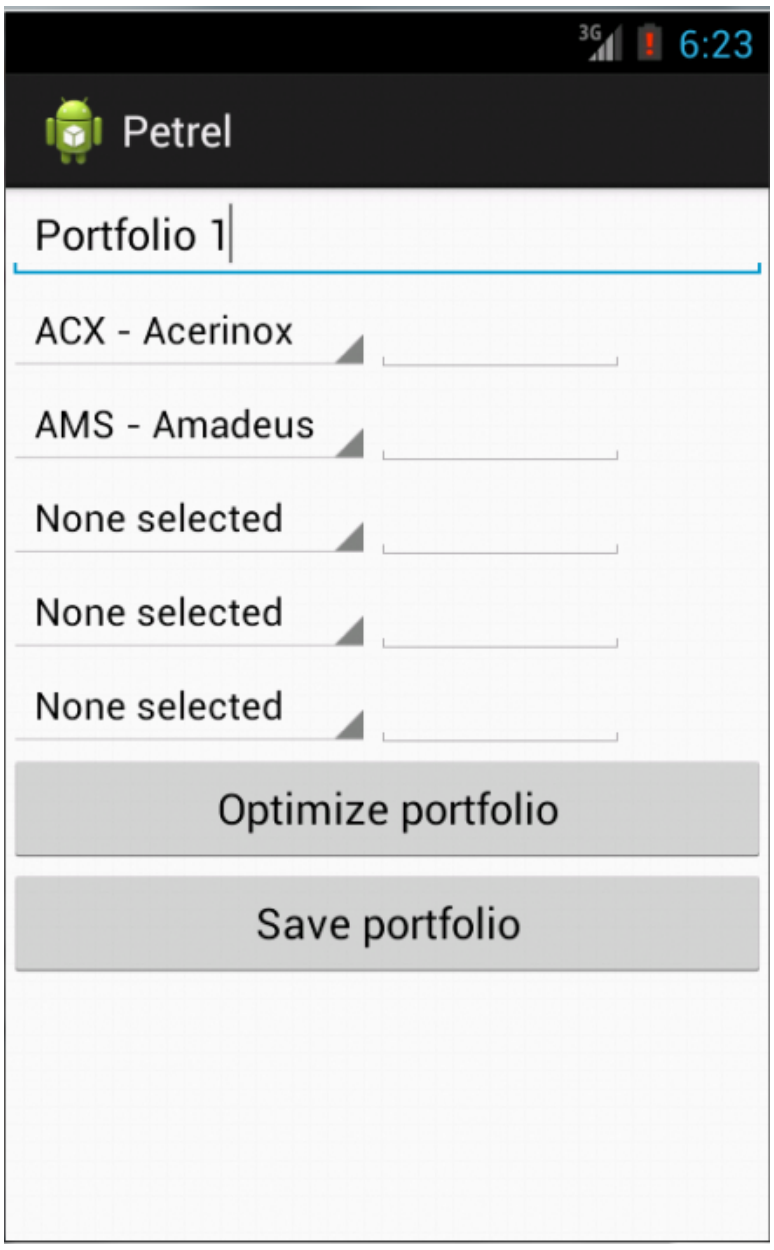

Figura 33: Pantalla de creació d'una cartera nova. *Font: elaboració pròpia*

En aquesta pantalla podem posar el nom que vulguem a la nostra cartera així com seleccionar els actius que hi volem posar. En aquesta primera versió podem posar- hi un màxim de 5 actius.

Un cop hem escollit els actius, hem de desar la cartera per tal de guardar-la a la base de dades. Ja que si no tenim la cartera desada a la base de dades, no la podem optimitzar.

Finalment disposem del botó d'optimitzar que enviarà la cartera amb els valors al servidor REST i aquest ens retornarà la cartera amb els pesos omplerts de manera automàtica.

En aquesta imatge podem veure el llista de carteres amb la nostra cartera seleccionada i optimitzada:

Figura 34: Pantalla de creació d'una cartera nova. *Font: elaboració pròpia*

9.3 Gestió de símbols

En aquesta pantalla podem consultar els símbols disponibles a la nostra aplicació.

En el cas que l'usuari no hagi executat mai anteriorment l'aplicació i no disposi de connexió a internet, es crearan els símbols automàticament a partir d'una plantilla de símbols que disposa el programa.

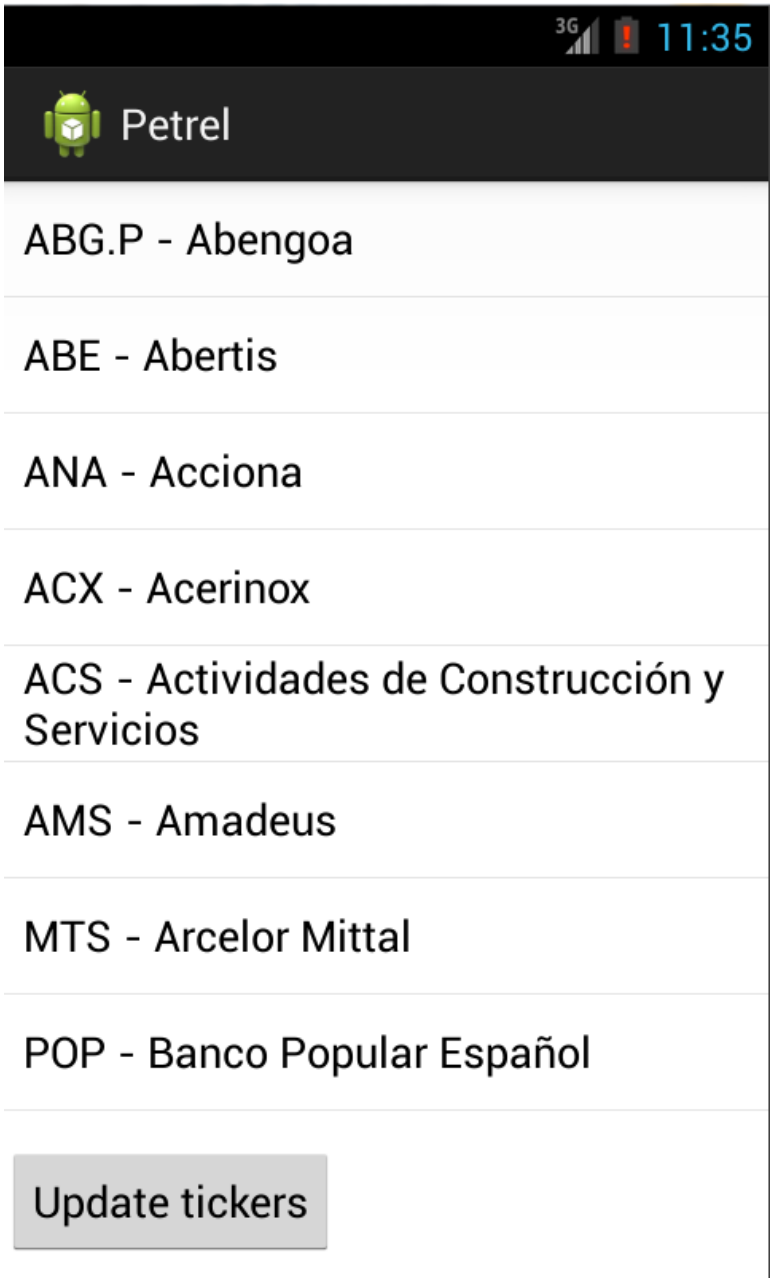

Figura 35: Pantalla de gestió de símbols. *Font: elaboració pròpia*

La llista de símbols s'actualitza periòdicament fent peticions al servidor. En

el cas que l'usuari vulgui forçar l'actualització, ho pot fer-ho tot fent clic al botó que té disponible a la part inferior de la pantalla:

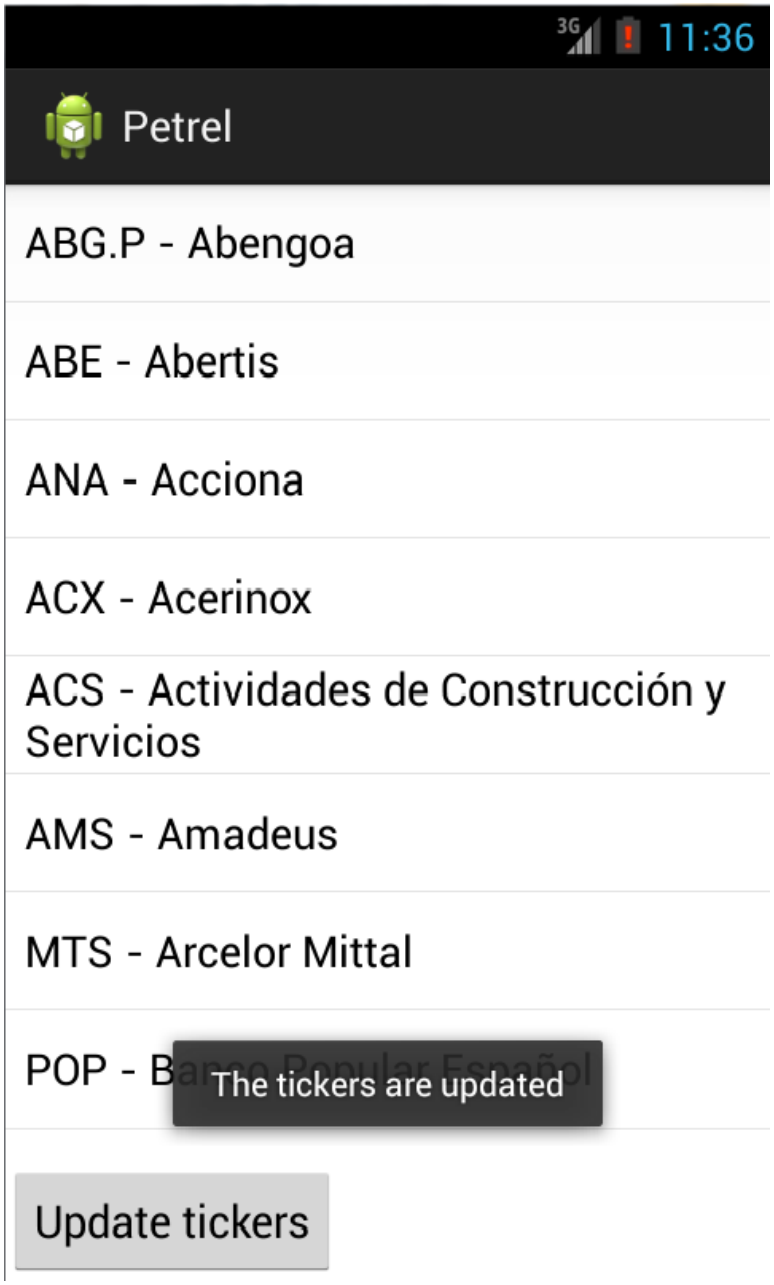

Figura 36: Actualització de símbols. *Font: elaboració pròpia*