UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Clasificadores multi-label en Scikit Learn Documentación Técnica

Presentado por Eduardo Tubilleja Calvo
en Universidad de Burgos — 31 de enero de 2018
Tutor: Dr. Álvar Arnaiz González
y Dr. Juan José Rodríguez Díez

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

La planificación es una parte importante de un proyecto. En esta parte se estima el trabajo, el tiempo y el dinero necesario para realizar el proyecto. Hay que analizar todas las partes que forman el proyecto, con esto sabemos los recursos que necesitaremos. Podemos dividir la planificación en planificación temporal y estudio de la viabilidad.

- Planificación temporal: Se elabora un calendario en el que se estima el tiempo que tardaremos en realizar cada una de las tareas del proyecto.
- Estudio de viabilidad: Si el proyecto es viable o no. Podemos dividirlo en dos:
  - Viabilidad económica: Se calculan los beneficios y costes del proyecto.
  - Viabilidad legal: Hay que ver si cumple todas las leyes, y en el software que tiene las licencias y la ley de protección de datos.

A.2. Planificación temporal

Para la planificación del proyecto hemos utilizado la metodología Scrum, aunque esta metodología está pensaba para trabajar en equipo, consiste en realizar unas entregas parciales y regulares del producto final, es recomendable para proyectos en entornos complejos, donde se necesitan obtener

resultados pronto, y la innovación, la competitividad, la flexibilidad y la productividad son fundamentales. En nuestro caso a través de GitHub:

- Creamos un Milestone correspondiente a la semana que estamos.
- Creamos las tareas que realizaremos esa semana habladas en la reunión semanal.
- Para gestionar el tiempo de las tareas utilizaremos ZenHub, que es una herramienta que incluye el navegador.
- Según vamos realizando las tareas las vamos cerrando, y así podremos observar el gráfico que nos muestra en *burndown chart*, en el que podremos ver el progreso.

A continuación se analizan y detallan las tareas realizadas en todos los sprints que se han realizado.

Sprint 0 (18/09/17 - 25/09/17)

En la reunión para planificar este sprint es cuando comenzó el proyecto. En ella se concreta por encima en que consiste el problema que vamos a llevar a cabo. En esta primera semana, se ha dedicado a la documentación de las herramientas que se van a usar y de distintos artículos, las tareas son:

- Refrescar los conocimientos de GitHub.
- Documentación sobre LaTeX [6], y su posterior instalación y configuración.
- Leer artículo Disturbing Neighbors [2].
- Documentación sobre Bagging [3].

Esta semana se han cumplido todas las tareas, aunque como todavía no sabía utilizar correctamente GitHub, no creé bien el Milestone por lo que esta semana no tenemos burndown.

Sprint 1 (26/09/17 - 02/10/17)

Esta semana, se ha dedicado a la documentación del funcionamiento de bagging y empezar con el primer clasificador Disturbing Neighbors, las tareas son:

- Reducir el conjunto de datos. Reducimos el conjunto de datos para quedarnos sólo con los datos que valoraremos para entrenar, estos datos se eligen mediante un subespacio aleatorio (un array aleatorio de 0 y 1).
- Calculamos las distancias al vecino más cercano para ello en nuestro caso utilizaremos la distancia de euclides. Los vecinos sobre los que calculamos dichas distancias son unas instancias del conjunto elegidas al azar.
- Funcionamiento del BaggingClassifier. Como nosotros también vamos a hacer un clasificador, entender como se usa el fit, predict y precict_proba.
- Entrenamiento. Entrenamos los datos mediante el método `fit` que tenemos que programar.
- Crear la clase Disturbing Neighbors. Es la clase en la que programaremos nuestro clasificador.

Esta semana se han cumplido todas las tareas.

Sprint 2 (02/10/17 - 10/10/17)

Esta semana, se ha dedicado a la corrección de errores de la semana pasada y a seguir avanzando con la clase, las tareas son:

- Corregir errores. En la reunión los tutores vieron algunos fallos que hay que corregir del método `fit`, en el que entrenamos nuestro conjunto de datos.
- Función `predict`. Empezamos con el siguiente método de la clase, en el que después de haber entrenado los datos ahora podemos predecir con ellos.
- Mostrar árbol de decisión. Como hemos entrenado nuestros datos podemos mostrar gráficamente los resultados para que sean más apreciables.

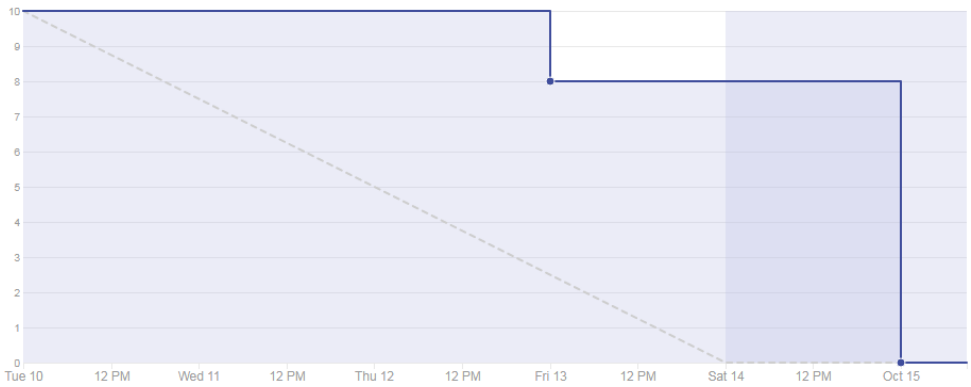

Figura A.1: Burndown del sprint 3

- Estructurar bien el código. Esto lo hacemos para que en un futuro sea más fácil trabajar.

Esta semana se han cumplido todas las tareas, esta semana terminé de entender como utilizar bien Github, por lo que no creé bien el Milestone por lo que esta semana no tenemos burndown.

Sprint 3 (10/10/17 - 15/10/17)

Esta semana, se ha dedicado a la corrección de errores de la semana pasada y a mostrar los resultados en un notebook, las tareas son:

- Corregir errores. La clase no funciona correctamente, ya que algunos de los métodos no hace lo que tienen que hacer, esta tarea no la conseguiremos acabar esta semana, tendremos que dedicar tiempo la próxima semana.

- Mostrar árbol en un notebook. Una vez conseguido que la primera versión del clasificador funcione correctamente, mostraremos un árbol de decisión en un notebook para poder observar mejor los resultados.

Esta semana se han cumplido todas las tareas, aunque una de las tareas estaba particionada entre esa semana y la siguiente.

En la figura A.1 se muestra el gráfico del Sprint 3.

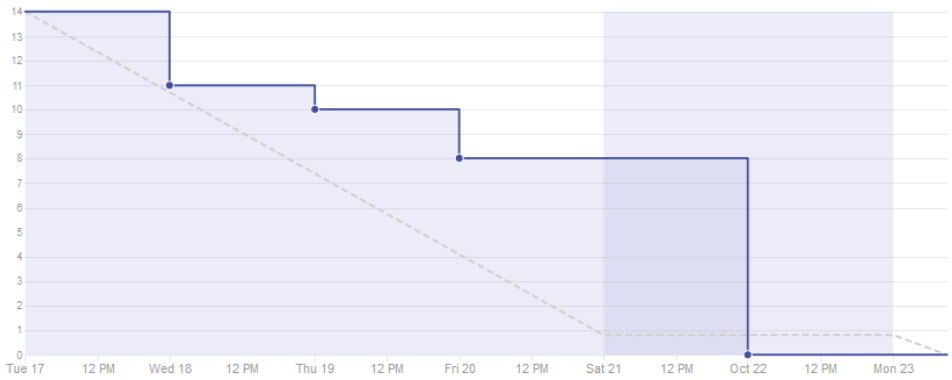

Figura A.2: Burndown del sprint 4

Sprint 4 (17/10/17 - 23/10/17)

Esta semana, se ha dedicado a corregir algún error más, a comentar el código y añadir algún nuevo método para que el clasificador funcione mejor:

- Comentar los métodos. Comentamos los diferentes métodos según el formato de Python [1].
- Semilla. Creamos una semilla, esto lo hacemos para inicializar los valores aleatorios y así conseguir que siempre empiece por el mismo, esto lo hacemos para cuando hacemos pruebas siempre las haga con los mismos datos.
- Vecinos molestones. No están calculados correctamente.
- Corregir fallos. Sigue dándome algún fallo, no hace lo que debería.
- Método `calculate_features`. Creamos un método en el que si recibe un numero menor de 1, es el porcentaje de las características con las que nos quedaremos, mientras que si lo que recibe es un número mayor de 1 es un número entero, que índica el valor exacto de las características que escogemos.

Esta semana se han cumplido todas las tareas, terminamos de corregir los fallos, aunque posteriormente iremos mejorando el clasificador.

En la figura A.2 se muestra el gráfico del Sprint 4.

Sprint 5 (24/10/17 - 30/10/17)

Esta semana, se ha dedicado mayormente a empezar a documentar en la memoria, a parte de alguna otra tarea:

- Semilla. No estaba hecha correctamente, hay que hacer alguna modificación.
- Subir estructura de proyecto a GitHub. Organizamos y estructuramos bien la estructura del proyecto en GitHub.
- Partición de entrenamiento. Dividimos el conjunto de entrenamiento para pasar una parte al fit y otra parte al predict.
- Memoria: Introducción. Sobre que va ir nuestro proyecto, una breve explicación.
- Memoria: Objetivos del proyecto. Los objetivos que cumpliremos en el proyecto.
- Memoria: Conceptos teóricos. Los conceptos necesarios para poder entender y trabajar en el proyecto.
- Anexo: Manual del programador. La información que necesitara un programador que quiera seguir con el proyecto.
- Anexo: Manual del usuario. Información que necesitará un usuario para poder usar las funcionalidad del proyecto.

Esta semana no se han cumplido todas las tareas, las partes del anexo de manual de programador y usuario no han podido llegar a realizar, aunque en el burndown muestre que están realizadas no es así.

En la figura A.3 se muestra el gráfico del Sprint 5.

Sprint 6 (31/10/17 - 05/11/17)

Esta semana, se ha dedicado mayormente a empezar a documentar en la memoria, a parte de alguna otra tarea:

- Clase funcional. Hacemos la clase funcional, como puede ser cambiar los for por map, para que luego se más rápido en la ejecución.
- Probar clase en Spyder. Para ver que todo la clase funciona correctamente la probamos.

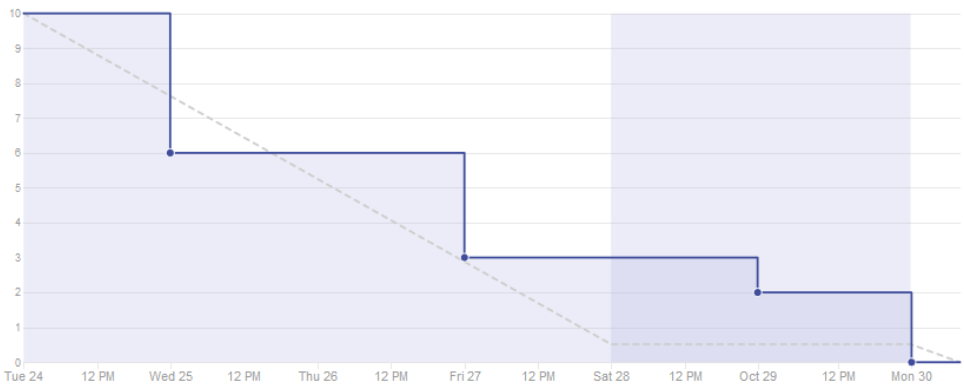

Figura A.3: Burndown del sprint 5

- Función predecir probabilidades. Creamos un nuevo método llamado `predict_proba` que nos devolverá las probabilidades.
- Memoria: Técnicas y herramientas. Añadiremos técnicas y herramientas necesarias para entender y poder trabajar con el proyecto.
- Método `train_test_split`: Usaremos el método para dividir nuestro conjunto de datos, dicho método es más eficaz que como lo estábamos haciendo la semana pasada.
- Correcciones de estilo y mejoras al código DN: Cambiaremos algunas variables a privadas, los comentarios los ponemos en inglés, y utilizaremos el chequeador de sintaxis `http://pep8online.com/` para tener un estilo adecuado.

Esta semana se han cumplido todas las tareas, al probar la clase en spyder ha dado algunos errores, sobre todo al importar la clase DisturbingNeihgbors, y algunos errores menores.

En la figura A.4 se muestra el gráfico del Sprint 6.

Sprint 7 (07/11/17 - 20/11/17)

Esta semana, se ha dedicado mayormente a empezar a documentar en la memoria, a parte de alguna otra tarea:

- Pasarle a bagging la clase DN. Probamos a pasarle a bagging nuestra clase Disturbing Neighbors. Como bagging no soporta múltiples sali-

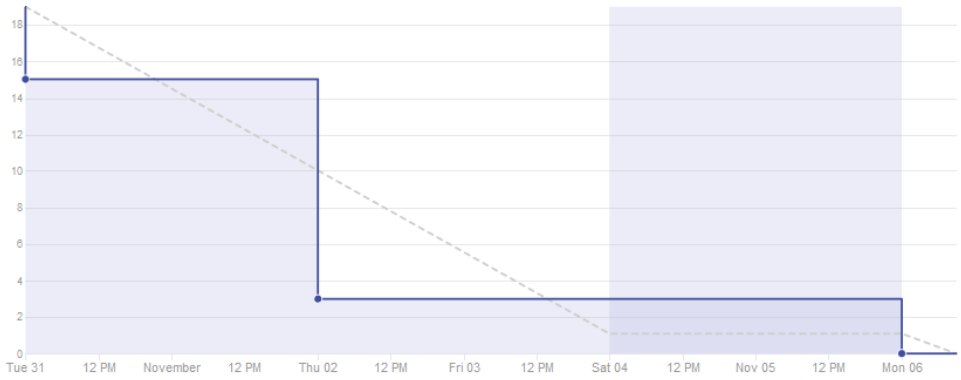

Figura A.4: Burndown del sprint 6

das, para ello lo solucionaremos utilizando OneVsRestClassifier, esto permitirá que bagging soporte múltiples salidas.

- Hacer funcional el método `nearest_neighbor`. Conseguir que el método `nearest_neighbor` funcione sin utilizar ningún bucle for.
- Memoria Conceptos teóricos. Añadir nuevos conceptos teóricos como Multi-Label o ensemble.
- Iteraciones sobre los métodos `fit`, `predict` y `predict_proba`. Hasta ahora solo se ejecutaba una vez cada método, pero para que esto sea eficaz queremos que se ejecute un número de iteraciones. El `fit` será el más fácil solo hay que realizar ese método el número de iteraciones requeridas. Para poder calcular el `predict` tenemos que ir guardando los valores del fit y calcular el promedio. Por último el `predict_proba` se calculará igual que el predict.
- Estructurar notebook. Estructuramos el notebook de jupyter, para que sea más fácil de entender.
- Método `cross validation`: Para crear los conjuntos de entrenamiento y test usar la validación cruzada.

Esta semana se han cumplido todas las tareas, aunque las tareas de hacer funcional el método `nearest_neighbor` y `cross validation`, me han hecho dedicar más horas de las esperadas, en el primer caso por mi poco conocimiento sobre el uso de los mapas en python y en el segundo caso por los diversos errores que me salían al intentar compilar.

En la figura A.5 se muestra el gráfico del Sprint 7.

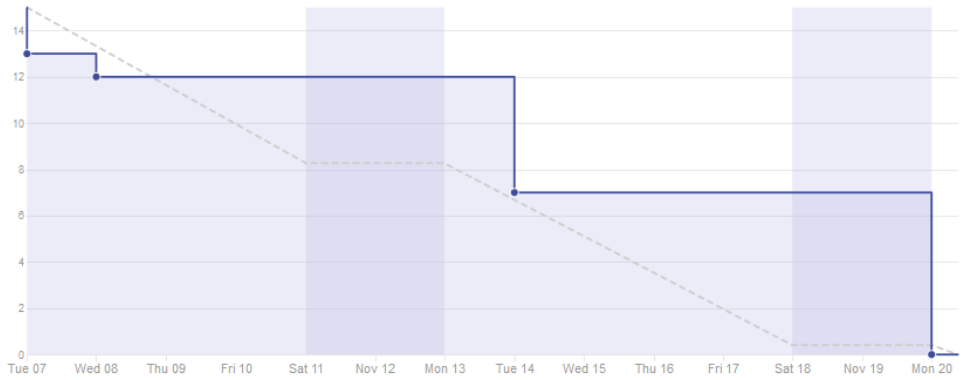

Figura A.5: Burndown del sprint 7

Sprint 8 (21/11/17 - 27/11/17)

Esta semana, se ha dedicado a terminar definitivamente el Disturbing Neighbors y documentarme sobre el siguiente algoritmo Random Oracles:

- Comentar Notebooks. Comentamos los notebooks, para ir explicando lo que realizamos en cada paso.
- Estilo. Como hemos realizado alguna modificación volvemos a comprobar que el estilo es el correcto.
- Excepciones. Ponemos excepciones para los casos que pueda ocurrir un error.
- Artículo Random Oracles [4]. Documentación sobre el nuevo algoritmo Random Oracles.
- Acabar `predict_proba` y comentar nuevos métodos. Acabamos el `predict_proba` para cuando hacemos iteraciones, y comentamos los nuevos métodos que hemos creado.
- Método `score` iteraciones. Hacer correctamente el método `score` cuando hacemos iteraciones.

Esta semana una de las tareas no se ha acabado por lo que aparecerá en el Sprint 9, ninguna de las tareas ha llevado mas tiempo del estimado, ya que ninguna de las tareas ha dado muchos problemas.

En la figura A.6 se muestra el gráfico del Sprint 8.

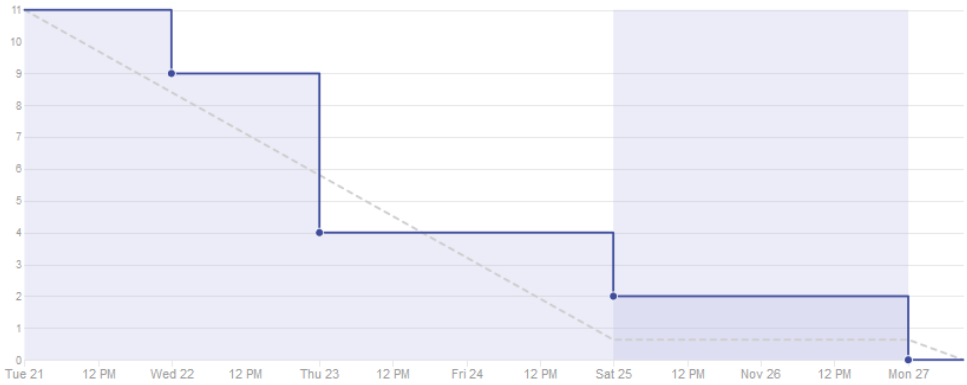

Figura A.6: Burndown del sprint 8

Sprint 9 (28/11/17 - 04/12/17)

Esta semana, se ha dedicado a probar unos datos reales en Disturbing Neighbors, a comenzar con el algoritmo Random Oracles y a otras tareas menores:

- Pasar un conjunto de datos reales a DN. Probamos el clasificador con un conjunto de datos reales, lo escogeremos de mulan o meka. Como el archivo es .arff, necesitaremos aprender como leer un archivo .arff en python.
- Fit Random Oracles. Creamos el `fit` y los métodos necesarios para su funcionamiento.
- Mejorar la descripción de las excepciones. Cambiamos la descripción de las excepciones para que no sean tan genéricas, y ayuden a entender al usuario el problema.
- Revisar Ortografía. Revisar los comentarios y todo lo escrito hasta la fecha para ver que no hay faltas de ortografía.
- Comentarios globales de los notebook. Poner comentarios encima de cada parte de código para que sea más entendible.

Esta semana se han cumplido todas las tareas menos la del predict del Random Oracles, las tareas de pasar un conjunto reales y el fit del random oracles, han llevado más tiempo del esperado, la primera por mi desconocimiento sobre los archivos arff, y la segunda porque no comprendí bien como debía funcionar el fit.

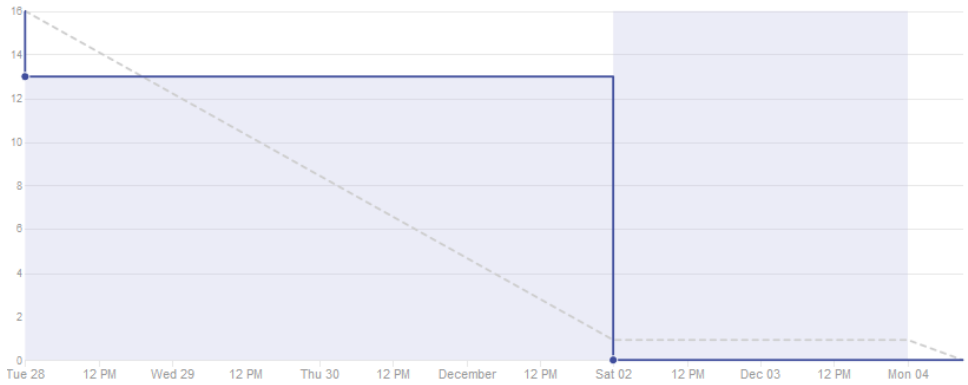

Figura A.7: Burndown del sprint 9

En la figura A.7 se muestra el gráfico del Sprint 9.

Sprint 10 (05/12/17 - 11/12/17)

Esta semana, se ha dedicado a dejar casi terminado el clasificador Random Oracles y a otras tareas menores:

- Predict Random Oracles. Realizamos el método `predict`.
- Avanzar en la documentación. Empezar con la documentación, ya que está muy retrasada.
- Mejorar método `fit`. Hacer funcional el código y si se puede reducirlo.
- Método `predict_proba`. Realizamos el método `predict_proba`.
- Iteraciones clase Random Oracle. Como en Disturbing Neighbors tenemos que hacer iteraciones sobre la clase Random Oracles.
- Revisión de código. Cambiar el nombre de algunas variables o funciones para que su significado tenga que ver con el nombre.
- Notebooks para el clasificador Random Oracle. Hacer notebooks para el clasificador Random Oracle, uno sin iteraciones, otro con iteraciones y otro con datos reales.

Esta semana se han cumplido todas las tareas, ninguna de las tareas ha dado muchos problemas a la hora de llevarla a cabo.

En la figura A.8 se muestra el gráfico del Sprint 10.

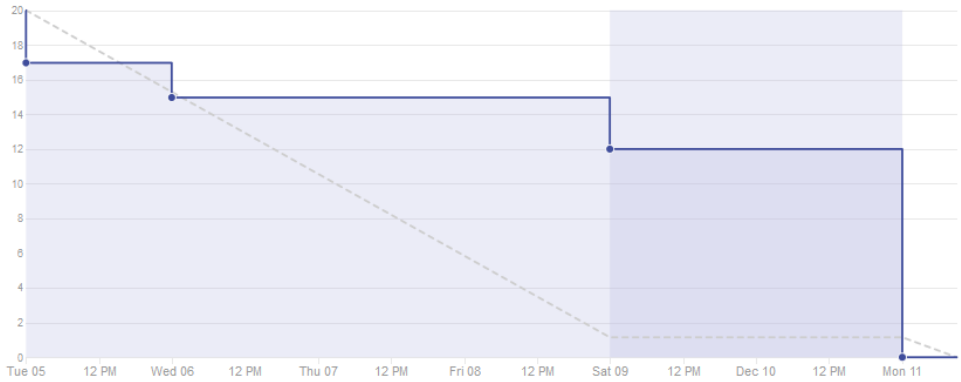

Figura A.8: Burndown del sprint 10

Sprint 11 (12/12/17 - 18/12/17)

Esta semana, se ha termina algunos detalles del Random Oracles, y se se ha empezado con el siguiente clasificador Random Forest:

- Artículo Rotation Forest [5]. Leer el artículo Rotation Forest, para entender el nuevo algoritmo que vamos realizar.
- Notebooks. Mejorar los notebooks, creando una función que según un parámetro nos llame al clasificador Disturbing Neighbors o al Random Oracles.
- Herencia. Usar la herencia, ya que una de las clases llamada `homogeneous_ensembles` , es la que hace las iteraciones, y sus herederos serían las clases de Disturbing Neighbours y Random Oracles.
- Rotation Forest dividir. Dividimos el conjunto de datos (X) en grupos de 3, usamos permutaciones para evitar repetidos.
- Rotation Forest PCA. Cuando ya tenemos los grupos, usaremos `PCA` sobre cada uno de los grupos, en los que haremos primero fit y luego transform.
- Rotation Forest fit. Volvemos a juntar los grupos en uno solo, y sobre este nuevo conjunto de datos (X') hacemos el `fit`.
- Correcto estilo Random Oracles. Utilizamos el chequeador de sintaxis de PEP8 online para que es el estilo que estamos siguiendo http://pep8online.com/.

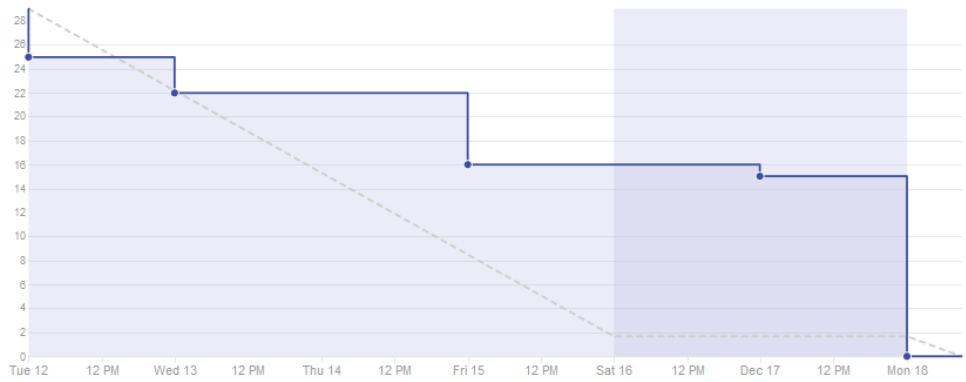

Figura A.9: Burndown del sprint 11

- Error encontrado en RO en el método `predict_proba`. Encontramos un error cuando realizamos iteraciones sobre el método `predict_proba` de la clase Random Oracles, ya que como lo probamos con un árbol cuando en una de las iteraciones si tiene el mismo labelset devuelve una predicción con tamaño uno, por lo tanto para corregir esto lo que hacemos es calcular las probabilidades con el método `predict`, ya que este sí funciona correctamente.

- Comentar correctamente las clases. Algunos comentarios están incompletos y algunos métodos les falta comentarios.

- Compactar los notebooks. Al final tener 3 notebooks en total, uno sin iteraciones, otro con iteraciones y otro con datos reales. Para diferenciar que clasificador queremos utilizar, instanciamos cada clasificador en una celda distinta, y después ejecutamos la celda del clasificador que queremos usar.

- Avanzar con la Memoria. Esta semana avanzaré en la parte de planificación temporal, añadimos algunos sprint.

Esta semana se han cumplido todas las tareas, el error encontrado en el método `predict_proba` me ha llevado mucho tiempo corregirlo, por lo que en la memoria no pude dedicarle todo el tiempo que quería.

En la figura A.9 se muestra el gráfico del Sprint 11.

Sprint 12 (19/12/17 - 25/12/17)

Esta semana, se ha dedicado a avanzar con el algoritmo Rotation Forest y se añadir herencias a los algoritmos creados anteriormente:

- `predict` RF. Programar el método `predict` de Rotation Forest.
- `predict_proba` RF. Programar el método `predict_proba` de Rotation Forest.
- Añadir a la herencia de DN y RO. Con la clase ya acabada ahora la añadiremos la herencia, dicha herencia es sobre `homogeneous_ensembles` y agregación sobre Rotation Forest.
- Notebook RF. Añadir a los notebooks creados el clasificador Rotation Forest, y ver que funciona correctamente.
- Seleccionar una muestra RF. Seleccionamos un muestra de cada uno de los subgrupos, por defecto el tamaño de dicha muestra será del 75 % de los datos. Dicha muestra la usaremos para entrenar pero para transformar le pasaremos el subgrupo entero.
- Seleccionar una instancias en base a las clases. Lo primero elegimos las distintas clases(y), después elegimos aleatoriamente una muestra de esas las distintas clases, y por último seleccionamos del conjunto de datos(X) las instancias correspondientes a la muestra de esas clases.
- Evitar código duplicado en los notebooks. La creación del conjunto de datos, la asignación de la semilla y la división del conjunto de datos en entrenamiento y test deben estar una única vez, y no repetidas en cada casilla de cada clasificador.

Esta semana se han cumplido todas las tareas, el tiempo empleado ha sido el previsto, ya que ninguna de las tareas ha dado demasiados problemas.

En la figura A.10 se muestra el gráfico del Sprint 12.

Sprint 13 (26/12/17 - 07/01/18)

Esta semana, se han tratado diversas temas, entre ellos, referencias en la documentación, dibujar gráficas para los clasificadores base, o que los algoritmos creados funciones también para Single-Label:

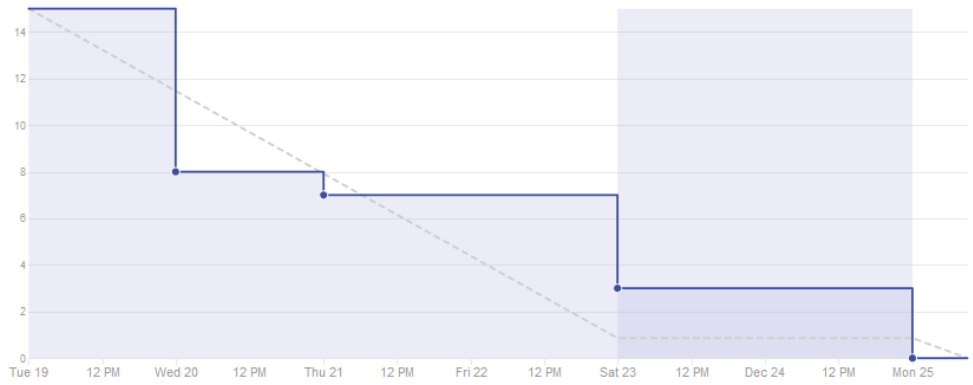

Figura A.10: Burndown del sprint 12

- Incluir referencias en la documentación. Añadimos referencias en la memoria y el anexo.
- Funcional RF. Hacemos funcional el método `fit` de Rotation Forest.
- Código repetido RF. Crear funciones para evitar el código repetido en Rotation Forest.
- Memoria. Añadir los sprint que faltan.
- Buscar información gráficas. Buscar información en otros Multi-Label de scikit learn para ver como dibujan los gráficos, que muestren los datos y como se dividen.
- Dibujar gráficas. Dibujamos las gráficas de cada uno de los algoritmos (DN, RO, RF), para ver como se divide el conjunto de datos.
- Comentar RF. Comentamos los métodos de Rotation Forest.
- Limpiar y sanear el repositorio. En el repositorio hay elementos que no deberían estar o deberían estar revisados.
- Single-Label para Disturbing Neighbors. El clasificador Disturbing Neighbors ahora mismo solo funciona para casos Multi-Label, y queremos que también funcione para casos Single-Label.
- Single-Label para Random Oracles. El clasificador Random Oracles ahora mismo solo funciona para casos Multi-Label, y queremos que también funcione para casos Single-Label.
- Memoria. Añadir los sprint que faltan.

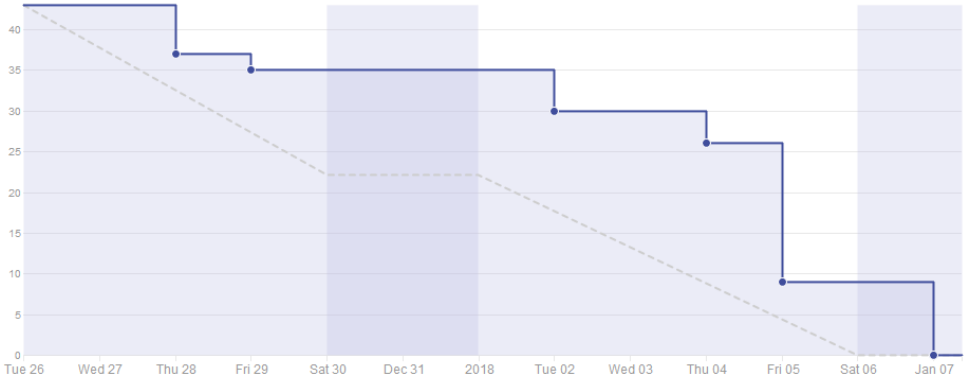

Figura A.11: Burndown del sprint 13

- Single-Label para Rotation Forest. El clasificador Rotation Forest ahora mismo solo funciona para casos Multi-Label, y queremos que también funcione para casos Single-Label.
- Correcciones sobre la memoria. Corregir las partes de las memorias y los anexos, revisadas por el tutor.

Esta semana se han cumplido todas las tareas, he dedicado algo más tiempo del pensado ya que surgió algún error.

En la figura A.11 se muestra el gráfico del Sprint 13.

Sprint 14 (08/01/18 - 14/01/18)

Esta semana, se ha avanzado con la memoria, se han hecho algunas correcciones y mejoras:

- Corregir los notebook. Corregir los notebook para que funcionen correctamente, ya que al descargarlos de Github no encuentra bien las librerías.
- Correcciones menores. Corregir fallos menores, entre ellos:
  - En Rotation Forest utilizar el conjunto reducido según las clases (instance_classes).
  - En Random Oracles ver porque no funciona al dibujar la gráfica.
  - En las clases ensemble modificar el base_estimator.

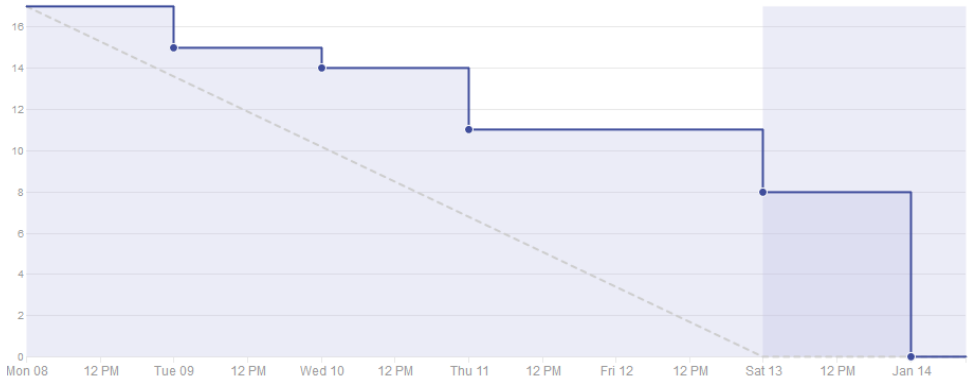

Figura A.12: Burndown del sprint 14

- Memoria correcciones. Corregir las partes de la memoria y anexos, que me han dicho los tutores..
- Comentar ensembles. Comentar los métodos ensembles, corregir los base, ya que estos no son ensembles.
- Mejorar el cómo detectar si es o no Multi-Label. En vez de la comprobación que hago, utilizar el método `is_multilabel`.
- `predict_proba` Random Oracles. Error encontrado en el `predict_proba`, no devuelve los valores de forma correcta..

Esta semana se han cumplido todas las tareas, en la tarea de las correcciones mínimas he tenido que dedicar más tiempo del esperado.

En la figura A.12 se muestra el gráfico del Sprint 14.

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

B.2. Objetivos generales

B.3. Catalogo de requisitos

B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

C.1. Introducción

En esta parte del anexo definiremos como se han resuelto los objetivos expuestos anteriormente.

C.2. Diseño de datos

El proyecto cuenta con las siguientes entidades:

- **Homogeneous Ensemble**: Es la superclase de los distintos algoritmos que hemos realizado, todos tienen unos parámetros en común. Por defecto, nos hace un `ExtraTreeClassifier`.
- **Disturbing Neighbors**: El método crea características nuevas que se agregarán al conjunto de datos del clasificador base. Dichas características se calculan con el clasificador Nearest Neighbour(NN), construido a partir de unas instancias seleccionadas al azar. Para probar la eficacia utilizamos árboles de decisión como clasificador base.
- **Random Oracles**: Cada clasificador del conjunto se reemplaza por un miniensemble de un par de subclasificadores con un oráculo para elegir entre ellos.
- **Rotation Forest**: Este método genera conjuntos de clasificadores basados en la extracción de características. Crea un conjunto de entrenamiento para un clasificador base, este conjunto se divide al azar

en subconjunto. La idea es mejorar la precisión y diversidad dentro del conjunto. La diversidad se basa en la extracción de características para cada clasificador base.

Diagrama de clases general

En esta parte vamos hacer una breve explicación de la función que tiene cada clase.

- **BaseEstimator**: Es la que utilizan todos los clasificadores de Scikit-Learn.
- **BaseEnsemble**: Hereda de **BaseEstimator**.Es la clase que utilizan todos los ensembles.
- **ClassifierMixin**: Es la clase para todos los clasificadores de Scikit-Learn, ya que esta tiene el método *score*.
- **HomogeneousEnsemble**: Hereda de **BaseEnsemble**. Es una superclase, que es la que utilizan los *ensembles* que hemos realizado, ya que todos tienen unos parámetros en común.
- **DisturbingNeighbors**: Hereda de **HomogeneousEnsemble**. Esta clase es un ensemble que usa el clasificador **BaseDisturbingNeighbors**, para realizar las predicciones de los conjuntos de datos.
- **RandomOracles**: Hereda de **HomogeneousEnsemble**. Esta clase es un ensemble que usa el clasificador **BaseRandomOracles**, para realizar las predicciones de los conjuntos de datos.
- **RotationForest**: Hereda de **HomogeneousEnsemble**. Esta clase es un ensemble que usa el clasificador **BaseRotationForest**, para realizar las predicciones de los conjuntos de datos.
- **BaseDisturbingNeighbors**: Hereda de **ClassifierMixin** y de **BaseEstimator**, y es una agregación de **DisturbingNeighbors**. Es un clasificador base.
- **BaseRandomOracles**: Hereda de **ClassifierMixin**, y es una agregación de **RandomOracles**. Es un clasificador base.
- **BaseRotationForest**: Hereda de **ClassifierMixin** y de **BaseEstimator**, y es una agregación de **RotationForest**. Es un clasificador base.

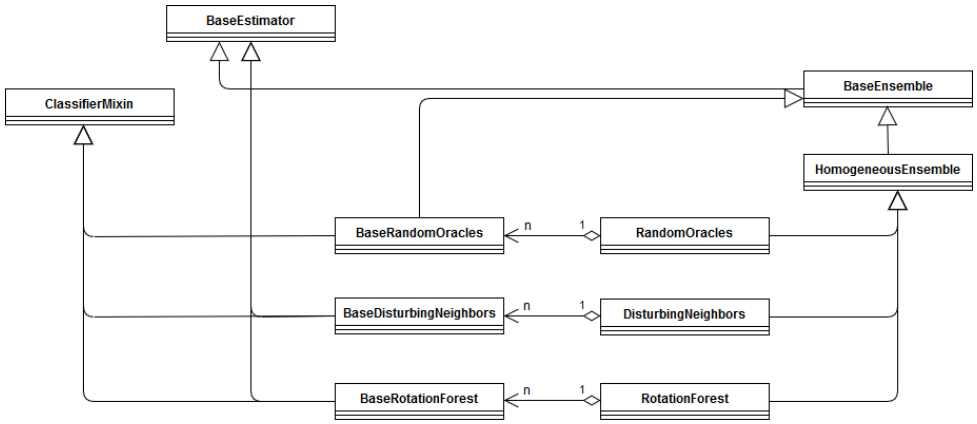

Figura C.1: Diagrama de clases general

Podemos ver un esquema de como son las relaciones en C.1

Diagrama de clases implementadas

En este diagrama podremos apreciar cada una de las clases implementadas, y sus parámetros y métodos C.2.

- **HomogeneousEnsemble**: Los parámetros que contiene esta clase son base_estimator, que por defecto tiene **ExtraTreeClassifier**, otro sería el nº de estimadores, que son las iteraciones que haremos sobre nuestro clasificador base. Tiene tres métodos *fit*, *predict* y *predict_proba*, en donde se calcula el promedio de los clasificadores base.

- Los tres ensembles **DisturbingNeighbors**, **RandomOracles** y **RotationForest**, tienen los parámetros que son específicos de cada clasificador base.

- Los clasificadores base **BaseDisturbingNeighbors**, **BaseRandomOracles** y **BaseRotationForest**, cada uno tiene sus parámetros específicos, y cada uno tiene los métodos necesarios para predecir el conjunto de datos.

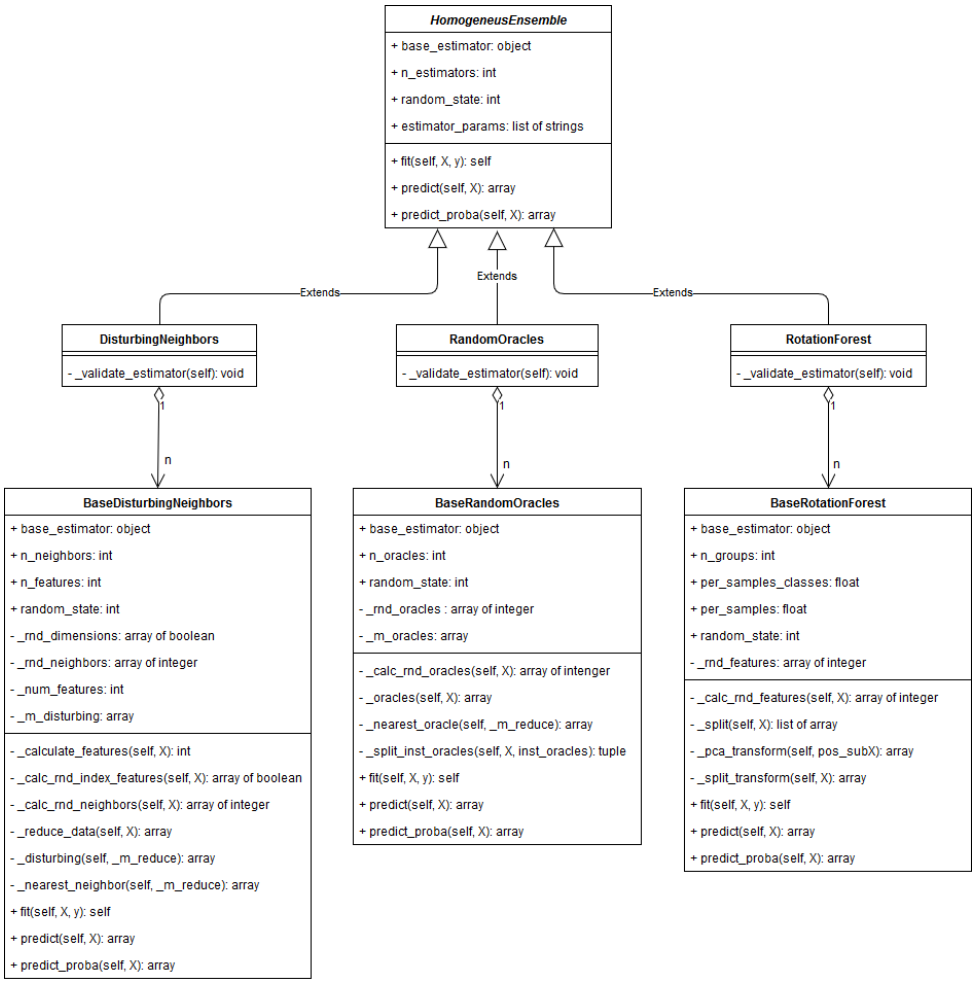

Figura C.2: Diagrama de clases implementadas

C.3. Diseño procedimental

Diagrama de secuencias

Se ha realizado un diagrama de secuencias de como seria un ejemplo de ejecución del *ensemble* Disturbing Neighbors, que podemos ver en C.3.

Los pasos que se han llevado son:

- Creamos el clasificador `DistubingNeighbors` que a su vez este creara el clasificador base `BaseDisturbingNeighbors`, y este a su vez `DecicisionTreeClassifier`.

- Una vez creado el clasificador, tendremos tantos clasificadores base como iteraciones tenga nuestro *ensemble* DisturbingNeighbors. Cada uno de estos clasificadores base entrara el conjunto de datos que le pasamos.
- Una vez tengamos nuestros clasificadores base entrenados, lo próximo es hacer la predicción de cada uno de ellos. Que igual que en el entrenamiento se hace sobre cada uno de los clasificadores. Pero lo que al final devolvemos es el promedio de todas las predicciones de los clasificadores base.
- Por último realizamos las predicciones de probabilidad, que estas lo que nos devolverán será la probabilidad de cada una de las clases de que sean 1 o 0. Al igual que en las predicciones, se calcula el promedio de todas. Aunque en la imagen no este no está reflejado, ya que básicamente sería igual que las predicciones.

C.4. Diseño arquitectónico

En esta sección, vamos a explicar y mostrar de forma general como esta diseñado el proyecto y como es la distribución de paquetes, que podemos ver en C.4.

Vamos hacer una descripción sobre cada paquete y su contenido.

- Sklearn-ubu: Este paquete contendrá todos los ficheros del código fuente de nuestro proyecto, es decir, los clasificadores base y los *ensembles*.
- Sklearn: Aunque este paquete no pertenece a este proyecto, como los algoritmos que hemos realizado, queremos implementarlos en la librería de Scikit-Learn, ha sido necesario hacer usos de este paquete en algunas ocasiones, por ello lo incluimos.

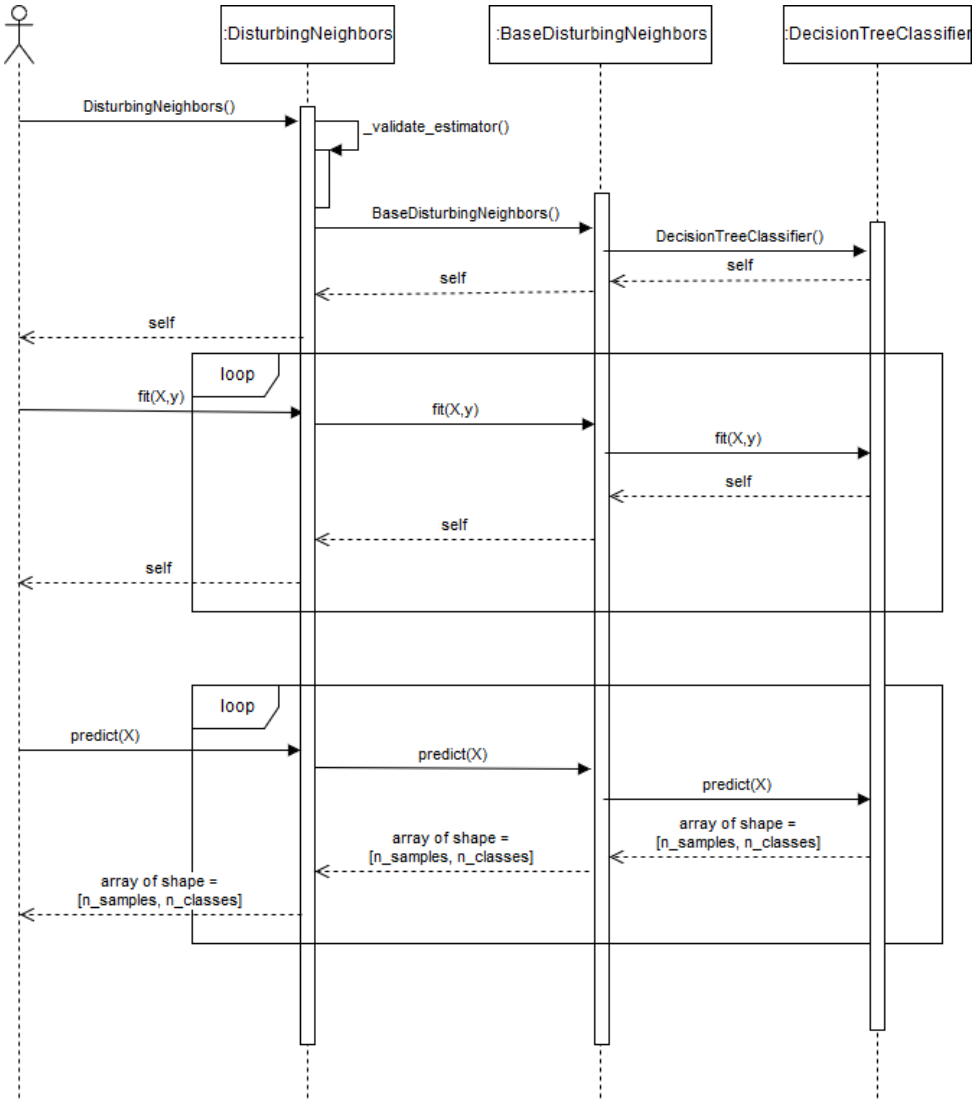

Figura C.3: Diagrama de secuencias del funcionamiento de un Notebook

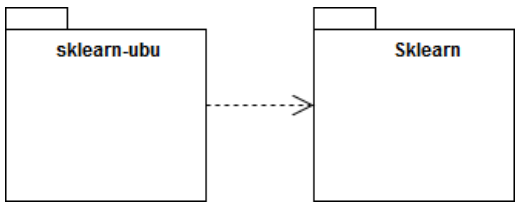

Figura C.4: Diagrama de la estructura

Apéndice D

Documentación técnica de programación

D.1. Introducción

Esta sección está dedicada para futuros desarroladores que quieran intención de continuar con este proyecto. Se describen el funcionamiento del proyecto, el software necesario, y los aspectos que se podrían mejorar.

D.2. Estructura de directorios

Nuestro proyecto se divide en dos partes, una donde tendremos nuestro código fuente, y otra con la documentación.

Documentación

En esta carpeta es donde podemos encontrar la documentación de la memoria y anexos.

- img: Esta carpeta contiene las distintas imágenes utilizadas en la memoria y anexos.
- tex: Están las distintas partes en las que se dividen la memoria y los anexos.
- Los pdfs de la memoria, anexos y la bibliografía.

Src

Esta carpeta contiene los ficheros código fuente del proyecto.

- `DecisionTreeClassifier` vs `Ensembles`: Notebook que muestra la comparación de los 3 algoritmos realizados con el `DecisionTreeClassifier`.
- `Example of base classifiers`: Notebook en el que podemos ejecutar los clasificadores base.
- `Example of ensembles classifiers`: Notebook donde ejecutamos los ensembles de cada uno de los clasficadores base.
- `Example with real ML data set`: Notebook donde probamos la ejecución de nuestros clasificadores en un conjunto de datos reales.
- `Graphics`: Notebook en el que podemos ver las gráficas de la comparación de los 3 clasificadores base realizados, en los que se puede ver como se dividen los datos.
- flags: Fichero con los datos reales. Este u otros conjuntos de datos podemos descargarlos de `mulan.sourceforge.net/datasets.html`
- sklearn_ubu: Esta carpeta contiene los ficheros con los códigos fuente de los algoritmos, sus clasificadores base y sus ensembles:
  - base_disturbing_neighbors.py
  - base_random_oracles.py
  - base_rotation_forest.py
  - disturbing_neighbors.py
  - random_oracles.py
  - rotation_forest.py
  - homogeneous_ensemble.py

D.3. Manual del programador

En esta sección vamos a describir como instalar las diferentes herramientas necesarias para realizar el proyecto.

SonarQube

Para analizar la calidad del código se ha analizado mediante la herramienta web de SonarQube. Si queremos comprobar nuestro código con esta herramienta hay que seguir una serie de pasos:

- Entrar en https://www.sonarqube.org/, podemos elegir entre descargarla o usar online, nosotros elegiremos esta segunda, que nos redirigirá a la página de https://about.sonarcloud.io/. Para poder utilizarla necesitamos loguearnos, para ello podemos hacerlo con nuestra cuenta de GitHub.
- Una vez estemos dentro, en la cabecera clickamos en el icono de la «?», que nos abrirá una ventana emergente. Y en el menú de la izquierda, pinchamos en tutorials, y dentro en el link de analizar un nuevo proyecto.
- En la primera opción elegiremos la opción por defecto de organización personal, porque hay está el proyecto que queremos analizar.
- En la segunda opción para generar el token, ponemos un nombre cualquiera.
- En la siguiente opción elegiremos el lenguaje y sistema operativo utilizados, en nuestro caso Python y Windows, y ponemos una clave única.
- Por último necesitamos descargar un pequeño archivo, lo añadiremos el bin al $PATH$.
- Para acabar deberemos entrar a la consola a la ubicación donde se encuentre nuestro proyecto, copiaremos el comando que nos ha creado en la página y lo pegamos en la consola. Esto analizará nuestro código y ya sabremos si tenemos un buen código o necesitamos modificarlo.

Nuestro proyecto tiene una calidad de A, ya que no tiene errores o código duplicado, solo tenemos unas advertencias en que algunos nombres de las variables, no son los adecuados. Aunque estos nombres no los cambiaremos, a continuación veremos las razones. Dichas variables son X, según las convenciones de Scikit-Learn, usa esa variable para los conjuntos de datos, por ello no la cambiaremos. Otras de las variables son Xp, pos_subX, $tuple_pos_subX$ y $subX$, como todas ellas son parte de X, se entiende mejor dejando el mismo nombre con el que nos referimos al conjunto de datos.

D.4. Compilación, instalación y ejecución del proyecto

Compilación

En Python no hace falta compilar el proyecto ya que es un lenguaje interpretado. Necesitaremos únicamente tener Python instalado, por medio de alguna distribución como puede ser Anaconda, aunque vale cualquier otra.

Python es un lenguaje interpretado y necesitamos tener Python 3.6. junto con las librerías usadas, pero instalar una a una dichas librerías es un trabajo complejo, para un usuario. Por eso se ha usado Anaconda, que contiene todas las librerías necesarias para ejecutar nuestro proyecto.

Instalación

Ejecución del proyecto

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Bibliografía

[1] Nick Coghlan Guido van Rossum, Barry Warsaw. Guia para comentar código python, 2013. [Online].

[2] Jesús Maudes, Juan José Rodríguez, and César Ignacio García-Osorio. Disturbing neighbors ensembles for linear svm. In *MCS*, pages 191–200. Springer, 2009.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Juan Rodríguez and Ludmila Kuncheva. Naïve bayes ensembles with a random oracle. *Multiple Classifier Systems*, pages 450–458, 2007.

[5] Juan José Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2006.

[6] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2015. [Internet; descargado 7-enero-2018].