Universidad de las Ciencias Informáticas

Facultad 6

Algoritmo de transformación LCNMTR para problemas de predicción con salidas compuestas integrado a MULAN.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autor (es): Juan M. Sierra Martínez

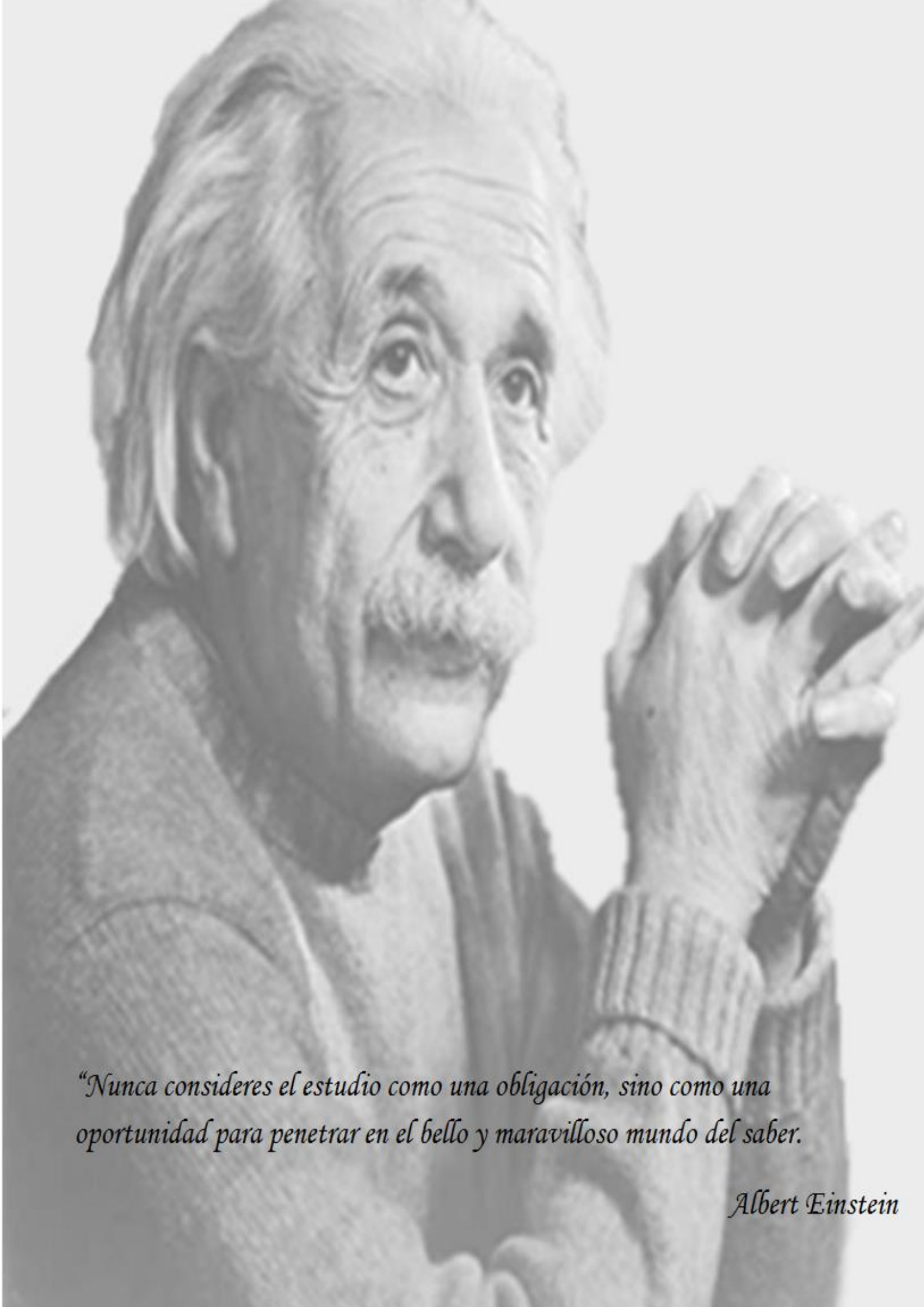
Pedro M. Pérez González

Tutor (es): MSc. Héctor Raúl González Diez

Co-tutor (es): Ing. Gabriela Santos Martínez

Ing. Frank R. Campos Almarales

La Habana, junio del 2017



"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.

Albert Einstein

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos a la UCI los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ___ días del mes de _____ del año _____.

_____ _____

Juan M. Sierra Martínez Pedro M. Pérez González

_____

MSc. Héctor Raúl González Diez

Resumen

La minería de datos como disciplina intenta reducir la brecha existente entre el volumen de información almacenado y el conocimiento que es capaz de extraerse de esta información**.** Dentro de la minería de datos el aprendizaje automático (AA) emplea información histórica para, a través de modelos, predecir el comportamiento de nuevas variables asociadas al problema. En los últimos años han surgido problemas de la vida práctica relacionados con la necesidad de predecir valores de salidas, modelados a través de diferentes estructuras complejas.

En el grupo de Inteligencia Artificial de la Universidad de las Ciencias Informáticas (UCI) se implementó el algoritmo *Linear Combination for Multi-target Regression* (LCNMTR) con un primer prototipo en MATLAB. Para lograr diseñar experimentos en condiciones similares y comparar el desempeño del algoritmo LCNMTR usando las mismas métricas de evaluación, aproximaciones de los datos sobre el mismo lenguaje y el mismo conjunto de datos, se propone la implementación de este algoritmo sobre la plataforma MULAN para que sea además evaluado por otros investigadores en este campo.

Se describe el proceso de implementación e integración a la herramienta MULAN siguiendo las pautas dictadas por la metodología XP. Se evalúan los resultados alcanzados aplicando la Prueba de Friedman y el test de Bonferroni-Dunn utilizando como medida el *aRRMSE*. Se obtiene como resultado que el algoritmo propuesto no presenta diferencias significativas con los algoritmos del estado de arte con un 95% de certeza. Además, es uno de los algoritmos con menor tiempo de ejecución.

Palabras clave: Predicción con salidas compuestas, LCNMTR, MULAN.

Contenido

Índice de tablas

Índice de Ilustraciones

Introducción

El volumen de información almacenada en el mundo ha ido creciendo exponencialmente en las últimas décadas. El hecho de que hoy en día el costo de almacenamiento se haya reducido considerablemente, hace que sea muy sencillo conservar información que hasta hace poco fuese desechada. En este contexto, en el que se calcula que cada 20 meses se duplica el volumen de la información almacenada en bases de datos digitales y en el que dispositivos electrónicos almacenan nuestras transacciones, el problema consiste en tener mecanismos que permitan no sólo almacenar y recuperar eficientemente la información sino convertirla en conocimiento para la toma de decisiones (Ávila Jiménez, 2013).

La minería de datos como disciplina intenta reducir la brecha existente entre el volumen de información almacenado y el conocimiento que es capaz de extraerse de esta información. El proceso, idealmente, debe ser automático y el objetivo principal es que los patrones descubiertos deben tener algún significado en el sentido de que puedan ser utilizados como conocimiento útil. Dentro de la minería de datos el aprendizaje automático (AA) está considerado una disciplina que emplea información histórica formalmente estructurada sobre un fenómeno, para a través de modelos predecir el comportamiento de nuevas variables asociadas al problema (Witten & Frank, Data Mining: Practical machine learning tools and techniques, 2005).

El AA estudia los agentes o programas que aprenden o evolucionan basados en su experiencia, para realizar una tarea determinada cada vez mejor. El objetivo principal de todo proceso dentro del AA es utilizar la evidencia conocida (Datos de Entrenamiento) para poder crear una hipótesis (Modelos) y dar una respuesta a nuevas situaciones no conocidas en un contexto dado (Sierra Araujo).

Una clasificación general de los algoritmos de AA los separa en métodos supervisados y no supervisados. La aplicación de uno u otro algoritmo depende de las características del problema a resolver, pues cada uno de ellos puede ser útil en determinadas circunstancias. Los algoritmos supervisados intentan definir un modelo matemático a partir de la información de las clases de un conjunto de datos de entrenamiento. Los algoritmos no supervisados encuentran una partición de los datos sin utilizar la información de la clase (Jin, Wang, & Yang) (Yang, 2007).

Dentro de las tareas de los algoritmos supervisados, los enfoques clásicos son aquellos donde se modela un problema a partir de un conjunto de variables predictoras y una única variable de salida la cual se desea medir. En los últimos años han surgido

problemas de la vida práctica que no sería adecuado modelarlos utilizando este tipo de enfoque clásico. Estos problemas están relacionados con la necesidad de predecir valores de salidas, modelados a través de diferentes estructuras complejas. Las estructuras complejas pueden ser: grafos, jerarquías o vectores con valores reales o binarios (Bakir, y otros, 2007). Este tipo de tarea es conocida como problemas de predicción con salidas compuestas (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014). Los algoritmos de predicción con salidas compuestas obtienen mejores resultados en cuanto al error predictivo si los comparamos con los algoritmos de salida única debido a que explotan la interdependencia entre las variables (Blockeel H. , 1998.).

Un ejemplo clásico es en la predicción de la calidad de la vegetación (Kocev D, 2009) donde se obtienen los datos utilizando el enfoque de "hábitat por hectáreas". Esta técnica de evaluación rápida produce múltiples puntuaciones que describen la condición de los diversos atributos de la vegetación de un sitio dado. Para el modelado de este tipo de datos se comparan los siguientes dos enfoques: el primero de ellos donde se aprende un modelo para cada una de las variables de salidas expresada en la puntuación y un segundo caso donde se construye un modelo único que toma en cuenta todas las variables de forma simultánea. Los resultados de dicho trabajo demuestran las ventajas de un enfoque de modelado multi-objetivo sobre uno de un solo objetivo.

Otras áreas donde se han modelado los problemas utilizando un enfoque de predicción con salidas compuestas son en la quimiometría (Dzeroski, Demsar, & J., 2000) al inferir concentraciones de varios analitos de calibración multivariable usando datos multivariados espectrales; la predicción de los precios de los billetes de avión para una determinada aerolínea tomando como variables de entrada detalles de vuelos observados durante varios días como precio, paradas o fecha de salida (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014) y en la predicción de la concentración de metales más costosos de medir utilizando metales que son más baratos para muestrear, entre otras.

Un artículo de revisión publicado recientemente sobre los problemas de predicción con salidas compuestas (Borchani, Varando , Bielza, & Larrañaga, 2015) establece dos grandes categorías para este tipo de tarea: El primer enfoque contiene los métodos que transforman el problema de salidas múltiples en diferentes problemas de predicción de una salida; mientras que el otro enfoque se basa en adaptar métodos clásicos conocidos de AA para estimar de manera simultánea las múltiples salidas. En ambas categorías los métodos pueden tener en cuenta o no la interdependencia entre las variables de salidas.

En el caso de los algoritmos basados en transformación generalmente se diseñan en dos etapas, la primera de ella permite estimar cada salida de manera independiente para luego en una segunda etapa establecer alguna estrategia que combine las diferentes salidas (Borchani, Varando , Bielza, & Larrañaga, 2015). Entre los algoritmos más representativos del estado del arte en esta categoría (por su capacidad predictiva) se encuentran el *Multi Target Staking* (MTS) y su variante corregida (MTSC) (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014), *Regressor Chains* (ERC) y (ERCC) (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014), *Curds and Wheys* (C&W) (Breiman & Friedman, Predicting multivariate responses in multiple linear regression , 1997), *Reduced Rank Regression* (RRR) (Izenman, 1975), *K Nearest Neighbor* (KNN-SP) (Pugelj & Dzeroski, 2011) (González Diez, Santos, Campos, & Morell Pérez, 2016) *Random Linear Combination* (RLC) (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014).

Estos algoritmos emplean diferentes regresores de base y estrategias para combinar las salidas. De manera particular los algoritmos C&W y RLC estiman una matriz como combinación lineal de las variables de salidas. En el caso de C&W es un algoritmo diseñado de conjunto con un modelo de regresión lineal y la estimación de la matriz de combinación lineal B, donde esta depende del modelo base de regresión. Por otra parte, el algoritmo RLC utiliza una matriz aleatoria con manejo de su rango para extender el espacio de salida a uno de mayor dimensión. Esta estrategia es independiente del regresor que se emplee, sin embargo, en la evaluación del algoritmo se emplean los regresores de base Bagging y RandonForest.

Tomando como inspiración estos dos modelos, se ha desarrollado, en el grupo de Inteligencia Artificial de la Universidad de las Ciencias Informáticas (UCI) el algoritmo *Linear Combination for Multi-target Regression* (LCNMTR) con un primer prototipo en MATLAB. En el caso de los restantes algoritmos del estado del arte se encuentran desarrollados en la herramienta MULAN (Tsoumakas, Spyromitros-Xioufis, Vilcek, & Vlahavas, 2011). MULAN es un software de código abierto para el aprendizaje automático cuyo objetivo principal es el manejo de problemas de clasificación mult-etiqueta. Recientemente este software ha extendido sus funcionalidades para el manejo de datos con salidas compuestas incorporando los algoritmos anteriormente descritos.

Para lograr diseñar experimentos en condiciones similares y comparar el desempeño del algoritmo LCNMTR usando las mismas métricas de evaluación, aproximaciones de los datos sobre el mismo lenguaje y el mismo conjunto de datos, se hace requiere la implementación de este algoritmo sobre la plataforma MULAN para que sea además evaluado por otros investigadores en este campo.

Tomando en cuenta los elementos anteriormente planteados se puede afirmar que:

1. Los principales algoritmos de predicción con salidas compuestas basados en transformación se encuentran implementados en la herramienta MULAN.
2. El algoritmo LCNMTR cuenta con un prototipo en MATLAB que no es suficiente para su rigurosa evaluación con los anteriormente desarrollados en la herramienta MULAN.

Por la problemática anteriormente descrita se plantea como **problema de la investigación:** ¿Cómo evaluar el algoritmo de predicción con salidas compuestas basado en trasformación LCNMTR en igualdad de condiciones con los del estado del arte?

Teniendo en cuenta el problema antes propuesto se define como **objeto de estudio:** Algoritmos de predicción con salidas compuestas.

Enmarcándose en el **campo de acción:** Algoritmos de predicción con salidas compuestas basados en transformación en la herramienta MULAN.

La presente investigación tiene como **objetivo general:** Implementar en MULAN el algoritmo de transformación LCNMTR para problemas de predicción con salidas compuestas.

Para alcanzar el objetivo trazado y dar solución al objetivo planteado se definen los siguientes **objetivos específicos:**

1. Caracterizar el marco teórico-conceptual de los algoritmos de predicción con salidas compuestas basados en transformación con énfasis en las herramientas de aprendizaje automático disponible para ello.
2. Implementar el algoritmo basado en transformación LCNMTR y sus variantes en la herramienta MULAN, así como la base experimental para probar el algoritmo con otros el estado del arte.
3. Validar la solución implementada mediante el diseño de experimentos comparando los resultados con algoritmos del estado del arte basados en transformación.

Para apoyar el desarrollo de la investigación se emplean los siguientes métodos científicos:

Métodos Teóricos:

Analítico-Sintético:

Se usa para conformar un estado del arte sobre los algoritmos de predicción con salidas compuestas basados en transformación y las herramientas que los implementan.

También se utilizó para un análisis general de las métricas y metodologías para la evaluación de los clasificadores.

Inductivo-deductivo:

Se hace uso de él para describir las bases teóricas del algoritmo LCNMTR y sus casos particulares.

Métodos Empíricos

Modelación:

Se utiliza en el diseño de los diferentes algoritmos estudiados y la implementación del LCNMTR; en el diseño de los diagramas de clases, diseño de la implementación y modelación de la arquitectura.

Histórico-Lógico:

Este método permite estudiar la trayectoria histórico-real de la evolución y desarrollo de los algoritmos de predicción con salidas compuestas basados en transformación.

Medición:

En la experimentación se utiliza el promedio del error cuadrático medio para el cálculo del error predictivo, así como la prueba de Friedman para comparar los resultados de los algoritmos estudiados.

El desarrollo de la investigación trae consigo dos beneficios fundamentales. Primeramente, el grupo de Inteligencia Artificial de la UCI podrá disponer de un nuevo algoritmo que permita darle valor de uso a los productos de software de la universidad. Además, la publicación de dicho algoritmo dentro de la herramienta MULAN generalizará su uso en la comunidad científica internacional.

En aras de estructurar el proceso de desarrollo del presente trabajo de una manera coherente y organizada, se dividió el mismo en un total de tres capítulos:

Capítulo 1. “Fundamentación teórica. Algoritmos de predicción con salidas compuestas”, aborda el estudio del estado del arte del tema relacionado con el trabajo que se desarrolla y los conceptos y definiciones investigados durante el análisis de las tendencias actuales.

Capítulo 2. “Exploración y Planificación”: Se describen las características de la solución propuesta, definidas a partir de la modelación de los procesos y la especificación de los requisitos de software. En esta fase se describen los requisitos funcionales a través de las historias de usuario, y se realiza una planificación para darle cumplimiento a las historias de usuario seleccionadas, así como el conjunto

de tareas que se ejecutan para su terminación. Además de la planificación y entrega de la aplicación.

Capítulo 3. “Implementación y Resultados”: En este capítulo se evalúa la calidad de la solución final mediante la realización de pruebas, con el objetivo de solucionar errores y no conformidades presentes en la implementación.

Capítulo 1. Fundamentación teórica. Algoritmos de predicción con salidas compuestas.

1.1. Introducción

En este capítulo se aborda en qué consisten los problemas de predicción con salidas compuestas. Se realiza un estudio del estado del arte de los algoritmos cuyo enfoque está basado en este tipo de problema. Además, se estudian y describen las herramientas existentes que implementan este tipo de algoritmo. Finalmente se describen las herramientas y metodología de desarrollo de software empleadas para elaborar una herramienta acorde a los requisitos planteados.

1.2. Problemas de predicción con salidas compuestas

En la actualidad, existen problemas que requieren cada vez más métodos no convencionales que obtengan su solución, donde los resultados puedan modelarse de maneras distintas en dependencia del tipo de problema. Los problemas de predicción generalmente involucran la predicción simultánea de varias salidas utilizando el mismo conjunto de variables predictoras. A este tipo de problemas se le conoce como problemas de predicción con salidas compuestas (Blockeel H. , 1998.). En dicho caso en vez de obtener una salida simple, se necesitan predecir un conjunto de estas de manera simultánea. Es decir, todas las salidas son igualmente importantes por lo que se predicen simultáneamente con un solo modelo (Blockeel, Ramon, & Raedt, 2000).

En términos más formales, sean x, y dos vectores aleatorios del espacio de entrada R^d donde $x = [x_1, \dots x_d]$ y $y = [y_1, \dots y_m]$ en el espacio de salida R^m para una instancia determinada. Dado el conjunto $D = \{(x^1, y^1), \dots (x^n, y^n \subset R^d \, X \, R^m)\}$ de n instancias de entrenamiento, el objetivo en un problema de predicción de salidas compuestas es aprender un modelo $h : R^d \rightarrow R^m$ de modo que dada una instancia con entrada conocida x^q es posible predecir, a través de un único modelo, todas las salidas $\hat{y}^q = h(x^q)$ de forma simultánea. La interdependencia para la j-ésima variable de salida queda expresada en un modelo $\check{h}(x, \tilde{y})$ con $\hat{y} = y \backslash y_j$ (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014).

El aprendizaje con salidas compuestas tiene como meta predecir las características de las salidas y describir explícitamente su relación con las características descriptivas. Capta implícitamente las dependencias entre las salidas y las representa en un modelo único generado. A través de este modelo, se determina el efecto de las características descriptivas de todas las salidas, y se analizan las relaciones, ya sea lineal o no lineal, entre las salidas (o grupos de salidas). En caso de que las salidas estén relacionadas,

se obtiene información sobre estas relaciones. Varios métodos estándar de aprendizaje han sido extendidos a problemas de predicción con salidas compuestas en los últimos años. Estos métodos pueden clasificarse en dos categoría fundamentales (Borchani, Varando , Bielza, & Larrañaga, 2015):

Los métodos basados en transformación (también conocidos como métodos locales) transforman los problemas de salida múltiple en problemas independientes de salida única, resolviendo cada uno usando un algoritmo de salida simple. Dentro de los métodos basados en transformación se encuentran el MTS, ERC, ERCC y MTSC (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014).

Los métodos basados en adaptación (también conocidos como métodos globales, o métodos big-bang) adaptan un método específico de salida simple para el manejo directo en un método de salida múltiple. Dentro de los métodos de adaptación se encuentran MORF (Tsoumakas , Spyromitros-Xioufis, Vrekou, & Vlahavas, 2014), RRR (Izenman, 1975), C&W (Breiman & Friedman, Predicting multivariate responses in multiple linear regression , 1997). A continuación, se describe la esencia de dichos algoritmos.

1.3. Algoritmos de predicción con salidas compuestas basados en transformación

1.3.1. Multi-target Stacking (MTS)

El entrenamiento en MTS está compuesto por dos etapas fundamentales: en la primera se aprenden m modelos $h_j : \mathrm{X} \rightarrow \mathrm{R}$ de salidas simples independientes y en la segunda se aprende un segundo conjunto de m meta-modelos $h^{\times}_j : \mathrm{X} \times R^{m-1} \rightarrow \mathrm{R}$ por cada salida Y_j. Para predecir las salidas de una instancia desconocida x^q se aplica la primera etapa obteniéndose un vector de salida $\tilde{y}^q$, luego al aplicar la segunda etapa en un vector de una entrada ${x^{\times}}_j{}^q$ transformado se produce el vector final de salida $\hat{\tilde{y}}_j{}^q$.

1.3.2. Ensemble of Regressor Chains (ERC)

El método de *Regressor Chains* (RC) se basa en la idea de modelos de encadenamiento de salidas simples. El entrenamiento del RC consiste en seleccionar una cadena aleatoria (permutación) del conjunto de variables de salida y luego construir un modelo de regresión separado para cada salida.

Una propiedad notable de RC es que es sensible en el ordenamiento de la cadena seleccionada. El principal problema que surge de la utilización de una sola cadena al azar, es que las salidas que aparecen antes en una cadena no pueden modelar posibles

relaciones estadísticas con las salidas que aparecen más adelante en la cadena. Además, el error de predicción es probable que sea propagado y amplificado a lo largo de la cadena cuando hacemos predicciones para una nueva instancia de prueba.

Para mitigar estos efectos, se propone un esquema de conjunto llamado *Ensemble of Classifier Chains* donde una prueba de k (típicamente k=10) modelos *Classifier Chains* (CC) con cadenas de diferente orden son construido sobre muestras de arranque del conjunto de entrenamiento y las predicciones finales provienen de la votación por mayoría. Este esquema ha demostrado mejorar consistentemente la exactitud de un solo CC en el dominio de clasificación. Se aplicala misma idea (sin muestreo) en RC y se calcula las predicciones finales tomando la media de las estimaciones de k para cada objetivo. El método resultante se llama *Ensemble of Regressor Chains.*

1.3.3. MTS y ERC Corrected (MTSC y ERCC)

El MTSC y el ERCC utilizan un enfoque interno *f-fold cross validation* para obtener estimados fuera de la muestra para las variables de salida. El estimado de esta validación cruzada puede ser menos preciso que los obtenidos durante la predicción ya que los modelos son hechos usando el $\frac{f-1}{\mathrm{f}}\%$ del conjunto de entrenamiento en su totalidad.

1.3.4. Curds and Whey (C&W)

La idea general del algoritmo *Curds and Whey* es tomar las regresiones de mínimos cuadrados, y luego de modificar los valores previstos de esas regresiones por la contracción de ellos, utilizando las correlaciones canónicas entre las variables de salida y las variables predictoras. Breiman y Friedman en su trabajo proponen la contracción simultánea tanto en el espacio de entrada como en el de salida.

El enfoque estándar de determinación de la matriz de coeficientes C es a través del OLS (*Ordinal Least Square*, mínimos cuadrados ordinales), donde el resultado que se obtiene es equivalente a la regresión de cada una de las variables de salida a través de las variables predictoras de forma separada. Para este enfoque el error predictivo es pobre en presencia de una correlación alta entre las variables predictivas o cuando existen una cantidad significativa de este tipo de variables.

1.3.5. Reduce Rank Regretion (RRR)

RRR es un método de estimación explícita en la regresión multivariante, que tiene en cuenta la restricción de rango reducido en la matriz de coeficientes. Una incidencia directa del RRR sobre la regresión lineal es el número de restricciones lineales sobre la

matriz de coeficientes C de la regresión, permitiendo que el número de parámetros efectivos se reduzca y la eficiencia de la estimación aumente.

1.3.6. Multi-objective Bagging (Clusbag) y Multi-objective Random Forest (MORF)

Clusbag y MORF surgen al aplicar métodos de ensamble Bagging (Breiman, “Bagging predictors,” Machine learning, 1996) y Random Forests (Breiman, Random forests,” Machine learning,, 2001) respectivamente al método Multi-objective decision trees (MODTs) (Kocev D, 2009). Los métodos de ensamble no son más que un conjunto de clasificadores construidos con un algoritmo dado. Cada nuevo ejemplo se clasifica mediante la combinación de las predicciones de cada clasificador desde el conjunto. Éstas se pueden combinar tomando el promedio (para las tareas de regresión) o el voto de la mayoría (para las tareas de clasificación) como se describe por (Breiman, “Bagging predictors,” Machine learning, 1996), o tomando combinaciones más complejas (Ho, Hull, & Sriari, 1994) (Kittler, Hatel, Duin , & Matas, 1998).

Para aplicar Bagging a MODTs, el procedimiento de Predictive Clustering Trees (PCTs) (Kocev D, 2009) se utiliza como un clasificador base. Para la aplicación de Random Forests se sigue el mismo enfoque cambiando el procedimiento BestTest. Con el fin de combinar la salida de las predicciones por los clasificadores base, se toma la media de la regresión, y se aplica un voto distribución de probabilidad en lugar de una simple mayoría de votos para la clasificación, según lo sugerido por Bauer y Kohavi (Bauer & Kohavi, 1999).

1.4. Herramientas para la resolución de problemas de predicción con salidas compuestas.

A continuación, se presenta una descripción de las herramientas empleadas en el contexto de los problemas de predicción con salidas compuestas.

1.4.1. Matlab

Matlab es un entorno de computación y desarrollo de aplicaciones totalmente integrado, orientado para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos y la visualización gráfica de los mismos. Cuenta con un lenguaje de programación propio (lenguaje M), disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux. Matlab integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo donde los problemas y sus soluciones son expresados del mismo modo en que se escribirían tradicionalmente, sin necesidad de hacer uso de la programación tradicional. Dentro de sus características

se encuentran: cálculos intensivos desde un punto de vista numérico, gráficos y visualización avanzada, lenguaje de alto nivel basado en vectores, arreglos y matrices; y una colección muy útil de funciones de aplicación. Recientemente ha sido empleado para desarrollar los prototipos de algoritmos del grupo de inteligencia artificial de la uci que investigan entorno a la presente tesis.

1.4.2. Weka

Weka (Durrant, y otros, 2013) (Witten & Frank, 2000) es una herramienta para el aprendizaje automático y la minería de datos escrito en JAVA y desarrollado en la Universidad de Waikato. Fue diseñado para ofrecer toda una gama de técnicas de aprendizaje automático en una interfaz común, aplicables consistentemente y que sea suficientemente flexible para que permita la incorporación de nuevos esquemas. Weka es una herramienta útil que logra reducir el nivel de complejidad que conlleva la obtención de datos del mundo real, así como la evaluación de su salida, proporcionado una ayuda flexible para la investigación de aprendizaje automático. Incluye algoritmos que permiten resolver problemas de clasificación, clasificación multiclase y de regresión. No permite resolver problemas de aprendizaje a gran escala.

1.4.3. Clus

Clus (Struyf, Zenko, Blockeel, Vens, & Dzeroski, 2010) es un sistema que implementa el marco de trabajo *predictive clustering* (PC) basado en árboles de decisión y reglas de inducción. Este marco de trabajo unifica los métodos no supervisados y los modelos predictivos permitiendo de forma natural la extensión a modelos predictivos más complejos tales como aprendizaje multitarea y clasificación multi-etiqueta. Este marco de trabajo está integrado a la herramienta MULAN desarrollado sobre tecnología JAVA y permite a través de un wrapper la ejecución desde MULAN de los algoritmos que se encuentran implementados en este.

1.4.4. MULAN

MULAN (Tsoumakas, Spyromitros-Xioufis, Vilcek, & Vlahavas, 2011) es un software de código abierto para máquinas de aprendizaje cuyo objetivo principal es el trabajo con datos multi-etiquetas. Ofrece los algoritmos que existen sobre clasificación multi-etiqueta y ranking de etiquetas, así como un marco de trabajo que contiene un compendio de medidas para evaluar la clasificación multi-etiqueta mediante la validación *hold-out* y la validación cruzada. MULAN está escrita en JAVA y se construye encima de Weka para emplear todos los recursos que contiene sobre algoritmos de aprendizaje supervisado, aunque esta es independiente. Una característica exclusiva de la biblioteca MULAN es la introducción reciente de un paquete de experimentos que

tiene como objetivo reproducir los resultados de los experimentos en el ámbito de la predicción con salidas compuestas.

En sentido general las herramientas analizadas se han desarrollado sobre tecnología JAVA y se encuentran integradas como parte de la herramienta MULAN lo cual facilita la ejecución de los diferentes algoritmos de predicción con salidas compuestas.

1.5. Bases de datos y medidas de evaluación.

En el trabajo (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014) se proponen varios algoritmos de predicción con salidas compuestas los cuales fueron evaluadas con 12 bases de datos estándares que se han establecido en este ámbito. De estas bases datos 4 fueron propuestas como resultado de este trabajo.

Estas bases de datos se relacionan en la ***Tabla 1*** donde la primera columna indica el nombre de la base de datos, la segunda se refiere al número de observaciones de la base de datos. En esta columna, además, se indica si los datos están particionados en datos de prueba (test) y datos de entrenamiento (train) o si la base de datos contiene todos los datos los cuales deben ser particionados para los experimentos. Para particionar los datos se utiliza un método de validación cruzada el cual se conoce como *K-Fold Cross Validation* (lo señalamos como CV). La tercera y cuarta columna indican la cantidad de atributos en el espacio de entrada y salida respectivamente. La última columna brinda una breve explicación del dominio de aplicación en el cual fueron colectados los datos.

Base de Datos	Instancias	Atributos de entrada	Atributos de salida	Dominio
water-quality	1060/cv	16	14	Abundancia de 14 plantas y especies animales en ríos de Slovenia.
scm20d	7463/1503	61	16	Predice los precios de los productos futuros en una competencia.
scm1d	8145/1658	280	16	Predice los precios de los productos futuros en una competencia.
rf2	4108/5017	576	8	Predice el comportamiento de las redes fluviales de ríos durante 48 horas.

rf1	4108/5017	64	8	Predice el comportamiento de las redes fluviales de ríos durante 48 horas.
oes97	343/cv	263	16	Estima el número de empleados a tiempo completo para una ciudad.
oes10	403/cv		16	Estima el número de empleados a tiempo completo para una ciudad.
flare2	1066/cv	10	3	Veces que se producen destellos repentinos en la superficie del sol durante 24 h.
flare1	323/cv	10	3	Veces que se producen destellos repentinos en la superficie del sol durante 24 h.
edm	154/cv	16	**2**	Generación de electricidad.
atp7d	188/108	411	**6**	Precio de los boletos de avión para una aerolínea durante intervalos de tiempo.
atp1d	201/136	411	**6**	Precio de los boletos de avión para una aerolínea durante intervalos de tiempo.

Tabla *1* Bases de datos para problemas de predicción con salidas compuestas

Como medida de evaluación en la mayoría de los trabajos se emplean el promedio del error relativo cuadrático medio (aRRMSE) entre lo predicho por el algoritmo y lo medido en cada base de datos para las variables de salidas. El mismo se determina de la siguiente manera:

$$RRMSE(h; D_{test}) = \sqrt{\frac{\sum_{(x,y_j)\in D_{test}} (\hat{y}_j - y_j)^2}{\sum_{(x,y_j)\in D_{test}} (\bar{Y}_j - y_j)^2}} \quad (7)$$

1.6. Metodologías de evaluación de algoritmos de aprendizaje automático

En esta sección se exponen las diferentes metodologías empleadas para la evaluación y comparación de los resultados en el ámbito del aprendizaje automático.

1.6.1. Validación cruzada

La validación cruzada es un método de evaluación de modelos que se emplea para mejorar el error predictivo. El problema de la evaluación del error es que no se conoce que tan bien el modelo ha aprendido para predecir datos desconocidos. Una forma de solucionar este problema es no emplear todo el conjunto de datos para entrenar el modelo. Una parte del conjunto de datos es removido del conjunto de entrenamiento y empleado para la prueba, una vez que el modelo haya sido entrenado.

El método hold-out es el más simple de los tipos de validación cruzada. Este consiste en dividir en dos conjuntos complementarios los datos de muestra, realizar el análisis de un subconjunto (denominado datos de entrenamiento o *training set*), y validar el análisis en el otro subconjunto (denominado datos de prueba o *test set*), de forma que la función de aproximación sólo se ajusta con el conjunto de datos de entrenamiento y a partir de aquí calcula los valores de salida para el conjunto de datos de prueba (valores que no ha analizado antes). La ventaja de este método es que es muy rápido a la hora de computar. Sin embargo, este método no es demasiado preciso debido a la variación del resultado obtenido para diferentes datos de entrenamiento. La evaluación puede depender en gran medida de cómo es la división entre datos de entrenamiento y de prueba, y por lo tanto puede ser significativamente diferente en función de cómo se realice esta división. Debido a estas carencias aparece el concepto de validación cruzada (Schneider, 1997).

En la validación cruzada de K iteraciones o *K-fold cross-validation* los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos se utiliza como datos de prueba y el resto (K-1) como datos de entrenamiento. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado. Este método es muy preciso puesto que evaluamos a partir de K combinaciones de datos de entrenamiento y de prueba, pero aun así tiene una desventaja, y es que, a diferencia del método de retención, es lento desde el punto de vista computacional. En la práctica, la elección del número de iteraciones depende de la medida del conjunto de datos. Lo más común es utilizar la validación cruzada de 10 iteraciones (*10-fold cross-validation*) (Joanneum, 2005).

Este método consiste al dividir aleatoriamente el conjunto de datos de entrenamiento y el conjunto de datos de prueba. Para cada división la función de aproximación se ajusta a partir de los datos de entrenamiento y calcula los valores de salida para el conjunto de datos de prueba. El resultado final se corresponde a la media aritmética de los valores obtenidos para las diferentes divisiones. La ventaja de este método es que la división de datos entrenamiento-prueba no depende del número de iteraciones. Pero, en cambio, con este método hay algunas muestras que quedan sin evaluar y otras que se evalúan más de una vez, es decir, los subconjuntos de prueba y entrenamiento se pueden solapar (Moore, 2001).

La validación cruzada dejando uno fuera o *Leave-one-out cross-validation* (LOOCV) implica separar los datos de forma que para cada iteración se tiene una sola muestra para los datos de prueba y todo el resto conformando los datos de entrenamiento. La evaluación viene dada por el error, y en este tipo de validación cruzada el error es muy bajo, pero en cambio, a nivel computacional es muy costoso, puesto que se tienen que realizar un elevado número de iteraciones, tantas como N muestras tengamos y para cada una analizar los datos tanto de entrenamiento como de prueba (Schneider, 1997).

De los métodos de validación cruzada antes expuesto se emplea para evaluar los modelos predictivos la validación cruzada de k iteraciones con k = 10, teniendo en cuenta que es el más empleado en la evaluación de los algoritmos del estado del arte. Además, este método es más preciso que el *hold-out*, puesto que evalúa a partir de k combinaciones de datos de entrenamiento y de prueba, todo el conjunto de datos y computacionalmente es menor el costo que el *Leave-one-out*.

1.6.2. Comparación estadística de múltiples clasificadores

Siguiendo las recomendaciones de (Demšar, 2006), el método estadístico común para esta tarea es ANOVA para análisis paramétrico. Una de las desventajas de ANOVA es que está basada en asunciones, las cuales, en el ámbito experimental de los algoritmos de aprendizaje automático, son posiblemente violadas. Primeramente, ANOVA asume que las muestras son extraídas de distribuciones normalizadas. La segunda y más importante asunción de ANOVA es la esfericidad. La esfericidad es una propiedad que requiere que las variables aleatorias tengan igual varianza y que debido a la naturaleza de los algoritmos de aprendizaje y las colecciones de datos no se puede dar por hecho. Violaciones de estas asunciones tiene un efecto aún mayor en las Pruebas *post-hoc*, por lo que ANOVA no es usada frecuentemente para estudios experimentales de aprendizaje automático.

Un equivalente no paramétrico de ANOVA es la Prueba de Friedman (Friedman M. , 1937) la cual jerarquiza los algoritmos por cada colección de datos separadamente y en caso de empate asigna un rango promedio. Tomando r_i^j como el rango del algoritmo j de k posibles algoritmos en la i-ésima colección de datos de N colecciones de datos, la Prueba de Friedman compara los rasgos promedios de los algoritmos $R_j = \frac{1}{N}\sum r_i^j$. La hipótesis nula afirma que todos los algoritmos son equivalentes y por eso sus rangos R_j deberían ser iguales. Bajo esta hipótesis nula, la estadística de la Prueba de Friedman se calcula:

$$X^2{}_F = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \quad (6)$$

De acuerdo a X_F^2 con $k-1$ grados de libertad cuando k y N son lo suficientemente grandes $k > 5$, $N > 10$. Si la hipótesis nula es rechazada se puede proceder con una Prueba de *post-hoc*. La Prueba de Nemenyi (Nemenyi, 1963) es similar a la de Tukey para ANOVA y es usada cuando los clasificadores son comparados entre sí. El desempeño de cualesquiera dos mediciones modeladas en el experimento es significativamente diferente si la diferencia entre estas excede al menos la Diferencia Crítica (CD). Los valores críticos son determinados usando el estadístico del rango *Studentized* dividida por $\sqrt{2}$. Esta diferencia crítica se calcula de la siguiente manera:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

Teóricamente la Prueba de Friedman no paramétrica tiene menos poder que el ANOVA paramétrico cuando las asunciones de ANOVA se cumplen, en caso contrario no necesariamente. Friedman en su trabajo (Friedman M. , 1940) realizó 56 experimentos relacionados con diversos problemas independientes y demostró que ambos coincidían generalmente. Cuando uno encontraba un nivel de significación en $p < 0.01$ el otro en al menos $p < 0.05$ respectivamente. Solamente en dos casos ANOVA encontró un nivel de significación que era insignificante para Friedman, mientras que en caso contrario ocurrió en 4 pruebas.

La imagen siguiente describe el proceso a seguir para la selección de la metodología estadística más idónea para comparar clasificadores.

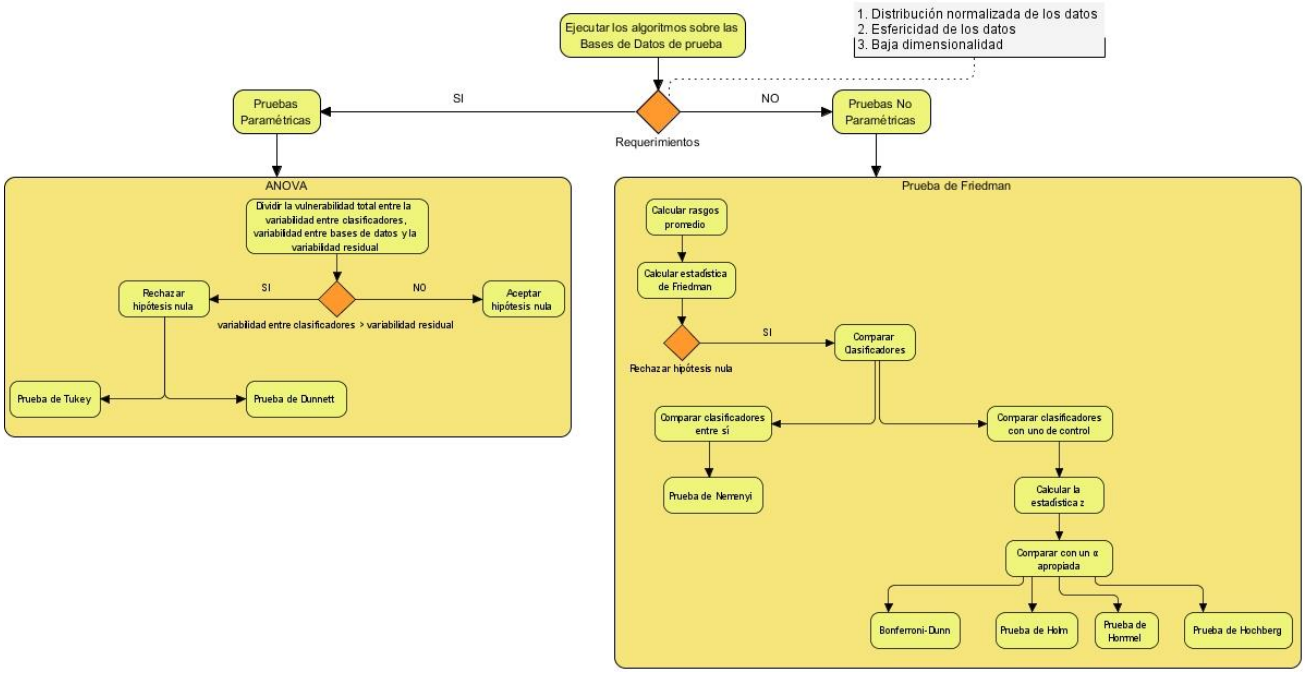


Ilustración *1* Proceso de selección de la metodología estadística para comparar clasificadores.

1.7. Metodología, lenguaje de programación y herramientas de desarrollo de software

A continuación, se presenta una descripción de la metodología, lenguaje de programación y herramientas a utilizar en el desarrollo de la solución.

1.7.1. Metodología de desarrollo.

Según (Pressman R. S., 2007) una metodología es un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un software. Está formada por fases (sub-fases) que guiarán el ciclo de vida del proyecto. Debido a la responsabilidad que tiene sobre la vida del software, es necesario que sea elegido el enfoque adecuado para el desarrollo eficiente de este. Existen dos enfoques: los ágiles y los tradicionales o prescriptivos.

Para la selección del enfoque se realiza el método Boehm y Turner también conocido como método de la estrella. Este caracteriza al proyecto de software a partir de 5 criterios y realiza una estimación acerca de cuan ágil o prescriptivo debe ser el enfoque a utilizar. Tiene 5 ejes, en cada uno se coloca un criterio, estos se explican a continuación (Boeras Velázquez, y otros, 2012):

Tamaño: Este criterio se utiliza para representar el número de personas involucradas en el proyecto. Pueden tenerse en cuenta el nivel de complejidad que pueda presentarse

en la comunicación entre los miembros del proyecto y los costos que pueden provocar cambios esperados.

Criticidad: Se utiliza para evaluar la naturaleza del daño ocasionado por defectos que no hayan sido detectados al producto. Su evaluación puede ser cualitativa.

Dinamismo: Representa la rapidez con la que pueden estar cambiando los requerimientos del proyecto.

Personal: Representa la proporción del personal con experiencia alta, media y baja. Los métodos orientados al plan no se ven afectados negativamente por este factor pues no interesa el nivel de experiencia con la que cuenten los miembros del equipo.

Cultura: Las organizaciones y las personas que relaciona el proyecto pueden depender de la confianza o de la relación contractual. Esto refleja el nivel de ceremonia necesario y aceptado: documentación, control, formalismo en las comunicaciones.

Es importante tener en cuenta el comportamiento de cada criterio en el proyecto, en correspondencia de los valores que alcancen se determinará el enfoque, mientras más cerca esté del centro el enfoque será el ágil, de lo contrario debe usarse un enfoque tradicional.

A continuación, se describe el comportamiento de los criterios en el proyecto:

Tamaño: El equipo de desarrollo está formado por dos estudiantes de quinto año para la implementación del algoritmo LCNMTR con ciertos elementos de complejidad. Estas características permiten clasificar el equipo de desarrollo y a la herramienta como pequeños por lo que será posible ubicar el punto cercano al centro.

Criticidad: El equipo de desarrollo tiene una elevada responsabilidad con la calidad del producto a obtener, debido al impacto social del mismo. Este sistema será multipropósitos por tanto permitirá ser usado tanto en el campo de la medicina, o en la realización de diversas tareas en la educación, y otros. El valor para este criterio dentro de la estrella se etiqueta como medio, debido a que los efectos por errores del producto, aunque no provocará pérdidas de vidas, provocarían otros.

Dinamismo: Pueden aparecer cambios en los requerimientos del sistema en cualquier fase del proceso de desarrollo, esto es un riesgo que se asumirá durante todo el ciclo de desarrollo del software. Para mitigarlo, deben adoptarse mecanismos que faciliten la asimilación y adaptación rápida a dichos cambios. Tomando en cuenta esta necesidad como una de las ventajas que brinda el enfoque ágil, se ha ubicado este punto bien cercano al centro de la estrella.

Personal: Los desarrolladores a pesar de no ser programadores experimentados dominan el trabajo con la tecnología que se pretende utilizar por lo que el valor para este criterio dentro de la estrella es medio.

Cultura: Se posee un buen ambiente entre el equipo de desarrollo, existe buena comunicación y confianza entre sus miembros puesto que han trabajado juntos en otras ocasiones. Las decisiones tomadas son previamente analizadas y consultadas en conjunto. El equipo presenta plena libertad en el desarrollo del proyecto, así como en la toma de decisiones de este, por lo que se distingue el valor del criterio como pequeño, muy alejado al enfoque formal o prescriptivo.

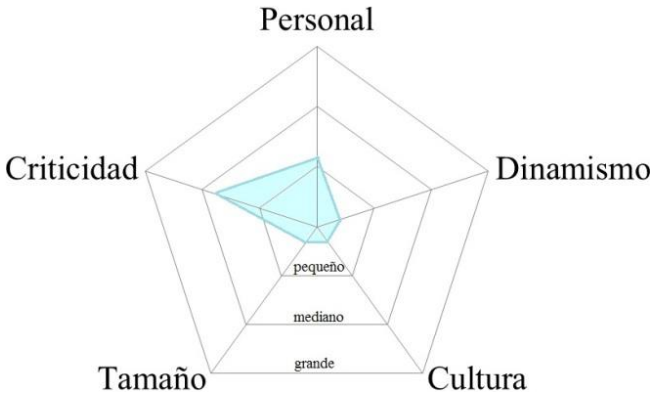


Ilustración *2* Estrella de Boehm y Turner.

Como se puede apreciar en la disposición de las aristas, se propone adoptar un enfoque ágil para el desarrollo del software en cuestión.

En la actualidad no existe una metodología general para desarrollar cualquier proyecto de software, sino que estas tienen prácticas específicas que las hacen mejores o no, dependiendo de las características del proyecto. Se decide utilizar XP por ser especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico, como es en el caso de la herramienta a desarrollar.

1.7.1.1. Programación Extrema.

Para emprender el desarrollo del proyecto se decidió utilizar XP (Programación Extrema, en inglés *Extreme Programming*), ya que los elementos que aporta se adaptan en gran medida a las condiciones de trabajo que se imponen para la herramienta a desarrollar. Además de que es sencilla, genera pocos artefactos, y es utilizada para proyectos de corto plazo, pequeños equipos de desarrolladores y breve tiempo de entrega. Los requisitos tienden a cambiar frecuentemente y según vaya avanzando el trabajo, se pueden agregar nuevas historias de usuario (HU), dividirlas e incluso eliminarlas. La

implementación se divide en iteraciones cortas y en cada iteración se entrega una versión funcional de la herramienta.

El ciclo de vida de XP consiste en 4 fases fundamentales:

Exploración: En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Planificación de la Entrega: Se establece la prioridad de cada historia de usuario, se estiman los esfuerzos que requieren cada una de las tareas al igual que el tiempo de duración de cada una de ellas.

Iteraciones: Se incluyen varias iteraciones sobre el sistema antes de ser entregado las mismas **no deben exceder las 3 semanas.**

Producción: En esta fase se realizan pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. De igual forma se toman decisiones sobre añadir nuevas características al sistema.

Con el fin de definir cada una de las etapas y sus artefactos en el transcurso del desarrollo de la aplicación, se decidió ajustar el ciclo de vida a la propuesta de solución como sigue:

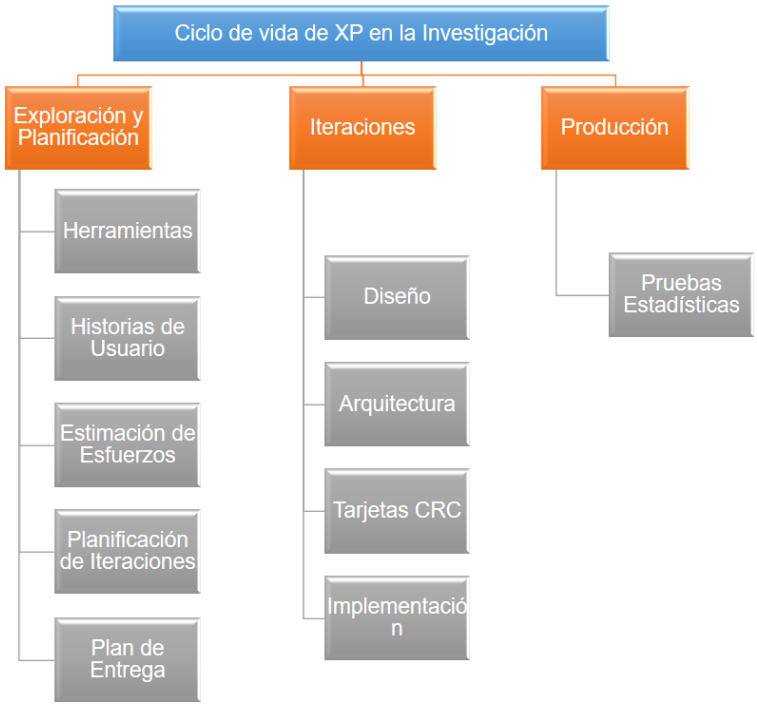


Ilustración *3* Ciclo de vida de XP en la Investigación

1.7.2. Lenguaje de programación.

Un lenguaje de programación es un conjunto de reglas, notaciones, símbolos y/o caracteres que permiten a un programador poder expresar el procesamiento de datos y sus estructuras en la computadora. En nuestra investigación adoptamos JAVA, potente lenguaje de programación, simple, sencillo de aprender y libre. Se propone su uso muy ligado a la selección de la plataforma MULAN.

La principal característica de JAVA es la de ser un lenguaje compilado e interpretado. Todo programa en JAVA ha de compilarse y el código que se genera es interpretado por una máquina virtual (Adamson, 2006). De este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma. JAVA es un lenguaje orientado a objetos de propósito general, se puede utilizar para construir cualquier tipo de proyecto. Finalmente posee un gestor de seguridad con el que poder restringir el acceso a los recursos del sistema.

1.7.3. Entorno de Desarrollo Integrado

Un IDE (Entorno de Desarrollo Integrado) es una aplicación de software que proporciona servicios integrales para el desarrollo de software. Un IDE normalmente consiste en un editor de código fuente que permite construir herramientas de automatización. En nuestro desarrollo haremos uso de NetBeans 8.0 debido al número de prestaciones que brinda y su especialización para ambientes de escritorio. Es un producto de código abierto, con todos los beneficios del software disponible en forma gratuita, el cual ha sido examinado por una comunidad de desarrolladores. Proporciona varias ventajas como es el caso de la facilidad de uso durante todo el ciclo de desarrollo y el soporte para lenguajes de modelado, lo que aumenta la productividad, y las razones que motivaron su uso.

1.8. Conclusiones parciales

Con la realización de este capítulo se dejaron fundamentados los conceptos relacionados con los algoritmos de predicción con salidas compuestas basados en transformación del estado del arte. Se realizó un estudio de los diferentes algoritmos basados en transformación y que explotan la interdependencia entre las variables de salidas. Se evidenció la utilidad del uso de dichos algoritmos en la actualidad para la resolución de problemas de la vida práctica estudiando diversas aplicaciones que forman parte del conjunto de problemas en este campo. Además, se describieron los métodos para evaluar los resultados luego de implementar el algoritmo propuesto, así como las herramientas y tecnologías a utilizar en la investigación.

Capítulo 2. Propuesta de Solución del algoritmo LCNMTR.

2.1. Introducción

En este capítulo se realiza una descripción teórica del algoritmo LCNMTR además del modelo conceptual de la propuesta de solución y los requerimientos que necesita la misma para poderse llevar a cabo a través de las historias de usuario (HU). También las especificidades de su implementación, así como una planificación que contiene una estimación del esfuerzo por HU, un plan de iteraciones y un plan de entrega.

2.2. Descripción teórica del algoritmo LCNMTR

El algoritmo LCNMTR fue desarrollado tomando en cuenta los modelos basados en transformación que explotan la interdependencia entre las variables de salidas en los problemas de predicción con salidas compuestas. En general estos algoritmos basados en transformación utilizan dos etapas en el proceso de entrenamiento donde en la primera etapa se hace una predicción de cada salida de manera independiente y luego se establecen las relaciones entre las variables de entrada y salida para la predicción final. Siguiendo este enfoque se define el algoritmo LCNMTR de la siguiente manera.

Sea D el conjunto de datos de entrenamiento, como se ha descrito con anterioridad, con n casos de entrenamiento p variables predictoras (o de entrada) y q variables objetivos (o de salidas). En la primera etapa de entrenamiento el algoritmo se estima un conjunto de predictores por cada salida $\{h_1, h_2, \dots h_q\}$ usando algún algoritmo de aprendizaje convencional. Con estos predictores se obtienen por cada salida el conjunto de predicciones para todos los casos de entrenamientos $\{\vec{y}_1^*, \vec{y}_2^*, \dots \vec{y}_n^*\} \in R^n \times R^q$. En nuestra propuesta evaluamos como regresores el algoritmo basado en la regla de los k-vecinos más cercanos KNN y además para comparar los resultados se emplearon la máquina de soporte vectorial para regresión SVR, regresión lineal LR, y árboles de decisión para regresión *Reptree*. Nótese que varios de estos modelos son no lineales al igual que KNN, lo cual permite modelar apropiadamente nuestro problema.

El algoritmo clásico KNN para predecir cada salida de manera independiente ha sido propuesto en (Zerosky et. al. 2011). Este algoritmo predice una nueva instancia a partir del conjunto de los k-vecinos $N_k(x_i)$ entre los datos de entrenamiento. Para obtener la predicción de cada salida se hace de manera independiente empleando la siguiente expresión general del algoritmo:

$$\widehat{y_j^l} = \frac{\sum_{x_j \in N_k(x_i)} K(x_i, x_j) y_j^l}{\sum_{x_j \in N_k(x_i)} K(x_i, x_j)}$$

En este enfoque $K(x_i, x_j)$ indica la función de peso la cual depende de la medida de distancia. El supraíndice l indica la salida que se está analizando. La medida de distancia entre pares de objetos será la euclidiana.

$$d(x_i, x_j) = (\vec{x}_i - \vec{x}_j)^T(\vec{x}_i - \vec{x}_j)$$

El principal aporte del LCNMTR radica en la segunda etapa de entrenamiento donde se pretende mejorar la predicción combinando las predicciones iniciales y estimando la influencia que tiene la predicción de la j-esima salida en la i-esima. Para ello se aprende un conjunto de pesos que representan el grado de importancia que tiene una variable en la predicción de otra. En este enfoque la predicción final quedaría expresada como combinación lineal de las predicciones individuales de cada salida de la siguiente manera:

$$\widehat{y_j^l} = \frac{\sum_m \sum_{x_j \in N_k(x_i)} W_{lm} K(x_i, x_j) y_j^l}{\sum_{x_j \in N_k(x_i)} K(x_i, x_j)}$$

Para estimar la matriz de pesos lo hacemos de manera análoga a la regresión lineal múltiple la cual en su forma normal estima los coeficientes de la matriz W_{lm} de la forma:

$$W = (\hat{Y}\hat{Y} + \gamma I)^{-1}\hat{Y}Y$$

Como se puede apreciar este enfoque basado en KNN permite aprender los pesos que establecen el nivel de influencia de una variable sobre otra en el conjunto de variables de salidas.

2.3. Modelo conceptual de la propuesta

La línea base que se propone como solución para el algoritmo LCNMTR se rige por la arquitectura de los sistemas clásicos de aprendizaje automático los cuales cuentan de dos procesos fundamentales: entrenamiento y clasificación.

En el proceso de entrenamiento se aprende un modelo matemático a partir de un conjunto de datos representativos de un problema real. Estos datos han sido colectados como parte de la información histórica del problema, el cual queda definido en forma de variables conocidas como predictoras. Por otra parte, esta colección de datos puede presentar, en algunos de los casos del conjunto de datos, valores anómalos, valores ausentes (valores que no han sido medidos para una variable) o variables en diferentes escalas de medición lo cual es muy común en estos problemas.

Para dar solución a estas problemáticas se emplean algoritmos de pre-procesamiento de datos para solucionar estos problemas. En la presente investigación el tratamiento

dado a los valores ausentes es ignorar estos casos en la base de datos, de igual manera se procede con los valores anómalos mientras que todos los conjuntos de datos se normalizaron a valores con media cero y varianza uno de modo que se encuentren en la misma escala de medición.

Cuando se cuenta con un conjunto de datos históricos sobre un problema es necesario separar los mismos en datos de entrenamiento y prueba teniendo muestras significativas en ambos conjuntos de datos relacionados con el problema. Para ello se emplean el procedimiento de validación cruzada con 5 *folds* según las recomendaciones de (Demšar, 2006). La siguiente ilustración ejemplifica el proceso de evaluación del algoritmo LCNMTR propuesto en el presente trabajo.

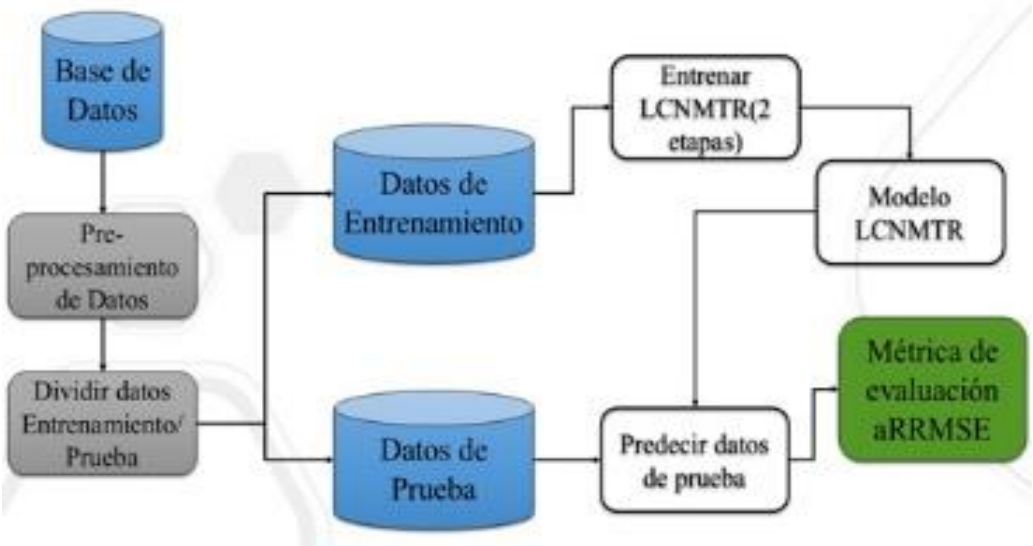


Ilustración *4* Diagrama conceptual del funcionamiento del algoritmo LCNMTR

2.4. Arquitectura del Sistema

La arquitectura de un software es el diseño de más alto nivel de la estructura de un sistema. Toda arquitectura de software debe definir diversos aspectos del software, y generalmente, cada uno de estos se describe de una manera más comprensible si se utilizan distintos modelos o vistas (Pressman R. S., 2007).

Para el desarrollo del sistema se empleará la arquitectura en tres capas, la misma consiste en dividir el sistema en varias capas lográndose de esta forma reducir el grado de complejidad. La aplicación contiene códigos para la presentación, procesamiento y almacenamiento de datos, la diferencia se encuentra en la organización del mismo. Ofrece ventajas de tipo organizativo debido a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La utilización de esta arquitectura permite desarrollar capas en paralelo, además posibilita un mantenimiento más sencillo ya que se puede modificar un componente de ser necesario en lugar de toda la aplicación y permite agregarle nuevos módulos aumentado su flexibilidad.

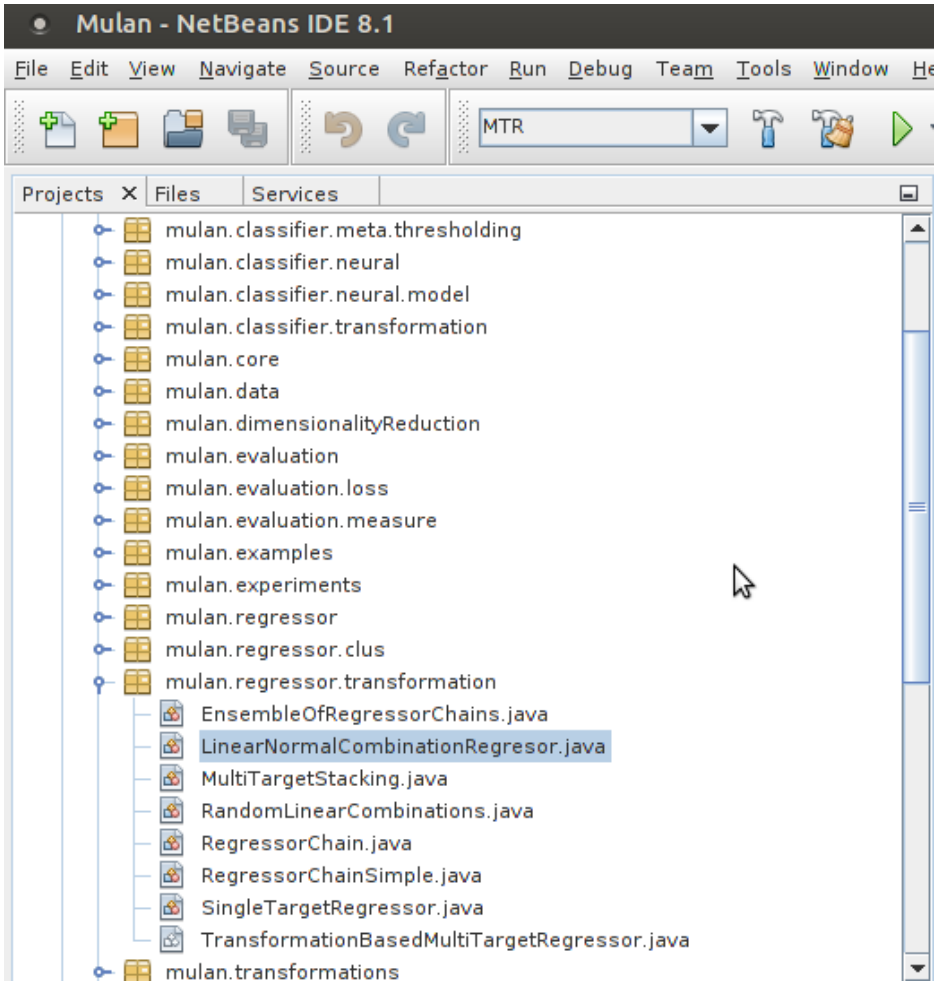


Ilustración *5* Arquitectura del sistema

En la siguiente ilustración se muestra la relación de la clase implementada *LinearNormalCombinationRegressor* y de los paquetes y clases principales de MULAN.

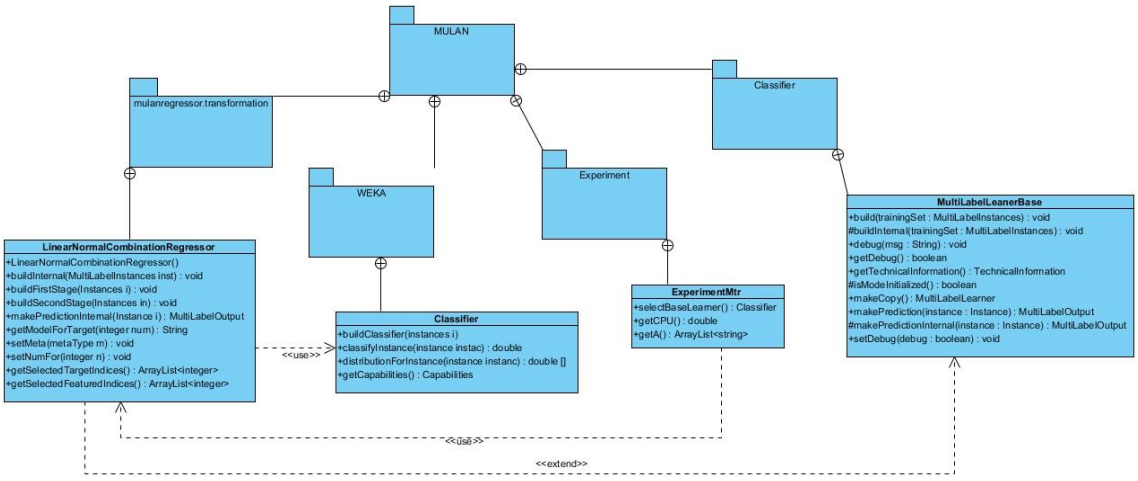


Ilustración *6* Diagrama de clases

2.5. Requerimientos de algoritmo propuesto

La primera fase de XP comienza con el proceso de identificación de las Historias de Usuario (HU) en dependencia de las necesidades del cliente, el cual define un conjunto de funcionalidades necesarias para el desarrollo del producto final. A partir de las funcionalidades requeridas se procede a identificar las HU necesarias para el funcionamiento de la aplicación. Las requisitos funcionales definidos por el cliente fueron:

- Leer uno o varios ficheros de datos en formato arff.
- Dividir los datos en entrenamiento y prueba siguiendo la metodología de validación cruzada para ejecutar el experimento.
- Entrenar los modelos de aprendizaje que son la base del algoritmo MTS.
- Graficar e imprimir los elementos que componen el modelo de aprendizaje para su inspección visual.
- Predecir cada conjunto de datos de pruebas usando los modelos de aprendizaje.
- Evaluar la predicción haciendo uso de la medida aRRMSE.
- Escribir los resultados en forma tabular en ficheros de texto y haciendo uso del formato latex.

Por otra parte, el algoritmo debe tener en cuenta los siguientes elementos **no funcionales**.

Software

- Es necesaria la instalación de la máquina virtual de JAVA en su versión 8 y la biblioteca WEKA 3.7.10.

Hardware

- La computadora donde se ejecutará el algoritmo debe tener las siguientes prestaciones mínimas: 4 Gb de RAM y microprocesador Intel Core i3 o AMD A6.

2.6. Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño (E. Gamma, R. Helm, R. Johnson, J. Vlissides, & ., n.d.).

Después de haber realizado un análisis sobre los criterios de expertos en el tema, se puede resumir, que los patrones de diseño se logran combinar en componentes que resuelven grandes problemas. Así como adaptar a problemas de menor complejidad siempre y cuando se haga un uso correcto de los mismos. Además, cabe destacar que existen varios grupos de patrones de diseño, pero la solución se basa en los más usados en función del objetivo que se ha planteado en este trabajo.

2.6.1. Patrones GRASP

Los patrones de software para la asignación de responsabilidades (GRASP), describen los principios esenciales de la asignación de responsabilidades a objetos, por tanto fueron considerados en la confección de las clases que componen el modelo de diseño (Craig Larman, 2010).

Patrón experto: Define el principio fundamental en virtud del cual se asignan las responsabilidades en un diseño orientado a objetos. Plantea que la responsabilidad debe recaer en la clase que cuenta con la información necesaria para cumplir con ella. De forma tal que el sistema sea más fácil de entender, mantener y ampliar, permitiendo reutilizar los componentes creados en futuras aplicaciones.

Ejemplo: Las clases MULAN cuentan con la información necesaria para cumplir con cada una de las responsabilidades que le corresponden.

Patrón creador: Define quien debe ser el responsable de crear una nueva instancia de una clase. Plantea que debe recaer en la clase que agrega, contiene, registra, utiliza o tiene los datos de inicialización de la nueva instancia. La prioridad en caso de que exista más de una posibilidad es en el orden enunciado. El propósito fundamental de este patrón es asignar responsabilidades relacionadas con la creación de objetos producidos en cualquier evento. Facilita un Bajo Acoplamiento, supone menor dependencia respecto al mantenimiento y mejores oportunidades de reutilización.

Ejemplo: La clase ExperimentMtr es la responsable de crear instancias de la clase LinearNormalCombinationRegressor.

Patrón Bajo Acoplamiento: Define como fomentar una menor dependencia de una mayor reutilización. Plantea mantener la fuerza de las conexiones entre clases lo más bajo posible. Este parámetro es conocido como acoplamiento. No puede verse de forma aislada como Experto o Alta Cohesión, el acoplamiento tal vez no sea muy importante, sino se busca la reutilización.

Ejemplo: La Ilustración *6* muestra cómo fue empleado el patrón bajo acoplamiento en la implementación de la aplicación.

Patrón Alta Cohesión: Define que las responsabilidades asignadas a una misma clase deben guardar relación y estar enfocadas al mismo objetivo. Una Alta Cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas donde todos sus elementos trabajan juntos para proporcionar algún comportamiento bien delimitado.

Ejemplo: La Ilustración *6* muestra cómo fue empleado el patrón bajo acoplamiento en la implementación de la aplicación.

2.7. Fase de Exploración

La metodología para desarrollo de software *Extreme Programming*, comienza su ciclo de desarrollo durante la fase de Exploración, que corresponde con la fase donde se identifican las historias de usuarios, que no serán más que los elementos rectores dentro del desarrollo del software. Además, en la misma se explora la familiarización del equipo de trabajo con las tecnologías y herramientas que se emplearán a lo largo del desarrollo.

2.7.1. Historias de Usuario

Como parte de la metodología de desarrollo seleccionada para la investigación, se debe definir en la primera fase las historias de usuarios (HU) las cuales detallan las características que deberán tomar la propuesta, así como la plataforma experimental sobre la cual está soportada. Es aquí donde el cliente plantea a grandes rasgos lo que necesita a través de las HU, lo que facilita a los programadores estimar el tiempo de desarrollo. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

El cliente es el encargado de asignarle una prioridad a cada HU y es el equipo de desarrollo el encargado de asignarle un costo, este se traduce en las semanas que llevará el desarrollo de las mismas. Si las HU según lo planificado demoran en desarrollarse, se sugiere dividirla en HU más pequeñas. También, es importante

destacar que las HU nuevas pueden describirse en cualquier momento, con esto se comprueba la flexibilidad de la metodología. Estas se escriben desde la perspectiva del cliente, aunque los desarrolladores pueden brindar también su ayuda en la identificación de las mismas (Beck 1999).

Estas son descritas en una tabla dividida por las siguientes secciones, aunque como en este caso, solo implementamos un algoritmo que no posee interfaz de usuario, entonces se adecúa la tabla y no se incluye la última sección:

- **Número**: número de la historia de usuario incremental en el tiempo.
- **Nombre de Historia de Usuario**: el nombre de la historia de usuario sería para identificarlas mejor entre los desarrolladores y el cliente.
- **Modificación de Historia de Usuario Número**: si sufrió alguna modificación anterior.
- **Usuario**: involucrados en el desarrollo de la HU.
- **Iteración Asignada**: número de la iteración.
- **Prioridad en Negocio**: Alta, Media o Baja.
- **Riesgo en Desarrollo**: Alta, Media o Baja.
- **Puntos Estimados**: tiempo estimado que se demorará el desarrollo de la HU.
- **Puntos Reales**: tiempo que se demoró en realidad el desarrollo de la HU.
- **Descripción**: breve descripción de la HU.
- **Observaciones**: señalamiento o advertencia del módulo.
- **Prototipo de Interfaz**: Prototipo de interfaz si aplica.

La prioridad en el negocio:

Alta: Se le otorga a las HU que resultan funcionalidades fundamentales en el desarrollo del módulo, a las que el cliente define como principales para el control integral del módulo.

Media: Se le otorga a las HU que resultan para el cliente como funcionalidades a tener en cuenta, sin que estas tengan una afectación sobre el módulo que se esté desarrollando.

Baja: Se le otorga a las HU que constituyen funcionalidades que sirven de ayuda al control de elementos asociados al equipo de desarrollo, a la estructura y no tienen nada que ver con el módulo en desarrollo.

El riesgo en su desarrollo:

Alta: Cuando en la implementación de las HU se considera la posible existencia de errores que lleven a la inoperatividad del código.

Media: Cuando pueden aparecer errores en la implementación de la HU que puedan retrasar la entrega de la versión.

Baja: Cuando pueden aparecer errores que serán tratados con relativa facilidad sin que traigan perjuicios para el desarrollo del proyecto.

Tomando en cuenta las características descritas por el cliente con anterioridad podemos plantear las siguientes HU en nuestra propuesta.

- HU1. Cargar base de datos.
- HU2. Dividir los datos en entrenamiento y prueba.
- HU3. Pre-procesar datos.
- HU4. Entrenar el algoritmo LCNMTR.
- HU5. Imprimir resultados de entrenamiento.
- HU6. Predecir datos de prueba.
- HU7. Evaluar predicción usando una o varias medidas.
- HU8. Imprimir los resultados finales de predicción.

A continuación, en las tablas 2 y 3 se muestran las descripciones detalladas de las HU1 y HU4 que son parte de las funcionalidades criticas del sistema el resto se podrán consultar en el ANEXO A.

Historia de Usuario		
Número: HU1	**Usuario:** Cliente	
Nombre de historia: Cargar base de datos.		
Prioridad en negocio: Media (Alta / Media / Baja)	**Riesgo en desarrollo:** Media	
Puntos estimados: 3/5	**Puntos reales:** 3	**Iteración asignada:** 1
Programador responsable: Juan Manuel		
Descripción: Permite cargar un archivo con extensión "*.arff*", de una dirección especificada que contenga una base de datos.		
Observaciones: Debe poder cargarse una o varias bases de datos.		

Tabla *2* HU1 Cargar base de datos

Historia de Usuario

Número: HU4	**Usuario:** Cliente	
Nombre de historia: Entrenar el algoritmo LCNMTR.		
Prioridad en negocio: Alta (Alta / Media / Baja)	**Riesgo en desarrollo:** Alta	
Puntos estimados: 3/5	**Puntos reales:** 3	**Iteración asignada:** 2
Programador responsable: Pedro		
Descripción: Permite entrenar el modelo LCNMTR en sus dos etapas aprendiendo la matriz de trasformación de los pesos a través del modelo de regresión lineal multivariado y de los datos de entrenamiento.		
Observaciones:		

Tabla *3* HU4 Entrenar algoritmo LCNMTR

2.8. Fase de Planificación

En la fase de **Planificación** de la metodología XP como se menciona en (Beck, 1999), se realiza una estimación del esfuerzo que costará implementar cada HU. La **estimación de esfuerzo** se expresa utilizando como medida el punto, generalmente una HU no excede los 3 puntos. Un punto es considerado por (Beck, 1999) como una semana ideal de trabajo, donde los miembros del equipo de desarrollo trabajan el tiempo planeado sin ningún tipo de interrupción. Esta estimación incluye todo el esfuerzo asociado a la implementación de una HU, por ejemplo, las pruebas unitarias, la preparación y ejecución de las pruebas de aceptación. Además, se define la **velocidad de desarrollo** del proyecto y el **plan de entregas**.

2.8.1. Estimación de esfuerzo por HU.

Para el desarrollo del algoritmo LCNMTR, se realizó una estimación del esfuerzo necesario para realizar cada una de las HU identificadas. El esfuerzo es estimado en puntos, donde un punto representa una semana ideal de implementación que no es más que el desarrollo ininterrumpido de las HU. La estimación se refleja en la descripción de las HU en la sección de puntos estimados. La estimación general se muestra en la siguiente tabla:

Historias de Usuario	Puntos estimados
HU1. Cargar base de datos.	1
HU2. Dividir los datos en entrenamiento y prueba.	2

HU3. Pre-procesar datos.	1
HU4. Entrenar el algoritmo LCNMTR.	2
HU5. Imprimir resultados de entrenamiento.	1
HU6. Predecir datos de prueba.	2
HU7. Evaluar predicción usando una o varias medidas.	2
HU8. Imprimir los resultados finales de predicción.	1

Tabla *4* Estimación de Esfuerzo por HU

2.8.2. Velocidad de Desarrollo

La velocidad del proyecto se define en dependencia de la cantidad de iteraciones necesarias para el desarrollo de la aplicación. El equipo de trabajo define la cantidad de iteraciones según (Beck 1999) teniendo en cuenta dos elementos: el total de puntos estimados y el tiempo de cada iteración (de 1 a 3 semanas).

La velocidad del proyecto se vuelve a calcular en cada iteración teniendo en cuenta la presente estimación y las tareas no implementadas en la iteración anterior (de ser así provocaría una demora en la entrega final del proyecto). Teniendo en cuenta que el total de puntos estimados es 12, el mayor número de semanas por iteración es 3 y la precedencia entre algunas de las HU para el correcto funcionamiento de la otra, se estiman un total de 4 iteraciones.

A continuación, se describen las iteraciones necesarias para la implementación del algoritmo LCNMTR y su base experimental:

Iteración 1: En la presente iteración se ejecuta la HU1 y HU2 de prioridad media pero necesaria para dar paso a la segunda iteración. Al finalizar la implementación, la primera versión será entregada al cliente, el cual es el encargado de realizarle pruebas y comprobar si satisfacen sus expectativas.

Iteración 2: En la segunda iteración se implementa la HU4 y HU5 de prioridad alta. Primeramente, se corrigen los errores encontrados por el cliente en la iteración anterior y se prosigue con el desarrollo de la HU4 con la HU5 se podrán comprobar los resultados. Al concluir, se muestra al cliente una segunda versión con los errores corregidos, verificando si satisface sus intereses.

Iteración 3: En esta iteración se implementan las HU3 y HU6 de prioridad media y alta. Al implementar la HU3 se vuelven a verificar los resultados de la iteración 2 ahora con los datos normalizados. Luego se implementa la HU 6 y se obtiene la predicción con el algoritmo LCNMTR. Una vez finalizada la implementación se obtendrá una porción media del módulo para la satisfacción del cliente, quien aprobará o no los resultados obtenidos de las pruebas realizadas en la presente iteración.

Iteración 4: En esta iteración se completará el desarrollo del algoritmo con la implementación de las HU 7 y 8. Al inicio se corrigen los errores encontrados en la iteración anterior y al finalizar se contará con la primera versión del producto final.

Al finalizar las iteraciones, se realizarán pruebas para definir si cumple con todos los requerimientos necesarios del cliente. Permitiéndose así tener un 100 % de la aplicación implementada y disponible para ser probada por el cliente en su totalidad.

La siguiente tabla muestra la velocidad del proyecto en semanas y el nombre de las HU que se implementan en cada iteración.

Iteraciones	Historias de Usuario	Semanas
1	HU1, HU2	3
2	HU4, HU5	3
3	HU3, HU6	3
4	HU7, HU8	3

Tabla *5* Velocidad del Proyecto

2.8.3. Plan de Entrega

El plan de entrega consiste en fijar la fecha de entrega de cada iteración y los riesgos a considerar en la implementación de una HU. Sí el grado de dificultad es determinado por los programadores como mínimo la HU pasa a formar parte de otra. Estos datos se almacenan en los campos: riesgo de desarrollo y puntos reales en la HU, el responsable de llenar estos datos es únicamente el programador. Al final de cada iteración se le realizarán pruebas al producto obtenido, las fechas de inicio y fin que se indican en la siguiente tabla:

Iteraciones	Semanas	Fecha de inicio	Fecha final
1	3	6-02-2017	24-02-2017
2	3	27-02-2017	17-03-2017
3	3	20-03-2017	7-04-2017
4	2	17-04-2017	05-05-2017

Tabla *6* Plan de Entrega

2.9. Conclusiones Parciales

Con el desarrollo de este capítulo se describió la propuesta de solución del algoritmo a implementar. Además de que se identificaron las características que el sistema debe cumplir para su correcto funcionamiento, también se describieron cada una de las HU y se realizó una estimación del esfuerzo dedicado a la implementación de cada una de ellas, según el orden establecido, de acuerdo a las necesidades del cliente. Con la obtención del plan de entregas se logró delimitar el ciclo de desarrollo del sistema con el tiempo requerido.

Capítulo 3. Implementación y Resultados

3.1. Introducción

La etapa implementación del software en la metodología XP es un proceso que se realiza de forma iterativa, obteniendo como resultado de cada una de estas un producto funcional que debe ser sometido a pruebas y mostrado al cliente para permitir una retroalimentación por parte de los desarrolladores. En este capítulo se evalúa la calidad de la solución a través de las pruebas de software realizadas, así como los resultados obtenidos.

3.2. Tarjetas CRC (Clase- Responsabilidad- Controlador)

El diseño de aplicaciones, en la metodología XP no requiere la representación del sistema mediante diagramas de clases utilizando notación UML[1], en su lugar se usan otras técnicas como las tarjetas CRC. Estas determinan responsabilidades y colaboraciones de las clases. El desarrollo de cualquier proyecto requiere de un buen diseño de sus clases para de esta forma realizarlo con la mejor calidad posible y así el cliente quede satisfecho.

En la metodología XP el diseño de las clases se realiza a través de las tarjetas CRC, para de esta forma ayudar al refinamiento de las clases. De forma organizada cada tarjeta representa una clase, donde se describen las responsabilidades que tiene y las clases colaboradoras que se relacionan con la misma. Las tarjetas CRC también ayudan a diseñar el sistema en conjunto entre todo el equipo de desarrollo, aunque su principal objetivo es propiciar el enfoque orientado a objetos y reducir el modo de pensar procedimental. Están diseñadas en cuatro secciones: nombre de la clase, descripción, responsabilidades y colaboradores. Una clase describe un objeto o evento del sistema, mediante sus atributos y métodos. Las responsabilidades de estas se describen por las tareas que realiza o por los métodos y los colaboradores son las demás clases con las que interactúa para cumplir con sus responsabilidades.

A continuación, se describen algunas de las tarjetas CRC diseñadas para la implementación del sistema:

3.2.1. Tarjeta CRC #1 Entrenamiento

Nombre de la clase: ExperimentMTR

[1] Lenguaje Unificado de Modelado, es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad para visualizar, especificar, construir y documentar un sistema.

Descripción: Clase principal, que contiene las diferentes llamadas a los métodos MTS, MTSC, ERC y ERCC.	
Responsabilidad	**Colaboradores**
Ejecutar los diferentes algoritmos de predicción con salidas múltiples implementados en MULAN.	LinearNormalCombinationRegression

Tabla *7* Tarjeta CRC#1 Entrenamiento

3.2.2. Tarjeta CRC #2 Múltiples bases de datos

Nombre de la clase: LinearNormalCombinationRegressor	
Descripción: Clase donde se implementa el algoritmo propuesto dividido en las etapas de aprendizaje y predicción**.**	
Responsabilidad	**Colaboradores**
Ejecutar las etapas de aprendizaje y predicción del algoritmo LNCR.	ExperimentMtr Classifier MultiLabelLearnerBase

Tabla *8* Tarjeta CRC#2 Múltiples bases de datos

3.2.3. Tarjeta CRC #3 Resultados

Para un mejor entendimiento de cómo están relacionadas estas clases y las funcionalidades que realizan cada una de ellas, se elaboró un diagrama de clases (ver Ilustración *6*).

3.3. Pruebas al Sistema

A continuación, se mostrarán los resultados obtenidos al comparar el algoritmo implementado con los algoritmos cuyo comportamiento es considerado como estado del arte según (Spyromitros-Xiou, Tsoumakas, Groves, & Vlahavas, 2014) que son el MTS, el ERC y el ST. Primeramente, se presenta una tabla donde se muestra la medida de evaluación *aRRMSE* que se obtiene al ejecutarlos en MULAN. Luego, los resultados tras aplicar la Prueba de Friedman y como prueba *post-hoc* la Prueba de Bonferroni-Dunn. Además, se muestra un Caso de Estudio para el algoritmo donde se detallan los resultados obtenidos al aplicarlo con una base de datos de prueba.

3.3.1. Resultados

En la siguiente tabla se puede apreciar el *aRRMSE* del clasificador implementado y su variante utilizando la validación cruzada en 13 bases de datos. Los campos en negrita representan los mejores resultados para cada base de datos.

Base de Datos	ST	LCNMTRCV	LCNMTR	MTS	ERC
edm	0,897	**0,802**	0,894	0,857	0,875

wq	0,979	0,936	0,990	0,982	**0,933**
sf1	1,078	**1,044**	1,149	1,084	1,045
sf2	1,145	1,321	1,154	**1,072**	1,249
jura	0,738	**0,733**	0,744	0,752	0,734
enb	0,280	0,280	0,278	**0,267**	0,273
oes10	0,463	0,477	0,467	**0,456**	0,459
oes97	0,556	0,609	**0,547**	0,555	0,552
SCPF	0,897	0,886	0,878	0,895	**0,856**
atp7d	0,638	0,654	0,649	0,618	**0,616**
atp1d	0,470	0,471	0,476	0,470	**0,461**
andro	0,570	0,675	0,546	0,560	**0,526**
slump	**0,733**	0,739	0,736	0,741	0,735

Tabla *9* Resultados de la medida aRRMSE

Como se puede observar, tomando como dominio específico las 13 bases de datos con las que se efectuaron las pruebas, el algoritmo propuesto se encuentra en segundo lugar con el algoritmo MTS obteniendo los mejores resultados en 3 bases de datos.

3.4. Prueba de Friedman

Tomando como base la Tabla, donde se muestra el *aRRMSE* de cada algoritmo para cada base de datos, se aplica la Prueba de Friedman para determinar si existen diferencias entre ellos, obteniendo un $\chi^2 = 1.7625$ con un *p_value* de 0.1505 mayor que $\alpha = 0.05$. Esto significa que se acepta la hipótesis nula, es decir, no existen diferencias significativas entre los algoritmos. Por tanto, el algoritmo LCNMTR es un algoritmo competitivo con los algoritmos del estado del arte para problemas de predicción con salidas compuestas. Finalmente se obtiene un *ranking* de los algoritmos como se puede observar en la siguiente ilustración:

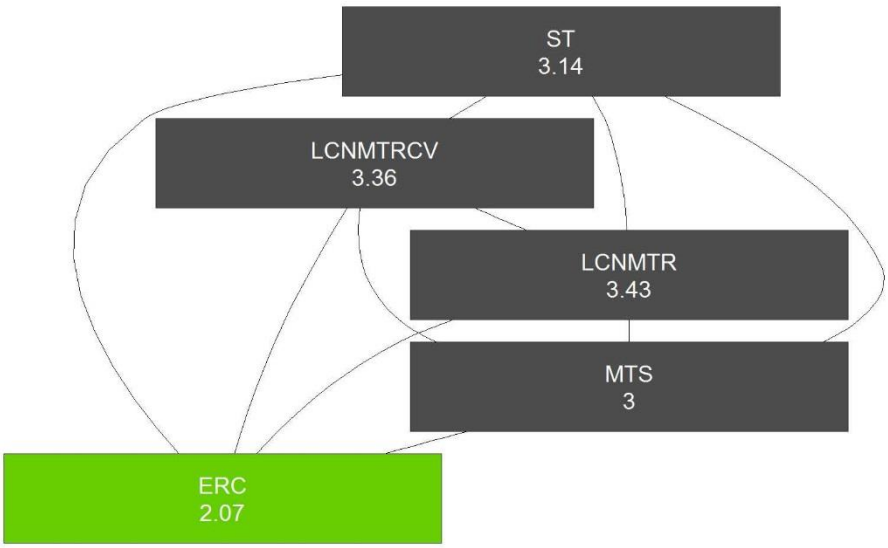


Ilustración *7* Rango promedio de los algoritmos

Obtenido este ranking se procedió a realizar la Prueba de Bonferroni-Dunn, que los compara con uno de control y utiliza la ecuación de diferencia crítica planteada por Nemenyi. Esta prueba *post-hoc* plantea que el desempeño de los clasificadores es significantemente diferente si el rango ordinario correspondiente difiere al menos la diferencia crítica (CD), obteniéndose los siguientes resultados en las 13 bases de datos de prueba para $\alpha = 0.05$:

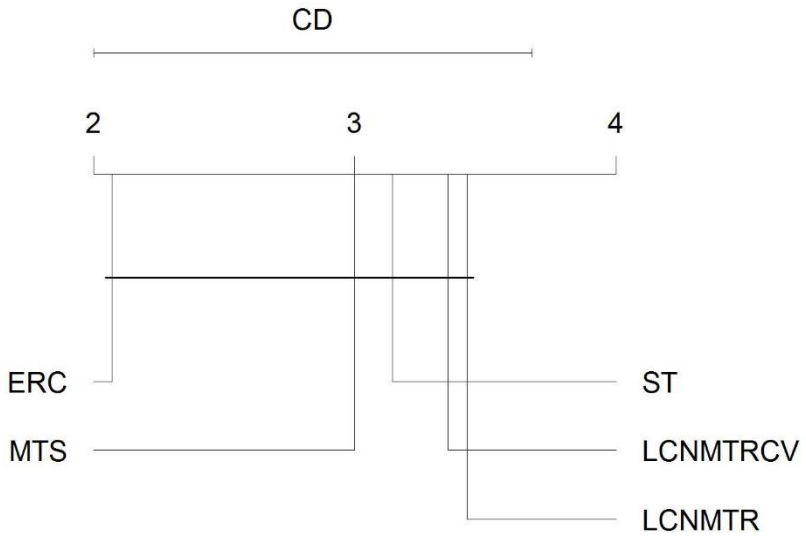


Ilustración *8* Resultados de la Prueba Bonferroni-Dunn

Como se puede apreciar en la ilustración, la distancia entre los algoritmos no supera la Diferencia Crítica establecida por Nemenyi, por lo tanto, se puede afirmar, que no existen diferencias significativas entre ellos con un 95% de certeza.

3.5. Caso de estudio de la herramienta gráfica para el diseño de experimentos en MULAN.

Tomando para este caso de estudio la base de datos Andro que tiene un total de 49 instancias, 30 atributos y 6 clases. La siguiente tabla muestra los resultados de aplicar los distintos algoritmos sobre esta base de datos (para ver la tabla completa ver ANEXO B):

Algoritmo	Salida	Clase	aRRMSE	Tiempo Real	Tiempo_CPU
ST	0	all	0,57008555	42	47
ST	1	Target	0,35788266	42	47
ST	2	Target_2	0,47798119	42	47
LCNMTRCV	0	all	0,67468382	109	109
LCNMTRCV	1	Target	0,42941204	109	109

LCNMTRCV	2	Target_2	0,66471559	109	109
LCNMTR	0	all	0,54644475	50	47
LCNMTR	1	Target	0,34541245	50	47
LCNMTR	2	Target_2	0,47818076	50	47
MTS	0	all	0,56024032	82	78
MTS	1	Target	0,34555061	82	78
MTS	2	Target_2	0,49647405	82	78
ERC	0	all	0,52558732	789	468
ERC	1	Target	0,32578553	789	468
ERC	2	Target_2	0,43104133	789	468

Tabla *10* Resultados de aplicar los algoritmos a la base de datos Andro.

Como se puede observar el algoritmo LCNMTR tiene uno de los menores tiempos de ejecución para el caso específico de la base de datos Andro. Generalizando estos resultados para las 13 bases de datos mencionadas anteriormente se obtuvieron los siguientes resultados con respecto al tiempo de ejecución en milisegundos.

Base de Datos	Instancias	Salida	Densidad	ST	LCNMTRCV	LCNMTR	MTS	ERC
andro	49	6	294	42	109	50	82	789
edm	154	2	308	246	224	89	137	175
slump	103	3	309	31	83	41	66	201
sf1	323	3	969	162	437	233	410	1080
jura	359	3	1077	253	687	364	609	1645
enb	768	2	1536	469	1259	773	1238	977
atp7d	296	6	1776	7562	19952	10956	18793	81348
atp1d	337	6	2022	9671	25487	14536	24282	102470
sf2	1066	3	3198	993	2771	1550	2522	6249
scpf	1137	3	3411	3021	8195	5246	8365	18874
oes97	334	16	5344	1877	49597	30771	50411	200284
oes10	410	16	6560	30378	80063	50685	82482	322556
wq	1060	14	14840	6755	18769	10452	29577	147254

Tabla *11* Resultado aplicación de los algoritmos en las 13 bases de datos

Tomando los resultados de la tabla anterior, el gráfico siguiente ejemplifica los tiempos de ejecución de los distintos algoritmos, evidenciándose que el LCNMTR y el LCNMTRCV están dentro de los algoritmos con menor tiempo de ejecución lo cual es altamente útil en el caso de que se estén realizando experimentos en grandes bases de datos.

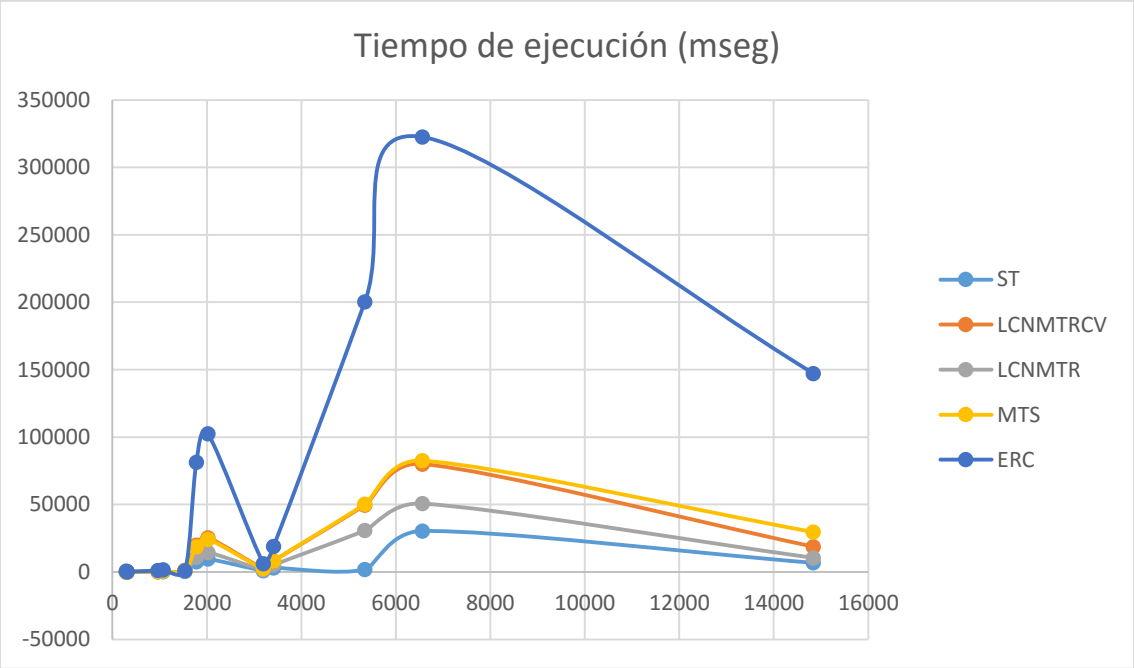


Gráfico *1* Tiempo de ejecución de los algoritmos

3.6. Conclusiones Parciales

En este capítulo se conformaron las tarjetas CRC como artefacto de la metodología seleccionada y muy útiles para guiar la implementación del algoritmo LCNMTR. Además, se realizaron las pruebas estadísticas de Friedman y el test *post-hoc* de Bonferroni-Dunn que arrojó como resultado que el algoritmo propuesto no tiene diferencias significativas con respecto a los algoritmos del estado del arte con un 95% de certeza. El desarrollo del caso de estudio demostró que el algoritmo propuesto y su variante utilizando validación cruzada están entre los algoritmos con menor tiempo de ejecución.

Conclusiones

A modo de conclusión se puede afirmar que se caracterizó el marco teórico conceptual de los algoritmos de predicción con salidas compuestas basados en transformación al fundamentar los conceptos relacionados del estado del arte donde explotan la interdependencia entre las variables de salidas. Esto evidenció la utilidad del uso de dichos algoritmos en la actualidad para la resolución de problemas de la vida práctica estudiando diversas aplicaciones que forman parte del conjunto de problemas en este campo. Además, se hizo énfasis en las herramientas existentes de aprendizaje automático disponibles. Se describieron los métodos para evaluar los resultados luego de implementar el algoritmo propuesto, así como las herramientas y tecnologías a utilizar en la investigación. Un resultado importante fue la selección correcta de la metodología a utilizar ya que marcó las pautas para la correcta realización de la investigación.

Se describió una propuesta del algoritmo LCNMTR a implementar. Además, se identificaron las características que el sistema debe cumplir para su correcto funcionamiento. Se describieron cada una de las HU identificadas y se realizó una estimación del esfuerzo dedicado a la implementación de cada una de ellas, según el orden establecido, de acuerdo a las necesidades del cliente. Con la obtención del plan de entregas se logró delimitar el ciclo de desarrollo del sistema con el tiempo requerido. Se implementó el algoritmo LCNMTR y su variante utilizando validación cruzada en la herramienta MULAN, así como la base experimental para probar el algoritmo y compararlo con otros del estado del arte.

Finalmente se conformaron las tarjetas CRC como artefacto de la metodología XP y se realizaron las pruebas estadísticas para la evaluación del funcionamiento del algoritmo. Al aplicarse la Prueba de Friedman y el *test post-hoc* de Bonferroni-Dunn utilizando la función de Nemenyi para la CD se obtuvo como resultado que el algoritmo propuesto no tiene diferencias significativas con respecto a los algoritmos del estado del arte con un 95% de certeza. Al desarrollar un caso de estudio donde se analizaron los resultados obtenidos, se demostró que el algoritmo propuesto y su variante utilizando validación cruzada, están entre los algoritmos con menor tiempo de ejecución. De esta manera se validó la solución implementada mediante el diseño de experimentos.

Recomendaciones

- ✓ Extender el algoritmo LCNMTR haciendo uso de un algoritmo iterativo de optimización y que emplee como regularizador un enfoque de *Elastic Net* de modo que mejore la eficacia del algoritmo y al mismo tiempo realice selección de atributos.
- ✓ Poner a disposición de la comunidad científica internacional una versión de MULAN que contenga las variantes de algoritmos implementadas, de modo que pueda ser empleada por investigadores de esta área del conocimiento.

Bibliografía

Adamson, C. (2006). *What Is Java.* Retrieved from http://www.onjava.com/pub/a/onjava/2006/03/08/what-is-java.html

Almarales, F. R., Santos Martínez, G., & González, H. (n.d.). *Adaptación del algoritmo LMNN para Problemas de Predicción con Salidas Compuestas.* La Habana.

Alonso Jimenez, J. A. (2000). *Introducción al Aprendizaje Automático.*

Ávila Jiménez, J. L. (2013, Abril). Modelos de Aprendizaje Basados en Programación Genética para Clasificación Multi-Etiqueta. Universidad de Córdova.

Bakir, G., Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., & Vishwanathan, S. (2007). Predicting Structured Data (Neural Information Processing).

Bauer, E., & Kohavi, R. (1999). "An empirical comparison of voting classification algorithms: Bagging boosting,and variants," Machine learning,. *36(1-2)*, 105-139.

Beck, K. (1999, Septiembre 29). Extreme Programming Explained. *1ra edición*, 137. (E. Change, Ed.)

Blockeel, H. (1998.). *Top-down Induction of First Order Logical Decision Trees.* Bélgica.

Blockeel, L., Ramon, J., & Raedt, D. (2000). "Top-down induction of clustering trees,". arXiv preprint cs/0011032.

Borchani, H., Varando , G., Bielza, C., & Larrañaga, P. (2015). A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 5*, 216-233.

Breiman, L. (1996). "Bagging predictors," Machine learning. *24(2)*, 123-140. Retrieved 4 20, 2016

Breiman, L. (2001). Random forests," Machine learning,. *45(1)*, 5-32. (R. Schapire, Ed.) Alemania.

Breiman, L., & Friedman, J. (1997). Predicting multivariate responses in multiple linear regression . *Journal of the Royal Statistical Society: Series B (Statistical Methodology), 59*, 3-54.

Computing, R. F. (2009). *R: A language and environment for statistical computing.* Viena, Austria.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 1-30. Retrieved 6 1, 2016

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *7*, 1-30. (D. Schuurmans, Ed.) The Journal of Machine Learning.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple datasets. *Journal of Machine Learning Research, 7*, 1-30.

Durrant, B., Frank, E., Hunt, L., Holmes, G., Mayo, M., Pfahringer, B., & Witten, I. (2013). Weka 3: Data Mining Software in Java.

Dzeroski, S., Demsar, D., & J., G. (2000). Predicting chemical parameters of river water quality from bio-indicator data. *Applied Intelligence, 13*, 7-17.

Ecured. (n.d.). Retrieved 3 9, 2016, from http://www.ecured.cu/Lenguaje_de_Programaci%C3%B3n

Ecured. (2011). Retrieved 3 10, 2016, from http://www.ecured.cu/Python

Eibe Frank, I. H. (2000). *WEKA Machine Learning Algorithms in Java.* New Zealand.

Eleftherios Spyromitros-Xioufis, G. T. (2014). Multi-label classification methods for multi-target regression.

Eleftherios Spyromitros-Xioufis, G. T. (2016). *Multi-Target Regression via Input Space Expansion: Treating Targets as Inputs, Ioannis Vlahavas.*

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. 675-701.

Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics 11*, 86-92.

García, Ó. B. (2011). *Etudio comparativo de descriptores visuales para la deteccion de escenas causi-duplicadas.* Madrid.

González Diez, H. R., Santos, G., Campos, F. R., & Morell Pérez, C. (2016, Julio 3). Evaluación del algoritmo KNN-SP para problemas de predicción con salidas compuestas. *Revista Cubana de las Ciencias Informáticas(10)*, 119-129. Habana, Cuba.

Google. (2017, 5 4). *Google.* Retrieved from www.google.com

Grigorios Tsoumakas, E. S.-X. (2011). *Mulan: A Java Library for Multi-Label Learning.* Greece.

Grigorios Tsoumakas, W. G.-X. (2016). *Multi-Target Regression via Input Space Expansion: Treating Targets as Inputs.* Grecia.

Hanen Borchani, G. V. (2015). *A survey on multi-output regression.*

Ho, T. K., Hull, J. J., & Sriari, S. N. (1994). “Decision combination in multiple classifier systems,”Pattern Analysis and Machine Intelligence,. *16(1)*, 66-75.

Izenman, A. J. (1975). Reduced-rank regression for the multivariate linear model. *5(2)*, 248-264. Journal of multivariate analysis.

J. Kittler, M. H. (1998). “On combining classifiers,” Pattern Analysis and Machine Intelligence,. *20*(3), 226-239.

Jan Struyf, B. Z. (septiembre, 2011). *Clus: User's Manual.*

Jin, R., Wang, S., & Yang, Z. (n.d.). Regularized Distance Metric Learning: Theory and Algorithm.

Joanneum, F. (2005). Cross-Validation Explained.

Kaufmann, M. (2005). *Practical Machine Learning Tools and Techniques.*

Kittler, J., Hatel, M., Duin , R., & Matas, J. (1998). On combining classifiers. *Pattern Analysis and Machine Intelligence(20)*, 226-239.

Kocev D, D. S. (2009). *Using single- and multitarget regression trees and ensembles to model a compound index of vegetation.Ecol. Model.*

Larman, C. (1999). *UML y Patrones.*

Larman, C. (2003). *UML y patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Prentice Hall.

Leo Breiman, J. H. (1997). Predicting Multivariate Responses in Multiple Linear Regression. *59*(1), 3-54.

M.U. (1998). *The mathworks.*

Ming Yuan and Ali Ekici, Z. L. (2007). Dimension reduction and coefficient estimation in multivariate linear regression. *69*(3), 329-347.

Moore, A. W. (2001). Cross-Validation for detecting and preventing overfitting.

Nemenyi, P. (1963). Distribution-free multiple comparisons. *PhD thesis*. Princeton University.

Ocio, J. Z. (2010). *Aplicación de la Quimiometría para el aprovechamiento analítico de reactivos generales Revisión de la incertidumbre instrumental y del Límite de Detección multivariable.*

Oré, A. (n.d.). *CalidadySoftware.* Retrieved mayo 25, 2015, from http://www.calidadysoftware.com/testing/pruebas_unitarias1.php.

Pressman, R. (2005). *Ingeniería del Software: Un Enfoque Práctico.* Retrieved from <www.mhhe.com/engcs/pressman/>

Pressman, R. S. (2007). *Ingeniería del Software. Un enfoque práctico.(6ta Parte I).*

Pugelj, M., & Dzeroski, S. (2011). Predicting structured outputs k-nearest neighbours method. *Discovery Science*, 262-276. Springer.

Schneider, J. (1997). Cross Validation. A Locally Weighted Learning Tutorial Using Vizier.

Sierra Araujo, B. (n.d.). *Aprendizaje Automático: Conceptos básicos y avanzados.* Pearson Education.

Spyromitros-Xiou, E., Tsoumakas, G., Groves, W., & Vlahavas, I. (2014). Drawing parallels between multi-label classification and multi-target regression. arXiv preprint arXiv:1211.6581v2.

Struyf, J., Zenko, B., Blockeel, H., Vens, C., & Dzeroski, S. (2010). Clus: Users Manual.

Tedeschi, N. (n.d.). *Microsoft Developer Network.* Retrieved abril 28, 2015, from https://msdn.microsoft.com/es-es/library/bb972240.aspx.

Tsoumakas , G., Spyromitros-Xioufis, E., Vrekou, A., & Vlahavas, I. (2014). Multi-Target Regression via Random Linear Target Combinations.

Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., & Vlahavas, I. (2011). Mulan: A Java library for multi-label learning. *The Journal of Machine Learning Research(12)*, 2411-2414.

Vaisala. (n.d.). Retrieved mayo 25, 2015, from http://es.vaisala.com/sp/services/projectservices/Pages/acceptancetesting.aspx.

Witten, I. H., & Frank, E. (2000). Weka. Machine Learning Algorithms in Java. 265-320.

Witten, I. H., & Frank, E. (2005). Data Mining: Practical machine learning tools and techniques. (M. Kaufmann, Ed.)

Yang, L. (2007). An Overview of Distance Metric Learning.

ANEXO A

Historia de Usuario		
Número: HU2	**Usuario:** Cliente	
Nombre de historia: Dividir los datos en entrenamiento y prueba.		
Prioridad en negocio: Media (Alta / Media / Baja)	**Riesgo en desarrollo:** Media	
Puntos estimados: 3/5	**Puntos reales:** 3	**Iteración asignada:** 1
Programador responsable: Juan Manuel		
Descripción: Desde un archivo con extensión "*.arff*", dividir los datos en entrenamiento y prueba		
Observaciones: Identificar si el archivo ya presenta dicha división		

Tabla *12* HU2 Dividir los datos en entrenamiento y prueba

Historia de Usuario		
Número: HU3	**Usuario:** Cliente	
Nombre de historia: Pre-procesar datos		
Prioridad en negocio: Media (Alta / Media / Baja)	**Riesgo en desarrollo:** Media	
Puntos estimados: 3/5	**Puntos reales:** 3	**Iteración asignada:** 3
Programador responsable: Pedro		
Descripción: Se deben normalizar los datos a media cero y varianza uno antes de ejecutar los algoritmos.		
Observaciones: Identificar si hay valores nulos o ausentes e ignorarlos. Se debe validar que si alguna partición tiene salida con varianza cero, no tenerla en cuenta en la evaluación.		

Tabla *13* HU3 Pre-procesar datos

Historia de Usuario	
Número: HU5	**Usuario:** Cliente

Nombre de historia: Imprimir resultados de entrenamiento		
Prioridad en negocio: Alta (Alta / Media / Baja)	**Riesgo en desarrollo:** Media	
Puntos estimados: 3/5	**Puntos reales:** 3	**Iteración asignada:** 2
Programador responsable: Juan Manuel		
Descripción: Mostrar los resultados obtenidos luego del entrenamiento		
Observaciones: Graficar e imprimir los resultados obtenidos luego del entrenamiento		

Tabla *14* HU5 Imprimir resultados de Entrenamiento

Historia de Usuario		
Número: HU6	**Usuario:** Cliente	
Nombre de historia: Predecir datos de prueba		
Prioridad en negocio: Alta (Alta / Media / Baja)	**Riesgo en desarrollo:** Media	
Puntos estimados: 3/5	**Puntos reales:** 3	**Iteración asignada:** 3
Programador responsable: Pedro		
Descripción: Predecir cada conjunto de datos de pruebas usando los modelos de aprendizaje		
Observaciones: Predecir utilizando el modelo aprendido.		

Tabla *15* HU6 Predecir datos de prueba

Historia de Usuario		
Número: HU7	**Usuario:** Cliente	
Nombre de historia: Evaluar predicción utilizando una o varias medidas		
Prioridad en negocio: Media (Alta / Media / Baja)	**Riesgo en desarrollo:** Media	
Puntos estimados: 3/5	**Puntos reales:** 3	**Iteración asignada:** 4
Programador responsable: Juan Manuel		

Descripción: Evaluar los resultados obtenidos utilizando una o varias medidas
Observaciones: Evaluar la predicción haciendo uso de la medida aRRMSE.

Tabla *16* HU7 Evaluar predicción utilizando una o varias medidas

Historia de Usuario		
Número: HU8	**Usuario:** Cliente	
Nombre de historia: Imprimir los resultados finales de predicción		
Prioridad en negocio: Media (Alta / Media / Baja)	**Riesgo en desarrollo:** Media	
Puntos estimados: 3/5	**Puntos reales:** 3	**Iteración asignada:** 4
Programador responsable: Pedro		
Descripción: Escribir los resultados en forma tabular en ficheros de texto y haciendo uso del formato latex		
Observaciones:		

Tabla *17* HU8 Imprimir los resultados finales de predicción

ANEXO B

Base de Datos	Tipo de Eval.	Algoritmo	Base	Salida	Clase	aRRMSE	Tiempo Real	Tiempo_CPU
andro	cv	ST	KNN	0	all	0,57008555	42	47
andro	cv	ST	KNN	1	Target	0,35788266	42	47
andro	cv	ST	KNN	2	Target_2	0,47798119	42	47
andro	cv	ST	KNN	3	Target_3	0,57101564	42	47
andro	cv	ST	KNN	4	Target_4	0,58618471	42	47
andro	cv	ST	KNN	5	Target_5	0,72230525	42	47
andro	cv	ST	KNN	6	Target_6	0,70514387	42	47
andro	cv	LCNMTRCV	KNN	0	all	0,67468382	109	109
andro	cv	LCNMTRCV	KNN	1	Target	0,42941204	109	109
andro	cv	LCNMTRCV	KNN	2	Target_2	0,66471559	109	109
andro	cv	LCNMTRCV	KNN	3	Target_3	0,65490482	109	109
andro	cv	LCNMTRCV	KNN	4	Target_4	0,66612403	109	109
andro	cv	LCNMTRCV	KNN	5	Target_5	0,82245236	109	109
andro	cv	LCNMTRCV	KNN	6	Target_6	0,81049405	109	109
andro	cv	LCNMTR	KNN	0	all	0,54644475	50	47
andro	cv	LCNMTR	KNN	1	Target	0,34541245	50	47
andro	cv	LCNMTR	KNN	2	Target_2	0,47818076	50	47
andro	cv	LCNMTR	KNN	3	Target_3	0,56986335	50	47
andro	cv	LCNMTR	KNN	4	Target_4	0,58695528	50	47
andro	cv	LCNMTR	KNN	5	Target_5	0,65934746	50	47
andro	cv	LCNMTR	KNN	6	Target_6	0,63890922	50	47
andro	cv	MTS	KNN	0	all	0,56024032	82	78
andro	cv	MTS	KNN	1	Target	0,34555061	82	78
andro	cv	MTS	KNN	2	Target_2	0,49647405	82	78
andro	cv	MTS	KNN	3	Target_3	0,54577999	82	78
andro	cv	MTS	KNN	4	Target_4	0,56560777	82	78
andro	cv	MTS	KNN	5	Target_5	0,72968113	82	78
andro	cv	MTS	KNN	6	Target_6	0,67834834	82	78
andro	cv	ERC	KNN	0	all	0,52558732	789	468
andro	cv	ERC	KNN	1	Target	0,32578553	789	468
andro	cv	ERC	KNN	2	Target_2	0,43104133	789	468
andro	cv	ERC	KNN	3	Target_3	0,53783563	789	468
andro	cv	ERC	KNN	4	Target_4	0,56282556	789	468
andro	cv	ERC	KNN	5	Target_5	0,65061919	789	468
andro	cv	ERC	KNN	6	Target_6	0,64541669	789	468

Tabla *18* Resultado de los algoritmos en la base de datos Andro.