Universidad de La Habana
Facultad de Matemática y Computación

Mitigación de sesgos con ensembles y optimización multiobjetivo

Autor:
Jorge Mederos Alvarado

Tutores:

Juan Pablo Consuegra Ayala
Alejandro Piad Morfis

Trabajo de Diploma
presentado en opción al título de
Licenciado en (Matemática o Ciencia de la Computación)

Fecha

github.com/jmederosalvarado/thesis

Dedicación

Agradecimientos

Agradecimientos

Opinión del tutor

Opiniones de los tutores

Resumen

Resumen en español

Abstract

Resumen en inglés

Índice general

Índice de figuras

Ejemplos de código

Introducción

En la actualidad los algoritmos de aprendizaje automático están siendo aplicados en disimilies areas de la vida humana. Es comun encontrarlos aplicados en sistemas de recomendacion de compras, aplicaciones de citas, solicitudes de prestamos, contratacion personal y muchas otras areas. A raiz de ello, ha surgido un creciete interes en estudiar las potencialidades y limitaciones de los modelos de aprendizaje automatico, asi como las posibles implicaciones de confiar ciegamente en sus predicciones.

En particular, su incorporacion a tareas de toma de decisiones de alto riesgo ha dirigido la atencion de muchos investigadores hacia una nueva interrogante: ¿estaran siendo "justos"los algoritmos de aprendizaje automatico al tomar sus decisiones?

En este escenario, ha ganado popularidad el desarrollo de tecnicas para detectar y mitigar los sesgos en colecciones de datos y algoritmos de aprendizaje automatico. Tales herramientas son cruciales para desarrollar sistemas de toma de decisiones mas justos. Los estudios orientados hacia la equidad en algoritmos de aprendizaje automatico se enfocan principalmente en desarrollar tecnicas que consideren tanto la precision como la equidad de los modelos.

Motivación

Un modelo de aprendizaje de maquina se entrena con el objetivo de optimizar una unica metrica, en la mayoria de los casos la precision. Esto significa que los modelos aprenden muy bien los patrones que se presentan en los datos de entrenamiento, incluyendo aquellos patrones que representan sesgos y prejuicios que estan desafortunadamente presente en la sociedad y por ende en los datos recopilados, en algunos casos incluso amplifican estos patrones negativos. Son varias la tecnicas que se han explorado para resolver este problema, algunas se enfocan en un preprocesamiento de los datos para eliminar aquellos elementos que puedan inducir un sesgo en el modelo, otras realizan variacinoes en el metodo de entrenamiento con el mismo objetivo. Sin embargo permanece relativamente poco explorado el uso de tecnicas de optimizacion multiobjetivo que permitan al modelo optimizar hasta encontrar un buen balance entre cuan justo es y cuan preciso.

Otra tecnica que ha demostrado ser de gran utilidad en la prevencion de los sesgos en los modelos de aprendizaje de maquina es la construccion de ensamblados de multiples modelos que maximizan la varianza entre si, por lo que se minimiza el sesgo del ensamblado final.

Problematica

A pesar de que existe AutoGOAL, una biblioteca de AutoML, que permite obtener modelos para resolver problemas arbitrarios utilizando entre otras tecnicas aprendizaje de maquina. No existe una biblioteca o herramienta que permita resolver de principio a fin un problema de clasificacion utilizando aprendizaje de maquina y donde exita alguna garantia de que el modelo aprendido sea justo.

Objetivo general

Proponer una herramienta que permita resolver problemas de clasificacion utilizando aprendizaje de maquina y que permita garantizar que el modelo aprendido sea justo.

Objetivos especifico

- Encontrar modelos que maximicen la varianza para minimizar el sesgo.
- Metodos basados en metaheuristicas para optimizar los modelos utilizando simultaneamente metricas de equidad y precision.
- Explorar adicion de optimizacion multiobjetivo a AutoGOAL para que el modelo aprendido sea justo.
- Metodos basados en la combinacion de diferentes metricas en una sola, para poder aprovechar los multiples metodos de optimizacion que existen.

Capítulo 1

Estado del Arte

1.1. Equidad

No existe en estos momentos una unica definicion ampliamnte aceptada de lo que es *equidad*, sino que diferentes definiciones codifican diferentes caracteristicas que se muestran utiles en diferentes contextos. Incluso, algunas de las definiciones mas comunes presentan conflictos entre si.

A continuacion se presentan algunas de las definiciones mas utilizadas. Sea Y la clasificacion (binaria) correcta, S el atributo protegido, y Y^p la clasificacion obtenida del modelo. Las definiciones mas comunes pueden ser agrupadas en tres categorias: (i) considerando el resultado obtenido dada la clasificacion correcta; (ii) considerando la clasificacion correcta dada la clasificacion obtenida; (iii) considerando solamente el resultado obtenido. Los siguientes son ejemplos de estos tipos de metricas.

- Equal Opportunity (EO) requiere que la proporcion de verdaderos positivos sea la misma en todos los subgrupos: $P(Y^p = 1|Y = 1, S = 0) = P(Y^p = 1|Y = 1, S = 1)$
- Equalized Odds (EOdd) requiere igual proporcion de falsos positivos, ademas de EO
- Statistical Parity (SP) require que las predicciones positivas no se vean afectadas por el valor del atributo protegido, sin tomar en cuenta la clasificacion correcta: $P(Y^p = 1|S = 0) = P(Y^p = 1|S = 1)$

Los tradoff inherentes a cada nocion de fairness han sido estudiados extensamente (Dwork y col. 2012; Friedler y col. 2016; Kleinberg 2018). Escoger la definicion correcta para un problema determinado es dificil, y en la practica no puede ser delegado a un agente automatico. En su lugar, una decision humana es preferida para asegurar una decision informada.

1.2. Mitigacion de sesgos

Las tecnicas de mitigacion de sesges pueden ser divididas fundamentalmente en pre-procesamiento, in-procesamiento, post-procesamiento.

Los metodos del primer grupo logran equidad modificando la representacion de los datos, i.e., pre-procesando los datos, y luago adoptan una solucion de machine learning estandard (Calmon y col. 2017; Kamiran y Calders 2011; Zemel y col. 2013). Por ejemplo aprender una representacion a partir de resolver un problema de optimzacion con dos objetivos, codificar la informacion preservando la mayor cantidad de informacion posible y ofuscar al mismo tiempo la pertenencia al conjunto de atributos protegidos (Zemel y col. 2013). Los metodos de pre-procesamiento son agnosticos al modelo, pero sus hiperparametros, asi como los del modelos de aprendizaje de maquina seleccionado todavia necesitan ser ajustados para mejor rendimiento.

El segundo grupo de la familia consiste de en metodos de in-procesamiento que se aseguran que se cumplan ciertas restricciones de equidad durante el entrenamiento (e.g. (Donini y col. 2018; Zafar, Valera, Gomez-Rodriguez y col. 2019; Zafar, Valera, Rogriguez y col. 2017)), sin embargo solo es aplicable a una cierta clase de modelos. Por ejemplo el algoritmo aplicado en (Donini y col. 2018) solo puede ser aplicado a *kernel machines* (tales como *maquinas de soporte vectorial*), y solo a una unica definicion de equidad (i.e., EO). Aunque metodos de in-procesamiento pueden brindar muy buenos resultados para la clase del modelo que estan diseñados, frecuentemente son dificiles, o a veces imposible de extender para nuevas clases de modelos. Estos metodos tambien pueden introducir nuevos hiper-parametros que podrian requerir conocimiento especificos del dominio y experimentacion.

Las tecnicas de post-procesamiento operan ajustando el umbral de decision de modelos pre-entrenados para eventualmente lograr resultados mas justos respecto a una metrica de equidad dada. El principal problema es que post-procecesar el umbral de decision es inherentemente suboptimo y puede llevar a peores tradeoffs de eficacia y equidad. Lo que es mas, estas tecnicas no son utilizables si la informacion sensible no esta disponible en el momento de realizar las predicciones, lo cual a su vez puede llevar a problemas legales por la utilizacion de informacion sensible para realizar predicciones (MacCarthy 2018).

Una clase de metodos recientemente propuesta para tareas de clasificacion justa, conocida como meta-algoritmos, reduce la tarea de clasificacion justa a una secuenia de problemas de clasificación con costo asociado a sus errores de predicción (Agarwal, Beygelzimer y col. 2018; Agarwal, Dudík y col. 2019; Kearns y col. 2018). Las soluciones a estos problemas producen un clasificador *randomizado*. Contrario alos metodos de *in-procesamiento*, los meta-algoritmos no dependen de la tipo del modelo que se utiliza en el clasificador, sino en la capacidad de los mismos para ser re-entrenados repetidamente. En el contexto de algoritmos de minimizacion del riesgo empirico, es-

tos metodos son agnosticos al modelo de aprendizaje de maquina, pero aun necesitan implementaciones especificas basadas en la definicion de equidad seleccionada y necesitan producir un ensamblado de modelos. Limitaciones similares caracterizan un numero de enfoques que utilizan optimizacion (Chiappa y Isaac 2018; Dimitrakakis y col. 2019) o inferencia bayesiana (Kearns y col. 2018; Thomas y col. 2019), sus implementaciones tienen que estar diseñadas especificamente para ciertas definiciones de equidad.

1.3. Metodos de ensamblado

Los metodos de ensamblado estan diseñados para intentar resolver el problema de *low bias/high variance* que muestran la mayoria de los modelos de aprendizaje de maquina, haciendolos apropieados para modelos de clasificacion mas robustos (Polikar 2006). Un modelo de ensamblado esta diseñado de muchos modelos con *low bias* cuyas predicciones son combinadas para producir una prediccion final. Se asume fundamentalemente que la combinacion de varias predicciones de bajo nivel producira una salida con baja varianza mientras mantiene un low bias. Tener un conjunto diverso de modelos de bajo nivel es una caracteristica fundamental para lograr esto (Polikar 2006). Sin embargo, esto requiere, que clasificadores individuales cometan errores en diferentes instancias. La intuicion es que si cada clasificador comete diferentes errores, entonces una combinacion estrategica de estos clasificadores puede reducir el error total. Luego, es necesario lograr que cada clasificador sea lo mas unico posible, particularmente con respecto a ejemplos erroneamente clasificados.

La naturaleza de multiples hipotesis de estos modelos asegura que, si son ajustados lo suficiente, tendran resultados mejores que cualquiera de los modelos individuales en el caso general. Esto les permite tambien estimar el grado de confianza o calidad de las predicciones que producen. Las tecnicas clasicas de ensamblado incluyen *voting* and *weighted voting* (Dietterich 2000), boosting (Schapire 1990), y bagging (Breiman 1996).

En un problema de clasificacion, *voting* produce como salidao la etiqueta que tiene la mayoria de los votos, tratando cada prediccion de los modelos ensamblados como un voto. *Weighted-voting* funciona de forma similar a *voting*, pero cada modelo del ensamblado es asignado un *peso* que indica la importancia de su voto. *Boosting* ejecuta un proceso iterativo donde los modelos son entrenados secuencialmente, cada uno tratando de mejorar su rendimineto en los ejemplos entrenantes donde los modelos anteriores tuvieron peor rendimiento. Durante este proceso, cada sub-modelo es tambien asignado un peso que marca la importancia de su prediccion. *Bagging* entrena cada sub-modelo en una seleccion diferente (con reemplazo) de los ejemplos entrenantes originales. De esta forma, el modelo con una alta varianza deberia producir modelos entrenados con alta diversidad. Alternativamente, *feature bagging* funciona de forma

similar, seleccionando un subconjunto de los features en lugar de los ejemplos entrenantes, causando que features correlacionados sean analizados de forma separada en algunos sub-modelos.

Las capacidad de los metodos de ensamblado para construir modelos mas robustos los ha hecho apropiados para multiples aplicaciones. El dominio de la salud es uno de los ejemplos donde estos metodos han sido aplicados con gran exito. Por ejemplo, una aplicacion hibrida de metodos de ensamblado con redes neuronales en un entorno de aprendizaje for reforzamiento es presentada para al prediccion de infeccion de COVID-19 con gran precision (W. Jin y col. 2022). De forma similar, un metodo de toma de decision multi-criterio basado en ensemblados es propuesto para la deteccion de COVID-19 a partir del sonido de la tos en pacientes (Chowdhury y col. 2022). Existen ejemplos tambien no relacionados a la medicina, por ejemplo, en (Livieris y col. 2020) se emplearon estrategias de ensamblado como *ensemble-averaging*, *bagging* y *stacking* con metodologias avanzadas de aprendizaje profundo para predecir los precios, a nivel de hora, de criptomonedas como *Ethereum*, *Bitcoin* y *Ripple*.

Una simple pero poderosa tecnica en el contexto de los metodos de ensamblado es la llamada *snapshot ensemble* (Huang y col. 2017). Esta tecnica genera multiples clasificadores base, entrenando una sola red neuronal mientras la hace converger de forma rapida y repetida a multiples optimos locales y salvando en cada uno de dichos puntos los parametros del modelo. Todas las redes neuronales son entonces ensambladas para producir el clasificador final. Estos *snapshot ensemble* son mas robustos y precisos que las redes individuales dada su naturaleza de ensamblado, a ningun costo adicional de entrenamiento.

1.4. Metodos de Aprendizaje de Maquina Automatico

Aprendizaje de Maquina Automatico (Auto-ML) es el proceso de automatizar la solucino de problemas del mundo real a traves de tecnicas de aprendizaje de maquina. El proceso involucra eliminar la necesidad de humanos expertos en aprendizaje de maquina para seleccionar apropiadamente las caracteristicas, flujos, paradigmas, algoritmos y sus hiperparametros para resolver un problema (Dimitrakakis y col. 2019). Las prinicpales ventajas de las tecnologias de AutoML incluyen: (1) reducir el tiempo empleado en resolver problemas bien estudiados; y, (2) eliminar la necesidad de conocimiento experto. Adicionalmente, estas tecnologias tienden a generar soluciones mas simples que a menudo tienen mejor desempeño que soluciones diseñadas por humanos.

Multiples tecnologias han sido propuestas para resolver el problema de Auto-ML. Auto-Weka (Thornton y col. 2013) fue una de las primeras soluciones presentadas.

Esta está basada en el software Weka (Witten y col. 2009), un software construido a partir de varias herramientas de visualizacion y algoritmos para analisis de datos y modelacion predictiva. *Auto-Weka* resuelve el problema de *Auto-ML* como un problema CASH segun definido a continuacion.

Definición 1.1 *Sea* $A = \{A^{(1)}, \ldots, A^{(R)}\}$ *un conjunto de algoritmos, y sea* $\Lambda^{(j)}$ *el dominio de los hiperparametros del algoritmo* $A^{(j)}$*. Sea,* $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ *el conjunto de entrenamiento, el cual es dividido en* K *cross-validation folds de la forma* $\{D^{(1)}_{valid}, \ldots, D^{(K)}_{valid}\}$ *y* $\{D^{(1)}_{train}, \ldots, D^{(K)}_{train}\}$ *tal que* $D^{(i)}_{train} = D \setminus D^{(i)}_{valid}$ *para todo* $i = 1, \ldots, K$*. Finalmente, denotese* $L(A^{(j)}_{\lambda}, D^{(i)}_{train}, D^{(i)}_{valid})$ *la perdida del algoritmo* $A^{(j)}$ *en* $D^{(i)}_{valid}$ *con hiperparametros* λ*. Entonces, el problema de Seleccion de Algoritmo y Optimizacion de Hiperparametros Combinado (CASH) es encontrar la configuracion conjunta de algoritmo e hiperparametros que minimiza la perdida:*

$$A^{\star}, \lambda_{\star} \in argmin_{A^{(j)}, \lambda \in \Lambda^{(j)}} \frac{1}{K} \sum_{i=1}^{K} L(A^{(j)}_{\lambda}, D^{(i)}_{train}, D^{(i)}_{valid}) \tag{1.1}$$

Otros sistemas populares de Auto-ML son Auto-Sklearn (Feurer y col. 2015) y Auto-Keras (H. Jin y col. 2018). Estos sistemas estan basados en las bibliotecas de aprendizaje de maquina, *Scikit-Learn* (Pedregosa y col. 2011) y *Keras* (*Keras* s.f.) respectivamente. Ambos sistemas proveen una interfaz para encontrar la mejor arquitectura de aprendizaje de maquina para resolver un problema. Una diferencia fundamental entre ellos es la forma en que sus espacios de busqueda son definidos. Mientras Auto-SkLearn explora espacio de busqueda condicional, i.e., un espacio con algunos hiperparametros condicionados a otros, Auto-Keras realiza una Busqueda de Arqutectura Neuronal (NAS (Elsken y col. 2018)), la cual implica explorar espacios jerarquicos de complejidad arbitraria.

AutoGOAL (Estévez Velarde y col. 2020; Estévez-Velarde y col. 2020) es una de las mas recientes contribucinoes al campo del Auto-ML. AutoGOAL es un sistema que utiliza tecnicas heterogeneas para resolver el problema CASH. La escencia de AutoGOAL radica en su espacio personalizable de pipelines y su pool de algoritmos de busqueda, que son usados para encontrar la mejor configuracion para resolver un problema. Cada pipeline esta definido como un conjunto de algoritmos interconectados que traducen una entrada predefinida a su salida correspondiente. El espacio de pipelines comprende no solo el conjunto de algoritmos, sino tambien sus hiperparametros.

Multiples fuentes de algoritmos estan incluidos en el espacio de AutoGOAL, tales como *Scikit-Learn*(Pedregosa y col. 2011), *NLTK*(Loper y Bird 2002), *Keras*(*Keras* s.f.), y *Pytorch*(Paszke y col. 2019). Sin embargo, AutoGOAL carece de la habilidad

de combinar multiples end-to-end pipelines para generar una solucion. Esta limitacion puede ser superada con la utilizacion de ensamblados.

El proceso fundamental de optimizacion utilizado en AutoGOAL en estos momentos esta baasado en una tecnica de continua optimizacion con Evolucion Gramatical para gramaticas probabilistas libres del contexto (Mégane y col. 2021). El proceso consiste de un cilo de generacion y evaluacion, utilizando una gramatica G apropiada para describir el problema de aprendizaje de maquina qe se intenta resolver. En cada iteracion un conjunto de N pipelines es generados a partir de samplear la gramatica G de acuerdo a las probabilidades θ asignadas a cada produccion. Estas probabilidades son inicializadas con una distribucion uniforme θ_0 para todas las producciones. Cada pipeline es evaluado (lo cual consiste simplemente en entrenamiento y ejecucion), y los de mejor desempeño son utilizados para modificar θ, con el objetivo de maximizar la probabilidad de que estos sean generados. Este proceso se ilustra en el algoritmo

Algoritmo 1.1: PGE

$N \leftarrow$ tamaño de la poblacion
$n \leftarrow$ numero de individuos seleccionados en cada iteracion
$\alpha \leftarrow$ factor de aprendizaje
$G \leftarrow$ gramatica que describe los pipelines de aprendizaje de maquina a explorar
$\theta_0 \leftarrow$ probabilidades iniciales (uniforme)
$f \leftarrow$ funcion de fitness (entrenamiento y evaluacion de pipelines)
$best \leftarrow none$
para *cada iteracion* i **hacer**
    $P_i \leftarrow$ generar poblacion utilizando gramatica G, con probabilidades θ_{i-1}
    para *cada solucion* $S \in P_i$ **hacer**
        $f(S) \leftarrow$ calcular fitness de S (evaluar el pipeline)
    fin
    $best \leftarrow argmax_{S \in P \cup \{best\}}\{f(S)\}$
    $P_i^* \leftarrow$ seleccionar los n mejores individuos de P_i
    $\theta_i^* \leftarrow$ calcular la distribucion marginal de P^*
    $\theta_i \leftarrow \alpha\theta_i^* + (1-\alpha)\theta_{i-1}$
fin
devolver *best*

1.5. Optimizacion Multiobjetivo

Los metodos de optimizacion multiobjetivo son aquellos que exploran el espacio de busqueda optimizando simultaneamente diferentes funciones.

Definición 1.2 (Optimizacion Multiobjetivo) *Dadas m funciones objetivo $f_1 : X \to \mathbf{R}, \ldots, f_m : X \to \mathbf{R}$ las cuales traducen el espacion X en $\mathbf{R}$, un problema de optimizacion multiobjetivo esta dado es expresado de la seguiente forma:*

$$minimizar f_1(x), \ldots, minimizar f_m(x), x \in X \tag{1.2}$$

Al trabajar con multiples funciones objetivos es necesario encontrar formas de comparar dos soluciones en el espacio de soluciones factibles. El concepto de *Pareto dominación* juega un papel fundamental en el ambito de la optimizacion multiobjetivo, dado que permite comparar objetivamente dos vectores de forma precisa, sin requerir informacion adicional de preferencia.

Definición 1.3 (Pareto Dominación) *Dados dos vectores en el espacio objetivo, digase $y^{(1)}, y^{(2)} \in \mathbf{R}^m$, entonces el punto $y^{(1)}$ se dice que* pareto-domina *a $y^{(2)}$ si y solo si:*

$$\forall_{i \in \{1,\ldots,m\}} : y_i^{(1)} \leq y_i^{(2)} y \exists_{j \in \{1,\ldots,m\}} : y_j^{(1)} \leq y_j^{(2)} \tag{1.3}$$

Definición 1.4 (Frente pareto) *Todos aquellos vectores x del espacio objetivo tal que no exista otro vector y en el espacio objetivo que pareto-domine a x.*

Tradicionalmente los problemas de optimizacion multiobjetivo han sido atacados utilizando tecnicas de escalarizacion (e.g. (Miettinen 2012)), estas tecnicas consisten en de alguna forma combinar todas las funciones objetivos en una sola, o reescribirlas como restricciones. Varias tecnicas existen en este contexto. *Linear Weighting* es una de estas, en la cual se construye una nueva funcion objetivo a partir de la combinacion lineal de las funciones objetivo del problema original, i.e. $min \sum w_i f_i(x), x \in X$. Este enfoque tiene un problema fundamental y es que si el frente pareto no es convexo, no es posible encontrar soluciones en esta zona, no importa los pesos w_i que se utilicen. Otra forma de escalarizacion es ϵ-*constrain*, esta tecnica selecciona una funcion como la *principal* y las demas se establecen como restricciones al conjunto de soluciones factibles, exigiendo que sean menores que un ϵ.

Existen metodos numericos que intentan resolver el problema haciendo cumplir las condiciones de Karush-Kuhn-Tucker (Kuhn y Tucker 2014). La idea va de encontrar al menos una solucion del sistema de ecuaciones que se produce al tratar el problema de *KKT*. Es posible utilizar metodos de continuacion y homotopia para obtener todas las soluciones (Hillermeier y col. 2001; Schütze y col. 2005). Estos metodos require que las soluciones satisfagan condiciones de convexidad local y diferenciabilidad.

1.5.1. NSGA-II

Los algoritmos geneticos utilizaan paradigmas basados en procesos evolutivos naturales, como *seleccion natural*, *mutación* y *recombinación* para mover un poblacion (conjunto de vectores de decision) a soluciones optimas o casi optimas (Back 1996). Los algoritmos geneticos multiobjetivos generalizan esta idea y son diseñados para en cada iteracion acercarse mas al frente pareto. En este contexto destaca *NSGA-II* (Deb y col. 2002), el cual se explica en mayor detalle a continuacion.

El algoritmo consiste basicamente de un ciclo generacional que se divide en dos partes. En la primera parte, la poblacion pasa por un proceso de variacion. En la segunda parte, un proceso de seleccion toma lugar, el cual resultaen la poblacion de la nueva generacion. Este proceso se repite hasta que se cumple algun criterio de convergencia o se excede una cantidad de computo predefinida.

En la parte de la variacion, λ nuevos individuos son generados. Para cada uno de ellos dos padres son seleccionados de la poblacion actual P_t. Para escoger estos, se utiliza una seleccion de torneo binario, es decir se escogen aleatoriamente dos individuos de la poblacion y se selecciona el mejor de acuerdo a su ranking en la poblacion. Los padres son entonces recombinados utilizando un operador de combinacion, el individuo resultante es luego mutado utilizando un operador de mutacion. De esta forma es creado un nuevo conjunto Q_t de individuos, los cuales son añadidos junto a la poblacion actual al conjunto de individuos a considerar para la siguiente generacion.

La segunda fase, fase de seleccion, los μ mejores individuos son seleccionados del conjunto $P_t \cup Q_t$ utilizando un mecanismo de ranking multiobjetivo, de esta forma la poblacion de la nueva generacion P_{t+1} es formada. El mecanismo de seleccion de *NSGA-II* es el ingrediente fundamental que lo distingue del resto de los algoritmos geneticos que son utilizados para resolver problemas de optimizacion de un unico objetivo. Este consiste de dos niveles. Primero se realiza un *non-dominated sorting*. Este depende unicamente del *pareto-orden* entre los individuos. Finalmente los individuos que comparten el mismo *pareto-orden* son ordenados de acuerdo al *crowding-distance*, la cual es una medidad de la diversidad.

Non-dominated sorting

Sea $ND(P)$ el conjunto de soluciones no dominadas 1.3 en una poblacion P. *Non-dominated sorting* particiona la poblacion en subconjuntos, pasado en la *pareto dominacion* 1.3 como especifica la siguiente recursion.

$$R_1 = ND(P) \tag{1.4}$$

$$R_{k+1} = ND(P \setminus \cup_{i=1}^{k} R_i) \quad k = 1, 2, \ldots \tag{1.5}$$

Como en cada paso de la recursion al menos una solucion es eliminada de la poblacion, el numero maximo de *capas* es $|P|$. El orden de una solucion esta dada por el subindice k del R_k en el cual queda dicha solucion.

Crownding distance

Si mas de una solucion quedan en el mismo subconjunto de la poblacion R_k luego de realizar la ordenacion anterior, se procesde a ordenar las soluciones dentro de dichos subconjunto a partir de su *crowding distance*. Esta es calculada para una solucion x como la suma de las contribuciones c_i a la i-ésima funcion objetivo:

$$l_i(x) = \text{máx}\{f_i(y)|y \in R \setminus \{x\} \wedge_i (y) \leq f_i(x)\} \cup \{-\infty\} \tag{1.6}$$

$$u_i(x) = \text{mín}\{f_i(y)|y \in R \setminus \{x\} \wedge f_i(y) \geq f_i(x)\} \cup \{+\infty\} \tag{1.7}$$

$$c_i(x) = u_i - l_i, \quad i = 1, \ldots, m \tag{1.8}$$

$$c(x) = \frac{1}{m} \sum_{i=1}^{m} c_i(x), x \in R \tag{1.9}$$

Intuitivamente mientras mas *espacio* exista alrededor de una solucion, mayor sera el *crowding distance* de la misma. Por tanto, aquellas soluciones con elevado *crowding distance* son preferidas a aquellas con baja distancia, con el proposito de mantener la diversidad en la poblacion.

1.6. Discusion

Capítulo 2

Propuesta

2.1. Descripcion general

El sistema toma como entrada un dataset $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ y una funcion de perdida $\mathcal{L}$ y una o varias metricas de equidad $F_1, F_2, \ldots, F_n$. El objetivo del sistema es producir un modelo de clasificacion que es a la vez efectivo segun L y justo segun $F_1, F_2, \ldots, F_n$. El sistema consiste en dos fases fundamentales. La primera es responsable de generar una coleccionde modelos, cada uno llamado modelo base. Esta coleccion es construida optimizando una funcion de perdida $\mathcal{L}_1$, mientras se asegura *diversidad* a lo largo de toda la poblacion. La segunda fase es responsable de producir un modelo de clasificacion. Este modelo es generado ensamblando la coleccion de modelos base de forma tal que optimice su efectividad segun L_2, a la vez que es lo mas *justo* posible segun $F_1, F_2, \ldots, F_n$. En el contexto de esta tesis se asume $L = L_1 = L_2$. Las secciones 2.2 y 2.4 abordan con mas detalles la primera y segunda fase, respectivamente.

2.2. Generacion de modelos base

En esta fase al sistema se le da la tarea de generar N modelos para ajustar D de acuerdo a la perdida $\mathcal{L}$.

La Definicion 1.1 se modifica para buscar una coleccion de modelos en lugar de un solo modelo, sujeto a una metrica de diversidad $\mathcal{D}$. Esto es, se desea encontrar una coleccion de modelos base (modelos que optimicen la efectividad en el dataset D de acuerdo a la funcion de perdida $\mathcal{L}$) mientras garantiza algunas diferencias entre sus hipotesis utilizando la metrica $\mathcal{D}$. Asegurar diversidad en la coleccion de modelos base es importante porque los metodos de ensemble no son capaces de mojorar su rendimiento si todos los modelos base tienen exactamente la misma hipotesis, i.e.,

todos realizan las mismas predicciones.

El procedimiento aplicado para generar la coleccion de modelos base esta resumido por la funcion *GenerateBaseModels* GenerateBaseModels. El espacio de algoritmos y hiperparametros es explorado utilizando una estrategia de busqueda pre-seleccionada. Todo esto es capturado por la funcion **explore**. Luego de evaluar las arquitecturas generadas y estimar la diversidad entre los modelos actualmente seleccionados y la nueva generacion de modelos, la coleccion de modelos base es actualizada para ajustarse a su capacidad N. Todo esto es capturado por la funcion **reselect**.

Función GenerateBaseModels$(N, D, A, \Lambda, \mathcal{L}, \mathcal{D})$

```
set base_models ← ∅
para generation ∈ explore(A, Λ) hacer
    scores ← ∅
    para A_λ^(j) ∈ generation hacer
        scores^(j) ← (1/K) Σ_{i=1}^{K} L(A_λ^(j), D_train^(i), D_valid^(i))
    fin
    diversity ← D(base_models ⌢ generation, D)
    base_models ← reselect(base_models ⌢ generation, scores, diversity, N)
fin
devolver base_models
```

Para explorar inteligentemente el espacio de algoritmos y hiperparametros, i.e., para resolver el problema *CASH* modificado, se utiliza la implementacion de *Probabilistic Grammatical Evolution Search* 1.1 presente en AutoGOAL. AutoGOAL se refiere a los modelos que construye como pipelines, dado que cada uno de ellos esta formado por algoritmos interconectados. La busqueda comienza con una estrategia de muestreo aleatorio, pero segun evalua mas pipelines, modifica el modelo de muestreo probabilista para que pipelines similares a los mejores encontrados hasta el momento, sean generados con mayor frecuencia. El espacio de algoritmos y hiperparametros utilizados es el utilizado por defecto en AutoGOAL, el cual incluye varios algoritmos clasicos de aprendizaje de maquina presentes en las diferentes bibliotecas utilizadas por AutoGOAL.

Para reseleccionar la coleccion de modelos base, i.e. la coleccion de pipelines de AutoGOAL, un enfoque greedy es utilizado y estudiado. La funcion **reselect** reselect resume la estrategia propuesta. El algoritmo siempre incluye el modelo que mejor se desempeña de acuerdo a L en la seleccion. Cada iteracion siguiente añade el modelo no todavia seleccionado, que maximiza la diversidad respecto a todos los modelos anteriormente seleccionados. El enfoque greedy no garantiza que la coleccion final logre la mejor posible diversidad respecto a $\mathcal{D}$. La precision tampoco es tomada en

cuenta, excepto para seleccionar el modelo de mejor desempeño.

Función reselect(M, $scores$, $diversity$, N)

set $R \leftarrow \emptyset$
set $R^{(0)} \leftarrow \underset{m^{(j)} \in M}{argmin}\ scores^{(j)}$
para $r \leftarrow 1$ **a** N **hacer**
  $R^{(r)} \leftarrow \underset{(m^{(j)} \in M \setminus R)}{argmax} \sum_{(m^{(i)} \in R)} diversity^{(i,j)}$
fin
devolver $R^{(0)} \cdots R^{(N)}$

Tres metodos *reselect* de referencia fueron implementaods para realizar comparaciones entre las metricas de diversidad *disagreement* y *double-fault*, las cuales son presentadas en la seccion 2.3.

- Shuffle: La coleccion de modelso base es construida barajeando aleatoriamente la selecciona ctual de modelos base y los nuevos encontrados. Los primeros N modelos luego de barajear son seleccionados para la siguiente generacion.
- Arbitrary: La coleccion de modelos base es construida de la misma forma que con la estrategia *shuffle*, pero el modelo de mejor rendimiento siempre es incluido en la coleccion de modelos base seleccionados.
- Best: La coleccion de modelos base es construida a partir de seleccionar los modelos de mejor rendimiento entre los previamente seleccionados y los recien encontrados.

A continuacion, la seccion 2.3 provee algunos detalles acerca de las metricas de diversidad estudiadas en este trabajo.

2.3. Metricas de diversidad

Dos metricas fueron implementadas para estimar la diversidad de una coleccion dada de modelos base. Ambas de ellas precomputan una matriz de clasificaciones incorrectas, la cual es usada entonces para computar una metrica que aporta informacion sobre la diversidad entre los modelos base dos a dos. La matriz de clasificaciones incorrectas se construye de la siguiente manera.

$$M_{i,j} = \begin{cases} 1 & \text{si el modelo j correctamente clasifica el ejemplo i } (D_valid) \\ -1 & \text{otherwise} \end{cases} \tag{2.1}$$

Las siguientes metricas son computadas entre pares de modelos base para estimar cuand diferentes son sus hipotesis, y por tanto la diversidad de la collecion incluyendo a ambos a la vez.

Disagreement. Esta mide la frecuencia con la cual uno de los mdelos falla cuando el otro no lo hace, y viceversa. Mientras mas alto el valor de la metrica, mas diferentes son los modelos.

$$disagreement(m^{(a)}, m^{(b)}) = \frac{|\{M_{i,a} \neq Mi, b | s^{(i)} \in D^{(\star)}_{valid}\}|}{|D^{(\star)}_{valid}|} \tag{2.2}$$

Double Fault. Esta mide cuan a menudo ambos modelos fallan a la vez. Mientras mas alta esta medida mayor la diferencia entre ambos.

$$double - fault(m^{(a)}, m^{(b)}) = 1 - \frac{|\{M_{i,a} = Mi, b = -1 | s^{(i)} \in D^{(\star)}_{valid}\}|}{|D^{(\star)}_{valid}|} \tag{2.3}$$

2.4. Ensamblado inteligente de modelos justos

En esta fase el sistema tiene la tarea de combinar las predicciones de N modelos para ajustar D de acuerdo tanto a la perdida $\mathcal{L}$ como a las metricas de equidad $F_1, \ldots, F_n$.

El sistema una vez mas resuelve un problema de *CASH* como se presenta en 1.1. En lugar de trabajar directamente en $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, esta vez el sistema trabaja sobre $D^e = \{(y_1^{(*)}, y_1), \ldots, (y_n^{(*)}, y_n)\}$, donde $y_i^{(*)} = [y_i^{(0)}, \ldots, y_i^{(n)}]$ y cada $y_i^{(j)}$ es la salida de un modelo base j para el ejemplo i, $i \in [1 \ldots n]$, $j \in [1 \ldots N]$. En otras palabras, al sistema se le pide encontrar la mejor combinacion E^* de algoritmos y sus hiperparametros para ensamblar las salidas de los modelos base. Particularmente, un espacio de algoritmos especializados en estrategias de ensamblados y sus hiperparametros es utilizado en esta fase. Estas estrategias son presentadas a continuacion.

- *Voting Classifiers*. Asigna la etiqueta mas comun entre las predichas por los modelos base. En caso de empate, se selecciona la etiqueta producida por el modelo mas preciso entre los modelos base.

- *Overfitted Voting Classifiers*. Asigna a cada combinacion de salida de los modelos base la etiqueta que asegura el mejor desempeño en D^e_{train}. En el momento de prediccion, si una combinacion no antes vista es encontrada, este selecciona la etiqueta predicha por el modelo base mas preciso (ignorando si esta fue la etiqueta mas votada).

- *ML Voting Classifiers*. Ajusta un modelo de aprendizaje de maquina sobre D^e_{train} para optimizar $\mathcal{L}$. La arquitectura del modelo de aprendizaje de maquina es tomado del pool de algoritmos por defecto de AutoGOAL.

Para explorar el espacio de algoritmos e hiperparametros se realiza una modificacion a *Probabilistic Grammatical Evolution*(1.1), en la cual la fase de seleccionar los n individuos de la poblacion actual que pasan a la siguiente generacion, se realiza utilizando *Non-dominated sorting* 1.5.1 y *Crowding distance* 1.5.1, de la misma forma en que se utiliza en *NSGA-II*. De esta forma la funcion de seleccion, y por tanto la exploracion del espacio, optimizan simultaneamente las metricas de equidad y la funcion de peridida.

Capítulo 3

Análisis Experimental

En este capitulo se evalúa la capacidad de nuestro sistema de optimización de dos fases de resolver un problema de clasificación y lograr buenos resultados en métricas de precisión y equidad a partir de ensamblar modelos base. La experimentación realizada consiste de dos etapas, en una se analiza la capacidad de la primera fase del sistema de obtener un conjunto de modelos base lo suficientemente diverso como para que ensamblar estos resultados en un modelo de mayor precisión que los modelos base. En la segunda etapa, se estudia si el algoritmo propuesto permite encontrar formas de ensamblar los modelos base resultantes de la primera etapa de forma tal que se obtengan valores satisfactorios tanto de precisión como en las métricas de equidad.

3.1. Marco Experimental

Una primera etapa experimental se lleva a cabo con el objetivo de estudiar la capacidad del sistema de producir un conjunto de modelos base capaz de, al ser ensamblado, generalizar y obtener mejores resultados que dichos modelos base. Adicionalmente se desea estudiar que influencia tienen las distintas métricas de diversidad utilizadas en esta capacidad del sistema.

Para estimar el mejor rendimiento obtenible a partir de ensamblar el conjunto de modelos base encontrado, dos medidas son propuestas a continuación. Estas medidas estiman el rendimiento que logran modelos de ensemble de prueba, esto es, modelos de ensemble que conocen los resultados correctos a priori, y por tanto son modelos que no tienen utilidad real fuera de ser utilizados con propósitos de comparación. Dichos modelos de ensemble son llamados *Oráculo Optimista* y *Oráculo Sobre-ajustado*.

Oráculo Optimista. Este modelo oráculo devuelve la etiqueta correcta si al menos uno delos modelos base predijo dicha etiqueta. La única forma de que este

modelo falle es si ninguno de los modelos base fue capaz de sugerir la etiqueta correcta.

$$O^{(i)}_{optimista}(M) = \begin{cases} y_i & \text{si } y_i \in \{\, y_i^{(j)} \mid m^{(j)} \in M \,\} \\ y_i^{(0)} & \text{en otro caso} \end{cases}$$

Los resultados alcanzados por este ensemble reflejan el grado de cobertura de los modelos base sobre la colección de evaluación, esto es, cuantos ejemplos son correctamente predichos por al menos uno de los clasificadores base.

Oráculo Sobre-ajustado. Este modelo oráculo computa todas las combinaciones de salidas de los modelos base y asigna a cada combinación la etiqueta mas frecuente encontrada en el conjunto correspondiente de etiquetas correctas. Este modelo falla cuando la misma combinación de modelos base tiene que producir diferentes etiquetas para que todas las posibles entradas sean clasificadas correctamente.

$$O^{(i)}_{sobreajustado}(M) = \textbf{max\_count}\left(\left\{ y_k \,\middle|\, (x_k, y_k) \in D, \underset{m^{(j)} \in M}{\forall} y_k^{(j)} = y_i^{(j)} \right\}\right)$$

La función **max_count** devuelve el elemento mas frecuente de la colección (en caso de un empate, siempre devuelve el primero que encuentra).

Los resultados que obtiene este ensemble, proveen una cota superior del mejor rendimiento que puede ser obtenido utilizando un conjunto de reglas **consistente** para ensamblar la colección de modelos base.

Primeramente se desea validar que nuestro sistema es capaz de generar modelos base en la primera fase que al ser ensamblados en la segunda fase dan lugar a un ensemble que obtiene mejores resultados sobre el conjunto de evaluación que cualquiera de los modelos base de los que se compone. Para ello, los resultados obtenidos por el ensemble producido por nuestro sistema son comparados con los obtenidos por el mejor de los modelos base, y analizados respecto a los resultados obtenidos por el *Oráculo Sobre-ajustado* descrito anteriormente.

Además en esta etapa experimental se desea también comprobar la influencia de las diferentes métricas de diversidad utilizadas en la capacidad del sistema para producir modelos base que generalicen luego de ser ensamblados. Con este propósito, se realiza un análisis de que tan bien los modelos base cubren el espacio de entrada de nuestro problema. Para este análisis resulta de suma utilidad el concepto de *Oráculo*

Optimista, el cual nos darán información acerca de la influencia de las distintas métricas de diversidad en la distribución de los distintos modelos base sobre el espacio de entrada de nuestros datos. Los resultados obtenidos por nuestro sistema son entonces comparados con aquellos del *Oráculo Optimista* para cada una de las métricas de diversidad utilizadas.

Finalmente, se analiza también en esta etapa el comportamiento del sistema bajo diferentes combinaciones de otros hiper-parámetros como la cantidad de modelos base. La mejor configuración de dichos hiper-parámetros, incluyendo la métrica de diversidad que mejores resultados proporciona al sistema, son utilizados en la segunda etapa experimental.

La segunda etapa experimental consiste del estudio de la capacidad del sistema para lograr producir modelos de ensemble que son precisos de acuerdo a una función de perdida determinada y justos según una o varias métricas de equidad. Con este fin se realiza una comparación entre los resultados obtenidos por nuestro sistema y los obtenidos por varios otros sistemas que resultan relevantes en la literatura para la solución de este problema.

3.1.1. Escenarios de Evaluación

La primera etapa experimental evalúa el sistema en la tarea *HAHA 2019*(*Humor Analysis based on Human Annotation*), con marco en el evento *IberLEF 2019* (Chiruzzo y col. 2019). El proceso de evaluación consiste primero de la ejecución de la primera fase de nuestro sistema, esto es, la exploración del espacio de modelos y obtención del conjunto de modelos base a ser ensamblados. Posteriormente la segunda fase de nuestro sistema es ejecutada, utilizando como única función objetivo la función de perdida que fue utilizada en la obtención de los modelos base. La función objetivo a optimizar en ambas fases del sistema es la precisión del sistema.

Finalmente la segunda etapa experimental consiste de la ejecución de extremo a extremo de nuestro sistema. En esta etapa se utiliza la mejor configuración de hiper-parámetros acorde a los resultados de la primera fase. Ambas fases optimizan la precisión del modelo, en particular la segunda fase incorpora al proceso de optimización métricas de equidad, dígase *Statistical Parity* y *Equalized Opportunity*, como funciones objetivo adicionales a optimizar simultáneamente con la precisión.

3.1.2. Corpus de Evaluación

Dos conjuntos de datos son utilizados para la evaluación del sistema en los experimentos realizados.

HAHA 2019 Colección de datos utilizada en la tarea *HAHA 2019* (*Humor Analysis based on Human Annotation*), con marco en *IberLEF 2019* (Chiruzzo y col.

2019). El corpus contiene 30 000 tweets en Español clasificados manualmente, de los cuales 24 000 son para entrenamiento y 6 000 para evaluación. Cada uno de estos tweets es clasificado en *humoroso* o *no-humoroso*.

La colección de datos es anotada a partir de asignar una clasificación de *gracioso* o *no-gracioso* a cada tweet, en caso de ser *gracioso* se da una puntuación de $[1, 5]$ de cuan *gracioso* es dicho tweet.

Tabla 3.1: Composición de los datos según la cantidad de votos para cada clase.

	Entrenamiento	Evaluación	Total
Tweets	24 000	6 000	30 000
Graciosos	9 253	2 342	11 595
No graciosos	14 757	3 658	18 405
Puntuación promedio	2.04	2.03	2.04
Total de Votos	59 440	13 605	73 045
Votos 1	19 063	4 818	23 881
Votos 2	14 713	3 777	18 490
Votos 3	10 206	2 649	12 855
Votos 4	4 493	1 122	5 615
Votos 5	1 305	275	1 580

Adult La colección de datos *Adult* (Dua y Graff 2017) presenta información extraída del censo de 1994 en los Estados Unidos por Barry Becker. Los datos contienen detalles personales de los individues, tales como nivel de educación, horas de trabajo a la semana, raza, sexo, etc., y el objetivo es predecir si el individuo ganará un salario mayor a \$50K al año. Hay un total de 48 842 filas de datos, y de estas, 3 620 contienen casillas con valores desconocidos, dejando 45 222 filas completas. Existen dos clases en las cuales clasificar a los individuos dependiendo de su salario anual, estas son, '$< 50K$' o '$\leq 50K$'. Las clases están desbalanceadas, con una tendencia hacia la etiqueta '$\leq 50K$', la cual representa aproximadamente el 75 % de los ejemplos.

3.1.3. Configuración Experimental

En todos los experimentos, el sistema fue configurado para permitir que cada fase ejecutara a lo sumo 10 000 iteraciones o por una hora. Los parámetros de búsqueda de AutoGOAL fueron los siguientes:

- `popsize=50`

- `selection=10`
- `cross_validation_steps=3`
- `validation_split=0.3`

Debido a limitaciones de infraestructura, los algoritmos de aprendizaje profundo fueron excluidos del conjunto de algoritmos disponibles para AutoGOAL, esto significa que flujos basados en *Keras* y *BERTA fueron excluidos y fundamentalmente* flujos basados en *ScikitLearn* fueron utilizados. AutoGOAL es una herramienta aun en desarrollo y la ultima versión estable en el momento de realizar los experimentos (`v0.6.0`) no soporta completamente algoritmos de aprendizaje profundo en todas las arquitecturas.

No incluir los algoritmos de aprendizaje profundo en la configuración experimental puede tener un impacto negativo en el mejor rendimiento que puede ser alcanzado por el sistema. Es decir, el rendimiento del sistema mientras resuelve el problema puede no ser directamente comparado con otras soluciones reportadas que utilizan técnicas de *Aprendizaje Profundo*. Sin embargo, la ausencia de estos algoritmos no deberían afectar la capacidad del sistema de mejorar el rendimiento respecto a los modelos base encontrados en la primera fase. Por tanto, si el sistema es capaz de mejorar el rendimiento de los modelos base, entonces el rendimiento del sistema se espera que mejore un mas una vez que las arquitecturas de aprendizaje profundo sean compatibles. Esto se espera que ocurra no solo porque los modelos base ahora tendrían mejor rendimiento, sino también porque arquitecturas de ensemble mas poderosas serían añadidas al mismo tiempo sin esfuerzo alguno. AutoGOAL ya ha probado lograr resultados competitivos cuando es configurado correctamente (Estevez-Velarde y col. 2020). Además, es importante destacar que, en el caso en que se optimiza buscando un buen balance entre la precisión y las métricas de equidad, por ejemplo en la segunda fase de nuestro sistema, no necesariamente modelos mas poderosos (como los de aprendizaje profundo) implican mejores resultados en dicho balance.

Biblioteca

El sistema propuesto en este trabajo es parte de la biblioteca en desarrollo `BFair`[1] La biblioteca tiene como objetivo atacar los problemas de sesgos que emergen de entrenar modelos de *Aprendizaje Automático* que en datos que muestran sesgos de los humanos.

[1] https://github.com/bfair-ml/bfair

Hardware

Los experimentos fueron ejecutados en un equipo con las siguientes propiedades: CPU Intel Core i9-9900K (-MT-MCP-) con velocidad máxima de 3651/5000 MHz, cache de 16384KB y RAM de 64GB.

3.2. Primera Etapa Experimental

A continuación, la sección 3.2.1 muestra los resultados de los obtenidos a partir de realizar los experimentos de la forma descrita en la sección 3.1 para la tarea *HAHA 2019*. Luego la sección 3.2.2 realiza un análisis en profundidad de dichos resultados y arriba a conclusiones a partir de los mismos.

A continuación se muestran los resultados de la primera y segunda fase experimental respectivamente, según descritas anteriormente.

3.2.1. Resultados

Las tablas 3.2, 3.3, 3.4, 3.5 resumen los resultados obtenidos por el sistema, en la tarea *HAHA 2019* correspondiente a la primera fase experimental, para configuraciones con número máximo de clasificadores base siendo 5, 20, 50 y 100 respectivamente. La métrica de puntuación utilizada es F_1 como se sugiere en la descripción de la tarea. Cinco estrategias de control de población fueron evaluadas con el objetivo de establecer comparaciones, de las cuales dos son las métricas de diversidad discutidas en la sección 2.3, y las otras tres son las implementaciones de referencia del método reselect presentados en la sección 2.2. Cada tabla muestra la métrica F_1 alcanzada por: (i) los oráculos optimista y sobreajustados (sección 3.1); (ii) el modelo obtenido a partir de ensamblar los modelos base de acuerdo a la función de perdida con la que estos fueron entrenados (sección 3.1); y, (iii) el mejor modelo base encontrado (sección 2.2). Además, el tipo de algoritmo de ensemble utilizado por el mejor de los modelos de ensemble encontrados es adicionado al final de cada tabla.

Como puede observarse, **los mejores resultados fueron obtenidos cuando el máximo número de clasificadores base esta entre 20 y 50 y la estrategia de selección de modelos base es *double-fault*.** A continuación, la sección 3.2.2 profundiza en los resultados presentados en esta sección 3.2.1.

3.2.2. Discusión

Algunos patrones interesantes pueden ser observados a partir de analizar el rendimiento de los ensembles oráculo para diferentes estrategias de reselección de clasificadores base. La tabla 3.6 resume el rendimiento de los oráculos optimista y sobre-

Tabla 3.2: HAHA 2019. Máximo número de modelos base es 5. Cada columna muestra el resultado obtenido utilizando la estrategia de selección de modelos base correspondiente. A^* y E^* representan el modelo base de mejor rendimiento y la mejor configuración de ensemble encontrada, respectivamente. Tipos de ensemble: *voting*, *overfit*, y *learning*, representan a *Voting Classifier*, *Overfitted Voting Classifier*, y *ML Voting Classifier*, respectivamente.

	shuffle	arbitrary	best	disagreement	double-fault
oráculo optimista (F_1)	0.996	0.917	0.893	1.000	0.970
oráculo sobreajustado (F_1)	0.715	0.711	0.744	0.748	0.754
A^* (F_1, entrenamiento)	0.989	0.973	0.945	0.846	0.876
A^* (F_1, evaluación)	0.690	0.711	0.722	0.749	0.757
E^* (F_1, entrenamiento)	0.913	0.961	0.917	0.846	0.941
E^* (F_1, evaluación)	0.731	0.726	0.755	0.749	0.761
tipo de ensemble	voting	learning	learning	voting	voting

Tabla 3.3: HAHA 2019. Máximo número de modelos base es 20. Cada columna muestra el resultado obtenido utilizando la estrategia de selección de modelos base correspondiente. A^* y E^* representan el modelo base de mejor rendimiento y la mejor configuración de ensemble encontrada, respectivamente. Tipos de ensemble: *voting*, *overfit*, y *learning*, representan a *Voting Classifier*, *Overfitted Voting Classifier*, y *ML Voting Classifier*, respectivamente.

	shuffle	arbitrary	best	disagreement	double fault
oráculo optimista (F_1)	1.000	1.000	0.936	1.000	0.998
oráculo sobreajustado (F_1)	0.729	0.771	0.831	0.735	0.889
A^* (F_1, entrenamiento)	0.876	0.901	0.855	0.875	0.856
A^* (F_1, evaluación)	0.705	0.721	0.759	0.732	0.755
E^* (F_1, entrenamiento)	0.870	0.930	0.883	0.875	0.942
E^* (F_1, evaluación)	0.719	0.740	0.765	0.732	0.767
tipo de ensemble	learning	overfit	voting	learning	voting

ajustado en el conjunto de evaluación. Algunos de estos patrones son presentados a continuación:

- La métrica *disagreement* asegura la máxima cobertura del conjunto de entrenamiento sin importar el número de clasificadores base seleccionado. Esto tiene sentido dado que la medida de *disagreement* premia la reselección de modelos

Tabla 3.4: HAHA 2019. Máximo número de modelos base es 50. Cada columna muestra el resultado obtenido utilizando la estrategia de selección de modelos base correspondiente. A^* y E^* representan el modelo base de mejor rendimiento y la mejor configuración de ensemble encontrada, respectivamente. Tipos de ensemble: *voting*, *overfit*, y *learning*, representan a *Voting Classifier*, *Overfitted Voting Classifier*, y *ML Voting Classifier*, respectivamente.

	shuffle	arbitrary	best	disagreement	double fault
oráculo optimista (F_1)	1.000	1.000	0.978	1.000	1.000
oráculo sobreajustado (F_1)	0.953	0.950	0.927	0.996	0.969
A^* (F_1, entrenamiento)	1.000	0.928	0.877	0.857	0.913
A^* (F_1, evaluación)	0.639	0.720	0.720	0.750	0.749
E^* (F_1, entrenamiento)	1.000	0.924	0.921	1.000	0.923
E^* (F_1, evaluación)	0.741	0.736	0.731	0.747	0.756
tipo de ensemble	overfit	learning	voting	overfit	voting

Tabla 3.5: HAHA 2019. Máximo número de modelos base es 100. Cada columna muestra el resultado obtenido utilizando la estrategia de selección de modelos base correspondiente. A^* y E^* representan el modelo base de mejor rendimiento y la mejor configuración de ensemble encontrada, respectivamente. Tipos de ensemble: *voting*, *overfit*, y *learning*, representan a *Voting Classifier*, *Overfitted Voting Classifier*, y *ML Voting Classifier*, respectivamente.

	shuffle	arbitrary	best	disagreement	double fault
oráculo optimista (F_1)	1.000	1.000	0.997	1.000	1.000
oráculo sobreajustado (F_1)	0.988	0.992	0.984	0.998	0.988
A^* (F_1, entrenamiento)	0.998	0.856	0.853	0.902	0.920
A^* (F_1, evaluación)	0.683	0.748	0.759	0.745	0.750
E^* (F_1, entrenamiento)	1.000	1.000	1.000	0.970	0.940
E^* (F_1, evaluación)	0.756	0.745	0.761	0.728	0.743
tipo de ensemble	overfit	overfit	overfit	learning	voting

que tienen predicciones con conflictos entre si. Observando el rendimiento del oráculo sobreajustado, se puede notar que a pesar de que provee un cubrimiento máximo, no hay un conjunto de reglas **consistente** que pueda ser aplicado para explotar dicho cubrimiento.

- La métrica *double-fault* provee el rendimiento mas consistente para ambos el

oráculo optimista y sobreajustado, en especial cuando el numero de clasificadores es bajo. Esto sugiere que la estrategia de diversificación basada en *double-fault* puede proveer el mejor rendimiento en el subsecuente fase de ensamblado, dado que obtiene una mayor puntuación cuando se utiliza un conjunto de reglas consistente.

- El cubrimiento mostrado por los ensembles oráculo se incrementa significativamente según se incrementa el número de clasificadores base, independientemente de la estrategia de reselección. Esto tiene sentido dado que un mayor conjunto de votantes incrementa la probabilidad de encontrar uno que correctamente clasifique los ejemplos de mayor conflicto si hay cierta diversidad entre los votantes. Esta idea es apoyada por el hecho de que la estrategia que reselecciona de forma golosa solo los modelos de mejor rendimiento es la que logra el menor cubrimiento (de acuerdo al ensemble optimista).

Tabla 3.6: Resumen de las métricas en los oráculos para cada estrategia de selección de modelos base según el numero máximo de clasificadores base se incrementa.

Oráculo Optimista (F_1)					
n-clasificadores	shuffle	arbitrary	best	disagreement	double fault
5	0.996	0.917	0.893	1.000	0.970
20	1.000	1.000	0.936	1.000	0.998
50	1.000	1.000	0.978	1.000	1.000
100	1.000	1.000	0.997	1.000	1.000
Oráculo Sobreajustado (F_1)					
n-clasificadores	shuffle	arbitrary	best	disagreement	double fault
5	0.715	0.711	0.744	0.748	0.754
20	0.729	0.771	0.831	0.735	0.889
50	0.953	0.950	0.927	0.996	0.969
100	0.988	0.992	0.984	0.998	0.988

La tabla 3.7 resume las diferencias entre el rendimiento logrado por el mejor ensemble encontrado E^* y el mejor modelo base A^*, en ambas la colección de entrenamiento y de evaluación. Es notable que los modelos de ensemble son capaces de sobrepasar a sus modelos base en la colección de entrenamiento, incluso cuando algunas estrategias requieren un mayor numero de modelos base para lograr esto. La estrategia de *double-fault* es capaz de mejorar el rendimiento incluso con un numero

bajo de clasificadores base. Esto tiene sentido dado que esta estrategia penaliza al sistema por construir una colección en la cual todos los modelos base fallen en los mismos ejemplos. Por tanto, estos resulta en una colección de modelos base, en la cual cada modelo arregla los fallos del otro. Este comportamiento es diferente al que muestra la estrategia de *disagreement*, esta premia a los modelos base por tener diferentes predicciones independientemente de si estas eran correctas o incorrectas. Esto puede llevar a situaciones en las cuales es difícil para el ensemble decidir en cual de los modelos base confiar, especialmente si el numero de modelos base es muy bajo.

Aun mas importante, la tabla 3.7 también muestra que la optimización de ensemble es capaz de producir ensembles que generalizan mejor que sus modelos base, es decir, los ensembles tienen mejor rendimiento en el conjunto de evaluación. Esto es particularmente cierto cuando el numero de clasificadores base no es muy grande. Según incrementa el numero de clasificadores base, también lo hace el numero de diferentes combinaciones, y por tanto, el ensemble es mas susceptible a sobre-ajustar el conjunto de entrenamiento.

Tabla 3.7: Resumen de la diferencia en rendimiento entre el mejor modelo base encontrado (A^*) y el ensemble correspondiente (E^*), para cada estrategia de selección de modelo base según incrementa el numero de clasificadores base.

$(E^* - A^*)$ en entrenamiento					
n-clasificadores	shuffle	arbitrary	best	disagreement	double fault
5	-0.076	-0.012	-0.027	0.000	0.065
20	-0.006	0.030	0.029	0.000	0.086
50	0.000	-0.004	0.044	0.143	0.010
100	0.002	0.144	0.147	0.069	0.020
$(E^* - A^*)$ en evaluación					
n-clasificadores	shuffle	arbitrary	best	disagreement	double fault
5	0.040	0.015	0.033	0.000	0.004
20	0.014	0.018	0.005	0.000	0.012
50	0.102	0.015	0.012	-0.003	0.006
100	0.073	-0.003	0.002	-0.017	-0.007

Algunas otras ideas que resaltan de la tabla 3.7 se resumen a continuación:

- Las métricas de diversidad, dígase *disagreement* y *double-fault*, tienen un mayor impacto en el rendimiento del ensemble en el conjunto de entrenamiento cuando el numero de clasificadores base es bajo. Esto tiene sentido dado que mientras

menor cantidad de modelos en la colección de modelos base, mas difícil será encontrar una *buena* selección de forma arbitraria. Sin embargo, las medidas de diversidad no parecen tener un impacto en las capacidades de generalización del ensemble.

- A la estrategia de *disagreement* le resulta difícil mejorar el rendimiento, tanto en el conjunto de entrenamiento como en el de evaluación. Esto es particularmente cierto cuando el numero de clasificadores base es bajo. Esta estrategia no penaliza a los modelos por realizar predicciones erróneas y en su lugar los premia si sus predicciones son diferentes a la del resto de los clasificadores. Luego, es usual que el ensemble simplemente decida confiar en la predicción de solo uno de los clasificadores base, y por tanto el ensemble produce los mismos resultados que su mejor modelo base. Esto muestra que lograr el mejor cubrimiento de acuerdo al oráculo optimista, como lo logra la medida de *disagreement*, no es necesariamente útil.

La tabla 3.8 resume el rendimiento obtenido por el ensemble E^* según el numero de clasificadores base y las estrategias de reselección cambian. La estrategia de *double-fault* siempre logra los mejores resultados, excepto en el ultimo escenario, en el cual el numero de clasificadores es el más alto. En ese caso, aunque el ensemble obtuvo un mejor rendimiento que sus modelos base en la colección de entrenamiento, el numero alto de votantes puede haber afectado sus capacidades de generalización. dado que el espacio de posibles combinaciones de votantes es mayor.

Como es de esperarse, la ausencia de algoritmos de aprendizaje profundo en el conjunto de métodos disponibles tuvo un impacto negativo en el mayor rendimiento alcanzado. La arquitectura obtenida con mayor rendimiento logra alcanzar 0,767 F_1, mientras que la mejor solución reportada por AutoGOAL en la tarea *HAHA 2019* es 0,789 (Estevez-Velarde y col. 2020), la cual utiliza algoritmos de aprendizaje profundo - una estrategia de pre-procesamiento utilizando BERT (Devlin y col. 2018) y una red neuronal con 2 nodos recurrentes (una capa *BiLSTM* y una *LSTM*) seguidas de dos capas densas distribuidas en tiempo -. En comparación, la mejor arquitectura obtenida en estos experimentos consiste de un *Voting Classifier* que ensambla un conjunto de simples modelos de *ScikitLearn* - una estrategia de pre-procesamiento utilizando tokenización (*BlanklineTokenizer*, *TreebankWordTokenizer*, etc.) y algoritmos de vectorizacion (*TfIdfVectorizer*, *CountVectorizer*, etc.) seguidos de una capa de clasificacion (*Perceptron*, *LinearSVC*, *NearestCentroid*, *MultinomialNB*, etc.) -. A pesar de utilizar una arquitectura mucho más simple, resultados competitivos son alcanzados.

Adicionalmente, como se muestra en la tabla 3.7, la optimización utilizando ensembles es capaz de producir ensembles que generalizan mejor que sus modelos base.

Tabla 3.8: Resumen del rendimiento obtenido por el ensemble (E^*) en el conjunto de entrenamiento y evaluación, para cada estrategia de selección de modelos base según el numero maximo de clasificadores incrementa.

E^* en entrenamiento					
n-clasificadores	shuffle	arbitrary	best	disagreement	double fault
5	0.913	**0.961**	0.917	0.846	0.941
20	0.870	0.930	0.883	0.875	**0.942**
50	**1.000**	0.924	0.921	**1.000**	0.923
100	**1.000**	**1.000**	**1.000**	0.970	0.940
E^* @ evaluación					
n-clasificadores	shuffle	arbitrary	best	disagreement	double fault
5	0.731	0.726	0.755	0.749	**0.761**
20	0.719	0.740	0.765	0.732	**0.767**
50	0.741	0.736	0.731	0.747	**0.756**
100	0.756	0.745	**0.761**	0.728	0.743

Luego, se espera que el rendimiento de los modelos producidos por nuestro sistema mejore una vez que algoritmos más poderosos estén disponibles.

Las tablas **??** y 3.10 resumen los diferentes tipos de ensemble que fueron encontrados como optimos en cada experimento. La técnica de ensemble "ML Voting Classifier" no parece ser utilizada por ninguna estrategia de reselección de forma consistente. La técnica de ensemble "Voting Classifier" es utilizada consistentemente en las estrategias basadas en *double-fault*, y también parece ser utilizada frecuentemente cuando el numero de clasificadores base es bajo. Además, la técnica de ensemble "Overfitted Voting Classifier" parece ser utilizada mas cuando el numero de clasificadores es bajo.

Para concluir, se muestra que el sistema brinda los resultados mas prometedores cuando es configurado para realizar la búsqueda sobre una colección de 20 o 50 modelos base y utilizando la estrategia de selección de modelos base *double-fault*.

3.3. Segunda Etapa Experimental

3.3.1. Resultados

3.3.2. Discusión

Tabla 3.9: Resume la estrategia de ensemble utilizada por la mejor configuracion de ensemble encontrada (E^*) para cada estrategia de seleccion de modelos base segun el numero maximo de clasificadores base incrementa. Tipos de ensemble: *voting*, *overfit*, y *learning*, representan *"Voting Classifier"*, *"Overfitted Voting Classifier"*, y *"ML Voting Classifier"*, respectivamente.

n-clasificadores	shuffle	arbitrary	best	disagreement	double fault
5	voting	learning	learning	voting	voting
20	learning	overfit	voting	learning	voting
50	overfit	learning	voting	overfit	voting
100	overfit	overfit	overfit	learning	voting

Tabla 3.10: Frecuencia de uso de cada tecnica de ensemble. Las columnas *voting*, *overfit*, y *learning*, representan *"Voting Classifier"*, *"Overfitted Voting Classifier"*, y *"ML Voting Classifier"*, respectivamente.

n-clasificadores	voting	overfitted	learning
5	3	0	2
20	2	1	2
50	2	2	1
100	1	3	1

Conclusiones

Conclusiones

Recomendaciones

Recomendaciones

Bibliografía

Agarwal, A., Beygelzimer, A., Dudík, M., Langford, J. & Wallach, H. (2018). A reductions approach to fair classification. *International Conference on Machine Learning*, 60-69 (vid. pág. 4).

Agarwal, A., Dudík, M. & Wu, Z. S. (2019). Fair regression: Quantitative definitions and reduction-based algorithms. *International Conference on Machine Learning*, 120-129 (vid. pág. 4).

Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press. (Vid. pág. 10).

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123-140 (vid. pág. 5).

Calmon, F., Wei, D., Vinzamuri, B., Natesan Ramamurthy, K. & Varshney, K. R. (2017). Optimized Pre-Processing for Discrimination Prevention. En I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/9a49a25d845a483fae4be7e341368e36-Paper.pdf. (Vid. pág. 4)

Chiappa, S. & Isaac, W. S. (2018). A causal Bayesian networks viewpoint on fairness. *IFIP International Summer School on Privacy and Identity Management*, 3-20 (vid. pág. 5).

Chiruzzo, L., Castro, S., Etcheverry, M., Garat, D., Prada, J. J. & Rosá, A. (2019). Overview of HAHA at IberLEF 2019: Humor analysis based on human annotation. *IberLEF@ SEPLN*, 132-144 (vid. pág. 19).

Chowdhury, N. K., Kabir, M. A., Rahman, M. M. & Islam, S. M. S. (2022). Machine learning for detecting COVID-19 from cough sounds: An ensemble-based MCDM method. *Computers in Biology and Medicine*, *145*, 105405. https://doi.org/https://doi.org/10.1016/j.compbiomed.2022.105405 (vid. pág. 6)

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182-197. https://doi.org/10.1109/4235.996017 (vid. pág. 10)

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (vid. pág. 27).

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. *Multiple Classifier Systems*, 1-15 (vid. pág. 5).

Dimitrakakis, C., Liu, Y., Parkes, D. C. & Radanovic, G. (2019). Bayesian Fairness. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 509-516. https://doi.org/10.1609/aaai.v33i01.3301509 (vid. págs. 5, 6)

Donini, M., Oneto, L., Ben-David, S., Shawe-Taylor, J. S. & Pontil, M. (2018). Empirical risk minimization under fairness constraints. *Advances in Neural Information Processing Systems*, *31* (vid. pág. 4).

Dua, D. & Graff, C. (2017). UCI Machine Learning Repository. http://archive.ics.uci.edu/ml. (Vid. pág. 20)

Dwork, C., Hardt, M., Pitassi, T., Reingold, O. & Zemel, R. (2012). Fairness through awareness. *Proceedings of the 3rd innovations in theoretical computer science conference*, 214-226 (vid. pág. 3).

Elsken, T., Metzen, J. H. & Hutter, F. (2018). Neural Architecture Search: A Survey. https://doi.org/10.48550/ARXIV.1808.05377 (vid. pág. 7)

Estévez Velarde, S., Gutiérrez, Y., Montoyo, A. & Almeida Cruz, Y. (2020). Automatic Discovery of Heterogeneous Machine Learning Pipelines: An Application to Natural Language Processing, 3558-3568. https://doi.org/10.18653/v1/2020.coling-main.317 (vid. pág. 7)

Estevez-Velarde, S., Gutiérrez, Y., Montoyo, A. & Almeida-Cruz, Y. (2020). Automatic Discovery of Heterogeneous Machine Learning Pipelines: An Application to Natural Language Processing. *Proceedings of the 28th International Conference on Computational Linguistics*, 3558-3568 (vid. págs. 21, 27).

Estévez-Velarde, S., Gutiérrez, Y., Almeida-Cruz, Y. & Montoyo, A. (2020). General-purpose hierarchical optimisation of machine learning pipelines with grammatical evolution. *Information Sciences*. https://doi.org/10.1016/j.ins.2020.07.035 (vid. pág. 7)

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M. & Hutter, F. (2015). Efficient and robust automated machine learning. *Advances in neural information processing systems*, *28* (vid. pág. 7).

Friedler, S. A., Scheidegger, C. & Venkatasubramanian, S. (2016). On the (im) possibility of fairness. *arXiv preprint arXiv:1609.07236* (vid. pág. 3).

Hillermeier, C. y col. (2001). *Nonlinear multiobjective optimization: a generalized homotopy approach* (Vol. 135). Springer Science & Business Media. (Vid. pág. 9).

Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E. & Weinberger, K. Q. (2017). Snapshot Ensembles: Train 1, get M for free. *CoRR*, *abs/1704.00109*. http://arxiv.org/abs/1704.00109 (vid. pág. 6)

Jin, H., Song, Q. & Hu, X. (2018). Efficient Neural Architecture Search with Network Morphism. *CoRR*, *abs/1806.10282*. http://arxiv.org/abs/1806.10282 (vid. pág. 7)

Jin, W., Dong, S., Yu, C. & Luo, Q. (2022). A data-driven hybrid ensemble AI model for COVID-19 infection forecast using multiple neural networks and reinforced learning. *Computers in Biology and Medicine*, *146*, 105560. https://doi.org/https://doi.org/10.1016/j.compbiomed.2022.105560 (vid. pág. 6)

Kamiran, F. & Calders, T. (2011). Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, *33*, 1-33 (vid. pág. 4).

Kearns, M., Neel, S., Roth, A. & Wu, Z. S. (2018). Preventing fairness gerrymandering: Auditing and learning for subgroup fairness. *International Conference on Machine Learning*, 2564-2572 (vid. págs. 4, 5).

Keras. (s.f.). https://github.com/fchollet/keras. (Vid. pág. 7)

Kleinberg, J. (2018). Inherent trade-offs in algorithmic fairness. *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*, 40-40 (vid. pág. 3).

Kuhn, H. W. & Tucker, A. W. (2014). Nonlinear programming. *Traces and emergence of nonlinear programming* (pp. 247-258). Springer. (Vid. pág. 9).

Livieris, I. E., Pintelas, E., Stavroyiannis, S. & Pintelas, P. (2020). Ensemble deep learning models for forecasting cryptocurrency time-series. *Algorithms*, *13*(5), 121 (vid. pág. 6).

Loper, E. & Bird, S. (2002). NLTK: The Natural Language Toolkit. *CoRR*, *cs.CL/0205028*. https://arxiv.org/abs/cs/0205028 (vid. pág. 7)

MacCarthy, M. (2018). Standards of Fairness for Disparate Impact Assessment of Big Data Algorithms. *Other Information Systems & eBusiness eJournal* (vid. pág. 4).

Mégane, J., Lourenço, N. & Machado, P. (2021). Probabilistic grammatical evolution. *European Conference on Genetic Programming (Part of EvoStar)*, 198-213 (vid. pág. 8).

Miettinen, K. (2012). *Nonlinear multiobjective optimization* (Vol. 12). Springer Science & Business Media. (Vid. pág. 9).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. y col. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32* (vid. pág. 7).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. y col. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, 2825-2830 (vid. pág. 7).

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, *6*(3), 21-45 (vid. pág. 5).

Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, *5*(2), 197-227 (vid. pág. 5).

Schütze, O., Dell'Aere, A. & Dellnitz, M. (2005). On Continuation Methods for the Numerical Treatment of Multi-Objective Optimization Problems. En J. Branke, K. Deb, K. Miettinen & R. E. Steuer (Eds.), *Practical Approaches to Multi-Objective Optimization* (pp. 1-15). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/DagSemProc.04461.16. (Vid. pág. 9)
Keywords: multi-objective optimization, continuation, k-manifolds

Thomas, P. S., Castro da Silva, B., Barto, A. G., Giguere, S., Brun, Y. & Brunskill, E. (2019). Preventing undesirable behavior of intelligent machines. *Science*, *366*(6468), 999-1004 (vid. pág. 5).

Thornton, C., Hutter, F., Hoos, H. H. & Leyton-Brown, K. (2013). Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 847-855. https://doi.org/10.1145/2487575.2487629 (vid. pág. 6)

Witten, I., Hall, M., Frank, E., Holmes, G., Pfahringer, B. & Reutemann, P. (2009). The WEKA data mining software: An update. *SIGKDD Explorations*, *11*, 10-18. https://doi.org/10.1145/1656274.1656278 (vid. pág. 7)

Zafar, M. B., Valera, I., Gomez-Rodriguez, M. & Gummadi, K. P. (2019). Fairness Constraints: A Flexible Approach for Fair Classification. *Journal of Machine Learning Research*, *20*(75), 1-42. http://jmlr.org/papers/v20/18-262.html (vid. pág. 4)

Zafar, M. B., Valera, I., Rogriguez, M. G. & Gummadi, K. P. (2017). Fairness constraints: Mechanisms for fair classification. *Artificial intelligence and statistics*, 962-970 (vid. pág. 4).

Zemel, R., Wu, Y., Swersky, K., Pitassi, T. & Dwork, C. (2013). Learning fair representations. *International conference on machine learning*, 325-333 (vid. pág. 4).