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

Los sesgos que muestran los algoritmos de aprendizaje automático provienen de diversas fuentes. Una de las fuentes mas comunes de sesgos son las colecciones de datos, de las cuales los modelos de aprendizaje automático implícitamente aprenden a expresar tales sesgos. Los sesgos contenidos en las colecciones de datos pueden ser a veces el resultado de errores durante su construcción, pero es común que tengan su origen en procesos históricos. Cuando estos sesgos son utilizados para realizar una predicción, puede dar lugar a una determinación injusta acerca de los individuos.

Los sesgos pueden incluso provenir de fuentes mas difíciles de detectar para el usuario. Por ejemplo, la popularización de representaciones semánticas autogeneradas, tales como *word embeddings* pueden contribuir a la propagación de sesgos contenidos en datos históricos (Bolukbasi y col. 2016). Luego, incluso cuando se trabaja con una colección de datos que no contiene explícitamente nada relacionado con genero o raza, por ejemplo, al utilizar *word embeddings* clásicos para procesar datos en forma de texto pueden obtenerse decisiones sesgadas a causa de sesos que se han probado contienen los *word embeddings*.

Es posible que los modelos de aprendizaje automático produzcan decisiones injustas incluso asumiendo que sus datos de entrenamiento no tenían sesgos. Es decir, el modelo de aprendizaje automático ha construido una hipótesis que no generaliza completamente a los datos no vistos durante el entrenamiento o incluso a los propios ejemplos entrenantes. Luego, cuantificar los sesgos que un modelo de aprendizaje automático tiene respecto a un conjunto de datos es una tarea usual. Se hace imprescindible proveer una definición que permita capturar el concepto de equidad y analizar los sesgos que presenta un modelo de forma objetiva.

1.1. Definiciones de Equidad

Múltiples definiciones han sido propuestas para capturar diferentes criterios de equidad. No existe en estos momentos una única definición ampliamente aceptada de lo que es *equidad*, sino que diferentes definiciones codifican diferentes características que se muestran útiles en diferentes contextos. Incluso, algunas de las definiciones más comunes presentan conflictos entre si. A continuación se presentan algunas de las definiciones mas utilizadas.

Sea Y la clasificación (binaria) correcta, S el atributo protegido, y Y^p la clasificación obtenida del modelo.

- **Statistical Parity (SP)**: Un clasificador $\hat{Y}$ satisface *statistical parity* si $P(\hat{Y} = 1|P = 1) = P(\hat{Y} = 1|P = 0)$. Esto es, la probabilidad de un resultados positivo debería ser la misma sin importar si el individuo pertenece a un grupo protegido (Verma y Rubin 2018).
- **Equal Opportunity (EO)**: Un binario clasificador $\hat{Y}$ satisface *equal opportunity* con respecto a P y Y si $P(\hat{Y} = 1|Y = 1, P = 1) = P(\hat{Y} = 1|Y = 1, P = 0)$. Esto significa que la probabilidad de que a una persona en la clase positiva le sea asignada un resultado positivo debería ser igual para miembros tanto de grupos protegidos como no protegidos (Verma y Rubin 2018).
- **Equalized Odds (EOdd)**: Un clasificador $\hat{Y}$ satisface *equalized odds* con respecto al atributo protegido P y predicción Y, si $P(\hat{Y} = 1|Y = y, P = 1) = P(\hat{Y} = 1|Y = y, P = 0)$, es decir, $\hat{Y}$ y A son independientemente condicionales a Y. Esto significa que la probabilidad de que a una persona en la clase positiva le sea asignada correctamente una predicción positiva y la probabilidad de que a una persona en la clase negativa le sea incorrectamente asignada una predicción positiva debería ser la misma para miembros de grupos protegidos y no protegidos Verma y Rubin 2018.

Las concesiones inherentes a utilizar cada noción de equidad han sido estudiados extensamente (Dwork y col. 2012; Friedler y col. 2016; Kleinberg 2018). Escoger la definición correcta para un problema determinado es difícil, y en la práctica no puede ser delegado a un agente automático. En su lugar, una decisión humana es preferida para asegurar una decisión informada.

1.2. Mitigación de sesgos

Las técnicas de mitigación de sesgos pueden ser divididas fundamentalmente en técnicas de pre-procesamiento, post-procesamiento y técnicas durante el procesamien-

to. Adicionalmente un conjunto de técnicas llamadas meta-algoritmos han surgido recientemente, presentando muy buenos resultados.

Las técnicas de pre-procesamiento logran equidad modificando la representación de los datos, es decir, pre-procesando los datos y luego adoptando una solución de aprendizaje automático estándar (Calmon y col. 2017; Kamiran y Calders 2011; Zemel y col. 2013). Un ejemplo de esto es: aprender una representación a partir de resolver un problema de optimización con dos objetivos, codificar la información preservando la mayor cantidad de información posible y ofuscar al mismo tiempo la pertenencia al conjunto de atributos protegidos (Zemel y col. 2013). Una ventaja de los métodos de pre-procesamiento son agnósticos al modelo. Por el contrario, sus hiperparámetros, así como los del modelos de aprendizaje automático seleccionado todavía necesitan ser ajustados para mejor rendimiento.

Los métodos aplicados durante el procesamiento aseguran que se cumplan ciertas restricciones de equidad durante el entrenamiento (p.e. (Donini y col. 2018; Zafar, Valera, Gomez-Rodriguez y col. 2019; Zafar, Valera, Rogriguez y col. 2017)), sin embargo, esto los hace aplicable solo a una cierta clase de modelos. Por ejemplo, el algoritmo propuesto en Donini y col. 2018 solo puede ser aplicado a *kernel machines* (tales como *maquinas de soporte vectorial*), y solo a una única definición de equidad, dígase *Equal Opportunity*). Aunque las técnicas de mitigación durante el procesamiento pueden brindar muy buenos resultados para la clase del modelo que están diseñados, frecuentemente son difíciles, o a veces imposible, de extender para nuevas clases de modelos. Estos métodos también pueden introducir nuevos hiperparámetros que podrían requerir conocimiento específicos del dominio y experimentación.

Las técnicas de post-procesamiento operan ajustando el umbral de decisión de modelos pre-entrenados para eventualmente lograr resultados más justos respecto a una métrica de equidad dada. El principal problema es que post-procesar el umbral de decisión es inherentemente subóptimo y puede llevar a peores balances de eficacia y equidad. Adicionalmente, estas técnicas no son utilizables si la información sensible no esta disponible en el momento de realizar las predicciones, lo cual a su vez puede llevar a problemas legales por la utilización de información sensible para realizar predicciones (MacCarthy 2018).

Una clase de métodos recientemente propuesta para tareas de clasificación justa, conocida como meta-algoritmos, reduce la tarea de clasificación justa a una secuencia 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 suelen producir un clasificador *randomizado*. Contrario a los métodos que funcionan durante el procesamiento, los meta-algoritmos no dependen del tipo de los modelos que se utilizan en el clasificador, sino en la capacidad de los mismos para ser reentrenados repetidamente. En el contexto de algoritmos de minimización del riesgo empírico, estos métodos son agnósticos al modelo de aprendizaje

automático, pero aun necesitan implementaciones específicas basadas en la definición de equidad seleccionada y necesitan producir un ensamblado de modelos. Limitaciones similares caracterizan un número de enfoques que utilizan optimización (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 específicamente para ciertas definiciones de equidad.

1.3. Métodos de ensamblado

Los métodos de ensamblado están diseñados para intentar resolver el problema de *low bias / high variance* que muestran la mayoría de los modelos de aprendizaje automático, haciéndolos apropiados para modelos de clasificación más robustos (Polikar 2006). Un modelo de ensamblado está diseñado de muchos modelos con *low bias* cuyas predicciones son combinadas para producir una predicción final. Se asume fundamentalmente que la combinación de varias predicciones de bajo nivel producirá una salida con baja varianza mientras mantiene un *low bias*. Tener un conjunto diverso de modelos de bajo nivel es una característica fundamental para lograr esto (Polikar 2006). Sin embargo, esto requiere, que clasificadores individuales cometan errores en diferentes instancias. La intuición es que si cada clasificador comete diferentes errores, entonces una combinación estratégica de estos clasificadores puede reducir el error total. Luego, es necesario lograr que cada clasificador sea lo mas único posible, particularmente con respecto a ejemplos erróneamente clasificados.

La naturaleza de múltiples hipótesis de estos modelos asegura que, si son ajustados lo suficiente, tendrán resultados mejores que cualquiera de los modelos individuales en el caso general. Esto les permite también estimar el grado de confianza o calidad de las predicciones que producen. Las técnicas clásicas de ensamblado incluyen *voting* y *weighted voting* (Dietterich 2000), *boosting* (Schapire 1990), y *bagging* (Breiman 1996).

En un problema de clasificación, *voting* produce como salida la etiqueta que tiene la mayoría de los votos, tratando cada predicción 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 rendimiento en los ejemplos entrenantes donde los modelos anteriores tuvieron peor rendimiento. Durante este proceso, a cada submodelo le es también asignado un peso que marca la importancia de su predicción. *Bagging* entrena cada submodelo en una selección diferente (con reemplazo) de los ejemplos entrenantes originales. De esta forma, el modelo con una alta varianza debería producir modelos entrenados con alta diversidad. Alternativamente, *feature bagging* funciona de forma similar, seleccionando un subconjunto de características en lugar de los

ejemplos entrenantes, causando que características correlacionados sean analizados de forma separada en algunos submodelos.

Las capacidad de los métodos de ensamblado para construir modelos más robustos los ha hecho apropiados para múltiples aplicaciones. El dominio de la salud es uno de los ejemplos donde estos métodos han sido aplicados con gran éxito. Por ejemplo, una aplicación híbrida de métodos de ensemble con redes neuronales en un entorno de aprendizaje por reforzamiento ha sido presentada para al predicción de infección de COVID-19 con gran precisión (W. Jin y col. 2022). De forma similar, un método de toma de decisión multicriterio basado en ensembles ha sido propuesto para la detección de COVID-19 a partir del sonido de la tos en pacientes (Chowdhury y col. 2022). Existen ejemplos también no relacionados a la medicina, por ejemplo, en (Livieris y col. 2020) se emplearon estrategias de ensemble como *ensemble-averaging*, *bagging* y *stacking* con metodologías avanzadas de aprendizaje profundo para predecir los precios, a nivel de hora, de criptomonedas como *Ethereum*, *Bitcoin* y *Ripple*.

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

1.4. Métodos de AutoML

AutoML (del inglés *Automated Machine Learning*) es el proceso de automatizar la solución de problemas del mundo real a través de técnicas de aprendizaje automático. El proceso intenta eliminar la necesidad de humanos expertos en aprendizaje automático para seleccionar apropiadamente las características, flujos, paradigmas, algoritmos y sus hiperparámetros para resolver un problema (Dimitrakakis y col. 2019). Las principales ventajas de las tecnologías de *AutoML* incluyen: (1) reducir el tiempo empleado en resolver problemas bien estudiados; y, (2) eliminar la necesidad de conocimiento experto. Adicionalmente, estas tecnologías tienden a generar soluciones más simples que a menudo tienen mejor desempeño que soluciones diseñadas por humanos.

Múltiples tecnologías han sido propuestas para resolver el problema de *AutoML*. *Auto Weka* (Thornton y col. 2013) fue una de las primeras soluciones presentadas. Esta solución está basada en el software *Weka* (Witten y col. 2009), un software construido a partir de varias herramientas de visualización y algoritmos para análisis de datos y modelación predictiva. *Auto Weka* resuelve el problema de *AutoML* como un problema

CASH según definido a continuación.

Definición 1.1 *Sea* $A = \{A^{(1)}, \ldots, A^{(R)}\}$ *un conjunto de algoritmos, y sea* $\Lambda^{(j)}$ *el dominio de los hiperparámetros 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, denótese* $L(A^{(j)}_{\lambda}, D^{(i)}_{train}, D^{(i)}_{valid})$ *la perdida del algoritmo* $A^{(j)}$ *en* $D^{(i)}_{valid}$ *con hiperparámetros* λ*. Entonces, el problema de Selección de Algoritmo y Optimización de Hiperparámetros Combinado (CASH) es encontrar la configuración conjunta de algoritmo e hiperparámetros 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 AutoML son AutoSklearn (Feurer y col. 2015) y AutoKeras (H. Jin y col. 2018). Estos sistemas están basados en las bibliotecas de aprendizaje automático *ScikitLearn* (Pedregosa y col. 2011) y *Keras* (*Keras* s.f.), respectivamente. Ambos sistemas proveen una interfaz para encontrar la mejor arquitectura de aprendizaje automático para resolver un problema. Una diferencia fundamental entre ellos es la forma en que sus espacios de búsqueda son definidos. Mientras AutoSkLearn explora espacio de búsqueda condicional, es decir, un espacio con algunos hiperparámetros condicionados a otros, AutoKeras realiza una *Busqueda de Arqutectura Neuronal (NAS)* (Elsken y col. 2018), la cual implica explorar espacios jerárquicos de complejidad arbitraria.

AutoGOAL (Estévez Velarde y col. 2020; Estévez-Velarde y col. 2020) es una de las mas recientes contribuciones al campo del AutoML. AutoGOAL es un sistema que utiliza técnicas heterogéneas para resolver el problema CASH. La esencia de AutoGOAL radica en su espacio personalizable de flujos y su conjunto de algoritmos de búsqueda, que son usados para encontrar la mejor configuración para resolver un problema. Cada flujo está definido como un conjunto de algoritmos interconectados que traducen una entrada predefinida a su salida correspondiente. El espacio de flujos comprende no solo el conjunto de algoritmos, sino también sus hiperparámetros.

Múltiples fuentes de algoritmos están incluidos en el espacio de AutoGOAL, tales como *ScikitLearn*(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 múltiples flujos de extremo a extremo para generar una solución. Esta limitación puede ser superada con la utilización de ensembles.

El proceso fundamental de optimización utilizado en AutoGOAL en estos momentos esta basado en una técnica de optimización con Evolución Gramatical para gramáticas probabilistas libres del contexto (Mégane y col. 2021). El proceso consiste

de un ciclo de generación y evaluación, utilizando una gramática G apropiada para describir el problema de aprendizaje de maquina que se intenta resolver. En cada iteración un conjunto de N flujos es generado a partir de tomar muestras de la gramática G de acuerdo a las probabilidades θ asignadas a cada producción. Estas probabilidades son inicializadas con una distribución uniforme θ_0 para todas las producciones. Cada flujo es evaluado (lo cual consiste simplemente en entrenamiento y ejecución), 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 población
$n \leftarrow$ numero de individuos seleccionados en cada iteración
$\alpha \leftarrow$ factor de aprendizaje
$G \leftarrow$ gramática que describe los flujos de aprendizaje de maquina a explorar
$\theta_0 \leftarrow$ probabilidades iniciales (uniforme)
$f \leftarrow$ función de *fitness* (entrenamiento y evaluación de flujos)
$best \leftarrow none$
para *cada iteración* i **hacer**
    $P_i \leftarrow$ generar población utilizando gramática G, con probabilidades θ_{i-1}
    para *cada solución* $S \in P_i$ **hacer**
        $f(S) \leftarrow$ calcular *fitness* de S (evaluar el flujo)
    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 distribución marginal de P^*
    $\theta_i \leftarrow \alpha\theta_i^* + (1-\alpha)\theta_{i-1}$
fin
devolver *best*

1.5. Optimización Multiobjetivo

Los métodos de optimización multiobjetivo son aquellos que exploran el espacio de búsqueda optimizando simultáneamente diferentes funciones.

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

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

Al trabajar con múltiples 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 ámbito de la optimización multiobjetivo, dado que permite comparar objetivamente dos vectores de forma precisa, sin requerir información adicional de preferencia.

Definición 1.3 (Pareto Dominación) *Dados dos vectores en el espacio objetivo, dígase* $y^{(1)}, y^{(2)} \in \mathbf{R}^m$ *, 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* z *en el espacio objetivo que* ***pareto-domine*** *a* x*.*

Tradicionalmente los problemas de optimización multiobjetivo han sido atacados utilizando técnicas de escalarización (p.e. (Miettinen 2012)), estas técnicas consisten en de alguna forma combinar todas las funciones objetivos en una sola, o reescribirlas como restricciones. Varias técnicas existen en este contexto. *Linear Weighting* es una de estas, en la cual se construye una nueva función objetivo a partir de la combinación lineal de las funciones objetivo del problema original, esto es $min \sum w_i f_i(x), x \in X$. 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 escalarización es ϵ-*constrain*, esta técnica selecciona una función como la *principal* y las demás se establecen como restricciones al conjunto de soluciones factibles, exigiendo que sean menores que un ϵ.

Existen métodos numéricos 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 solución del sistema de ecuaciones que se produce al tratar el problema de *KKT*. Es posible utilizar métodos de continuación y homotopía para obtener todas las soluciones (Hillermeier y col. 2001; Schütze y col. 2005). Estos métodos requiere que las soluciones satisfagan condiciones de convexidad local y diferenciabilidad.

1.5.1. NSGA-II

Los algoritmos genéticos utilizan paradigmas basados en procesos evolutivos naturales, como *selección natural*, *mutación* y *recombinación* para mover una población (conjunto de vectores de decisión) a soluciones óptimas o casi óptimas (Back 1996). Los algoritmos genéticos multiobjetivo generalizan esta idea y son diseñados para en cada iteración acercarse más al frente pareto. En este contexto destaca *NSGA-II* (Deb y col. 2002), el cual se explica en mayor detalle a continuación.

El algoritmo consiste básicamente de un ciclo generacional que se divide en dos partes. En la primera parte, la población pasa por un proceso de variación. En la segunda parte, un proceso de selección toma lugar, el cual resulta en la población de la nueva generación. Este proceso se repite hasta que se cumple algún criterio de convergencia o se excede una cantidad de computo predefinida.

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

La segunda fase, fase de selección, los μ mejores individuos son seleccionados del conjunto $P_t \cup Q_t$ utilizando un mecanismo de ordenación multiobjetivo, de esta forma la población de la nueva generación P_{t+1} es formada. El mecanismo de selección de *NSGA-II* es el ingrediente fundamental que lo distingue del resto de los algoritmos genéticos que son utilizados para resolver problemas de optimización de un único objetivo. Este consiste de dos niveles. Primero se realiza un ***non-dominated sorting***. Este depende únicamente 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 medida de la diversidad.

Non-dominated sorting

Sea $\mathrm{ND}(P)$ el conjunto de soluciones no dominadas 1.3 en una población P. *Non-dominated sorting* particiona la población en subconjuntos, basado en la *pareto dominación* 1.3 como especifica la siguiente recurrencia.

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

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

Como en cada paso de la recurrencia al menos una solución es eliminada de la población, el número máximo de *capas* es $|P|$. El orden de una solución esta dado por el subíndice k del R_k en el cual queda dicha solución.

Crownding distance

Si más de una solución quedan en el mismo subconjunto de la población R_k luego de realizar la ordenación anterior, se procede a ordenar las soluciones dentro de dichos

subconjunto a partir de su *crowding distance*. Esta es calculada para una solución x como la suma de las contribuciones c_i a la i- ésima función 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 solución, 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 propósito de mantener la diversidad en la población.

1.6. Discusión

Capítulo 2

Propuesta

En este capítulo ...

2.1. Descripción general

El sistema toma como entrada una colección de datos $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ y una función de pérdida $\mathcal{L}$ y una o varias métricas de equidad $F_1, F_2, \ldots, F_n$. El objetivo del sistema es producir un modelo de clasificación que es a la vez efectivo según L y justo según $F_1, F_2, \ldots, F_n$. El sistema consiste en dos fases fundamentales. La primera es responsable de generar una colección de modelos, cada uno llamado modelo base. Esta colección es construida optimizando una función de perdida $\mathcal{L}_1$, mientras se asegura *diversidad* a lo largo de toda la población. La segunda fase es responsable de producir un conjunto de modelos de clasificación. Estos modelos son generados ensamblando la colección de modelos base de forma tal que optimice su efectividad según L_2, a la vez que es lo mas *justo* posible según $F_1, F_2, \ldots, F_n$. 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.

Párrafo con la imagen del overview, y que describe la imagen.

2.2. Generación de modelos base

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

La Definición 1.1 se modifica para buscar una colección de modelos en lugar de un solo modelo, sujeto a una métrica de diversidad $\mathcal{D}$. Esto es, se desea encontrar una colección de modelos base (modelos que optimicen la efectividad en el conjunto de

datos D de acuerdo a la función de pérdida $\mathcal{L}$) mientras garantiza algunas diferencias entre sus hipótesis utilizando la métrica $\mathcal{D}$. Asegurar diversidad en la colección de modelos base es importante porque los métodos de ensemble no son capaces de mejorar su rendimiento si todos los modelos base tienen exactamente la misma hipótesis, es decir si todos realizan las mismas predicciones.

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

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

set base_models $\leftarrow \emptyset$
para generation $\in$ **explore**(A, Λ) **hacer**
  scores $\leftarrow \emptyset$
  para $A_\lambda^{(j)} \in$ generation **hacer**
    scores$^{(j)} \leftarrow \frac{1}{K}\sum_{i=1}^{K} \mathcal{L}(A_\lambda^{(j)},\ D_{train}^{(i)},\ D_{valid}^{(i)})$
  fin
  diversity $\leftarrow \mathcal{D}($base_models $\frown$ generation$,\ D)$
  base_models $\leftarrow$ **reselect**(base_models $\frown$ generation, scores, diversity, N)
fin
devolver base_models

Para explorar inteligentemente el espacio de algoritmos y hiperparámetros, es decir, para resolver el problema *CASH* modificado, se utiliza la implementación de *Probabilistic Grammatical Evolution Search* (algoritmo 1.1) presente en AutoGOAL. AutoGOAL se refiere a los modelos que construye como flujos, dado que cada uno de ellos esta formado por algoritmos interconectados. La búsqueda comienza con una estrategia de muestreo aleatorio, pero según evalúa más flujos, modifica el modelo de muestreo probabilista para que flujos similares a los mejores encontrados hasta el momento, sean generados con mayor frecuencia. El espacio de algoritmos y hiperparámetros utilizados es el utilizado por defecto en AutoGOAL, el cual incluye varios algoritmos clásicos de aprendizaje automático presentes en las diferentes bibliotecas utilizadas por AutoGOAL.

Para reseleccionar la colección de modelos base, o sea, la colección de flujos de AutoGOAL, un enfoque goloso es utilizado y estudiado. La función **reselect** reselect resume la estrategia propuesta. El algoritmo siempre incluye el modelo que mejor se

desempeña de acuerdo a L en la selección. Cada iteración siguiente añade el modelo, no todavía seleccionado, que maximiza la diversidad respecto a todos los modelos anteriormente seleccionados. El enfoque goloso no garantiza que la colección final logre la mejor posible diversidad respecto a $\mathcal{D}$. La precisión 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 métodos *reselect* de referencia fueron implementados para realizar comparaciones entre las métricas de diversidad *disagreement* y *double-fault*, las cuales son presentadas en la sección 2.3.

Shuffle. La colección de modelos base es construida barajando aleatoriamente la selecciona actual de modelos base y los nuevos encontrados. Los primeros N modelos luego de barajar son seleccionados para la siguiente generación.

Arbitrary. La colección 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 colección de modelos base seleccionados.

Best. La colección de modelos base es construida a partir de seleccionar los modelos de mejor rendimiento entre los previamente seleccionados y los recién encontrados.

A continuación, la sección 2.3 provee algunos detalles acerca de las métricas de diversidad estudiadas en este trabajo.

2.3. Métricas de diversidad

Dos métricas fueron implementadas para estimar la diversidad de una colección dada de modelos base. Ambas de ellas precomputan una matriz de clasificaciones incorrectas, la cual es utilizada entonces para computar una métrica que aporta información sobre la diversidad entre los modelos base dos a dos. La matriz de clasificaciones incorrectas se construye de la siguiente manera.

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

Las siguientes métricas son computadas entre pares de modelos base para estimar cuando diferentes son sus hipótesis, y por tanto la diversidad de la colección incluyendo a ambos a la vez.

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

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

Pon algun párrafo de clausura. En plan, algo que diga que recordemos que la diversidad de los modelos influye en el performance de los ensembles, y que cual de estas 2 metricas es mejor se estudiará en la experimentación, y se compararán con otros approach usando selecciones aleatorias.

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 métricas 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 combinación E^* de algoritmos y sus

hiperparámetros para ensamblar las salidas de los modelos base. Particularmente, un espacio de algoritmos especializados en estrategias de ensamblados y sus hiperparámetros es utilizado en esta fase. Estas estrategias son presentadas a continuación.

Voting Classifiers. Asigna la etiqueta mas común 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 combinación de salida de los modelos base la etiqueta que asegura el mejor desempeño en D^e_{train}. En el momento de predicción, si una combinación 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 automático sobre D^e_{train} para optimizar $\mathcal{L}$. La arquitectura del modelo de aprendizaje automático es tomado del conjunto de algoritmos disponibles por defecto a AutoGOAL.

los hiperparámetros de esto? Puedes incluir un pseudo codigo ademas de como se construye el espacio a partir de un sampler

Anade una image que ilustre el comportamiento de los 3 ensemblers

Para explorar el espacio de algoritmos e hiperparámetros se realiza una modificación a *Probabilistic Grammatical Evolution*(1.1), en la cual la fase de seleccionar los n individuos de la población actual que pasan a la siguiente generación, se realiza utilizando *Non-dominated sorting* (sección 1.5.1) y *Crowding distance* (sección 1.5.1), de la misma forma en que se utiliza en *NSGA-II*. De esta forma la función de selección, y por tanto la exploración del espacio, optimizan simultáneamente las métricas de equidad y la función de pérdida.

Parrafo describiendo cómo se combinan PGE y NSGA-II (no decirme qué se hace, sino cómo se hace). Con el parrafo anterior dices "qué pero del çomo.ªpenas dices una cosa

Pseudo codigo de NS-PGE. Tienen que quedar claro en él que (1) se itera por generaciones con PGE, (2) se actualizan los hiperparámetros segun SN, (3) cómo se construye la seleccion de modelos final, (4) como se actualiza entre generaciones

.

Párrafo explicando por qué funciona (o se espere que funcione) esa forma para optimizar fairness

Explicar cómo se construye la colección final. Y por qué quedarse con los no dominados de cada generación tiene sentido

Explicar como el usuario puede encontrar de todas las soluciones la que quiere al final. Además como puede especificar constraints para dirigir la busqueda, y donde se insertan en el codigo.

Hay todo un grupo de cosas que existen fuera de las 2 fases, que va de como integrarlas, que podría ir en una ultima sección de ejecución end to end. En general deberias revisar el codigo del Diversifier, Ensembler, y Mitigator, y ver qué cosas están y no están dichas en la propuesta, como eso mismo de los contraints, el detriment, etc.

Capítulo 3

Análisis Experimental

En este capítulo se evalúa la capacidad de nuestro sistema de optimización para resolver problemas de clasificación y lograr buenos resultados según métricas de precisión y equidad. La experimentación realizada consiste de dos etapas. Inicialmente se analiza la capacidad de la primera fase del sistema para obtener un conjunto de modelos base lo suficientemente diverso como para que ensamblar sus predicciones resulte en un modelo de mayor precisión que los modelos base. Finalmente, se estudia si el algoritmo propuesto permite encontrar formas de ensamblar los modelos base resultantes de la primera etapa de manera tal que se obtengan valores satisfactorios tanto de precisión como en las métricas de equidad.

3.1. Marco Experimental

Las dos etapas en las que queda divida nuestra experimentación se describen a continuación

La **primera etapa experimental** tiene como objetivo estudiar la capacidad del sistema de producir un conjunto de modelos base que, al ser ensamblado, pueda 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 artificiales, estos son 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. Llamaremos a dichos ensembles: *Oráculo Optimista* y *Oráculo Sobreajustado*.

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 hiperparámetros como la cantidad de modelos base. La mejor configuración de dichos hiperparámetros, 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. Como parte de esta etapa se realizan múltiples ejecuciones con diferentes configuraciones de hiperparámetros. La configuración de hiperparámetros para la cual se observen los mejores resultados en esta etapa, sera la utilizada en la segunda etapa experimental, como se describe a continuación.

Una segunda etapa experimental se lleva a cabo, la cual consiste de la ejecución de extremo a extremo de nuestro sistema, utilizando la colección de datos *Adult* (Dua y Graff 2017) como escenario de evaluación. En esta etapa se utiliza la mejor configuración de hiperparámetros acorde a los resultados de la primera fase, es relevante destacar que entre estos hiperparámetros preseleccionados en la etapa anterior se encuentran la medida de diversificación a utilizar en la selección de modelos base y la cantidad máxima de estos. 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 `<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 *BERT* fueron excluidos y fundamentalmente flujos basados en *ScikitLearn* fueron utilizados.

No incluir los algoritmos de aprendizaje profundo en la configuración experimental puede tener un impacto negativo en el rendimiento máximo que puede ser alcanzado por el sistema. Es decir, la precisión del sistema no puede ser comparada directamente con otras soluciones reportadas, en términos de magnitud, pues aquellas soluciones que utilizan técnicas de aprendizaje profundo tienen una ventaja inherente en problemas donde modelos más simples no son tan competitivos. 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. Por tanto, la comparativa con otros métodos de mitigación no debería verse afectada significativamente.

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.

Aquí quizas puedes poner un ejemplo de cómo queda un llamado a la interfaz para resolver Adult. En plan, la parte de cargar los datos la pones como una funcion maquica y abajo lo que pones es el código de python con los hiperparámetros que tocan.

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*, para configuraciones con número máximo de clasificadores base siendo 5, 20, 50 y 100 respectivamente. El sistema fue configurado para utilizar en ambas fases una función de pérdida basada en F_1, la cual fue propuesta como métrica de puntuación 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 tres restantes 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 (sección 2.4); y, (iii) el mejor modelo base encontrado (sección 2.2). Además,

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

el tipo de algoritmo de ensemble utilizado por el mejor de los modelos de ensemble encontrados es adicionado al final de cada tabla.

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

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

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 sobreajustado 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 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.

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

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

Las tablas 3.9 y 3.10 resumen los diferentes tipos de ensemble que fueron encontrados como óptimos 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.

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

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

Tabla 3.11: Evaluation in the Adult dataset in terms of .... Maximum number of classifiers set to 50. The base model selection strategy was set to *double-fault*. No detriment constraints in use.

Accuracy Score	
evaluations	adult
A^* (training)	0.839 (±0,069)
E^* (training)	**0.860** (±0,060)
delta (training)	0.021
A^* (testing)	0.823 (±0,019)
E^* (testing)	**0.831** (±0,013)
delta (testing)	0.008

Adult				
method	accuracy	statistical parity	equal opportunity	equalized odds
FERM	0.81	-	0.01	-
Zafar Adversarial	0.78	-	0.05	-
FERM preprocess SMOTE				
FBO (RF)	$\approx 0{,}785$	$\leq 0{,}05$	$\leq 0{,}05$	$\leq 0{,}05$
FBO (XGBoost)	$\approx 0{,}835$	$\leq 0{,}05$	$\leq 0{,}05$	$\leq 0{,}05$
FBO (NN)	$\approx 0{,}775$	$\leq 0{,}05$	$\leq 0{,}05$	$\leq 0{,}05$
FBO (LL)	$\approx 0{,}810$	$\leq 0{,}05$	$\leq 0{,}05$	$\leq 0{,}05$
BFair (20)	0.817	0.049	0.014	0.007
BFair (20)	0.820	0.057	0.002	0.005
BFair (50)	0.833	0.089	0.086	0.052
BFair (20)	0.860	-	0.032	-

Tabla 3.12: Restriccion $DSP \leq 0{,}1$.

Precisión	
Método	Resultados en *Adult*
FERM	$0{,}164 \pm 0{,}010$
Zafar	$0{,}187 \pm 0{,}001$
Adversarial	$0{,}237 \pm 0{,}001$
FERM preprocess	$0{,}228 \pm 0{,}013$
SMOTE	$0{,}178 \pm 0{,}005$
FairBO (LL)	$0{,}175 \pm 0{,}007$
BFair (20)	0,170 $_{\text{SP}=0{,}087}$

3.3.1. Resultados

3.3.2. Discusión

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. 5).

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. 5).

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

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V. & Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems*, *29* (vid. pág. 3).

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24* (2), 123-140 (vid. pág. 6).

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. 5)

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. 6).

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. págs. 21, 22).

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. 7)

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. 30).

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. *Multiple Classifier Systems*, 1-15 (vid. pág. 6).

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. 6, 7)

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. 5).

Dua, D. & Graff, C. (2017). UCI Machine Learning Repository. http://archive.ics.uci.edu/ml. (Vid. págs. 21, 22)

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. 4).

Elsken, T., Metzen, J. H. & Hutter, F. (2018). Neural Architecture Search: A Survey. https://doi.org/10.48550/ARXIV.1808.05377 (vid. pág. 8)

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. 8)

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. 23, 30).

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. 8)

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. 8).

Friedler, S. A., Scheidegger, C. & Venkatasubramanian, S. (2016). On the (im) possibility of fairness. *arXiv preprint arXiv:1609.07236* (vid. pág. 4).

Hillermeier, C. y col. (2001). *Nonlinear multiobjective optimization: a generalized homotopy approach* (Vol. 135). Springer Science & Business Media. (Vid. pág. 10).

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. 7)

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. 8)

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. 7)

Kamiran, F. & Calders, T. (2011). Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, *33*, 1-33 (vid. pág. 5).

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. 5, 6).

Keras. (s.f.). https://github.com/fchollet/keras. (Vid. pág. 8)

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. 4).

Kuhn, H. W. & Tucker, A. W. (2014). Nonlinear programming. *Traces and emergence of nonlinear programming* (pp. 247-258). Springer. (Vid. pág. 10).

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. 7).

Loper, E. & Bird, S. (2002). NLTK: The Natural Language Toolkit. *CoRR*, *cs.CL/0205028*. https://arxiv.org/abs/cs/0205028 (vid. pág. 8)

MacCarthy, M. (2018). Standards of Fairness for Disparate Impact Assessment of Big Data Algorithms. *Other Information Systems & eBusiness eJournal* (vid. pág. 5).

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. 10).

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. 8).

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. 8).

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

Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, *5*(2), 197-227 (vid. pág. 6).

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. 10)
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. 6).

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. 7)

Verma, S. & Rubin, J. (2018). Fairness definitions explained. *2018 ieee/acm international workshop on software fairness (fairware)*, 1-7 (vid. pág. 4).

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. 5)

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. 5).

Zemel, R., Wu, Y., Swersky, K., Pitassi, T. & Dwork, C. (2013). Learning fair representations. *International conference on machine learning*, 325-333 (vid. pág. 5).