

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

Introducción	1
1. Estado del Arte	3
1.1. Definiciones de Equidad	4
1.2. Mitigación de sesgos	5
1.3. Métodos de ensemble	7
1.4. Métodos de AutoML	8
1.5. Optimización Multiobjetivo	11
1.5.1. NSGA-II	12
1.6. Discusión	14
2. Propuesta	15
2.1. Descripción general	15
2.2. Fase 1: Generación de modelos base	16
2.2.1. Métricas de diversidad	17
2.3. Fase 2: Ensamblado de modelos justos	18
2.3.1. Espacio de búsqueda	19
2.3.2. Encontrando modelos justos y efectivos	19
3. Análisis Experimental	23
3.1. Marco Experimental	23
3.1.1. Escenarios de Evaluación	25
3.1.2. Corpus de Evaluación	26
3.1.3. Configuración Experimental	27
3.2. Primera Etapa Experimental	28
3.2.1. Resultados	28
3.2.2. Discusión	31
3.3. Segunda Etapa Experimental	36
3.3.1. Resultados	36
3.3.2. Discusión	37

Conclusiones	38
Recomendaciones	39
Bibliografía	40

Índice de figuras

Ejemplos de código

Introducción

En la actualidad los algoritmos de aprendizaje automático están siendo aplicados en disímiles áreas de la vida humana. Es común encontrarlos aplicados en sistemas de recomendación de compras, aplicaciones de citas, solicitudes de préstamos, contratación personal y muchas otras áreas. A raíz de ello, ha surgido un creciente interés en estudiar las potencialidades y limitaciones de los modelos de aprendizaje automático, así como las posibles implicaciones de confiar ciegamente en sus predicciones.

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

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

Motivación

Un modelo de aprendizaje de máquina se entrena con el objetivo de optimizar una única métrica, en la mayoría de los casos la precisión. 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 están desafortunadamente presentes en la sociedad y por ende en los datos recopilados, en algunos casos incluso amplifican estos patrones negativos. Son varias las técnicas 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 variaciones en el método de entrenamiento con el mismo objetivo. Sin embargo permanece relativamente poco explorado el uso de técnicas de optimización multiobjetivo que permitan al modelo optimizar hasta encontrar un buen balance entre cuán justo es y cuán 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 exista 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 metaheurísticas para optimizar los modelos utilizando simultaneamente metricas de equidad y precision.
- Explorar adición de optimización 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 optimización 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 (4). 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.

- **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 (51).
- **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 (51).
- **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 51.

Las concesiones inherentes a utilizar cada noción de equidad han sido estudiados extensamente (18, 24, 34). 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.

Es posible transformar las definiciones de equidad anteriores para obtener métricas, que de cierta forma cuantifiquen el sesgo de un clasificador. Por ejemplo *Differential Statistical Parity* (DSP) puede ser obtenida utilizando la definición de *Statistical Parity* para obtener una medida de la diferencia entre la cantidad de resultados positivos entre los diferentes grupos protegidos. Luego, estas métricas pueden ser utilizadas como funciones de perdida a emplear en algoritmos de optimización que permitirán mitigar los sesgos de los modelos.

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 procesamiento. 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 (6, 31, 55). 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 (55). 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. (16, 54, 53)), sin embargo, esto los hace aplicable solo a una cierta clase de modelos. Por ejemplo, el algoritmo propuesto por *Donini, Michele et. al.* (16) 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. El conocimiento por parte del sistema de información sensible de los individuos reales en el momento de realizar las predicciones puede llevar a problemas legales (39).

Finalmente, resultan relevantes los llamados meta-algoritmos, una clase de métodos recientemente propuesta para tareas de clasificación justa. Estos reducen la tarea de clasificación justa a una secuencia de problemas de clasificación con costo asociado a sus errores de predicción (2, 1, 32). 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 repetidamen-

te. En el contexto de algoritmos de Minimización del Riesgo Empírico (*ERM* por sus siglas en inglés), 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 (8, 14) o inferencia bayesiana (32, 49), sus implementaciones tienen que estar diseñadas específicamente para ciertas definiciones de equidad.

Varios algoritmos han sido propuestos en la literatura, que o bien se acogen a uno de los enfoques anteriormente descritos o combinan varios de ellos. A continuación se exploran algunos de estos métodos que resultan sumamente relevantes para el presente trabajo.

***Fair Bayesian Optimization* (45)** Motivado por la versatilidad del ajuste de hiperparámetros, propone un enfoque basado en *Optimización Bayesiana* (*BO*) general restringida. La *Optimización Bayesiana* es una metodología bien establecida para optimizar funciones de caja negra costosas de evaluar. La técnica propuesta por *FBO* optimiza los hiperparámetros de una función de caja negra de forma agnóstica al modelo, mientras se satisfacen restricciones de equidad.

***SMOTE* (7)** Propone un metodo de preprocesamiento para colecciones de datos desbalanceadas. El enfoque de este método se basa fundamentalmente en el sobre-muestreo de los datos pertenecientes a las clases menos representadas en la colección. Utiliza un algoritmo que genera nuevos ejemplos a partir de modificaciones en los ejemplos pertenecientes a dicha clase existentes en los datos.

***Fair Empirical Risk Minimization* (*FERM*) (15)** Introduce una generalización de las diferentes nociones de equidad, las cuales restringen el *riesgo condicional* de un clasificador de acuerdo a una función de pérdida predeterminada y un parámetro de aproximación. Este intenta resolver el problema de minimizar el riesgo esperado para un conjunto predeterminado de funciones, utilizando una modificación de *Empirical Risk Minimization* a la cual se refieren como *Fair Empirical Risk Minimization*. Además, se proponen ejemplos concretos de algoritmos como *Máquinas de Soporte Vectorial* mejorados para satisfacer estas restricciones de equidad. Finalmente se muestra como para el caso lineal de la restricción de equidad modificada, este enfoque se reduce a un paso de preprocesamiento sobre la colección de datos.

***Zafar et. al.* (54)** El trabajo realizado por los autores introduce una métrica para la equidad del umbral de decisión, la *covarianza del umbral de decisión*. Utilizando esta métrica como base, se definen dos problemas a resolver, la optimización

de la equidad con restricciones sobre la precisión, y la optimización de la precisión con restricciones sobre la equidad. Los autores realizan modificaciones a *Regresión Logística* y *Maquinas de Soporte Vectorial* para utilizar estas restricciones y resolver estos problemas propuestos.

Mitigación de Sesgos mediante Aprendizaje Adversarial (56) replantea el problema de mitigación de sesgos como un problema de aprendizaje profundo adversarial (25). Esto significa que la tarea se reduce a encontrar un modelo predictor que resuelva el problema de clasificación a la vez que intenta que un adversario no pueda inferir el valor de los atributos protegidos a partir de la predicción.

1.3. Métodos de ensemble

Los métodos de ensamble 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 (46). Un modelo de ensamble 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 (46). 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 los métodos de ensemble 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* (13), *boosting* (47), y *bagging* (5).

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. 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 (29). 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 (10). Existen ejemplos también no relacionados a la medicina, por ejemplo, en (37) 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* (27). 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.

Como se puede observar, los métodos de ensemble son muy poderosos. Estos son capaces de construir sistemas que a partir de las predicciones de un conjunto de modelos (quizás no lo suficientemente expresivo) generalizan bien y obtienen resultados satisfactorios. Sin embargo, es necesario para la construcción de estos ensembles se debe partir de un conjunto de modelos base previamente obtenidos, que cumplan determinadas características. Luego, resulta sumamente interesante explorar enfoques que permitan obtener estos modelos base de forma automática para un problema determinado, de forma tal que no sea necesario manualmente tomar decisiones acerca de el tipo de modelos a utilizar, lo hiperparámetros de los mismos, etc.

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 (14). 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*. *AutoWeka* (50) fue una de las primeras soluciones presentadas. Esta solución está basada en el software *Weka* (52), un software construido a partir de varias herramientas de visualización y algoritmos para análisis de datos y modelación predictiva. *AutoWeka* resuelve el problema de *AutoML* como un problema *CASH* según definido a continuación.

Definición 1.1 Sea $A = \{A^{(1)}, \dots, A^{(R)}\}$ un conjunto de algoritmos, y sea $\Lambda^{(j)}$ el dominio de los hiperparámetros del algoritmo $A^{(j)}$. Sea, $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ el conjunto de entrenamiento, el cual es dividido en K cross-validation folds de la forma $\{D_{valid}^{(1)}, \dots, D_{valid}^{(K)}\}$ y $\{D_{train}^{(1)}, \dots, D_{train}^{(K)}\}$ tal que $D_{train}^{(i)} = D \setminus D_{valid}^{(i)}$ para todo $i = 1, \dots, K$. Finalmente, denótese $L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$ la pérdida del algoritmo $A^{(j)}$ en $D_{valid}^{(i)}$ 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 pérdida:

$$A^*, \lambda_\star \in \operatorname{argmin}_{A^{(j)}, \lambda \in \Lambda^{(j)}} \frac{1}{K} \sum_{i=1}^K L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)}) \quad (1.1)$$

Otros sistemas populares de *AutoML* son *AutoSklearn* (23) y *AutoKeras* (28). Estos sistemas están basados en las bibliotecas de aprendizaje automático *ScikitLearn* (44) y *Keras* (33), 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 Arquitectura Neuronal (NAS)* (19), la cual implica explorar espacios jerárquicos de complejidad arbitraria.

AutoGOAL (20, 22) es una de las mas recientes contribuciones al campo del *AutoML*. *AutoGOAL* es un sistema que utiliza técnicas heterogéneas para resolver el problema *CASH*. *AutoGOAL* se refiere a los modelos que construye como flujos, dado que cada uno de ellos esta formado por algoritmos interconectados. 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*(44), *NLTK*(38), *Keras*(33), y *Pytorch*(43). 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 está basado en una técnica de optimización con Evolución Gramatical para gramáticas probabilistas libres del contexto (40). El proceso consiste de un ciclo de generación y evaluación, utilizando una gramática G apropiada para describir el problema de aprendizaje de máquina 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

```

1  $N \leftarrow$  tamaño de la población
2  $n \leftarrow$  numero de individuos seleccionados en cada iteración
3  $\alpha \leftarrow$  factor de aprendizaje
4  $G \leftarrow$  gramática que describe los flujos de aprendizaje de máquina a explorar
5  $\theta_0 \leftarrow$  probabilidades iniciales (uniforme)
6  $f \leftarrow$  función de fitness (entrenamiento y evaluación de flujos)
7  $best \leftarrow none$ 
8 para cada iteración  $i$  hacer
9    $P_i \leftarrow$  generar población utilizando gramática  $G$ , con probabilidades  $\theta_{i-1}$ 
10  para cada solución  $S \in P_i$  hacer
11     $f(S) \leftarrow$  calcular fitness de  $S$  (evaluar el flujo)
12  fin
13   $best \leftarrow \operatorname{argmax}_{S \in P \cup \{best\}} \{f(S)\}$ 
14   $P_i^* \leftarrow$  seleccionar los  $n$  mejores individuos de  $P_i$ 
15   $\theta_i^* \leftarrow$  calcular la distribución marginal de  $P^*$ 
16   $\theta_i \leftarrow \alpha \theta_i^* + (1 - \alpha) \theta_{i-1}$ 
17 fin
18 devolver  $best$ 

```

A pesar de la gran variedad de algoritmos que AutoGOAL tiene a su disposición y los satisfactorios resultados que obtiene para una gran variedad de problemas, es una limitación significativa el hecho de que solo es posible optimizar una función objetivo. La capacidad de AutoGOAL para encontrar modelos que optimicen múltiples funciones objetivos simultáneamente posibilitaría la solución de un amplio espectro de problemas que son inherentemente multi-objetivo.

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 $f_1 : X \rightarrow \mathbf{R}, \dots, f_m : X \rightarrow \mathbf{R}$ las cuales traducen el espacio X en \mathbf{R} , un problema de optimización multiobjetivo esta dado es expresado de la siguiente forma:*

$$\text{minimizar } f_1(x), \dots, \text{minimizar } f_m(x), x \in X \quad (1.2)$$

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:*

$$\forall_{i \in \{1, \dots, m\}} : y_i^{(1)} \leq y_i^{(2)} \text{ y } \exists_{j \in \{1, \dots, m\}} : y_j^{(1)} < y_j^{(2)} \quad (1.3)$$

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. (41)), 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 . *Random Scalarization* (42) propone una estrategia basada en escalarizaciones aleatorias de las funciones objetivo. En lugar de optimizar una única escalarización de las funciones objetivos, este enfoque itera sobre un conjunto de escalarizaciones durante un proceso de *Optimización Bayesiana*. Finalmente, *ParEGO* (35) es una extensión de *EGO* (30) para la utilización en escenarios multiobjetivos. Utiliza un enfoque que combina las diferentes funciones objetivo en una sola a partir de una escalarización utilizando un vector de pesos parametrizado, utilizando diferentes parámetros para dicho vector de pesos el *Frente Pareto* es construido de forma gradual.

Existen métodos numéricos que intentan resolver el problema haciendo cumplir las condiciones de *Karush-Kuhn-Tucker* (36). 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 (26, 48). Estos métodos requiere que las soluciones satisfagan condiciones de convexidad local y diferenciabilidad.

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 (3). 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* (11), el cual se explica en mayor detalle a continuación.

1.5.1. NSGA-II

NSGA-II es un algoritmo sumamente sencillo, pero aun así ha mostrado ser muy efectivo en la resolución de problemas de optimización multiobjetivo. 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 $ND(P)$ 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.

$$R_1 = ND(P) \quad (1.4)$$

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

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.

Crowding 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:

$$l_i(x) = \max\{f_i(y) | y \in R \setminus \{x\} \wedge f_i(y) \leq f_i(x)\} \cup \{-\infty\} \quad (1.6)$$

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

$$c_i(x) = u_i - l_i, \quad i = 1, \dots, m \quad (1.8)$$

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

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

El método propuesto en este trabajo explora la utilización de AutoGOAL como vía para obtener un conjunto de modelos *diversos* base que puedan ser posteriormente ensamblados. El proceso de construir el ensemble a partir de estos modelos base se realiza mediante una modificación del algoritmo de optimización de AutoGOAL, la cual permite explorar el espacio de búsqueda optimizando simultáneamente varias métricas, como por ejemplo la precisión y una o varias métricas de equidad. Para la modificación de AutoGOAL este trabajo se inspira en los metodos de selección propuestos por *NSGA-II*, estos son, *Non-dominated Sorting* y *Crowding Distance*.

Capítulo 2

Propuesta

En este capítulo se presenta un sistema que permite la resolución de problemas de clasificación arbitrarios. Este sistema tiene entre sus objetivos fundamentales producir clasificadores que sean justos respecto a una o varias métricas de equidad, a la vez que minimizan una función de pérdida determinada. Para ello se propone un enfoque basado en dos fases, que utiliza una combinación de técnicas de *AutoML*, métodos de ensemble y optimización multiobjetivo. La sección 2.1 provee una descripción general del sistema. Las secciones 2.2 y 2.3 detalla el funcionamiento de la primera y segunda fase del sistema respectivamente.

2.1. Descripción general

El sistema toma como entrada una colección de datos $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ y una función de pérdida \mathcal{L} y una o varias métricas de equidad F_1, F_2, \dots, 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, \dots, 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 la función de pérdida \mathcal{L} , 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 , a la vez que es lo mas *justo* posible según F_1, F_2, \dots, F_n . Las secciones 2.2 y 2.3 abordan con mas detalles la primera y segunda fase, respectivamente.

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

2.2. Fase 1: 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 preseleccionada. 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($N, D, A, \Lambda, \mathcal{L}, \mathcal{D}$)`

```

1 set base_models  $\leftarrow \emptyset$ 
2 para generation  $\in \text{explore}(A, \Lambda)$  hacer
3   scores  $\leftarrow \emptyset$ 
4   para  $A_\lambda^{(j)} \in \text{generation}$  hacer
5     scores(j)  $\leftarrow \frac{1}{K} \sum_{i=1}^K \mathcal{L}(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$ 
6   fin
7   diversity  $\leftarrow \mathcal{D}(\text{base\_models} \cup \text{generation}, D)$ 
8   base_models  $\leftarrow \text{reselect}(\text{base\_models} \cup \text{generation}, \text{scores}, \text{diversity}, N)$ 
9 fin
10 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. La búsqueda comienza con una estrategia de muestreo aleatorio, pero según evalúa más flujos, modifica el modelo de muestreo probabilístico 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 reelegir la colección de modelos base, o sea, la colección de flujos de AutoGOAL, un enfoque goloso es utilizado. La función **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 $\text{reselect}(M, \text{scores}, \text{diversity}, N)$

```

1 set  $R \leftarrow \emptyset$ 
2 set  $R^{(0)} \leftarrow \underset{m^{(j)} \in M}{\operatorname{argmin}} \text{scores}^{(j)}$ 
3 para  $r \leftarrow 1$  a  $N$  hacer
4    $R^{(r)} \leftarrow \underset{(m^{(j)} \in M \setminus R)}{\operatorname{argmax}} \sum_{(m^{(i)} \in R)} \text{diversity}^{(i,j)}$ 
5 fin
6 devolver  $R^{(0)} \dots R^{(N)}$ 

```

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 2.2.1.

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 2.2.1 provee algunos detalles acerca de las métricas de diversidad estudiadas en este trabajo.

2.2.1. 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 inco-

rectas, 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.

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

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.

$$\text{disagreement}(m^{(a)}, m^{(b)}) = \frac{|\{M_{i,a} \neq M_{i,b} | s^{(i)} \in D_{valid}^{(*)}\}|}{|D_{valid}^{(*)}|} \quad (2.2)$$

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

$$\text{double-fault}(m^{(a)}, m^{(b)}) = 1 - \frac{|\{M_{i,a} = M_{i,b} = -1 | s^{(i)} \in D_{valid}^{(*)}\}|}{|D_{valid}^{(*)}|} \quad (2.3)$$

La diversidad de los modelos base conocemos que es un factor fundamental en la capacidad de generalización y en general el rendimiento de los ensembles. Los métodos de ensemble, como se verá a continuación son una parte clave de nuestro sistema y de los resultados que este logra. Por tanto, el estudio de la capacidad de estas métricas para producir conjuntos de clasificadores base diversos es de suma importancia. La influencia de estas métricas en los resultados de nuestro sistema será estudiada en detalle en la sección 3.2.1.

2.3. Fase 2: Ensamblado 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 pérdida \mathcal{L} como a las métricas de equidad F_1, \dots, F_n .

El sistema una vez más resuelve un problema de *CASH* como se presenta en la definición 1.1. En lugar de trabajar directamente en $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$, esta vez el sistema trabaja sobre $D^e = \{(y_1^{(*)}, y_1), \dots, (y_n^{(*)}, y_n)\}$, donde $y_i^{(*)} = [y_i^{(0)}, \dots, y_i^{(n)}]$ y cada $y_i^{(j)}$ es la salida de un modelo base j para el ejemplo i , $i \in [1 \dots n]$, $j \in [1 \dots N]$. En otras palabras, al sistema se le pide encontrar las mejores combinaciones de algoritmos y sus hiperparámetros para ensamblar las salidas de los modelos base.

2.3.1. Espacio de búsqueda

Con el propósito de encontrar el ensemble que optimiza las funciones objetivos en cuestión, el sistema utiliza técnicas de AutoML para explorar un espacio de posibles soluciones. Este espacio de búsqueda esta conformado por las diferentes maneras de formar un ensemble a partir de los modelos base. Más específicamente, el espacio de búsqueda se forma a partir de tomar decisiones sobre una serie de hiperparámetros. Estos hiperparámetros son, la cantidad de modelos base que serán escogidos para formar el ensemble, el subconjunto de modelos base a partir de la cantidad escogida anteriormente y por ultimo el tipo de ensemble que será utilizada. Los diferentes tipos de ensemble se describen con mas detalle 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_{train}^e . 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_{train}^e para optimizar \mathcal{L} . La arquitectura del modelo de aprendizaje automático es tomado del conjunto de algoritmos disponibles por defecto a AutoGOAL.

2.3.2. Encontrando modelos justos y efectivos

Probabilistic Grammatical Evolution (1.1) es un algoritmo que a pesar de ser sumamente sencillo, permite explorar espacios de búsqueda muy variados. Adicionalmente, su utilización en AutoGOAL ha mostrado que puede brindar excelentes resultados en la tarea de encontrar modelos capaces de aprendizaje automático. Sin embargo una de las limitaciones fundamentales de este algoritmo es la estrategia de selección de

los individuos mas aptos dentro población, dado que solamente considera una métrica como objetivo de la optimización. Nuestra propuesta realiza una modificación en este proceso de selección, el cual define la forma en que se explora el espacio, para acomodar más de una métrica en el mismo. Esto nos permitirá explorar el espacio de posibles ensembles e hiperparámetros de los mismos, mientras simultáneamente se dirige la búsqueda en direcciones que optimicen tanto métricas de equidad como de efectividad.

El funcionamiento de nuestro algoritmo consiste fundamentalmente de tres procesos siendo ejecutados continuamente en un ciclo. Primeramente, de forma similar a *PGE*, un conjunto de N flujos es generado a partir de tomar muestras de una gramática G de acuerdo a las probabilidades θ asignadas a cada producción. Seguido de esto, cada flujo generado es evaluado en cada una de las funciones objetivos a optimizar y se procede a un proceso de selección en el cual se determinan los mejores candidatos en la población para pasar a la siguiente generación. Este proceso de selección es el que guía las direcciones en las que se explora el espacio de búsqueda, por tanto necesita tener en cuenta la evaluación de cada una de las funciones objetivo. Luego, dicho proceso 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*, los N individuos de la población son seleccionados según el orden que se obtiene de aplicar *Non-dominated Sorting* y empleando *Crowding Distance* como métrica para desambiguar entre individuos con el mismo *pareto-orden*. Además, se mantiene en todo momento durante la ejecución del algoritmo la mejor aproximación del *Frente Pareto* encontrada hasta ese instante. Para esto actualizamos al final de cada iteración este conjunto aproximación con las soluciones **no** *pareto-dominadas* de la unión de aquellas que ya pertenecían al conjunto y las nuevas soluciones encontradas. Finalmente, como parte de la ejecución tradicional de *PGE*, se actualiza el modelo probabilístico de la gramática a partir de las soluciones que pasan el proceso de selección. La propuesta anteriormente descrita se resume en el algoritmo 2.1.

Esta propuesta tiene su motivación en tratar de aprovechar las ventajas que brindan individualmente *Probabilistic Grammatical Evolution* y *NSGA-II*. *PGE* es extremadamente versátil respecto al tipo de espacio que permite explorar gracias a permite explorar cualquier espacio que pueda ser capturado por una gramática. Al mismo tiempo *PGE* es muy flexible y no impone restricciones algunas sobre la forma en que se realiza el proceso de selección de los individuos que continúan de una generación a otra. *NSGA-II* por otro lado ha mostrado ser muy efectivo en converger a buenas aproximaciones del *Frente Pareto* para problemas de optimización multi-objetivo. Se considera entonces que una combinación de estos algoritmos puede ser capaz de explorar el espacio de búsqueda de la tarea en cuestión, de forma tal que la estrategia de exploración induzca a cada vez mejores aproximaciones del *Frente Pareto*, a partir de utilizar el método de selección de *NSGA-II*.

Algoritmo 2.1: NSPGE(D, G, \mathcal{F})

```

1 set  $\sigma \leftarrow \text{uniform-init}(G)$ 
2 set models  $\leftarrow \emptyset$ 
3 para generation  $\in \text{sample}(G, \sigma)$  hacer
4   set scores  $\leftarrow \emptyset$ 
5   para  $\mathcal{L}^{(i)} \in \mathcal{F}$  hacer
6     para  $A_\lambda^{(j)} \in \text{generation}$  hacer
7       scores $^{(i,j)} \leftarrow \mathcal{L}^{(i)}(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$ 
8     fin
9   fin
10  set indices  $\leftarrow \text{non-dominated-sort}(\text{generation}, \text{scores})$ 
11  set updates  $\leftarrow \text{select-best-parameters}(\text{generation}, \text{scores}, \text{indices})$ 
12
13   $\sigma \leftarrow \text{update-probabilities}(\sigma, \text{updates})$ 
14  models  $\leftarrow \text{select}(\text{generation}, \text{indices})$ 
15 fin
16 devolver models

```

Una de las ventajas de nuestra propuesta es que mantiene en todo momento un conjunto de soluciones que aproximan el *Frente Pareto*. Esto significa que al terminar la ejecución, el sistema no da una única solución al usuario. Dar una única solución es problemático debido a que la naturaleza de estos problemas de optimización multi-objetivo implica que existe un conjunto de soluciones entre las cuales no hay manera general de determinar un orden. Cada una de dichas soluciones representa un balance diferente entre las diferentes métricas a optimizar. Nuestro sistema le provee al usuario un conjunto de soluciones óptimas entre sí, y el usuario puede posteriormente seleccionar la solución que mejor considere de acuerdo al problema en cuestión que se esta resolviendo y las restricciones del mismo. Por ejemplo, en determinados casos puede ser tolerable ceder en la justeza del modelo con el objetivo de ganar precisión, mientras que otros escenarios mas sensibles puede que la restricción sobre la equidad del modelo sea primordial y no pueda ser comprometida. Luego, el usuario tiene a su disposición una serie de soluciones que representan diferentes balances entre los objetivos, cada una útil en escenarios con diferentes características, todo esto luego de una única ejecución de extremo a extremo de nuestro sistema.

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 constraints, 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 dividida 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 de los 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.

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

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.

$$O_{sobreajustado}^{(i)}(M) = \mathbf{max_count} \left(\left\{ y_k \mid (x_k, y_k) \in D, \forall_{m^{(j)} \in M} y_k^{(j)} = y_i^{(j)} \right\} \right)$$

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 pérdida 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* (9). 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 pérdida 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, será 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* (17) 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, dígame *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* (9). 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 gracioso	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* (17) presenta información extraída del censo de 1994 en los Estados Unidos por Barry Becker. Los datos contienen detalles personales de los individuos, 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 10000 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 (21). 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

¹<https://github.com/bfair-ml/bfair>

entrenar modelos de *Aprendizaje Automático* que en datos que muestran sesgos de los humanos.

Aquí quizás 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 función maquina 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 2.2.1, 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 2.3); 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.

Tabla 3.2: HABA 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: HABA 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: HABA 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: HABA 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 reelección 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 reelección 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 reelección. 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 reelecciona 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, esto 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, dígame *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 reelección 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 (21), la cual utiliza algoritmos de aprendizaje profundo - una estrategia de pre-procesamiento utilizando BERT (12) 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 vectorización (*TfidfVectorizer*, *CountVectorizer*, etc.) seguidos de una capa de clasificación (*Perceptron*, *LinearSVC*, *Nea-*

restCentroid, *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 reelección 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 prometido-

res 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

Tabla 3.11: Evaluación en *Adult*, con restricción en las tres medidas de equidad. El máximo número de clasificadores utilizado es 20. La estrategia de selección de modelos base es *double-fault*.

método	precisión	statistical parity	equal opportunity	equalized odds
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

Tabla 3.12: Evaluación en *Adult*, con restricción sobre *statistical parity* como métrica de equidad. El máximo número de clasificadores utilizado es 20. La estrategia de selección de modelos base es *double-fault*.

Método	Error	statistical parity
FERM	$0,164 \pm 0,010$	$\leq 0,1$
Zafar	$0,187 \pm 0,001$	$\leq 0,1$
Adversarial	$0,237 \pm 0,001$	$\leq 0,1$
FERM preprocess	$0,228 \pm 0,013$	$\leq 0,1$
SMOTE	$0,178 \pm 0,005$	$\leq 0,1$
FairBO	$0,175 \pm 0,007$	$\leq 0,1$
BFair (20)	0,170	0,087

Tabla 3.13: Evaluación en *Adult*, con restricción sobre *statistical parity* como métrica de equidad. El máximo número de clasificadores utilizado aparece entre paréntesis. La estrategia de selección de modelos base es *double-fault*.

Método	Precisión	statistical parity
ParEGO	$\approx 0,764$	$\approx 0,0175$
Dragonfly	$\approx 0,763$	$\approx 0,0188$
BFair (20)	0,813	0,0487
BFair (20)	0,803	0,0332
BFair (50)	0,800	0,0269
BFair (50)	0,777	0,0072
BFair (50)	0,777	0,0072
BFair (20)	0,764	0,0000

3.3.2. Discusión

Conclusiones

Conclusiones

Recomendaciones

Recomendaciones

Bibliografía

- [1] Alekh Agarwal, Miroslav Dudík y Zhiwei Steven Wu. «Fair regression: Quantitative definitions and reduction-based algorithms». En: *International Conference on Machine Learning*. PMLR. 2019, págs. 120-129 (vid. pág. 5).
- [2] Alekh Agarwal y col. «A reductions approach to fair classification». En: *International Conference on Machine Learning*. PMLR. 2018, págs. 60-69 (vid. pág. 5).
- [3] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996 (vid. pág. 12).
- [4] Tolga Bolukbasi y col. «Man is to computer programmer as woman is to homemaker? debiasing word embeddings». En: *Advances in neural information processing systems* 29 (2016) (vid. pág. 3).
- [5] Leo Breiman. «Bagging predictors». En: *Machine learning* 24.2 (1996), págs. 123-140 (vid. pág. 7).
- [6] Flavio Calmon y col. «Optimized Pre-Processing for Discrimination Prevention». En: *Advances in Neural Information Processing Systems*. Ed. por I. Guyon y col. Vol. 30. Curran Associates, Inc., 2017. URL: <https://proceedings.neurips.cc/paper/2017/file/9a49a25d845a483fae4be7e341368e36-Paper.pdf> (vid. pág. 5).
- [7] Nitesh V Chawla y col. «SMOTE: synthetic minority over-sampling technique». En: *Journal of artificial intelligence research* 16 (2002), págs. 321-357 (vid. pág. 6).
- [8] Silvia Chiappa y William S Isaac. «A causal Bayesian networks viewpoint on fairness». En: *IFIP International Summer School on Privacy and Identity Management*. Springer. 2018, págs. 3-20 (vid. pág. 6).
- [9] Luis Chiruzzo y col. «Overview of HABA at IberLEF 2019: Humor analysis based on human annotation.» En: *IberLEF@ SEPLN*. 2019, págs. 132-144 (vid. págs. 25, 26).

- [10] Nihad Karim Chowdhury y col. «Machine learning for detecting COVID-19 from cough sounds: An ensemble-based MCDM method». En: *Computers in Biology and Medicine* 145 (2022), pág. 105405. ISSN: 0010-4825. DOI: <https://doi.org/10.1016/j.compbiomed.2022.105405>. URL: <https://www.sciencedirect.com/science/article/pii/S0010482522001974> (vid. pág. 8).
- [11] K. Deb y col. «A fast and elitist multiobjective genetic algorithm: NSGA-II». En: *IEEE Transactions on Evolutionary Computation* 6.2 (abr. de 2002), págs. 182-197. ISSN: 1941-0026. DOI: 10.1109/4235.996017 (vid. pág. 12).
- [12] Jacob Devlin y col. «Bert: Pre-training of deep bidirectional transformers for language understanding». En: *arXiv preprint arXiv:1810.04805* (2018) (vid. pág. 34).
- [13] Thomas G. Dietterich. «Ensemble Methods in Machine Learning». En: *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, págs. 1-15. ISBN: 978-3-540-45014-6 (vid. pág. 7).
- [14] Christos Dimitrakakis y col. «Bayesian Fairness». En: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (jul. de 2019), págs. 509-516. DOI: 10.1609/aaai.v33i01.3301509. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/3824> (vid. págs. 6, 9).
- [15] Michele Donini y col. «Empirical Risk Minimization Under Fairness Constraints». En: *Advances in Neural Information Processing Systems*. Ed. por S. Bengio y col. Vol. 31. Curran Associates, Inc., 2018. URL: <https://proceedings.neurips.cc/paper/2018/file/83cdcec08fbf90370fcf53bdd56604ff-Paper.pdf> (vid. pág. 6).
- [16] Michele Donini y col. «Empirical risk minimization under fairness constraints». En: *Advances in Neural Information Processing Systems* 31 (2018) (vid. pág. 5).
- [17] Dheeru Dua y Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml> (vid. págs. 25, 26).
- [18] Cynthia Dwork y col. «Fairness through awareness». En: *Proceedings of the 3rd innovations in theoretical computer science conference*. 2012, págs. 214-226 (vid. pág. 4).
- [19] Thomas Elsken, Jan Hendrik Metzen y Frank Hutter. «Neural Architecture Search: A Survey». En: (2018). DOI: 10.48550/ARXIV.1808.05377. URL: <https://arxiv.org/abs/1808.05377> (vid. pág. 9).
- [20] Suilan Estévez Velarde y col. «Automatic Discovery of Heterogeneous Machine Learning Pipelines: An Application to Natural Language Processing». En: ene. de 2020, págs. 3558-3568. DOI: 10.18653/v1/2020.coling-main.317 (vid. pág. 9).

- [21] Suilan Estevez-Velarde y col. «Automatic Discovery of Heterogeneous Machine Learning Pipelines: An Application to Natural Language Processing». En: *Proceedings of the 28th International Conference on Computational Linguistics*. 2020, págs. 3558-3568 (vid. págs. 27, 34).
- [22] Suilan Estévez-Velarde y col. «General-purpose hierarchical optimisation of machine learning pipelines with grammatical evolution». En: *Information Sciences* (2020). DOI: 10.1016/j.ins.2020.07.035 (vid. pág. 9).
- [23] Matthias Feurer y col. «Efficient and robust automated machine learning». En: *Advances in neural information processing systems* 28 (2015) (vid. pág. 9).
- [24] Sorelle A Friedler, Carlos Scheidegger y Suresh Venkatasubramanian. «On the (im) possibility of fairness». En: *arXiv preprint arXiv:1609.07236* (2016) (vid. pág. 4).
- [25] Ian Goodfellow y col. «Generative Adversarial Nets». En: *Advances in Neural Information Processing Systems*. Ed. por Z. Ghahramani y col. Vol. 27. Curran Associates, Inc., 2014. URL: <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf> (vid. pág. 7).
- [26] Claus Hillermeier y col. *Nonlinear multiobjective optimization: a generalized homotopy approach*. Vol. 135. Springer Science & Business Media, 2001 (vid. pág. 12).
- [27] Gao Huang y col. «Snapshot Ensembles: Train 1, get M for free». En: *CoRR* abs/1704.00109 (2017). arXiv: 1704.00109. URL: <http://arxiv.org/abs/1704.00109> (vid. pág. 8).
- [28] Haifeng Jin, Qingquan Song y Xia Hu. «Efficient Neural Architecture Search with Network Morphism». En: *CoRR* abs/1806.10282 (2018). arXiv: 1806.10282. URL: <http://arxiv.org/abs/1806.10282> (vid. pág. 9).
- [29] Weiqiu Jin y col. «A data-driven hybrid ensemble AI model for COVID-19 infection forecast using multiple neural networks and reinforced learning». En: *Computers in Biology and Medicine* 146 (2022), pág. 105560. ISSN: 0010-4825. DOI: <https://doi.org/10.1016/j.combiomed.2022.105560>. URL: <https://www.sciencedirect.com/science/article/pii/S0010482522003523> (vid. pág. 8).
- [30] Donald R. Jones, Matthias Schonlau y William J. Welch. «Efficient Global Optimization of Expensive Black-Box Functions». En: *Journal of Global Optimization* 13.4 (dic. de 1998), págs. 455-492. ISSN: 1573-2916. DOI: 10.1023/A:1008306431147. URL: <https://doi.org/10.1023/A:1008306431147> (vid. pág. 12).

- [31] Faisal Kamiran y Toon Calders. «Data preprocessing techniques for classification without discrimination». En: *Knowledge and Information Systems* 33 (2011), págs. 1-33 (vid. pág. 5).
- [32] Michael Kearns y col. «Preventing fairness gerrymandering: Auditing and learning for subgroup fairness». En: *International Conference on Machine Learning*. PMLR. 2018, págs. 2564-2572 (vid. págs. 5, 6).
- [33] *Keras*. URL: <https://github.com/fchollet/keras> (vid. págs. 9, 10).
- [34] Jon Kleinberg. «Inherent trade-offs in algorithmic fairness». En: *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*. 2018, págs. 40-40 (vid. pág. 4).
- [35] J. Knowles. «ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems». En: *IEEE Transactions on Evolutionary Computation* 10.1 (2006), págs. 50-66. DOI: 10.1109/TEVC.2005.851274 (vid. pág. 12).
- [36] Harold W Kuhn y Albert W Tucker. «Nonlinear programming». En: *Traces and emergence of nonlinear programming*. Springer, 2014, págs. 247-258 (vid. pág. 12).
- [37] Ioannis E Livieris y col. «Ensemble deep learning models for forecasting cryptocurrency time-series». En: *Algorithms* 13.5 (2020), pág. 121 (vid. pág. 8).
- [38] Edward Loper y Steven Bird. «NLTK: The Natural Language Toolkit». En: *CoRR* cs.CL/0205028 (2002). URL: <https://arxiv.org/abs/cs/0205028> (vid. pág. 10).
- [39] Mark MacCarthy. «Standards of Fairness for Disparate Impact Assessment of Big Data Algorithms». En: *Other Information Systems & eBusiness eJournal* (2018) (vid. pág. 5).
- [40] Jessica Mégane, Nuno Lourenço y Penousal Machado. «Probabilistic grammatical evolution». En: *European Conference on Genetic Programming (Part of EvoStar)*. Springer. 2021, págs. 198-213 (vid. pág. 10).
- [41] Kaisa Miettinen. *Nonlinear multiobjective optimization*. Vol. 12. Springer Science & Business Media, 2012 (vid. pág. 11).
- [42] Biswajit Paria, Kirthevasan Kandasamy y Barnabás Póczos. «A flexible framework for multi-objective bayesian optimization using random scalarizations». En: *Uncertainty in Artificial Intelligence*. PMLR. 2020, págs. 766-776 (vid. pág. 12).
- [43] Adam Paszke y col. «Pytorch: An imperative style, high-performance deep learning library». En: *Advances in neural information processing systems* 32 (2019) (vid. pág. 10).

- [44] Fabian Pedregosa y col. «Scikit-learn: Machine learning in Python». En: *the Journal of machine Learning research* 12 (2011), págs. 2825-2830 (vid. págs. 9, 10).
- [45] Valerio Perrone y col. «Fair Bayesian Optimization». En: *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*. AIES '21. Virtual Event, USA: Association for Computing Machinery, 2021, págs. 854-863. ISBN: 9781450384735. DOI: 10.1145/3461702.3462629. URL: <https://doi.org/10.1145/3461702.3462629> (vid. pág. 6).
- [46] Robi Polikar. «Ensemble based systems in decision making». En: *IEEE Circuits and systems magazine* 6.3 (2006), págs. 21-45 (vid. pág. 7).
- [47] Robert E Schapire. «The strength of weak learnability». En: *Machine learning* 5.2 (1990), págs. 197-227 (vid. pág. 7).
- [48] Oliver Schütze, Alessandro Dell'Aere y Michael Dellnitz. «On Continuation Methods for the Numerical Treatment of Multi-Objective Optimization Problems». En: *Practical Approaches to Multi-Objective Optimization*. Ed. por Jürgen Branke y col. Vol. 4461. Dagstuhl Seminar Proceedings (DagSemProc). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2005, págs. 1-15. DOI: 10.4230/DagSemProc.04461.16. URL: <https://drops.dagstuhl.de/opus/volltexte/2005/349> (vid. pág. 12).
- [49] Philip S Thomas y col. «Preventing undesirable behavior of intelligent machines». En: *Science* 366.6468 (2019), págs. 999-1004 (vid. pág. 6).
- [50] Chris Thornton y col. «Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms». En: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA: Association for Computing Machinery, 2013, págs. 847-855. ISBN: 9781450321747. DOI: 10.1145/2487575.2487629. URL: <https://doi.org/10.1145/2487575.2487629> (vid. pág. 9).
- [51] Sahil Verma y Julia Rubin. «Fairness definitions explained». En: *2018 ieee/acm international workshop on software fairness (fairware)*. IEEE. 2018, págs. 1-7 (vid. pág. 4).
- [52] Ian Witten y col. «The WEKA data mining software: An update». En: *SIGKDD Explorations* 11 (nov. de 2009), págs. 10-18. DOI: 10.1145/1656274.1656278 (vid. pág. 9).
- [53] Muhammad Bilal Zafar y col. «Fairness Constraints: A Flexible Approach for Fair Classification». En: *Journal of Machine Learning Research* 20.75 (2019), págs. 1-42. URL: <http://jmlr.org/papers/v20/18-262.html> (vid. pág. 5).

- [54] Muhammad Bilal Zafar y col. «Fairness constraints: Mechanisms for fair classification». En: *Artificial intelligence and statistics*. PMLR. 2017, págs. 962-970 (vid. págs. 5, 6).
- [55] Rich Zemel y col. «Learning fair representations». En: *International conference on machine learning*. PMLR. 2013, págs. 325-333 (vid. pág. 5).
- [56] Brian Hu Zhang, Blake Lemoine y Margaret Mitchell. «Mitigating unwanted biases with adversarial learning». En: *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. 2018, págs. 335-340 (vid. pág. 7).