Universidad de La Habana
Facultad de Matemática y Computación

Una estrategia de meta-learning para flujos genéricos de AutoML

Autor: **Loraine Monteagudo García**

Tutores:
Msc. Suilan Estéves Velarde
Lic. Daniel Alejandro Valdés Pérez

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

Agradecimientos

Opinión del tutor

Resumen

Índice general

Índice de tablas

Índice de Algoritmos

Introducción

En los últimos tiempos ha habido una explosión en la investigación y aplicación del aprendizaje de máquinas, del inglés *machine learning* (ML). Sin embargo, el rendimiento de muchos métodos de aprendizaje de máquinas es muy sensible a una gran variedad de decisiones, lo que constituye una barrera para nuevos usuarios. Por ejemplo, el científico de datos debe seleccionar entre una amplia gama de posibles algoritmos, incluidas las técnicas de clasificación o regresión (por ejemplo, máquinas de vectores de soporte, redes neuronales, modelos bayesianos, árboles de decisión, etc.) y ajustar numerosos hiperparámetros del algoritmo seleccionado. Además, el rendimiento del modelo también se puede juzgar por varias métricas (por ejemplo, precisión, sensibilidad, especificidad, puntuación F1). Incluso los expertos requieren gran cantidad de recursos y tiempo para crear modelos con buen rendimiento a causa del tedioso proceso de prueba y error que es repetido en cada aplicación para de desarrollar buenos modelos de ML.

Debido a los grandes costos de desarrollo, ha emergido una nueva idea para automatizar el proceso de ML, aprendizaje de máquinas automático, del inglés *automated machine learning* (AutoML). Mientras que los primeros trabajos se centraron en tareas específicas de ML (como el ajuste de hiperparámetros), los estudios recientes buscan automatizar la creación de los pipelines de AutoML de extremo a extremo. El pipeline de AutoML suele consistir de varios procesos: preparación de datos, ingeniería de características, selección del modelo, ajuste de hiperparámetros y evaluación del modelo.

AutoML está diseñado para reducir la demanda de los científicos de datos y permitir a los expertos construir automáticamente aplicaciones de ML sin mucho requerimiento de conocimiento estadístico y de ML. Por lo tanto, AutoML hace accesible enfoques de machine learning a los usuarios no expertos que están interesados en aplicarlos, pero no tienen los recursos para aprender sobre las tecnologías involucradas en detalle. Con el crecimiento exponencial del poder computacional y de los datos digitales, AutoML se ha convertido en un tema de creciente importancia tanto en la industria como en la academia.

0.1. Problema

A pesar de la creciente cantidad de herramientas de AutoML desarrolladas en los últimos años, la generación automática de *machine learning* es computacionalmente costosa y toma mucho tiempo. Las razones de estos defectos incluyen un espacio de búsqueda muy grande, tanto para algoritmos simples de ML como para otras arquitecturas más complejas como redes neuronales, y que la evaluación de incluso un solo pipeline en un dataset grande puede requerir horas. La optimización de los hiperparámetros, que es el núcleo de los sistemas de AutoML, es también un proceso generalmente lento, principalmente al principio al buscar en un espacio de hiperparámetros tan grande como aquellos presentes en un *framework* de ML.

Otro de los fallos de la mayoría de los enfoques existentes de AutoML es su inhabilidad de aprender de los datasets previamente analizados, lo que los fuerza a aprender desde cero con cada nuevo dataset. Esta "pérdida de memoria" hace que las herramientas de AutoML procesen innecesariamente soluciones inválidas. Por ejemplo, no son capaces de preveer que con pocos datos una red neuronal no va a obtener buenos resultados.

0.2. Motivación

En esta tesis se realiza una propuesta para superar estos obstáculos. El principal objetivo del enfoque propuesto es asistir a los profesionales de la industria y a los investigadores en ciencias de datos para la selección de modelos incorporando un componente en los sistemas AutoML. Este componente acelerará la búsqueda de AutoML proveyendo un conjunto inicial de algoritmos y las configuraciones de sus hiperparámetros. Esta recomendación inicial deberá en última instancia dar mejores resultados mediante el análisis previo de problemas relacionados. Por lo tanto, esta característica tiene como objetivo emular el rol del experto en el campo de aprendizaje de máquinas. Con el fin de lograr esto, hacemos uso del concepto de meta-learning.

Meta-learning, o *aprender a aprender*, es la ciencia de observar sistemáticamente cómo se desempeñan los diferentes enfoques de aprendizaje automático en una amplia gama de tareas de aprendizaje, y luego aprender de esta experiencia, o metadatos, para aprender nuevas tareas mucho más rápido de lo que sería posible de otra manera. Esto no solo acelera y mejora drásticamente el diseño de pipelines de aprendizaje automático, sino que también nos permite reemplazar algoritmos diseñados a mano con enfoques novedosos aprendidos de una manera basada en datos.

Mediante meta-learning, intentamos ganar un poco de perspectiva sacada de los meta-datos de experimentos de ML. Los resultados de cada entrenamiento es guardado con caracterizaciones del dataset y sus detalles de rendimiento y son usados en las ejecuciones futuras. Ha habido substancial interés en el espacio de meta-learning en los recientes años y muchos sistemas de AutoML lo han integrado.

0.3. Antecedentes

Esta tesis forma parte de las líneas de investigación del grupo de Inteligencia Artificial de la facultad de Matemática y Computación de la Universidad de La Habana. En dicho grupo se ha llevado a cabo el sistema de AutoGOAL, por lo que esta herramienta es la usada para la incorporación de conocimiento experto mediante meta-learning. AutoGOAL es un sistema AutoML implementado como una biblioteca de código abierto en el lenguaje de programación Python. AutoGOAL utiliza técnicas heterogéneas, que a diferencia de otros sistemas, puede construir automáticamente flujos de aprendizaje automático que combinen técnicas y algoritmos de diferentes bibliotecas, incluidos clasificadores lineales, herramientas de procesamiento de lenguaje natural y redes neuronales.

0.4. Propuesta

meta-learning, el cual está compuesto por dos fases principales: la fase offline que es de aprendizaje y la fase online que es de recomendación. Dado un dataset, una tarea de evaluación (por ejemplo, clasificación o regresión) y una métrica, el algoritmo de meta-learning propuesto produce una lista de ranking de los modelos candidatos. Dicha lista está basada en el rendimiento esperado de los modelos candidatos con respecto a la métrica dada. Esta lista es producida solamente con meta-

conocimiento ganado del análisis de datasets relacionados y el entrenamiento de combinaciones de algoritmos en dichos datasets, sin ejecutar ninguno de los algoritmos candidatos.

- Propuesta, contribuciones No estoy segura si con lo que he hablado de la propuesta es suficiente, o si debería entrar un poco más en detalles... Por otro lado esto creo que va mejor antes de los antecedentes.

0.5. Estructura de la tesis

El resto de la tesis está organizada de la siguiente manera:

- El capítulo 2 (Estado del Arte) introduce los problemas de meta-learning y AutoML y las técnicas que a menudo son aplicadas para lidiar con estos problemas. Esto es seguido por un resumen de los trabajos relacionados de meta-learning para la selección de algoritmos y de distintas herramientas de AutoML.
- El capítulo 3 (Propuesta) describe la estrategia de meta-learning propuesta para AutoML, incluyendo un análisis de los meta-features usados y los meta-modelos desarrollados.
- El capítulo 4 (Experimentación y resultados) describe brevemente los aspectos de la metodología experimental adoptada, investiga los resultados obtenidos y se realiza una discusión de los mismos.
- Finalmente, en el capítulo 5 (conclusiones) se presentan las conclusiones de esta tesis, y se discuten las limitaciones y consideraciones finales. Por último, se realizan sugerencias para estudios futuros.

Capítulo 1

Estado del arte

- ✓ Realizar una introducción sobre el contenido
- ✓ Hablar de la estructura del capítulo.
  - La sección de AutoML se define el problema de AutoML, se da una (o varias) definiciones del concepto de AutoML y se realiza un estudio del mismo desde los diferentes tipos de métodos de optimización usados para la selección de modelos, destacando el uso de meta-learning.
  - En la sección de Meta-Learning se define el problema de meta-learning, se dan definiciones del mismo y se realiza un estudio de meta-learning aplicado al problema de la selección de modelos separados por el tipo de solución que se da para el mismo.

1.1. Meta-Learning

- ✓ Introducir esta sección hablando un poco del concepto de meta-learning y un poco de su historia (referirse a los papers: *Metalearning: A survey of trends and technologies* y *A Comprehensive Overview and Survey of Recent Advances in Meta-Learning*)

El término meta-learning ocurrió por primera vez en el área de psicología educacional. Uno de los investigadores más citados en este campo, John Biggs, describió meta-learning como *ser consciente y tomar el control del conocimiento de uno*. Por lo tanto, meta-learning es visto como un entendimiento y adaptación del aprendizaje en sí en un nivel más alto que simplemente adquirir conocimiento de una materia. De una forma, una persona consciente y capaz de meta-learning es capaz de evaluar su enfoque de aprendizaje de acuerdo a los requerimientos de una tarea en específico.

Meta-learning usada en el contexto de aprendizaje de máquina tiene muchas similitudes a esta descripción. El conocimiento de una matera se traduce en *base-learning*, donde la experiencia es acumulada para una tarea en específico. Meta-learning empieza en un nivel mayor y se encarga de acumular experiencia sobre varias aplicaciones de un sistema de aprendizaje de acuerdo a..[poner link]

En los últimos 20 años, la investigación de aprendizaje de máquinas enfrentó un número creciente de algoritmos disponibles incluyendo multitudes de parametrizaciones, enfoques de pre-procesamiento y post-procesamiento, así como una gran variedad de aplicaciones debido al creciente poder de computación y gran disponibilidad de conjuntos de datos. Promoviendo un mejor entendimiento del aprendizaje de máquinas en sí, meta-learning puede ser de un ayuda invaluable, evitando procedimientos extensivos de prueba y error para la selección de algoritmos. Además, puede permitir

entender mejor que hace a un determinado algoritmo desempeñarse bien en un determinado problema.

1.1.1. Definición

✓ Hablar de distintas definiciones formales del término de meta-learning (referirse a: *Metalearning: A survey of trends and technologies*)

La primera definición de *meta-learning* en el campo de aprendizaje de máquinas fue dado por Jügen Schmidhuber en 1987, el cual lo considera la interacción entre agente y el ambiente impulsando la superación personal en el agente. Un número de definiciones ha sido dada desde entonces, la siguiente lista fue obtenida de:

1. Meta-learning estudia como los sistemas de aprendizaje puede incrementar en eficiencia a través de la experiencia; el objetivo es entender cómo el aprendizaje en sí puede hacerse flexible de acuerdo al dominio o la tarea sujeto de estudio.

2. El principal objetivo de *meta-learning* es entender la interacción entre el mecanismo de aprendizaje y los contextos concretos en los cuales ese mecanismo es aplicable.

3. *Meta-learning* es el estudio de los métodos principales que explotan el meta-conocimiento para obtener modelos eficientes y soluciones adaptando los procesos de aprendizaje de máquinas y minería de datos.

4. *Meta-learning* monitorea el proceso de aprendizaje automático en sí, en el contexto de aprender los problemas que encuentra, e intenta adaptar su comportamiento para desempeñarse mejor.

En la mayoría de las definiciones son conceptos claves sistemas de aprendizaje que se adaptan y mejoran con la experiencia, excepto en la definición 2. Esta enfatiza en una mejor comprensión de la interacción entre dominios y mecanismos de aprendizaje, lo cual no necesariamente implica el objetivo de un sistema de aprendizaje mejorado, sino la búsqueda de un mejor entendimiento de cuales tareas los aprendices tienen éxito o fallan.

Meta-learning es mejor entendido comúnmente como *aprendiendo a aprender*, lo cual se refiere al proceso de mejorar un algoritmo de aprendizaje a través de múltiples episodios de aprendizaje. En contraste, el aprendizaje de máquinas convencional mejora las predicciones del modelo sobre múltiples instancias de datos. Durante el *aprendizaje base*, un algoritmo de aprendizaje interior (o inferior/base) resuelve una *tarea* como clasificación de imágenes, definida por una dataset y un objetivo. Durante *meta-learning*, un algoritmo externo (o superior/meta) actualiza el algoritmo interior de tal forma que el modelo que aprende mejore el objetivo exterior. Los episodios de aprendizaje de la tarea base pueden ser vistos como una forma de proveer las instancias necesitadas por el algoritmo externo para aprender el algoritmo de aprendizaje base.

De esta forma, muchos algoritmos convencionales tales como la búsqueda aleatoria de hiperparámetros mediante validación cruzada podrían caer en la definición de *meta-learning*. La característica destacada del *meta-learning* contemporáneo es un meta-objetivo explícitamente definido, y una optimización de extremo a extremo del algoritmo interior con respecto a este objetivo.

1.1.2. Campos relacionados

✓ Mencionar y resumir algunas áreas relacionadas con meta-learning, para incluir AutoML (referirse a: *Meta-Learning in Neural Networks: A Survey*)

Aquí posicionamos meta-learning contra áreas relacionadas cuya relación con meta-learning es a menudo una fuente de confusión:

Transfer Learning (TL): TL usa experiencia pasada de una tarea fuente para mejorar el aprendizaje (la velocidad, la eficiencia de los datos, la precisión) de una tarea destino. TL se refiere tanto a esta área de problemas como familia de soluciones, mejor conocidas conmúnmente como transferencia de parámetros más ajuste de los mismos opcional. En contraste, meta-learning se refiere al paradigma que puede ser usado para mejorar TL así como otros problemas. En TL el modelo final es extraído por aprendizaje *vainilla* en la tarea fuente sin el uso de un meta-objetivo. En *meta-learning*, el modelo final estaría definido por la optimización externa que evalúa el beneficio del modelo cuando aprende una nueva tarea. De una forma más general, *meta-learning* trata con un rango mucho más grande de meta-representaciones.

Aprendizaje multi-tarea (MTL): intenta aprender en conjunto algunas tareas relacionadas, para beneficiarse de la regularización debido al intercambio de parámetros y la diversidad de la representación compartida resultante. Como TL, MTL convencional es una optimización de un solo nivel sin un meta-objetivo. Además, el propósito de MTL es el de aprender de una cantidad fija de tareas conocidas, mientras que el punto de meta-learning es a menudo aprender de taras futuras no vistas.

Optimización de Hiperparámetros (HO): está dentro de las consideraciones de meta-learning, en el sentido de que hiperparámetros como la tasa de aprendizaje o la fuerza de regularización describen "como aprender". Aquí se suelen incluir tareas de HO que definen un meta-objetivo que es entrenado de extremo a extremo con redes neuronales, tales como el aprendizaje de hiperparámetros basados en gradientes y la búsqueda de arquitectura neuronal. Pero excluyen otros enfoques como búsqueda random y optimización bayesiana, las cuales raramente están consideradas como meta-learning.

AutoML: AutoML es más bien una sombrilla amplia para los enfoques con el objetivo de automatizar partes del proceso de aprendizaje de máquinas que son típicamente manuales, tales como la preparación de los datos, la selección de algoritmos, ajuste de hiperparámetros, y búsqueda de arquitecturas. AutoML a menudo hace uso de numerosas heurísticas afuera del alcance de meta-learning tal y como está definido aquí, y se centra en tareas como limpieza de datos, que son menos centrales a meta-learning. Sin embargo, AutoML a veces hace uso de optimizaciones de extremo a extremo de un meta-objetivo, así que meta-learning puede ser visto como una especialización de AutoML.

1.1.3. Aplicaciones de Meta-Learning para la Selección de Modelos

- ✓ Introducir mencionando varias de las aplicaciones que se mencionan en *Metalearning: A survey of trends and technologies*
- ✓ La idea principal es estructurarlo como en el paper de *Meta-learning: A survey*

Meta-learning puede ser empleada en una variedad de configuraciones, con cierto desacuerdo en la literatura sobre lo que constituye exactamente un problema de aprendizaje. Meta-learning es extremandamente útil en los casos donde el modelo interno es requerido y hay poca cantidad de datos, ya que el modelo interno contiene muchos parámetros que no pueden ser estimados precisamente con pocos datos. Algunas de las aplicaciones comunes son en la investigación robótica, donde se espera que los robots tengan un mayor nivel de autonomía en IA general, en el descubrimiento de drogas

para manejar los datos de altas dimensiones con un tamaño de muestra pequeño y en la traducción de lenguajes raramente usados.

La principal área de investigación de meta-learning estudiada en este trabajo es la selección de algoritmos, la cual ha recibido una considerable cantidad de investigación. En este caso especial de meta-learning, el aspecto de interés es la relación entre las características de los datos y el rendimiento del algoritmo, con el objetivo final de predecir un algoritmo o un conjunto de algoritmos adecuado para un problema específico. Como motivación está el hecho de que es inviable examinar todas las posibles alternativas de algoritmos en un procedimiento de prueba y error. La aplicación de meta-learning en este campo puede por lo tanto ser útil tanto para proveer una recomendación para un usuario final o seleccionar automáticamente o ponderar los algoritmos que son más prometedores.

El desafío en meta-learning para la selección de modelos es aprender de experiencia pasada de una forma sistemática e impulsada por los datos. Primero, es necesario coleccionar los meta-datos que describen las tareas de aprendizaje anteriores y los modelos previamente aprendidos. Ellos comprenden las configuraciones exactas de los algoritmos usados para entrenar los modelos, incluyendo:

- Las configuraciones de los hiperparámetros, composiciones de los pipelines y/o arquitecturas de redes neuronales.
- Las evaluaciones del modelo resultante, tales como la precisión y el tiempo de entrenamiento.
- Los parámetros del modelo aprendidos, tales como los pesos entrenados de una red neuronal.
- Propiedades medibles de la tarea en sí, también conocidas como meta-features.

Luego necesitamos aprender de estos meta-datos previos, para extraer y transferir conocimiento de la búsqueda de los modelos óptimos para nuevas tareas. El resto de esta sección presenta una visión general de diferentes enfoques de meta-learning para hacer esto efectivamente.

Las técnicas de meta-learning basadas en el tipo de meta-datos que ellas aprovechan son categorizadas. Primero, se describe como se aprende solamente de evaluaciones de los modelos y luego se discute cómo caracterizar las tareas para expresar más explícitamente la similaridad entre las tareas y construir meta-modelos para aprender las relaciones entre las características de los datos y aprender el rendimiento.

Aprendiendo de las evaluaciones de modelos

- Falta un poco de itroducción...

✓ Explicar en qué consiste

Consideremos que tenemos acceso a:

- Tareas anteriores $t_j \in T$, siendo esto último el conjunto de todas las tareas conocidas.
- Un conjunto de algoritmos de aprendizaje, completamente definidos por sus configuraciones $\omega_i \in \Omega$; aquí Ω representa un espacio de configuración discreto, continuo o mixto que puede cubrir configuraciones de hiperparámetros, componentes de pipeline y/o componentes de una arquitectura de redes.
- P el conjunto de todas las evaluaciones escalares anteriores $P_{i,j} = P(\omega_i, t_j)$ de una configuración ω_i en una tarea t_j, de acuerdo a una medida de evaluación predefinida, por ejemplo accuracy, y una técnica de evaluación de modelos, por ejemplo validación cruzada.

- P_{new}, el conjunto de todas las evaluaciones conocidas $P_{i,new}$ en una nueva tarea t_{new}

Ahora queremos entrenar un *meta-learner* L que predice las configuraciones recomendadas Ω^*_{new} para una nueva tarea t_{new}. El meta-learner es entrenado en meta-data $P \cup P_{new}$ donde:

- P usualmente es recogido de antemano, o extraído de repositorios de meta-data.
- P_{new} es aprendido por la técnica de meta-lerning en sí de una forma iterativa, algunas veces inicializado con un P'_{new} inicial generado por otro método.

Estas técnicas son usadas generalmente para recomendar configuraciones y espacios de búsqueda útiles, así como transferir conocimiento de tareas *empíricamente* similares. Algunas de estas técnicas son explicadas a continuación.

Recomendaciones independientes de la tarea

✓ Explicar en qué consiste

- Poner ejemplos
  - *Selecting Classification Algorithms with Active Testing*
  - *Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results*
  - *Speeding up algorithm selection using average ranking and active testing by introducing runtime*
  - *Learning hyperparameter optimization initializations*

Supongamos que $P_{new} = 0$, por lo que no tenemos acceso a ninguna evaluación de t_{new}. Aún así se puede aprender una función $f : \Omega x T ß\{\omega^*_k\}, k = 1..K$, resultando en un conjunto de evaluaciones recomendadas *independientes* de t_{new}. Estos ω^*_k pueden ser evaluados en t_{new} para seleccionar el mejor, o para inicializar otros enfoques de optimización.

Tales enfoques a menudo producen un ranking, es decir, un conjunto ordenado ω^*_k. Esto es típicamente hecho discreteando Ω en un conjunto de configuraciones candidatas ω_i, también llamada *portafolio*, evaluado en un gran número de tareas t_j. Luego podemos construir un ranking por clase, por ejemplo usando *success rates*, *AUC*, o *significant wins*. Sin embargo, a menudo es deseable que algortimos igualmente buenos pero más rápidos estén rankeados mejor, y múltiples métodos han sido propuestos para compensar tiempo de entrenamiento y precisión. Luego podemos agregar estos rankings de una sola tarea en rankings globales, por ejemplo calculando el rank promedio a través de todas las tareas. Cuando hay insuficientes datos para construir un ranking global, uno puede recomendar un subconjunto de configuraciones basados en la configuración mejor conocida para cada tarea anterior, o retornar quasi-lineal rankings.

Para encontrar el mejor ω^* para una tarea t_{new} nunca vista antes, un método simple es seleccionar las K mejores configuraciones, bajando en la lista y evaluando cada configuración en t_{new}. Esta evaluación puede ser terminada después de un valor predefinido para K, un presupuesto de tiempo, o cuando un modelo lo suficientemente preciso es encontrado.

Diseño del espacio de configuración

✓ Explicar en qué consiste

- Poner ejemplos

- Functional ANOVA
- *Hyperparameter Importance Across Datasets*
- *Hyperparameter Search Space Pruning - A New Component for Sequential Model-Based Hyperparameter Optimization*

Las evaluaciones previas también pueden ser usadas para aprender mejores *espacios de configuración* Ω^*. Incluso siendo independientes de t_{new}, esto puede radicalmente acelerar la búsqueda para modelos óptimos, ya que solo las regiones más relevantes de los espacios de configuración son explorados. Esto es crítico cuando los recursos computacionales están limitados.

Transferencia de la configuración

Si se quiere dar recomendaciones para una tarea específica t_{new}, necesitamos información sobre que tan similar t_{new} es a las tareas anteriores t_j. Una manera de hacer esto es evaluando un número de configuraciones recomendadas (o parcialmente aleatorias) en t_{new}, dando lugar a nueva evidencia P_{new}. Si luego observamos que las evaluaciones de $P_{i,new}$ son similares a $P_{i,j}$, entonces t_j y t_{new} pueden ser consideradas similares, basadas en evidencia empírica. Se puede incluir este conocimiento para entrenar un meta-learner que predice un conjunto de configuraciones recomendadas Ω^*_{new} para t_{new}. Además, cada ω^*_{new} seleccionado puede ser evaluado y luego incluido en P_{new}, repitiendo el ciclo y coleccionando más evidencia empírica para aprender cuales tareas son similares entre sí.

Landmarks relativos

✓ Explicar en qué consiste

- Hablar de Active Testing
- Poner ejemplos
- *Pairwise meta rules for better meta learning based algorithm ranking*
- *Toward Automatic Generation of Meta-features*

Una primera medida para la similaridad de las tareas considera las diferencias en el rendimiento relativas (pairwise), también llamadas *relative landmarks*, $RL_{a,b,j} = P_{a,j} - P_{b,j}$ entre dos configuraciones ω_a y ω_b en una tarea particular t_j. (hablar de los papers que tratan de esto)

Modelos sustitutos

✓ Explicar en qué consiste

- Me falta explicar qué es un modelo sustituto
- Poner ejemplos
- *Collaborative hyperparameter tuning*
- *Efficient Transfer Learning Method for Automatic Hyperparameter Tuning*
- *Hyperparameter Optimization with Factorized Multilayer Perceptrons*
- *Scalable Gaussian process-based transfer surrogates for hyperparameter optimization.*

Una forma más flexible de transferir información es construir un *modelo sustituto* $s_j(\omega_i) = P_{i,j}$ para todas las tareas anteriores t_j, entrenadas usando todo el conjunto P disponible. Uno puede definir la similitud entre tareas en términos de error entre $s_j(\omega)$ y $P_{i,new}$: si el modelo sustituto para t_j puede generar predicciones precisas para t_new, entonces esas tareas son intrínsicamente similares. Esto es realizado usualmente en combinación con la optimización Bayesiana para determinar el siguiente ω_i.

Aprendiendo de las propiedades de las tareas

Otra rica fuente de meta-datos son caracterizaciones (meta-features) de la tarea en cuestión. Cada tarea $t_j \in T$ es descrita por un vector $m(t_j) = (m_{j,1}, \ldots, m_{j,K}$ de K meta-features, $m_{j,k}$ es el conjunto de todas las meta-features conocidas. Esto puede ser usado para definir la medida de la similitud de las tareas, por ejemplo, la distancia euclidiana entre $m(t_i)$ y $m(t_j)$, así podemos transferir información de las tareas más similares a la nueva tarea t_{new}. Además, junto con las evaluaciones anteriores P, podemos entrenar un *meta-learner* L para predecir el rendimiento $P_{i,new}$ de las configuraciones ω_i en la nueva tarea t_{new}.

Meta-Features

✓ Poner algunos ejemplos de meta-features dividido por el tipo de meta-feature qué es.

- Hablar de varios papers que definen los distintos tipos de meta-features, puedo además mencionar varios survey de meta-features.
- *Meta-Learning and the Full Model Selection Problem*
- *Pairwise meta rules for better meta learning based algorithm ranking*
- *Selection Classification Algorithms with Active Testing*

Las meta-features caracterizan un dataset y generalmente en la literatura se encuentran divididos en 5 grupos:

Simple: son medidas que son fácilmente extraídas de los datos, son conocidas comúnmente y no requieren recursos computacionales significativos. Representan información básica sobre el dataset. Hasta un determinado punto son concebidas para medir la complejidad del problema subyacente. Algunas de las caracterizaciones incluidas en este grupo son: el número de instancias, el número de atributos, la dimensionalidad del dataset, el radio de valores faltantes, etc. También son llamadas medidas *generales*.

Estadísticas: son medidas que capturan las propiedades estadísticas de los datos. Estas métricas capturan los indicadores de distribución de datos, tales como la media, la desviación estándar, la correlación y curtosis. Solo caracterizan los atributos numéricos. Las caracterizaciones estadísticas son deterministas y algunas de ellas requieren la definición de valores de hiperparámetros, mientras otras pueden generar excepciones.

Teóricas de la información: son medidas del campo de teoría de la información. Estas medidas están basadas en la entropía, la cual captura la cantidad de información en los datos y su complejidad. Ellos pueden ser usados para caracterizar los atributos discretos. Además, son computadas directamente, libres de hiperparámetros, determinísticas y robustas. Semánticamente, describen la variedad y la redundancia de los atributos predictivos para representar las clases.

Basados en modelos: on medidas extraídas de un modelo inducido de los datos de entrenamiento. Las meta-características en este grupo están caracterizadas por la extracción de información de un modelo de aprendizaje de predicción, generalmente, un árbol de decisión. Las medidas caracterizan la complejidad de los problemas basados en las hojas, los nodos y la forma del árbol. Están diseñadas para caracterizar problemas supervisados, todas las medidas son determinísticas, robustas y requieren la definición de los hiperparámetros que hay en el algoritmo de árboles de decisión para inducir el modelo.

***Landmarking*:** son medidas que usan el rendimiento de algoritmos de aprendizaje simples y rápidos para caracterizar los datasets. Los algoritmos deben tener diferentes sesgos y capturar información importante con un costo computacional bajo. Las medidas caracterizan problemas supervisados y son indirectamente extraídos. Requieren la definición de hiperparámetros: el algoritmo de aprendizaje, la medida de evaluación usada para comprobar el rendimiento del modelo y el procedimiento usado para calcularla (por ejemplo, validación cruzada).

Los primeros tres grupos representan los enfoques más comunes y tradicionales de las caracterizaciones de los datos. Los últimos dos requieren el uso de algoritmos de aprendizaje de máquinas, porque extraen la complejidad del modelo o medidas de rendimiento.

Selección Automática de Meta-Features

✓ Introducir la idea de aprender una representación automáticamente de un grupo de tareas

- Ejemplos
- *Automatic classifier selection for non-experts*
- *Towards Automatic Generation of Meta-Features*
- *On the predictive power of meta-features in OpenML*
- *Towards Reproducible Empirical Research in Meta-Learning*
- *A Framework to decompose and develop metafeatures*

En vez de definir manualmente los meta-features, uno puede además aprender una representación conjunta para un grupo de tareas.

Meta-modelos

✓ Explicar en qué consiste

Además, podemos aprender las relaciones complejas entre los meta-features de una tarea y la utilidad de una configuración específica construyendo un meta-modelo L que recomienda la configuración más útil Ω^*_{new} dados los meta-features M de la nueva tarea t_{new}. Existe un gran grupo de trabajo previo construyendo modelos para la selección de algoritmos y recomendación de hiperparámetros.

Ranking

✓ Explicar en qué consiste

- Poner ejemplos de papers

- *KNN*
  - *Selection of Time Series Forecasting Models based on Performance Information*
  - *Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results. (its somewhat old)*
  - *Approximate Ranking Tree Forests (ART Forests)*
  - *Pairwise meta rules for better meta learning based algorithm ranking*
- *Clustering Trees:*
  - *Ranking with Predictive Clustering Trees*
- *XGBoost*
  - *RankML: Meta Learning-Based Approach for Pre-Ranking Machine Learning Pipelines*

Los meta-modelos también pueden generar un *ranking* de las K configuraciones más prometedoras.

Predicción de Rendimiento

✓ Explicar en qué consiste

- Poner ejemplos de papers
- *Predicting the Performance of Learning Algorithms Using Support Vector Machines as Meta-regressors*
- *Selection Classification Algorithms with Active Testing*
- *Automatic classifier selection for non-experts*

Los meta-modelos además pueden predecir directamente el rendimiento, es decir, la predicción o el tiempo de entrenamiento de una configuración en una tarea dada, dada sus meta-features. Esto nos permite estimar si una configuración será lo suficientemente interesante para evaluar cualquier procedimiento de optimización.

Sintesis de Pipeline

✓ Explicar en qué consiste

- Poner ejemplos de papers
- *Towards Automatic Machine Learning Pipeline Design*
- *AlphaD3M*
- En general puedo poner varios ejemplos de sistemas de AutoML

Cuando se crean pipelines enteros de machine learning, el número de opciones de configuración crece dinámicamente, haciendolo incluso más importante para aprovechar la experiencia previa. Uno puede controlar el espacio de búsqueda imponiendo una estructura fija en el pipeline, completamente descrita por sus hiperparámetros. Luego uno puede usar los pipelines más prometedores para inicializar otros procesos de optimización.

Ajustar o no ajustar?

- ✓ Introducir este tema
- Poner ejemplos de papers
- *Using Metalearning to Predict When Parameter Optimization Is Likely to Improve Classification Accuracy*
- *Informing the Use of Hyperparameter Optimization Through Metalearning*

Para reducir el número de configuraciones de parámetros a ser optimizados, y para salvar valioso tiempo de optimización en las configuraciones con restricciones de tiempo. Los meta-modelos también han sido propuestos para predecir si un algoritmo dado vale la pena ser ajustado dados las meta-features de la tarea en cuestión y cuánta mejora puede ser esperada del ajuste de un algoritmo específico contra la inversión de tiempo adicional.

Otras técnicas

- Introducir otras técnicas y poner ejemplos de papers
- Collaborative Filtering
- *Recommending Learning Algorithms and Their Associated Hyperparameters*
- Los ejemplos de sistemas de AutoML que usan CF

Conclusión

- Concluir

1.2. AutoML

- Introducir el tema de AutoML y su próposito

1.2.1. Definición

- Definición de AutoML, quizá desde distintas perspectivas usando diferentes definiciones
- Explicar además varias de las subtareas de AutoML
- *Automated feature enginineering*
- *Automated model selection*
- *Neural architecture search*

1.2.2. Definición del problema

- Definir el problema de automl
- Definir el problema de optimización de hiperparámetros (HPO)
- Definir el problema de *Combined Algorithm Selection and Hyperparameter tuning* (CASH)
- Definir el workflow de un sistema de AutoML,
- Su entrada es un conjunto de datos, una métrica de optimización (poner ejemplos de métricas de optimización) y (generalmente) restricciones de tiempo
- Su salida es un modelo (pipeline?) de machine learning con sus hiperpárametros
- El sistema de AutoML está fundamentalmente definida por la técnica de optimización usada y la definición del espacio de búsqueda.

1.2.3. Métodos de Optimización

- Introducir esta sección
- A lo largo de esta sección, mi idea es explicar los algoritmos que usan cada una de estas técnicas de optimización a medida que las describo, pero quizá sería un poco más organizado si hago otra subsección poniendo ejemplos de los sistemas de AutoML aparte para además no darle tanta importancia a los algoritmos de optimización (hmmm...)

A diferencia de los parámetros del modelo que se aprenden durante el entrenamiento, el científico de datos establece los hiperparámetros del modelo antes de los aspectos de implementación de control y entrenamiento del modelo. Los hiperparámetros se pueden considerar como configuraciones de modelo. Estos deben ajustarse porque los hiperparámetros ideales para un conjunto de datos no serán los mismos en todos los conjuntos de datos.

Grid Search y Random Search

- ✓ Hablar de la necesidad de Grid Search y Random Search como métodos populares para el ajuste de hiperparámetros y del hecho de que son simples métodos de búsqueda que no hacen ninguna suposición sobre el espacio de búsqueda pero son muy ineficientes.
- ✓ Explicar en lo que consiste Grid Search
- ✓ Explicar en lo que consiste Random Search.
- ✓ Compararlos
- Poner ejemplos de sistemas que usan RS y GS (casi todos los hacen), hablar quizá de que estos métodos están implementados en sklearn

Al ajustar los hiperparámetros de un estimador, *grid search* y *radom search* son métodos populares usados por su simplicidad y por el hecho de que no realizan suposiciones sobre el espacio de búsqueda. Cada configuración en el espacio de búsqueda pueden ser evaluado independientemente. Estos enfoques simples de búsqueda recopilan los comentarios del evaluador simplemente para realizar un seguimiento de las buenas configuraciones. Usualmente es ineficiente porque no explota el conocimiento ganado de evaluaciones pasadas.

Grid Search (GS) es el proceso de discretizar cada HP y evaluar exhaustivamente cada combinación de valores. Los valores numéricos de los hiperparámetros suelen estar espaciados equidistantemente en sus restricciones de cuadro. Para los HP categóricos, se consideran un subconjunto o todos los valores posibles. Es la manera más tradicional de ajustar hiper-parámetros. Para obtener la configuración de hiper-parámetros óptima, *grid search* tiene que enumerar cada configuración posible en el espacio de búsqueda, lo que lo hace muy ineficiente. Por ejemplo, la búsqueda de 20 valores de parámetros diferentes para cada uno de los 4 parámetros requerirá 160.000 ensayos de validación cruzada. Esto equivale a 1.600.000 ajustes de modelo y 1.600.000 predicciones si se utiliza una validación cruzada de 10 veces. extremadamente costosa tanto en potencia como en tiempo. Además la discretización es necesaria cuando el espacio de búsqueda es continuo.

Por otro lado, *random search* (RS) configura una cuadrícula de valores de hiperparámetros y selecciona combinaciones aleatorias para entrenar el modelo y la puntuación. Esto permite controlar explícitamente el número de combinaciones de parámetros que se intentan. El número de iteraciones de búsqueda se pueden establecer en función del tiempo o los recursos.

RS a menudo tiene un rendimiento mucho mejor que GS en entornos de HPO de mayor dimensión. GS sufre directamente de la maldición de la dimensionalidad, ya que el número requerido de evaluaciones aumenta exponencialmente con el número de hiperparámetros para un número de valores de hiper-parámetros fijo. Esto también parece ser cierto para RS a primera vista, y ciertamente necesitamos un número exponencial de puntos en $dim(\Lambda)$ para cubrir bien el espacio. Pero en la práctica, los problemas de HPO a menudo tienen una dimensionalidad efectiva baja: el conjunto de hiperparámetros que influyen en el rendimiento suele ser un pequeño subconjunto de todos los hiperparámetros disponibles. Otra ventaja de RS es que se puede ampliar fácilmente con más muestras; por el contrario, el número de puntos en una cuadrícula debe especificarse de antemano y refinar la resolución de GS posteriormente es más complicado. Todo esto hace que RS sea preferible a GS y una base sorprendentemente sólida para HPO en muchos entornos prácticos. Tanto GS como RS se utilizan e implementan ampliamente en casi todos los marcos de software de ML.

Optimización Bayesiana

- ✓ Introducir quizá diciendo su popularidad como técnica de optimización
- ✓ Hablar de manera general en qué consiste el algoritmo de optimización, hablar de las funciones de adquisición y los modelos sustitutos, especialmente de su forma de equilibrar exploración (la evaluación de tantos conjuntos de hiperparámetros como sea posible) y explotación (asignar más recursos a los hiperparámetros más prometedores)
- ✓ Hablar de distintos tipos de modelos sustitutos y funciones de adquisicion.
- Hablar de los sistemas:
  - Sin meta-learning: AutoWeka, HyperOpt, (*maybe* AutoNet)
  - Con Meta-learning:
  - Meta-features: Auto-Sklearn, SmartML
  - Sin mf: Auto-Sklearn 2.0

La optimización bayesiana (del inglés *Bayesian Optimization*) (BO) se ha vuelto cada vez más popular como técnica de optimización global para funciones costosas de caja negra, y específicamente para HPO.

BO es un algoritmo iterativo, cuya idea clave es modelar el mapeo $\lambda \longmapsto c(\lambda)$ basado en valores de rendimiento observados encontrados en un archivo A mediante regresión (no lineal). Este

modelo aproximado se denomina modelo sustituto, o modelo probabilístico, para el cual es normalmente utilizado un proceso gaussiano o un bosque aleatorio. BO comienza en un archivo A lleno de configuraciones evaluadas. Este archivo es normalmente generado a partir de una configuración de diseño inicial, típicamente muestreada al azar. BO luego usa el archivo para ajustar el modelo sustituto, que para cada λ produce una estimación del rendimiento $\hat{c}(\lambda)$, así como una estimación de la incertidumbre del modelo $\hat{\sigma}(\lambda)$. Sobre la base de estas predicciones, BO establece un función de adquisición $u(\lambda)$ fácil de evaluar, cuyo resultado codifica una compensación entre explotación (el modelo sustituto predice que el candidato λ tiene un valor c bueno y bajo) y exploración (el modelo sustituto es muy incierto acerca de la $c(\lambda)$, probablemente porque el área circundante no ha sido exploradam minuciosamente).

La elección del modelo sustituto tiene una gran influencia en el rendimiento de BO y a menudo está relacionado con las propiedades del espacio de búsqueda $\hat{\Lambda}$.

Si $\hat{\Lambda}$ es puramente numérico, los procesos guasianos (GP), son los usados más a menudo. Extensiones para los espacios jerárquicos mezclados que son basados en kernels especiales existen, y el uso de embeddings aleatorios ha sido sugerido para los espacios de grandes dimensiones. Más importante, GPs tiene una complejidad de tiempo de ejecución que es cúbica en el numero de ejemplos, lo que puede resultar en una sobrecarga significativa cuando el archivo A se vuelve grande.

Los bosques aleatorios, son más notablemente utilizados en SMAC (Hutter et al., 2011), también han mostrado buenos rendimientos como modelos sustitutos para BO. Su ventaja es su habilidad nativa para manejar HP discretos y, con modificaciones menores, incluso los HP dependientes sin la necesidad de preprocesamiento. Las implementaciones estándar de bosque aleatorio todavía pueden manejar HP dependientes al tratar los valores de HP no factibles como faltantes y realizar la imputación. Los bosques aleatorios tienden a funcionar bien con archivos más grandes e introducen menos gastos generales que los GPs. SMAC usa la desviación estándar de las predicciones de los árboles como una estimación de incertidumbre heurística $\hat{\sigma}(\lambda)$, sin embargo existen alternativas más sofisticadas para proporcionar estimaciones no sesgadas. Dado que los árboles no son modelos espaciales basados en la distancia, el estimador de incertidumbre no aumenta cuanto más extrapolamos de los puntos de entrenamiento observados. Este podría ser una explicación de por qué GP supera a los sustitutos basados en árboles en la búsqueda puramente numérica espacios.

Las redes neuronales han mostrado un buen rendimiento en particular con espacios de entrada no triviales, por lo que se consideran cada vez más como modelos sustitutos de BO. Los NN ofrecen eficientes y versátiles implementaciones que permiten el uso de gradientes para una optimización más eficiente de la función de adquisición. Los límites de incertidumbre en las predicciones se pueden obtener, por ejemplo, utilizando Redes Neuronales Bayesianas (BNN) que combinan NNs con un modelo probabilístico de los pesos de la red o regresión de base adaptativa donde solo se agrega un regresor lineal bayesiano a la última capa de la NN.

La función de adquisición equilibra la predicción del modelo sustituto $\hat{c}(\lambda)$ y su incertidumbre posterior $\hat{\sigma}(\lambda)$ para asegurar tanto la exploración de las regiones inexploradas de $\hat{\Lambda}$, así como la explotación de las regiones que han tenido un buen rendimiento en las evaluaciones previas.

Diferentes funciones de adquisición han sido diseñadas para la optimización bayesiana: *probability of improvement* (probabilidad de mejora), *entropy search*, *upper confidence bound*, *lower confidence bound*, *expected improvement* (mejora esperada), siendo esta última una de las más populares.

Basada en teoría de probabilidades básica, esta puede ser calculada relativa a la estimación actual del rendimiento óptimo. Suponiendo que nuestra métrica de rendimiento debería ser maximizada (por ejemplo, accuracy, precission, recobrado, etc), que para cualquier ajuste de combinación de hiperparámetro λ tenemos la media predicha y el error estándar de esa métrica ($\mu(\lambda)$ y $\sigma(\lambda)$ respectivamente) y de los datos anteriores el mejor valor (medio) de rendimiento fue m_{opt}. *Expected improvement* es determinado usando:

$$EI(\lambda, m_{opt}) = \delta(\lambda)\Phi\left(\frac{\delta(\lambda)}{\sigma(\lambda)}\right) + \sigma(\lambda)\phi\left(\frac{\delta(\lambda)}{\sigma(\lambda)}\right)$$

Donde $\delta(\lambda) = m_{opt} - \mu(\lambda)$, la función $\Phi(\cdot)$ es la función de distribución acumulativa y $\phi(\cdot)$ es la densidad normal estándar.

El valor de $\delta(\lambda)$ mide cuán cerca estamos (en promedio) al mejor valor actual de rendimiento. Cuando se necesitan nuevos parámetros de ajuste candidatos, el espacio de θ es buscado por el valor que maximiza la mejora esperada.

Algoritmos Evolutivos

- ✓ Introducción de los algoritmos evolutivos
- ✓ Definir conceptos importantes como *individuos*, *población* y sus principales operaciones *mutation* y *crossover*. Explicarlos en la terminología de HPO
- ✓ Hablar un poco de sus desventajas y propiedades.
- Mencionar los distintos paradigmas de los algoritmos evolutivos, para hacer incapié en varios usados en los sistemas de AutoML: GGA, GA, PGE
  - Al explicar estos algoritmos concretos, hablar de los distintos sistemas de AutoML
  - GA: TPOT. No tiene mtl, pero hay trabajos incorporándolo, explicarlos. Autostacker
  - GGA: Auto-MEKA_GPP, RECIPE (sin mtl)
  - PGE: Auto-GOAL sin mtl

Un algoritmo evolutivo (EA) es un subconjunto de la computación evolutiva, un algoritmo genérico de optimización metaheurística basado en la población. Un EA utiliza mecanismos inspirados en la evolución biológica, como la reproducción, la mutación, la recombinación y la selección. Las soluciones candidatas al problema de optimización juegan el papel de individuos en una población, y la función de aptitud determina la calidad de las soluciones. La evolución de la población tiene lugar luego de la aplicación repetida de los operadores anteriores.

En un algoritmo evolutivo, una *población* de soluciones candidatas (llamadas individuos, criaturas o fenotipos) a un problema de optimización evoluciona hacia mejores soluciones. Cada solución candidata tiene un conjunto de propiedades (sus cromosomas o genotipo) que se pueden mutar y alterar.

La evolución generalmente comienza a partir de una población de individuos generados aleatoriamente y es un proceso iterativo, con la población en cada iteración llamada *generación*. En cada generación, se evalúa la aptitud de cada individuo de la población; la aptitud suele ser el valor de la función objetivo en el problema de optimización que se resuelve. Los individuos más aptos se seleccionan estocásticamente de la población actual, y el genoma de cada individuo se modifica (recombina y posiblemente muta al azar) para formar una nueva generación. La nueva generación de soluciones candidatas se utiliza luego en la siguiente iteración del algoritmo. Por lo general, el algoritmo termina cuando se ha producido un número máximo de generaciones o se ha alcanzado un nivel de aptitud satisfactorio para la población.

En la terminología de la optimización de hiperparámetros un *individuo* es un HPC único, la *población* es un conjunto de HPC actualmente mantenido y la *aptitud* de un individuo es su error de generalización (invertido) $c(\lambda)$. La mutación es el cambio (aleatorio) de uno o unos pocos valores de hiperparámetros en una configuración. El cruce crea un nuevo HPC mezclando (aleatoriamente) los valores de otras dos configuraciones.

Los EA estaban limitados a espacios numéricos en su formulación original, pero pueden extenderse fácilmente para manejar espacios mixtos tratando componentes de diferentes tipos de forma independiente, por ejemplo, agregando un valor aleatorio normalmente distribuido a HP de valor real mientras se agrega la diferencia de dos geométricamente valores distribuidos a HP con valores enteros (R. Li et al., 2013). Al definir operaciones de mutación y cruce que operan en estructuras de árbol o gráficos, incluso es posible realizar la optimización de tuberías de preprocesamiento (Olson et al., 2016; Escalante et al., 2009) o arquitecturas de redes neuronales (Real et al., 2019) utilizando algoritmos evolutivos.

Los algoritmos evolutivos a menudo funcionan bien al aproximar soluciones a todo tipo de problemas porque, idealmente, no hacen ninguna suposición sobre la función de aptitud subyacente. En la mayoría de las aplicaciones reales de EA, la complejidad computacional es un factor prohibitivo. De hecho, esta complejidad computacional se debe a la evaluación de la función de aptitud. La aproximación al fitness es una de las soluciones para superar esta dificultad.

Multifidelity Optimization

- ✓ Introducir este concepto mediante su necesidad de uso para las redes neuronales.
- ✓ Explicar de forma concreta en que consiste el método de optimización.
- ▪ Hablar de AutoPytorch como ejemplo de sistema de AutoML (usa mtl sin meta-features)

La optimización bayesiana es popular para optimizar los objetivos de caja negra que consumen mucho tiempo. No obstante, para el ajuste de hiperparámetros en redes neuronales profundas, el tiempo necesario para evaluar el error de validación incluso para unos pocos ajustes de hiperparámetros sigue siendo muy costosa. La optimización de multifidelity promete alivio al usar proxies más baratos para tales objetivos, por ejemplo, error de validación para una red entrenada usando un subconjunto de puntos de entrenamiento o menos iteraciones de las requeridas para la convergencia. (poner referencia del paper de *multifidelity optimization*)

El concepto de *Multifidelity* en HPO se refiere a todos los enfoques de ajuste que pueden manejar de manera eficiente a un learner $I(D, \lambda)$ con un hiperparámetro de presupuesto λ_{budget} como un componente de λ que influye en el costo computacional del procedimiento de ajuste de una manera monótonamente creciente. Los valores de λ_{budget} más altos implican un tiempo de ejecución más largo del ajuste. Suponemos conocer las restricciones de λ_{budget} en forma de límite inferior y superior. Por lo general, asumimos que más presupuesto es mejor en términos de rendimiento predictivo (pero, naturalmente, más caro), pero el *overfitting* puede ocurrir en algún momento. Además, suponemos que la relación entre el presupuesto y el rendimiento de la predicción cambia de forma algo suave, por lo que al evaluar múltiples HPC con un presupuesto λ pequeño, esto proporciona al menos una indicación con respecto a su clasificación cuando se evalúa en el presupuesto completo. Normalmente, esto implica un procedimiento de ajuste secuencial, donde λ_{budget} es, por ejemplo, el número de pasos de descenso de gradiente (estocásticos) o el número de miembros del conjunto agregados (de refuerzo) secuencialmente. Otra opción, de aplicación general, es realizar una submuestra de los datos de formación del 0 % al 100 % antes de la formación y tratar esto como control de presupuesto. Los algoritmos HPO que explotan dicho parámetro λ_{budget}, generalmente gastando el presupuesto en HPC baratos antes para la exploración y luego concentrándose en los más prometedores más tarde, se denominan métodos de multifidelity.

Reinforcement Learning

- ✓ Introducir este método como algoritmo de optimización

- ✓ Hablar de la retroalimentación retardada: las retroalimentaciones (la recompensa y el estado) no necesitan ser devueltos inmediantamente una vez que se toma una acción pero una arquitectura solo se puede evaluar después de que se componga toda su estructura
- ✓ Explicar su desventaja de que consume gran cantidad de datos
- Hablar de los sistemas que usan Reinforcement Learning: AlphaD3M (mtl con meta-features)

El aprendizaje por refuerzo (RL) es un marco de optimización muy general y sólido, que puede resolver problemas con retroalimentación retardada. A diferencia de los métodos anteriores, las retroalimentaciones (es decir, la recompensa y el estado) no necesitan ser devueltos inmediatamente una vez que se toma una acción. Se pueden devolver después de realizar una secuencia de acciones.

Sin embargo, el rendimiento de una arquitectura solo se puede evaluar después de que se componga toda su estructura, lo que implica una recompensa retrasada. Por lo tanto, la generación de arquitectura iterativa sigue naturalmente la propiedad de RL.

Debido a las retroalimentaciones retrasadas, AutoML con aprendizaje por refuerzo consume una gran cantidad de datos y es necesario explorar métodos más eficientes. Algunos esfuerzos actuales para abordar estos problemas son el aprendizaje de arquitecturas transferibles a partir de conjuntos de datos más pequeños y la reducción del espacio de búsqueda compartiendo parámetros.

Multi-Armed Bandit (MAB)

- ✓ Introducir el problema de MAB, mencionar que es un caso particular de RL
- ✓ Explicar la solución del problema en el contexto de la optimización de hiperparámetros
- Hablar de los sistemas que usan MAB:
  - ATM usa BO y MAB Híbrido (explicar como cambia el algoritmo). Usa meta-learning
  - Alpine Meadow usa cost-based MAB con BO (explicar como cambia el algoritmo). Usa meta-learning con meta-features

El problema de los *Multi-Armed Bandit* es un problema en el que se debe asignar un conjunto limitado fijo de recursos entre opciones en competencia (alternativas) de una manera que maximice su ganancia esperada, cuando las propiedades de cada opción se conocen solo parcialmente en el momento de la asignación, y pueden entenderse mejor a medida que pasa el tiempo o mediante la asignación de recursos a la elección. Este es un problema clásico de aprendizaje por refuerzo que ejemplifica el dilema de compensación de exploración-explotación.

El problema de MAB puede definirse de la siguiente forma:

Dado un conjunto de acciones $a \in A$ y un presupuesto de tiempo T, en cada ronda $t \in [T]$:

1. Un algoritmo elige un brazo $a \in A$
2. Los algoritmos observan una recompensa del brazo elegido a_t

Dado las distribuciones de recompensa D que son desconocidas e independientes, encontrar el algoritmo que mejor aproxima la solución con la menor pérdida de recompensa.

En el contexto de HPO, cada brazo a_t es asociado con un conjunto de hiperparámetros λ_t y la recompensa (estocástica) por seleccionar (alar) un conjunto de hiperparámetro previo está definido en términos de error en las predicciones.

Monte Carlo Tree Search (MCTS)

- ✓ Introducir el algoritmo
- ✓ Explicar los pasos en los que consite MCTS: selección, expansión, simulación y backpropagation.
- ✓ Mencionar como MCTS extiende el algoritmo de MAB [link de mosaic]
- ✓ Ejemplificar la adaptación del procedimiento estándar de MCTS al problema de HPO
  - Poner ejemplos de sistemas
  - MOSAIC: usa meta-learning con meta-features

Monte Carlo Tree Search (MCTS) es un algoritmo de búsqueda heurística para algunos tipos de procesos de decisión, más notablemente aquellos empleados en software que juegan juegos de mesa. El enfoque de MCTS es en el análisis de los movimientos más prometedores, expandiendo el árbol de búsqueda basado en un muestreo aleatorio del espacio de búsqueda. Cada ronda en MCTS consiste en 4 pasos:

Selección: comienza desde la raíz R y selecciona los nodos secundarios sucesivos hasta que se alcance un nodo hoja L. La raíz es el estado actual del juego y una hoja es cualquier nodo que tenga un hijo potencial a partir del cual aún no se ha iniciado ninguna simulación (reproducción). La siguiente sección dice más sobre una forma de sesgar la elección de los nodos secundarios que permite que el árbol del juego se expanda hacia los movimientos más prometedores, que es la esencia de la búsqueda del árbol de Monte Carlo.

Expansión: a menos que L termine el juego de manera decisiva (por ejemplo, ganar/perder/empatar) para cualquiera de los jugadores, crear uno (o más) nodos secundarios y elija el nodo C de uno de ellos. Los nodos secundarios son cualquier movimiento válido desde la posición del juego definida por L.

Simulación: Completa una reproducción aleatoria del nodo C. Este paso a veces también se denomina *playout* o *rollout*. Un *playout* puede ser tan simple como elegir movimientos aleatorios uniformes hasta que se decida el juego (por ejemplo, en el ajedrez, el juego se gana, se pierde o se empata).

***Backpropagation*:** usa el resultado del *playout* para actualizar la información en los nodos en la ruta de C a R.

MCTS extiende el algoritmo de *Multi-Armed Bandit* (MAB) a espacios de búsqueda estructurados como árboles. Un problema MAB define un problema de RL con un solo estado. MCTS relaja esta limitación de un solo estado y es conocido por ser un enfoque de RL eficiente [poner link de MOSAIC]

Un ejemplo de adaptación del procedimiento estándar de MCTS al problema de HPO es el llevado a cabo por MOSAIC que sigue la siguiente estrategia:

- El camino del árbol del nodo raíz a un node interno representa una solución parcial
- En cada nodo no-terminal, la elección del nodo hijo es realizado usando la regla UCB (*Upper Confidence Bound*) en el caso finito, seleccionando el nodo hijo que maximiza:

$$\mu_i + c\sqrt{\frac{logN}{n_i}}$$

donde μ_i es el valor del nodo i, N es el número de veces que el nodo padre fue visitado, n_i es el número de veces que el nodo i es visitado, y c es la constante dependiente del problema que equilibra exploración y explotación.

- Cuando se alcanza los límites del árbol visitado, una estrategia de *rollout* es aplicada hasta alcanzar un nodo terminal y calcular la recompensa asociada.
- La recompensa asociada al camino completo del árbol (nodo terminal, solución completa m) es calculada.
- Después de la evaluación del nodo terminal, la recompensa es *backpropagated* para actualizar el valor en cada nodo visitado del camino del árbol; este valor apoyará la elección entre los nodos hijos en el siguiente recorrido por el árbol.

Descenso por Gradientes

- ✓ Introducir el método hablando de los problemas de su uso para HPO
- ✓ Explicar cómo se usa en la solución del problema
- Hablar de HyperGD, y poner el ejemplo de Oracle, que es un sistema que usa este método de optimización. Usa meta-learning

Los problemas de HPO suelen ser complejos ya que en general el objetivo no suele ser diferenciable o incluso continuo. Cada HP que está definido por un conjunto discreto introduce superficies no suaves. Además en general el problema de HPO no es convexo. Toda la teoría de convergencia de Gradient Descent (GD) asume, que el problema subyacente es convexo. Por lo tanto, el descenso por gradientes no es tan popular como los métodos anteriores. Sin embargo, centrándose en alguna función de pérdida diferenciable, por ejemplo, *square loss* y *logistic loss*, los hiperparámetros continuos se pueden optimizar mediante el descenso de gradiente.

En comparación con los métodos anteriores, los gradientes ofrecen la información más precisa donde se ubican las mejores configuraciones. A diferencia de los problemas de optimización tradicionales cuyos gradientes se pueden derivar explícitamente a partir del objetivo, en los problemas de AutoML, los gradientes deben calcularse numéricamente.

Por lo general, esto se puede hacer con métodos de diferenciación finita pero con costos elevados. Para algunos métodos tradicionales de aprendizaje automático, por ejemplo, regresión logística y SVM, se propone el gradiente aproximado para buscar hiperparámetros continuos. El cálculo de gradientes exactos se basa en la convergencia del entrenamiento del modelo. A través de un gradiente inexacto, los hiperparámetros se pueden actualizar antes de que converja el entrenamiento del modelo, lo que hace que el método de descenso del gradiente sea más eficiente.

Otra forma de calcular gradientes es mediante el aprendizaje reversible [poner link del artículo sobre reverse learning] (también denominado diferenciación automática). Calcula gradientes con la regla de la cadena, que también se utiliza en el proceso de retropropagación del entrenamiento de la red. Se ha aplicado en la búsqueda de hiperparámetros de aprendizaje profundo.

HyperGD es un optimizador de hiperparámetros basado en gradientes altamente paralelo que realiza una optimización asincrónica en diferentes dimensiones de hiperparámetros en paralelo.

Filtrado Colaborativo

- ✓ Introducir este método, quizá hablando de cómo no es una técnica de optimización pero se usa para la selección de modelos

- ✓ Explicar en que cosiste este método
- ✓ Explicar el método en el contexto de HPO
- Hablar de los sistemas que usan filtrado colaborativo
  - OBOE: Usa meta-learning sin meta-features
  - PMF: Usa CF con BO (explicar cómo se modifica el algoritmo). Usa Meta-Learning sin meta-features
  - Describir de CofiRank, un algoritmo particular de filtrado colaborativo que es una alternativa de MMMF (*Maximum Margin Matrix Factorization*) para hablar de ActivMetaL. Este sistema usa meta-learning sin meta-features

El filtrado colaborativo es un método para realizar predicciones automáticas (filtrado) sobre los intereses de un usuario mediante la recopilación de preferencias o información sobre gustos de muchos usuarios (colaborando). El supuesto subyacente del enfoque de filtrado colaborativo es que si una persona A tiene la misma opinión que una persona B sobre un tema, es más probable que A tenga la opinión de B sobre un tema diferente que la de una persona elegida al azar. Por ejemplo, un sistema de recomendación de filtrado colaborativo para las preferencias en la programación de televisión podría hacer predicciones sobre qué programa de televisión le gustaría a un usuario dada una lista parcial de los gustos de ese usuario (gustos o disgustos). Esto difiere del enfoque más simple de dar una puntuación media (no específica) para cada elemento de interés, por ejemplo, en función de su número de votos.

En HPO podemos realizar una analogía entre los datasets (usuarios) que proveen calificaciones $P_{i,j}$, para las configuraciones de hiperparámetros λ_i, y técnicas de factorización son usadas para predecir los valores desconocidos $P_{i,j}$ y recomendar las mejores configuraciones para cualquier tarea. Un problema importante es el *problema de arranque en frío*, ya que la factorización de matrices requiere al menos algunas evaluaciones en λ_{new}.

Sobre CofiRank

Para la calificación colaborativa, *Maximum Margin Matrix Factorization* (MMMF) ha demostrado ser un medio eficaz para estimar la función de calificación. MMMF aprovecha los efectos de colaboración: los patrones de calificación de otros usuarios se utilizan para estimar las calificaciones del usuario actual. Una ventaja clave de este enfoque es que funciona sin extracción de características. La extracción de características es específica del dominio, por ejemplo, los procedimientos desarrollados para películas no se pueden aplicar a libros. Por lo tanto, es difícil encontrar un conjunto de características consistente en aplicaciones con muchos tipos diferentes de elementos, como por ejemplo en Amazon. El algoritmo CofiRank se basa en esta idea de MMMF, pero optimiza las medidas de clasificación en lugar de las medidas de clasificación. [poner link del artículo sobre CofiRank]

Dado que solo se presentarán al usuario los elementos mejor clasificados, es mucho más importante clasificar correctamente los primeros elementos que los últimos. En otras palabras, es más importante predecir qué al usuario le gusta de lo que no le gusta. En términos más técnicos, el valor del error para la estimación no es uniforme entre las calificaciones.

1.2.4. Conclusión

Concluir

1.3. Conclusión

Concluir el capítulo

Capítulo 2

Propuesta

Capítulo 3

Resultados

Conclusiones

Recomendaciones

Bibliografía