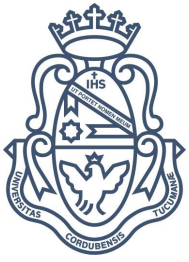
Facultad de Matemática, Astronomía,
Física y Computación

Universidad Nacional de Córdoba



# Framework para Aprendizaje Activo

Tesis

que para obtener el título de

Licenciado en Ciencias de la Computación

presenta

Agustín Daniel Márquez Braconi

Directora: Laura Alonso Alemany

Córdoba, Argentina 2018

# Agradecimientos

Quiero agradecer a todas las personas que de una forma u otra me han ayudado a hacer este sueño realidad.

Principalmente a mi directora de tesis, Laura Alonso, quien me ha guiado y acompañado todos estos meses. De ella he aprendido muchísimo este tiempo, tanto para poder realizar este trabajo, como también para desarrollarme en mi vida profesional.

También a mi familia, quienes siempre me han apoyado a lo largo de todos estos años. Han sufrido y celebrado conmigo cada instancia de este camino y han estado siempre brindando su apoyo en los momentos difíciles.

A FaMAF, esta hermosa facultad, tan única y especial. A todos los profesores, que siempre han estado presentes, y siempre se han hecho del tiempo necesario para enseñarme todo lo que he aprendido en estos años.

Y como no agradecer a mis compañeros, con quienes más he compartido a lo largo de todo este camino, de quienes también he aprendido tantas cosas. En especial a Diego Piloni quien ha sido una pieza clave para mis estudios y para llegar hoy a donde estoy.

¡Muchas gracias a todos!

# Resumen

Muchos proyectos de Aprendizaje Automático (*Machine Learning*) de la actualidad precisan de un gran número de datos etiquetados para poder entrenar los algoritmos. El etiquetado de los mismos tiene un gran costo tanto económico como de tiempo.

Una solución a este problema es el Aprendizaje Activo (*Active Learning*), una forma inteligente de seleccionar qué instancias etiquetar para maximizar el aprendizaje. Para facilitar esta tarea propongo realizar un framework de software que sirva para desplegar proyectos de este tipo.

El framework desarrollado fue puesto a prueba logrando excelentes resultados, mostrando que a partir de un mismo conjunto de datos, si se seleccionan las instancias a etiquetar inteligentemente se puede lograr el rendimiento máximo con una cantidad considerablemente menor de ejemplos.

# Índice general

# Capítulo 1

# Introducción y Motivación

## 1.1. Qué es el aprendizaje Automático Activo?

Aprendizaje Activo es un caso especial de Aprendizaje Automático. El Aprendizaje Automático es un tipo de Inteligencia Artificial que proporciona a las computadoras la capacidad de aprender a realizar una tarea sin ser explícitamente programadas. Esto lo logran analizando grandes cantidades de datos y buscando patrones en los mismos. Los algoritmos de Aprendizaje Automático se pueden clasificar como supervisados, no supervisados y semi supervisados. Los primeros requieren de datos etiquetados para poder entrenarse, esto es, para cada dato de entrenamiento, se requiere la etiqueta esperada que prediga el algoritmo. Mientras los segundos solo requieren de datos, estos aprenden de los mismos haciendo inferencias sobre estos. Los últimos son parte de los dos anteriores, donde se poseen tanto datos etiquetados como no. El Aprendizaje Activo es un caso particular del aprendizaje Semi Supervisado. Esto es una tarea de Aprendizaje Automático de aprender una función que mapea una entrada a una salida. Para obtener dicha función se utilizan dos conjuntos de datos, etiquetados y no etiquetados. El primero de ellos consiste en un conjunto de pares

donde un elemento es la entrada y el otro la salida correspondiente. El conjunto de datos no etiquetados consiste solo del dato de entrada. Usualmente se cuenta con un pequeño conjunto de datos etiquetados y un conjunto mucho mayor de datos no etiquetados.

A diferencia del aprendizaje semi supervisado, en el Aprendizaje Activo el algoritmo participa activamente de su propio entrenamiento, seleccionando de forma inteligente instancias no etiquetadas para ser enviadas a un humano *Oráculo* para obtener su etiqueta. De esta forma se pude lograr aprender esta función con mayor precisión y etiquetando muchos menos ejemplos que con otros métodos, como la selección aleatoria de ejemplos, como afirma (Settles 2011b)

La forma en la que el algoritmo decide sobre que ejemplos consultar al oráculo es el punto clave del Aprendizaje Activo. Para esto existen diferentes métodos que veremos más adelante.

## 1.2. Motivación

Como mencionamos anteriormente, para entrenar algoritmos de Aprendizaje Automático se necesitan grandes cantidades de datos. Hoy en día, gracias a Internet, en muchos casos esta necesidad ya no es un problema. Esto permitió un gran crecimiento en el área incluso en la parte industrial. Pero para etiquetar estos grandes volúmenes de datos se requiere de mucho esfuerzo tanto en términos monetarios como de tiempo. Por lo que se debe conseguir la forma más eficiente de hacer este trabajo, y es aquí donde entra en juego el Aprendizaje Activo. Para grandes empresas conseguir datos de calidad y etiquetados no es un problema mayor, e.g. para Facebook conseguir imágenes etiquetadas con caras de personas. Pero para pequeñas compañías, grupos de investigación o desarrolladores independientes este es un problema real y actual. Con la implementación de este framework estos contarán con una herramienta más que los ayude a mitigar este problema.

El Aprendizaje Activo presenta a su vez mayor complejidad que

el Aprendizaje Automático tradicional. A las tareas de modelado del algoritmo de aprendizaje, se le suma la tarea de implementar un selector, con las decisiones de cuál y cómo. Y tal como sucede con los algoritmos de aprendizaje, cuando uno realiza un proyecto de este tipo no solo prueba con un algoritmo, sino con diferentes opciones, en el caso de los selectores es igual, no se sabe a priori cual tendrá el mejor desempeño para un proyecto dado por lo que se deberá probar con diferentes opciones. Es por no tener este conocimiento y esta tecnología al alcance que en muchos proyectos se termina llevando gran parte del tiempo etiquetar instancias para el entrenamiento.

Como objetivo de esta tesis entonces me propongo a diseñar e implementar un framework que acerque los beneficios del Aprendizaje Activo a usuarios sin conocimiento técnico sobre el mismo y que sirva como punto de partida para proyectos futuros así como para aprender sobre el tema.

Un framework cuenta con las ventajas de tener la flexibilidad suficiente para ser utilizado en diversos proyectos, mientras que a la vez resuelve problemas y decisiones comunes a todos ellos.

## 1.3. Estructura de la tesis

Este documento está organizado de la siguiente forma. En el Capítulo 2 describo algunos trabajos en el área de Aprendizaje Activo en los que me he basado para tomar las decisiones de implementación del framework y sus parámetros. En el Capítulo 4 se delimita el problema, se describe la arquitectura del framework, su flujo de datos y los distintos módulos que se pueden utilizar en diferentes partes (*hotspots*. En el Capítulo 5 comparamos el rendimiento de diferentes parámetros para el núcleo del framework, que son las estrategias de selección de ejemplos no etiquetados, para que los usuarios tengan una referencia a la hora de elegir un valor para este parámetro. Finalmente, concluimos con un resumen de las contribuciones de este trabajo y algunas líneas que quedan abiertas para trabajo futuro.

El framework y todo el código relacionado al mismo se encuentra disponible públicamente en:

`https://github.com/agusdmb/Active-Learning-Framework`

# Capítulo 2

# Aproximaciones al problema

## 2.1. Trabajos Previos

Nos enfocaremos en las decisiones ingenieriles de los siguientes trabajos para determinar la mejor estructura para realizar proyectos de Aprendizaje Activo.

### 2.1.1. DUALIST

Burr Settles (2011a) es un gran referente del Aprendizaje Activo a nivel mundial. Él mismo ha aportado mucho en desarrollo e investigación sobre el área. Uno de sus aportes fue DUALIST, un framework para crear modelos para clasificación de texto basado en Aprendizaje Activo.

A diferencia de otros estudios sobre el área que se han hecho, DUALIST esta pensando para caso reales de uso, y no solo en la teoría detrás de los distintos métodos de Aprendizaje Activo. Por ello tiene una interfaz pensada y optimizada para el usuario etiquetador. El proyecto es open source y se puede encontrar en su repositorio de GitHub. Está implementado mayormente en Javascript y algo de Java.

Una de las características principales de esta herramienta es que no solo se pueden etiquetar instancias, sino también sus características.

Para lograr este objetivo Settles fijó un algoritmo de aprendizaje, Multinomial Naive Bayes, esto es un clasificador probabilístico basado en el teorema de Bayes y en la asumpción de independencia de las variables predictoras. También Settles fija selectores, tanto para instancias como para las características, para el primero se usa un selector basado en entropía, y para el segundo un rankeo basado en ganancia de información. Otra característica que posee es que al momento de etiquetar una instancia, si la misma es ambigua incluso para el usuario etiquetador, esta se puede ignorar, forzando a la herramienta a que no haga otra consulta.

Si bien fue pensado y diseñado para generar modelos de clasificación de texto, se lo ha utilizado para otros fines, como ser: Desambiguación de sentido de palabra, extracción de información, filtrado de twitter y análisis de sentimiento.

Fue diseñado para correr en una única computadora, su algoritmo de entrenamiento corre bien hasta algunos cientos de miles de instancias, pero este sistema no está pensando para escalar.

Aunque se describe en el artículo que se puede cambiar el algoritmo de entrenamiento, en la práctica no parece sencillo si así se quisiera ya que requiere conocimientos técnicos lo cual dificulta realizarlo para alguien que no los tenga.

### 2.1.2. Aprendizaje Activo para clasificación de preguntas

Otro trabajo fuertemente relacionado a este y al ya mencionado framework de Settles, es la tesis de grado de Milagro Teruel (2015). Esta introduce Quepy, un framework desarrollado en Python para transformar preguntas de lenguaje natural a consultas en una base de datos. Para esto, Quepy se basa en plantillas definidas por un usuario programador, por lo que una pregunta es reconocida por Quepy sólo si cumple con el formato de algunas de las plantillas previamente definidas. Esto genera un problema de escalabilidad para el framework, ya

que para aumentar la expresividad del mismo, el número de plantillas debe crecer linealmente, lo cual es muy costoso.

Lo que se propone en esta tesis es entonces ampliar la cobertura de preguntas aceptadas por Quepy sin la necesidad de depender de nuevas plantillas generadas por un técnico. Para esto se propone reconocer preguntas semánticamente equivalentes a otras ya definidas en el sistema a través de un clasificador que dada una pregunta decida a que clase semántica pertenece.

Para implementar esta solución se requiere entrenar un clasificador, pero para esto no se cuenta con datos de entrenamiento. Por este motivo, se anotan manualmente unas preguntas de Quepy y se agregan reformulaciones de las mismas como equivalencias semánticas. Por otro lado, sí existen muchos corpus de preguntas utilizados en otras tareas de clasificación que no están etiquetados. Por lo tanto se decide usar Aprendizaje Activo sobre estos para aumentar los datos de manera inteligente optimizando el tiempo.

Para esto, Teruel se apoya en el trabajo de Settles y en su herramienta de etiquetado DUALIST para entrenar el clasificador etiquetando la menor cantidad de preguntas posibles. Al igual que Settles, usa un algoritmo Multinomial Naive Bayes, y etiqueta tanto instancias como en características. Pero a diferencia de Settles, separa la arquitectura del sistema en tres componentes claves pensando en que estas partes puedan ser independientes y reutilizables. Las mismas son: el algoritmo de entrenamiento (FeatMultinomialNB), el proceso de selección de instancias (ActivePipeline) y el preprocesado de los datos (Preprocess). Al igual que en DUALIST, en este sistema también las instancias a etiquetar se seleccionarán basándonse en la entropía, y las características basándose en la ganancia de información.

Una de las restricciones fuertes de diseño es que los datos de entrenamiento, sobre los cuales se harán las predicciones deben estar ya preprocesados antes de ser leídos por ActivePipeline

### 2.1.3. Information Extraction with Active Learning: A Case Study in Legal Text

Otros campos de estudio y exploración que se han dado en esta área es el paper *Information Extraction with Active Learning: A Case Study in Legal Text* de Cardellino and et. al. (2015). El mismo trata un tema crítico: Licencias sobre los datos y términos de uso y limitaciones que poseen datos publicados en la web como datos enlazados En el mismo, se presenta un estudio donde se aplica Aprendizaje Activo para la extracción de información de estas licencias escritas en lenguaje natural para traducirlas a RDF. La importancia de esto es expresar estas condiciones de uso en un formato que la computadora pueda procesarlas automáticamente para verificarlas. Esto es crucial por que hay que ver detalles muy finos para evitar problemas legales. Por ejemplo, si en la licencia se dice que tal acción es prohibida, y esto no queda reflejado en el documento RDF, puede terminar en un uso indebido de la misma.

Para resolver esta problemática, se basaron en la herramienta de etiquetado DUALIST. Y como este, han usado como modelo de predicción un Multinomial Naive Bayes, no sin antes comparar esta performance con otro modelo más robusto como SVM y viendo si ajustando los parámetros del primero lograban igualar o mejorar la performance del segundo. También han etiquetado instancias y características, aunque a diferencia del trabajo realizado por la gente de DUALIST, en este caso esto se hacía de manera secuencial y no de manera simultanea. Además, en este trabajo una instancia podía tener más de una etiqueta, lo que se conoce como problema multi-etiqueta. Como método de selección se utilizó muestreo por certeza para las instancias y ganancia de información para las características.

La arquitectura de este proyecto describe un loop principal el cual recibe como entrada un corpus anotado, entrena un algoritmo del cual se obtiene un modelo, se etiqueta automáticamente con este modelo, se seleccionan sobre estas instancias etiquetadas automáticamente las que serán enviadas al oráculo, luego este etiqueta manualmente dichas

instancias cuales son enviadas como parte del corpus anotado para empezar una nueva iteración.

Para la evaluación del algoritmo así como del método de selección se realizaron experimentos con oráculos simulados. En estos experimentos se probó anotando de a una instancia a la vez, tres, cinco y hasta diez por iteración. Como dijimos anteriormente, el método de selección que mejor resultado obtuvo fue Certainty, el inverso de Uncertainty que es en algún sentido el estandard en Aprendizaje Activo. Curiosamente Uncertainty se desempeñó peor que elegir las instancias a etiquetar de manera aleatoria.

Esto demuestra la complejidad de Aprendizaje Activo y que cada proyecto es un caso particular.

### 2.1.4. Otros frameworks y librerias

Una de las librerías más robustas en la actualidad para Aprendizaje Activo es Libact. Este es un proyecto open source implementado en Python. El mismo ha alcanzado un nivel de estabilidad suficiente por lo que su desarrollo se ha discontinuado hace ya cerca de dos años. Tiene implementado varios selectores, *"query_strategies"*, populares. También tiene implementado *Active Learning by Learning* el cual emplea diferentes selectores a la vez y los hace competir y va aprendiendo sobre cual es mejor para usar según el tipo de conjunto de datos que se tenga. Al ser este proyecto una librería, no posee una estructura definida, dejando mucho trabajo para el usuario, el cual tendrá que definir el loop principal del mismo. Tampoco posee capacidad para etiquetar features. Este framework posee un "oráculo" para hacer simulaciones con un conjunto de datos completamente etiquetado, y otro que a través de matplotlib muestra una imágen a partir de sus features. Carece de un oráculo por defecto y mayormente un usuario debe implementar el suyo propio. Contiene tres algoritmos de aprendizaje y además una clase para adaptar algoritmos de *scikit learn*.

Otro proyecto más reciente y aún en crecimiento es modAL. mo-

dAL también es open source e implementado en su totalidad en Python3. A diferencia de Libact este tiene una forma de uso mucho más sencilla. Esto lo logra seteando defaults para sus hotspots haciendo posible que un usuario con poco conocimiento técnico logre aplicarlo en su proyecto con pocas líneas de código. Este framework tampoco tiene implementado un oráculo, dejando esta tarea para el usuario programador.

Un framework muy prometedor es Prodigy. A diferencia de los anteriores, este proyecto no es de código abierto y para poder hacer uso del mismo se debe pagar una licencia anual. De los creadores de Spacy. Actualmente posee soporte para algoritmos de entrenamiento de Clasificación de textos y Reconocimiento de entidades nombradas. Tiene un oráculo basado en un anotador que funciona aparte implementado como una interfaz gráfica web muy bien lograda con la cual se pueden etiquetar diversos tipos de proyectos como ser: Anotar instancias, entidades nombradas, part-of-speech, dependencias sintácticas, imágenes. Diseñado para correr en una sola computadora.

### 2.1.5. Conclusión

En todos los casos, estas librerías ofrecen funcionalidades de selección de instancias para etiquetar. Algunos con mayor o menor flexibilidad, otros poniendo más énfasis en las instancias que seleccionan, o en la experiencia del etiquetador.

En cambio, estas librerías no ofrecen la facilidad de anotar las características, solamente las instancias, ya que esa funcionalidad requiere fijar el tipo de algoritmo de aprendizaje, que es justamente uno de los parámetros más flexibles de los frameworks y por lo tanto no pueden ser fijados.

Como hemos visto, la mayor parte de proyectos con Aprendizaje Activo requieren un notable conocimiento técnico, lo cual dificulta su aplicación en proyectos modestos. Parece entonces que hay una necesidad de un Software que facilite más el despliegue de Aprendizaje Activo. Mi propuesta es un framework basado en librerías estándar de

Aprendizaje Automático (`Scikit Learn`) de fácil lectura y manipulación (con Jupyter Notebooks) y con la documentación también en castellano.

# Capítulo 3

# Delimitación del problema

En base a lo visto en el capítulo anterior se propone la implementación de un framework. En este capítulo vamos a delimitar las funcionalidades del framework a modo de requerimientos del software.

## 3.1. Tipos de problema que se cubren

El objetivo de este framework será proveer un punto de partida para problemas de clasificación multi-clase. Queremos cubrir la mayor cantidad de casos posibles. Por este motivo la entrada podrá ser una matriz de características o una lista de instancias. Esto dependerá de qué algoritmo de aprendizaje se utilice, si el mismo es solo un estimador (un algoritmo para predecir) o si se utiliza un pipeline (un algoritmo que primero transforma los datos y luego predice). Muchas veces las características dependen de los datos que se tengan en un momento dado, como por ejemplo, el vocabulario o la frecuencia de las palabras en los documentos. Esto no se puede determinar de ante mano en proyectos donde nuevos datos se obtienen constantemente, como cuando se está haciendo un trabajo sobre tweets que ocurren en tiempo real. De no poder calcularse las características para todos los datos de entrenamiento, la solución es usar un pipeline de preprocesa-

do. Esta solución lleva mucho más tiempo de entrenamiento ya que se debe recalcular las características en cada iteración del loop principal como veremos más adelante. Por este motivo, salvo que el proyecto lo requiera, se recomienda que el procesamiento no sea parte del framework, debiendo primero preprocesar todos los datos, etiquetados y no, y luego instanciar el framework.

Como salida de este framework se podrá obtener un modelo entrenado capaz de predecir sobre datos no etiquetados. A la vez, se podrán obtener las instancias que se han ido etiquetando junto a sus etiquetas, haciendo uso del framework como una herramienta de etiquetado.

## 3.2. Encontrando el punto medio de especificidad y generalidad

Analizaremos los problemas que surgen al momento de determinar que tan específico o genérico queremos que sea el framework, buscando así el balance entre estos dos que mejor se adecúe para captar la mayor cantidad de casos de usos con el menor esfuerzo posible.

### 3.2.1. Problema de demasiado específico

Un framework demasiado específico tiene la ventaja de dejar menos trabajo de implementación al usuario programador. De esta forma se logra que con poco esfuerzo pueda estar aplicando Aprendizaje Activo sobre sus datos. Pero esta ganancia en esfuerzo es consecuencia de decisiones ya tomadas a la hora de implementar el framework. Estas decisiones limitan las posibilidades del usuario programador de adaptar el framework a su caso particular. Por esta razón un framework de estas características sólo será útil y conveniente de utilizar en proyectos puntuales donde coincidan las decisiones tomadas al momento del diseño e implementación con las necesidades del proyecto particular del usuario programador.

Un buen ejemplo de decisión demasiado específica se encuentra en los datos de entrada. Si queremos un framework agnóstico del problema, podemos pensar que las funcionalidades de extracción de características queden como responsabilidad del usuario, porque así el framework puede acomodarse a casos que no fueron previstos en su diseño. Si aplicamos esta visión, el usuario debería proveer el dataset ya vectorizado, es decir una lista de instancias donde cada una a su vez es una lista conteniendo los valores de las catacterísticas.

Sin embargo, esta decisión implica también que el conjunto de características de los datos quedaría fijo desde la inicialización del framework. Esta decisión es inadecuada para varios problemas donde el conjunto de características puede ir aumentando o cambiando. Por ejemplo, en el caso de que se generen instancias con variaciones en el tiempo. Estos casos no se pueden tratar adecuadamente si el formato de los datos de entrada está fijo desde la inicialización. Por esta razón es necesario proveer junto con los datos un método para extraer las características de los mismos y convertirlos en el formato de entrada requerido por el hotspot `Dataset`.

### 3.2.2. Problema de demasiado genérico

Un framework demasiado genérico tendrá la ventaja de adaptarse a más casos de usos. Haciéndolo más atractivo de aprender y utilizar por el hecho de ser más versátil. Pero esta generalización viene acompañada de muchas decisiones de diseño e implementación dejadas en manos del usuario programador. Por este motivo, el usuario, deberá primero tomar estas decisiones antes de poder crear una instancia del framework. Si estas decisiones de diseño y/o implementación requieren de mucho esfuerzo, desalentarán al usuario programador a utilizar el framework en primera instancia, ya que probablemente una solución Ad-hoc conlleve un esfuerzo similar pero funcione mejor.

### 3.2.3. Propuesta

Mi propuesta es desarrollar un framework que permita al usuario programador implementar instancias de proyectos de anotación basados en *Aprendizaje Automático* con el mínimo de programación sin perder generalidad. Esto es, lograr un framework el cual pueda instanciarse con un mínimo de esfuerzo pero facilitando también la posibilidad al usuario programador de adaptar el mismo a su proyecto en particular. Para esto, el framework proveerá un workflow ya establecido y módulos pre-programados que permitan elegir diferentes métodos para los hotspots del mismo. A su vez, se proveerá de interfaces para que el usuario programador pueda crear sus propias instancias de los hotspots para ajustar el framework aun más a su proyecto particular.

Por lo expuesto en el capítulo anterior, este framework debe tener las siguientes componentes:

- un método para seleccionar instancias para ser anotadas por el oráculo. (Ver 4.1.2).
- un algoritmo de aprendizaje, capaz de medir la probabilidad de sus predicciones. (Ver 4.1.2).
- un conjunto de datos anotado, por más mínimo que este sea. (Ver 4.1.2).
- un gran conjunto de datos sin anotar, o la capacidad de obtener constantemente nuevas instancias. (Ver 4.1.2).
- una interfaz para el usuario anotador, pensada y diseñada para facilitar su tarea, evitando así errores humanos. (Ver 4.1.2).

# Capítulo 4

# Diseño y Arquitectura

## 4.1. Diseño del framework propuesto

Se trata de una arquitectura de framework, con frozen spots y hotspots. Los hotspots señalan los puntos parametrizables del framework, donde el usuario programador proveerá las funcionalidades requeridas: `Oracle`, `Selector`, `Model` y `Dataset`.

Las **interfaces** (marcadas con I en la Figura 4.2) son clases abstractas con métodos y/o atributos sin implementar, por lo tanto requieren que se instancie una clase que implemente los métodos requeridos por la interfaz.

### 4.1.1. Flujo del framework

Para la arquitectura del framework se tomó como base la arquitectura desarrollada por Cardellino and et. al. (2015) para su caso de estudio. A partir de su arquitectura se pensó una similar que admita modificaciones para poder adaptarla a otros proyectos.

En el diagrama de flujo (Figura 4.1) podemos distinguir entre *figuras azules* y *rojas* siendo estas los *frozen spots* y *hotspots* respectivamente. El workflow determinado por el framework puede observarse como el loop formado por los frozen spots (figuras azules) en dicha

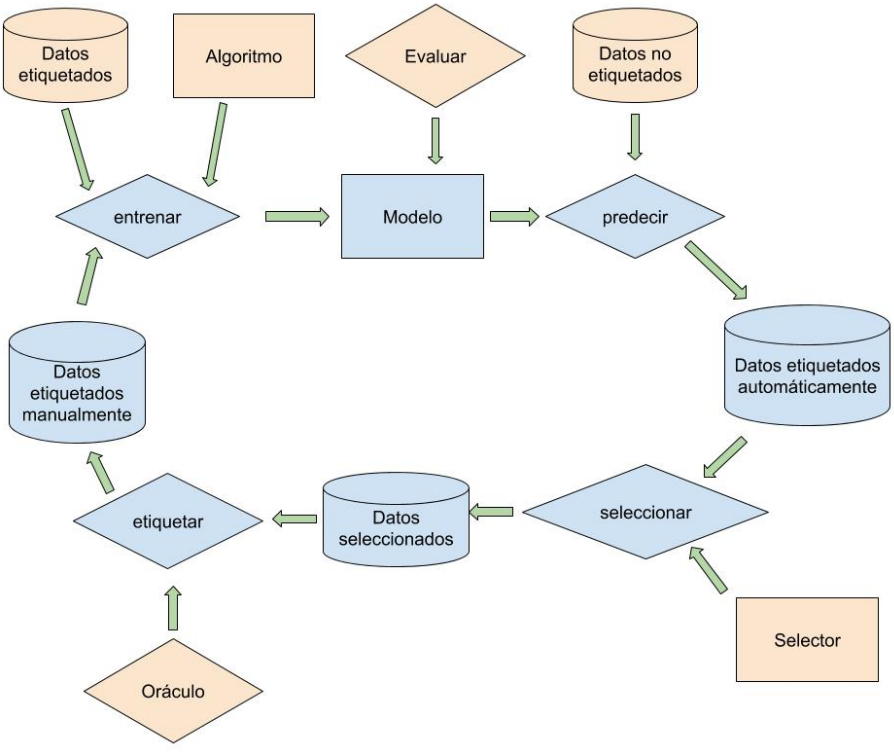



Figura 4.1: Diagrama de flujo del framework propuesto.

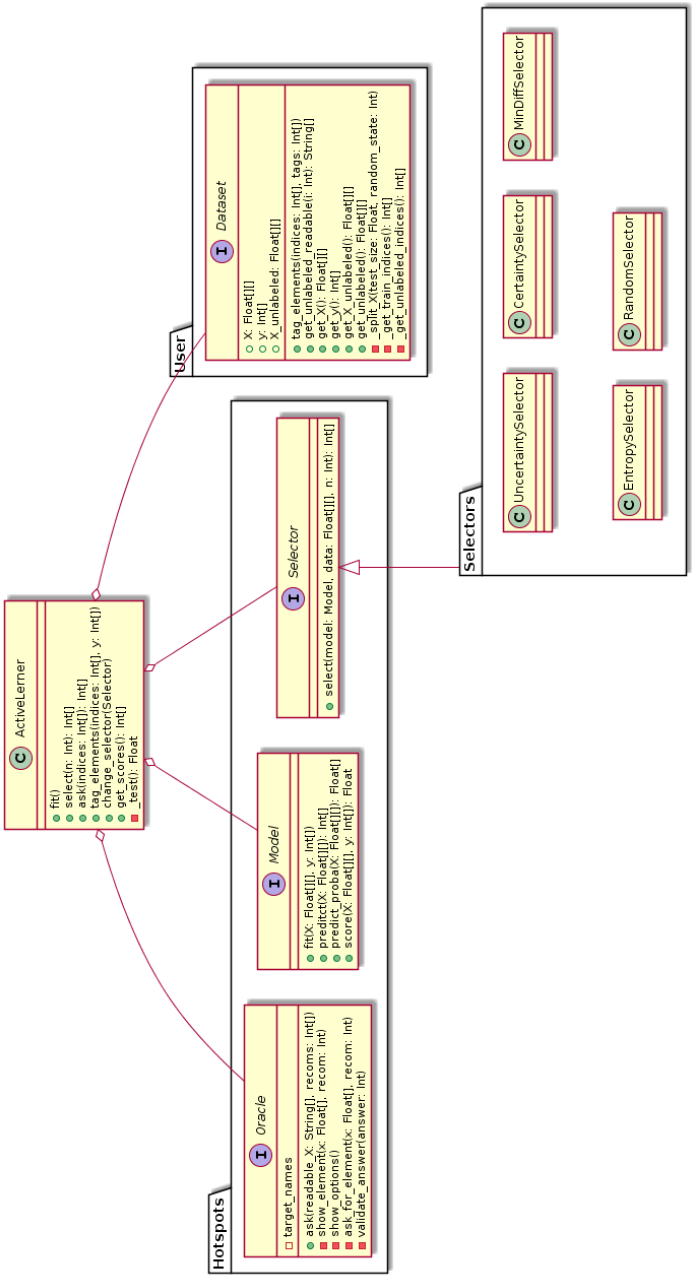



Figura 4.2: Diagrama de clases del framework propuesto.

figura. El mismo comienza entrenando un *modelo* a partir de un *algoritmo* y *datos etiquetados*. El entrenamiento del algoritmo puede incluir el preprocesamiento de los datos de entrenamiento. Esto se consigue pasando en lugar de un estimador en este hotspot, un pipeline con el preprocesamiento y el estimador al final del mismo. Esto sirve para cuando las características que se usan para el entrenamiento y posterior predicción sufren variaciones en el tiempo, como es habitual en procesamiento de texto. Este modelo, una vez entrenado, será evaluado con una parte de los datos etiquetados que se separará para tal fin, haciendo que los mismos no participen en el entrenamiento del modelo. Luego se procede a *predecir* sobre los *datos no etiquetados* sus posibles etiquetas. Las mismas serán enviadas al *selector* el cual seleccionará una cierta cantidad de instancias para ser enviadas al *oráculo*. Luego este le solicitará al usuario etiquetador que ingrese las etiquetas de cada una de estas instancias. Una vez obtenidas las etiquetas, se iniciará de nuevo el loop principal, esta vez usando para entrenar el modelo los datos etiquetados originales y los datos etiquetados manualmente.

Como podemos apreciar en la Figura 4.1 el loop principal ya está determinado por el framework. Pero es el usuario programador quien decide como se comportan estos componentes en el loop. El mismo deberá siempre proveer de los datos etiquetados y no etiquetados, pero además podrá ajustar este loop a sus necesidades. Por ejemplo, el usuario programador puede elegir con que *algoritmo* se entrenará el *modelo*, o que *método de selección* utilizará sobre las instancias etiquetadas automáticamente. Para esto, podrá valerse de las instancias ya implementadas en el framework, o implementado su propia versión de las mismas. Así también podrá hacerlo con el *oráculo*, personalizando así la experiencia del usuario etiquetador.

### 4.1.2. Componentes del sistema

#### Oracle

Esta clase es la encargada de mostrar al oráculo (persona etiquetadora) las instancias que el selector seleccione para etiquetar, obtener la etiqueta y devolverla al *ActiveLearner* para guardar las nuevas instancias etiquetadas.

Es una clase abstracta, por lo que el usuario programador deberá implementar una clase que cumpla esta interfaz e instanciarla. Esta nueva clase deberá cuando menos definir un atributo `target_names`. El mismo será explicado a continuación junto con el resto de los métodos de esta clase:

- `target_names` Atributo que contendrá una lista con el nombre de cada clase objetivo. La misma se deberá corresponder en orden con los valores del atributo `y` del `Dataset`.
- `ask(readable_X, recoms)` Método el cual será llamado por el *ActiveLearner* en cada iteración del loop principal para obtener nuevas etiquetas. El mismo recibe los siguientes parámetros:
  1. `readable_X` Es una lista que contiene para cada elemento que se desea etiquetar, la salida del método `get_unlabeled_readable()` del `Dataset`.
  2. `recoms` Es una lista con las etiquetas predichas por el modelo para utilizar como recomendación de etiqueta al oráculo.

  Este método llamará por cada elemento que se le pase a los otros métodos de esta clase para procesar los elementos a mostrar y validar las etiquetas.
- `ask_for_element(x, recom)` Este método es llamado internamente por cada elemento que se le pasa al método `ask()` anteriormente descripto para procesar cada elemento individualmente. Sus parámetros son:

1. `x` La información obtenida del elemento por el método `get_unlabeled_readable()` del `Dataset`.
2. `recom` La etiqueta predicha por el modelo para utilizar como recomendación para oráculo.

- `show_options()` Imprime las etiquetas posibles, con sus nombres y números que las representan.
- `show_element(x, recom)` Este método muestra el dato con una simple impresión por pantalla y su etiqueta predicha. Se recomienda reimplementar este método para personalizar la experiencia de usuario del oráculo.

  1. `x` La información obtenida del elemento por el método `get_unlabeled_readable()` del `Dataset`.
  2. `recom` La etiqueta predicha por el modelo para utilizar como recomendación para oráculo.

- `validate_answer(answer)` Este método controla que el oráculo haya ingresado una etiqueta válida.

Se aconseja al usuario programador reimplementar los métodos `show_element()` y `show_options()` en conjunto con el método `get_unlabeled_readable()` del `Dataset` para adecuar la visualización de las etiquetas y las instancias al contexto de etiquetado particular de cada proyecto.

### Selector

Aunque es un módulo sencillo, es el corazón del *Aprendizaje Activo*. Se trata de una interfaz que especifica los requerimientos para la funcionalidad que selecciona las instancias que van a ser enviadas al `Oracle` para su etiquetado, a partir de la información provista por el modelo sobre un dataset. Esta interfaz requiere sólo del siguiente método:

- `select(model, data, n)` Este método recibe el modelo entrenado, los datos sobre los cuales seleccionar y la cantidad de datos a seleccionar. Devuelve una lista con los índices de los elementos seleccionados.

    1. `model` Es una instancia que satisface la interfaz de `Model`, la cual ya ha sido entrenada (ya se ha llamado al método `fit()`).
    2. `data` Son los datos que devuelve el método `get_unlabeled()` del `Dataset`.
    3. `n` Es un número entero especificando la cantidad de datos a seleccionar.

El usuario programador puede instanciar selectores siempre que implemente esta interfaz. Sin embargo el framework provee selectores ya implementados, listos para instanciarse en este hotspot, los mismos se describen a continuación:

*UncertaintySelector:* Selecciona las instancias sobre las cuales el modelo tiene menos seguridad. Esto es, para cuales la probabilidad logarítmica más alta es baja. Sirve para cuando los núcleos ya están bien definidos, para delimitar los bordes de decisión.

*CertaintySelector:* Selecciona las instancias sobre las cuales el modelo tiene mayor seguridad. Esto es, las que la probabilidad logarítmica alta es más alta. Sirve más que nada para las primeras iteraciones ya que caracteriza mejor los núcleos de las clases.

*MinDiffSelector:* selecciona los que tengan menor diferencia entre la clase con mayor probabilidad y la segunda clase con mayor probabilidad.

*EntropySelector:* basado en teoría de la información de Shannon (1948), selecciona las instancias con mayor entropía. La formula: $-\Sigma(ci * log(ci))$ se hace 0 para clases etiquetadas y se hace más grande mientras más parecidos sean las probabilidades de cada clase.

*RandomSelector:* Selecciona instancias al azar, esto es para comparar la efectividad del Aprendizaje Activo usando distintos selectores,

contra este método que sería no hacer Aprendizaje Activo, ya que al elegir al azar las instancias a etiquetar es como si simplemente estuviéramos etiquetando instancias para mejorar el aprendizaje del modelo.

### Model

Este módulo es una interfaz que especifica los requerimientos para el algoritmo de aprendizaje automático que se utilizará para entrenar un modelo de predicción a partir del conjunto de datos de entrenamiento. El mismo se ejecutará al principio de cada vuelta del bucle principal para entrenar un modelo nuevo. Y luego será nuevamente llamado para predecir las etiquetas sobre los datos no entrenados. Este módulo ha sido pensado de tal forma que pueda pasarse como argumento al Aprendizaje Activo una instancia ya entrenada de la misma. De esta forma logramos que se pueda entrenar en un servidor donde el tiempo de uso tiene un alto costo, y se pueda etiquetar localmente para evitar el uso del mismo. Además esto nos permite utilizar instancias pre entrenadas las cuales son cada vez más comunes.

Para instanciar un objeto de esta clase es necesario que el usuario programador implemente los métodos descriptos en la interfaz. Los mismos son:

- `fit(X, y)` Ajusta el modelo a los datos de entrenamiento.

  1. `X` Son los datos de entrenamiento, los que devuelve el método `get_X()` del `Dataset`.
  2. `y` Los objetivos de los datos de entrenamiento, los que devuelve `get_y()` del `Dataset`.

  Este método no retorna nada.

- `predict(X)` Predice las clases de los datos que recibe como entrada. Retorna una lista con la clase predicha para cada elemento.

- `predict_proba(X)` predice las probabilidades de cada clase de los datos que recibe. Retorna por cada elemento en X una lista de n-uplas donde n es la cantidad de clases objetivo del proyecto y donde cada valor n es la probabilidad de que el elemento sea de dicha clase.

- `score(X, y)` Devuelve la media de precisión sobre los datos `X` con los objetivos `y`. `X` e `y` deben tener el mismo type que el `X` y el `y` de entrenamiento.

Estos métodos son equivalentes a los que utiliza la librería scikit-learn en su mayoría de clasificadores. Por esta razón, en este hotspot se pueden usar estos métodos instanciándolos directamente con los que proveen los clasificadores de scikit-learn o implementar los propios siguiendo la interfaz anteriormente descrita.

El entrenamiento siempre se hace en batch, no incremental, porque hay muy pocos algoritmos que tengan su versión de entrenamiento incremental y además porque de esta forma permitimos que se pueda ir cambiando la porción del corpus que se usa para evaluación.

### Dataset

Este módulo contendrá a los datos etiquetados para entrenar, y a los no etiquetados sobre los cuales aplicar el aprendizaje activo.

Es la única clase que necesita como mínimo que el usuario programador inicialice, pasando como parámetros los datos etiquetados y no etiquetados. A su vez, el usuario programador puede heredar de esta clase para reimplementar algunos de los métodos de la misma.

Para mejorar la eficiencia de esta clase se decidió guardar internamente las variables `X` y `X_unlabeled` en una misma variable, llamada `X`. Como estos deben ser del mismo tipo, y contienen información del mismo tipo también (las características de cada instancia) esto no es un problema, y a su vez nos hace ganar mucha performance, ya que sino, cada vez que se etiqueta una instancia (o una selección de instancias) las mismas deben ser eliminadas de `X_unlabeled` y agregadas

a `X`. Pero esto genera un overhead grande, ya que estos datos son inmutables por lo que hay que crear nuevas instancias cada vez. Con esta implementación solucionamos ese problema ya que mantengo los datos en una misma variable, y accediéndolos usando máscaras, una para acceder al `X` original y otra para acceder a los `X_unlabeled` logramos que etiquetar una nueva instancia sea tan fácil como cambiar las máscaras.

De la misma manera, el `y` que se guarda en esta clase, no solo contiene las etiquetas de los datos etiquetados, sino que además funciona como máscara. En la misma se almacena un valor para cada elemento en `X,` esta valor puede ser la etiqueta de ese dato (de ser un valor entre 0 y n-1 siendo n la cantidad de clases) o puede ser -1 indicando que no se contiene la etiqueta de ese dato, o -2 indicando que ese dato ya se ha mostrado para etiquetar pero el oráculo no ha podido determinar con exactitud la etiqueta de la misma, ignorándola.

Por lo tanto la forma correcta de acceder a `X` (datos etiquetados usados para entrenar) y a `X_unlabeled` (datos sobre los cuales seleccionar) es a través de los métodos `get_X()` y`get_unlabeled()`. De esta forma se abstrae fuera de este módulo como están guardados los datos internamente.

- `__init__(X, y, X_unlabeled)` esta función se ejecuta al crearse una instancia nueva de la clase. La misma requiere de 4 parámetros obligatorios y dos opcionales. Los mismos se describen a continuación:

    1. `X` Contiene los datos sobre los cuales se posee la etiqueta. Debe ser una instancia ndarray de la librería numpy o csr_matrix de la librería scipy.
    2. `y` Contiene, en el mismo orden que `X,` las etiquetas de dichos datos. Debe ser una instancia de ndarray de la librería numpy.
    3. `X_unlabeled` Contiene los datos de los cuales no se posee la etiqueta y sobre los cuales se quiere seleccionar para

etiquetar. Debe ser una instancia del mismo tipo que `X`.

4. `test_size` Parámetro opcional. Determina el porcentaje de datos de entrenamiento que se separará para usar como evaluación del modelo. Los mismos serán separados al intanciarse la clase y no se usarán para entrenar en ningún momento.
5. `random_state` Parámetro opcional. Es la semilla que se usa para dividir los datos de entrenamiento dejando un porcentaje de los mismos para evaluar.

- `_split_X(X, y, test_size, random_state)` Este método debe ser llamado solo internamente y en la inicialización de la clase. Es el que realiza la separación de los datos de entrenamientos en dos, para entrenar y para evaluar. El mismo para hacerlo utiliza la función `train_test_split()` de sklearn.
- `get_X()` Método para acceder a los datos de entrenamiento desde afuera, utilizando la máscara descrita en la introducción de este módulo.
- `get_y()` Método para obtener los valores de las etiquetas de los datos de entrenamiento. Utiliza para eso internamente la máscara descrita en la descripción de este módulo.
- `get_unlabeled()` Método para obtener los datos no etiquetados, y que además no se hayan ignorado para etiquetar anteriormente. Utiliza interiormente la máscara descrita en la introducción de este módulo para tal fin.
- `tag_elements(indices, tags)` Método para agregar nuevas etiquetas a los datos que estaban en `X_unlabeled.`. Los parámetros de esta función se describen a continuación:

1. `índices` Posee los índices que dan con el dato a etiquetar filtrados por la máscara. Esto es para opacar el funcionamiento interno de esta clase.

2. `tags` Lista con los valores de las etiquetas de los datos cuyos índices fueron pasados. Deben tener el mismo orden.

- `get_unlabeled_readable(i)` Este método será llamado al momento de solicitarle las etiquetas de algún elemento al oráculo. Recibe el parámetro i el cual es el índice del elemento a presentar en el `Oracle`. Debe devolver lo necesario para poder mostrar el objeto al usuario, por ejemplo, si se trata de imágenes, este método podría devolver la ubicación de dicha imagen para que luego el `Oracle` la muestre.

- `_get_train_indices()` Método para ser utilizado sólo dentro de la clase. El mismo devuelve una lista con los índices de los elementos en la variable X que son datos etiquetados para entrenamiento. Se utiliza para la máscara.

- `_get_unlabeled_indices()` Método para ser utilizado sólo dentro de la clase. El mismo devuelve una lista con los índices de los elementos en la variable X que son datos sin etiquetar. Se utiliza para la máscara.

### Active Learner

Este es el módulo que conecta todos los anteriores. Para crear una instancia primero se deben crear instancias de cada uno de los módulo anteriores para pasarlas como parámetro. Esta clase está completamente implementada, y posee los siguientes métodos:

- `fit()` Este método llama al método `fit()` del modelo que se le pasó como parámetro al momento de la inicialización de la instancia o por el cual se haya cambiado con el método `change_selector()`. Al mismo le pasa como parámetros los `X` e `y` para entrenar, obtenidos del método `get_X()` y `get_y()` del `dataset`.

- `select(n)` Este método es para seleccionar los datos no anotados que van a ir al oráculo para su anotación. El mismo recibe

un n como parámetro de entrada que especifica la cantidad de instancias que el seleccionador tiene que devolver. El método llama al método `select()` del `Selector` al cual le pasa además el modelo entrenado y los datos no etiquetados (obtenidos por medio de la interfaz `get_unlabeled()` y devuelve la salida del mismo.

- `ask(indices)` Este método busca las instancias sin anotar solicitadas (indices) del `Dataset` a través del método `get_unlabeled_readable()` del mismo. Luego calcula la etiqueta predicha por `Model` y estos dos parámetros se los pasa al `Oracle` por medio del método `ask()`.

- `tag_elements(indices, y)` Este método recibe los índices de los elementos que devuelve el método `get_unlabeled()` de `Dataset (indices)` y la etiqueta correspondiente a cada una de estas instancias `(y)`. Lo que hace el método es llamar al método `tag_elements()` de `Dataset`.

- `get_scores()` Método que devuelve una lista con los valores del score de cada `fit()` realizado.

- `change_selector()` Este método recibe una instancia nueva de `Selector` como parámetro y lo cambia en el `ActiveLearner` por el actual para futuras selecciones.

## 4.2. Evaluación

Al momento de crear la instancia de `Dataset` al inicializar el framework, el conjunto de datos de entrenamiento se divide en dos. De estas partes se usará una para entrenar el modelo y otra para evaluar la precisión del mismo. Con la parte que se deja para evaluar en ningún momento se entrenará al modelo. Esta parte servirá para que al final de cada iteración del bucle principal se prediga sobre ellos las etiquetas, luego se comparan con las etiquetas originales y se calcula

el porcentaje de acierto del modelo. Estos valores se guardan en una lista, para luego poder graficar y así analizar el desempeño a lo largo de las iteraciones. Esto nos da una idea de la tasa de aprendizaje, y nos sirve para detectar posibles problemas y para ver cuándo se está llegando a la meseta de aprendizaje[1].

El porcentaje de datos dejado para tests se define al inicializar la instancia de `Dataset` pasándole como parámetro en `test_size` un valor entre 0 y 1 indicando el porcentaje que se deja para test. Por defecto este valor es 0.2, lo que se traduce a que el 20 % de los datos son dejados para test. No es necesario que el corpus de test sea muy grande, porque no afecta al funcionamiento del sistema, que usa solamente el corpus de training para entrenar el modelo y tomar decisiones.

Sin embargo, según el porcentaje de datos con el que se cuente al momento de inicializar el framework, este conjunto de datos dejado para testing puede que sea muy chico. Si es muy chico, resultará difícil identificar los fenómenos relevantes, como la meseta de aprendizaje. Se consideró ir aumentando este conjunto de manera progresiva a medida que iteraba el framework, incorporando un porcentaje de instancias etiquetadas por el oráculo después de haber sido seleccionadas por el selector. Sin embargo, en algunos casos esto podía llevar a mediciones demasiado optimistas, sesgadas por los selectores. En particular, si se incorporan instancias para test seleccionadas por el `CertaintySelector`, los resultados son muy altos porque se evalúan sobre instancias que resultan muy fáciles al predictor.

## 4.3. Formato de entrada

El framework toma un conjunto de datos de entrada y un pipeline para vectorizarlos, es decir, convertirlos en una lista de instancias donde cada una a su vez es una lista de características con sus valores

[1] Llamamos meseta a la curva que se produce cuando en el gráfico de aprendizaje respecto al tiempo, el algoritmo deja de mejorar, creando la ilusión óptica de una meseta en el mismo.

respectivos. En cada iteración del framework se vectorizan de nuevo las instancias ya que el conjunto de características puede modificarse al incorporar nuevas instancias, por ejemplo, añadiendo características nuevas o eliminando características que no cumplen con alguna condición (e.g. un umbral de probabilidad de ocurrencia). El proceso de vectorización puede ser costoso en cálculo pero provee la flexibilidad necesaria para modelar adecuadamente diferentes tipos de problemas. Por este motivo, si el preprocesado no cambia en el tiempo, se recomienda ingresar al framework los datos ya preprocesados en primer lugar y utilizar solo el estimador en el hotspot del modelo y no un pipeline para evitar este overhead.

Actualmente el framework espera que el conjunto de datos de entrada sea del tipo *numpy array* o *scipy sparse*. Esto es por como el dataset almacena los datos etiquetados y no etiquetados. De requerirse otro formato de entrada será necesario agregar funcionalidad al método `vstack()` con el nuevo tipo de entrada y con la implementación de código necesaria para que se puedan apilar dos matrices. Las mismas son el conjunto de datos etiquetados y el conjunto de datos no etiquetados.

# Capítulo 5

# Experimentos

Para evaluar el rendimiento de los diferentes parámetros del sistema, se realizaron algunos experimentos sobre datasets estándares.

## 5.1. Evaluación con pocos datos etiquetados

Es muy difícil evaluar un modelo del cual se tienen pocos datos. El framework para evaluar su progreso selecciona un porcentaje de datos de los del entrenamiento inicial que ya están etiquetados, y los separa para evaluar sobre ellos la precisión del mismo. El problema es que cuando se tiene pocos datos para la primer iteración, este porcentaje deja muy pocas instancias para testing. Con pocas instancias, los resultados de las evaluaciones tienen mucha variación, por lo que se observan saltos abruptos en la curva de precisión. Esto hace que tengamos poca estabilidad a la hora de evaluar el modelo, porque los datos con los que estamos evaluando son poco representativos del total del universo. Por esta razón, la evaluación que provee el framework debe tomarse con recaudos, como una indicación del rendimiento de los sucesivos modelos que se entrenan, y no como una medida totalmente fidedigna.

Sin embargo, para evaluar el framework queremos realizar una evaluación más representativa y reproducible. Por esa razón usamos datasets estándares que están completamente etiquetados y para los cuales ya se conocen los rendimientos de los algoritmos estándares. Dado que estos datasets están totalmente etiquetados, podemos utilizar una porción más grande de datos para evaluación.

## 5.2. Evaluación por simulación de oráculo

Los experimentos se realizaron simulando un oráculo sobre datasets que estaban completamente etiquetados. Esta decisión fue para acelerar el proceso y poder probar y comparar más configuraciones en un menor tiempo.

Entonces de los datasets originales se tomó sólo una muestra pequeña a las cuales se les dejó las etiquetas y sobre las cuales se iniciaron las iteraciones del Framework. El resto del dataset se separó en datos y etiquetas, pero guardando estas últimas para pasárselas al `Oracle` cuando este las requiera, en lugar de solicitarle a un usuario etiquetador y tener que esperar por las mismas. Esto se logró reimplementando el método `ask()` del mismo, y el método `get_X_readable()` del `Dataset` para que en lugar de seguir el workflow normal, el oráculo obtenga la información de las etiquetas directamente del dataset.

Para que la simulación sea lo mas realista posible, primero se experimentó etiquetando manualmente con diferentes configuraciones. Así se encontró que cuando el etiquetador es uno sólo, lo más llevadero es seleccionar de a 10 instancias y etiquetarlas antes de volver a iterar. Esto es, teniendo en cuenta el tiempo que lleva entrenar y volver a seleccionar instancias para etiquetar, hacen que seleccionar de a una, aunque en teoría funcione mejor, en la práctica es un trabajo más lento y pesado para el etiquetador. Y seleccionando muchas instancias, se hace muy tedioso, además de que muchas veces las instancias que se seleccionan son muy similares (dependiendo del selector que se

use) esto genera que probablemente con un par de instancias ya etiquetadas el modelo las aprenda y las otras etiquetas hayan sido más redundantes y esto se traduce en una pérdida de tiempo. Por lo que se decidió pedirle al selector 10 instancias a etiquetar en cada iteración del framework. Para más cantidades de etiquetadores trabajando a la vez, el número de etiquetas a seleccionar debería ser mayor, perdiendo en calidad de selección pero aumentando la velocidad con la que se consiguen las etiquetas.

## 5.3. Datos de evaluación

Para experimentar entonces con la configuración anteriormente descrita del oráculo se tomaron diferentes datasets estándares sobre diferentes tipos de datos para ver cómo se adaptaba el framework a cada uno de ellos. A continuación describiremos dos de ellos: El 20 Newsgroups y el MNIST.

### 5.3.1. 20 Newsgroups

El 20 Newsgroups es una colección de aproximadamente 20.000 documentos, repartidos de forma considerablemente uniforme en 20 categorías diferentes. Es uno de los datasets más populares para experimentar en proyectos de Aprendizaje Automático que involucren textos, tanto para clasificación como para clustering.

Podemos observar que las categorías de este dataset pueden tener problemas para caracterizarse de forma unívoca, ya que son muy específicas. Podríamos esperar que categorías más estándares cómo "deportes" o "política" fueran más fáciles de caracterizar y por lo tanto también de diferenciar. En la anotación manual llevada a cabo por el oráculo se observó que efectivamente los textos eran difíciles de categorizar para un humano. Estas 20 categorías pueden agruparse en 6 conjuntos distintos según el tema que contengan. Por ejemplo, juntando a las que contienen temas relacionados a deportes, o a los de

políticas. De esta manera se logra diferenciar más a las mismas. Por lo tanto, para paliar el problema de la representatividad, seleccionamos 6 clases diferentes, cada una de un subconjunto distinto, que van a ser nuestras clases objetivo finales: 'alt.atheism', 'comp.graphics', 'sci.med', 'rec.motorcycles', 'misc.forsale' y 'talk.politics.guns'.

Los documentos se representaron usando como características las palabras más frecuentes en el dataset, con el valor *tf-idf* de cada palabra en cada documento. El *tf-idf* es una medida de la frecuencia de una palabra en un documento atenuada por la frecuencia de la palabra en todo el dataset, lo cual da una buena medida de la fuerza de asociación entre una palabra y un documento.

Para inicializar el framework se utilizaron 10 instancias de cada clase a predecir como dataset etiquetado de partida.

En este caso se utilizó el subset de test que provee scikit learn para analizar el desempeño del modelo, y así ver qué selector maximizaba más su aprendizaje.

Durante la etapa de exploración mediante etiquetado manual de este experimento se vio que había varias instancias de datos a etiquetar que no poseían información relevante respecto al tema que les correspondía, lo que imposibilitaba la tarea de clasificarlos incluso para un usuario etiquetador. Dichas instancias no tenían caracteres alfabéticos, o sólo decían "`[Deleted]`". Por lo que se propuso hacer un segundo experimento, donde se decidió limpiar el dataset, quitando todas las instancias vacías de los datos, y todas las que no contenían ningún caracter alfabético. Para eliminar posibles causas de ruido, se decidió también eliminar todos los documentos con menos de 100 palabras, que en general, eran documentos con contenidos espúreos o poco clasificables. También se utilizó el mismo preproceso en el dataset de testing.

Comparamos entonces el dataset original del dataset del cual eliminamos las instancias antes descriptas. La cantidad de instancias por clase quedó conformada entonces de la siguiente forma:

| Clase | Train | Test | Train Limpio | Test Limpio |
|---|---|---|---|---|
| alt.atheism | 470 | 319 | 211 | 158 |
| comp.graphics | 574 | 389 | 170 | 137 |
| misc.forsale | 575 | 390 | 210 | 124 |
| rec.motorcycles | 588 | 398 | 202 | 114 |
| sci.med | 584 | 396 | 275 | 182 |
| talk.politics.guns | 536 | 364 | 275 | 161 |

Donde *Train* son los datos de entrenamiento y *Test* los datos sobre los que se evaluó el modelo.

### 5.3.2. MNIST

El MNIST [1] es un dataset de imágenes, donde cada instancia es una imagen de un dígito escrito a mano. Este dataset también es muy estándar para trastear con clasificación de imágenes. Se usó una muestra de 1800 instancias de las 77000 que se encuentran en el corpus, ya que con esta cantidad resultaba suficiente para comprobar el funcionamiento del framework. Para poder mostrar estas imágenes en el `Oracle` el mismo se modificó para que la función `show_element()` en vez de mostrar texto plano, muestre un ploteo de los features del dígito a etiquetar para poder visualizarlo. Los mismos se los pasa la función `get_unlabeled_readable()` la cual también se modificó ligeramente.

Se dejó el 33 % del dataset etiquetado original para evaluar a los selectores. Del restante, se tomó el 5 % (60 instancias) de manera aleatoria para usar como datos etiquetados para inicializar el framework y se dejó el resto como datos no etiquetados.

Para el etiquetado manual, los datos se representaron como una figura de 8 por 8 creada con la herramienta *Matplotlib*. En esta instancia, también se encontraron dificultades para etiquetar algunos números, pero la gran mayoría de instancias a etiquetar eran fácilmente determinadas por un humano.

[1] `http://yann.lecun.com/exdb/mnist/`

## 5.4. Comparación de selectores

Usando la simulación de oráculo sobre los dos datasets mencionados, se compararon los diferentes selectores provistos en el framework y cinco selectores aleatorios. En la Figura 5.1 podemos ver los resultados para el 20Newsgroups usando *multinomial naïve bayes* como algoritmo de aprendizaje y en las Figuras 5.3 y 5.4 vemos los resultados para el MNIST usando *logistic regression* y *random forests* respectivamente. Además podemos ver en la Figura 5.2 que limpiar los datos de entrenamiento hace que se logre un mejor desempeño al momento de predecir y el mismo se alcanza con muchas menos instancias. También se puede observar, que luego de seguir entrenando una vez alcanzado el máximo desempeño, el algoritmo empieza a sobre-entrenarse (*overfitting*).

Se puede observar que, independientemente del algoritmo de aprendizaje, todos los selectores alcanzan el mismo rendimiento cuando incorporan todas las instancias del dataset. El mejor rendimiento de un selector se obtiene cuando éste alcanza buen rendimiento con un menor número de instancias.

En el caso del 20 Newsgroups observamos que todos los selectores, menos el selector basado en certidumbre (`CertaintySelector`), obtienen sistemáticamente resultados mucho mejores que los selectores aleatorios, alcanzando buena performance con muchas menos instancias. Por lo tanto, el uso de selectores inteligentes ofrece mejores resultados que usar selectores aleatorios.

En el primer experimento podemos observar cómo los modelos entrenados usando los selectores comienzan a converger antes de la iteración 150. Mientras que los selectores aleatorios no lo hacen hasta la iteración 250. Además los selectores (excepto el que está basado en certeza), cada iteración de las primeras 150 logran resultados consistentemente superiores a los selectores aleatorios, acercándose estos sólo sobre el final de la simulación cuando ya no quedan muchas opciones para seleccionar.

Entre los diferentes selectores inteligentes no observamos grandes

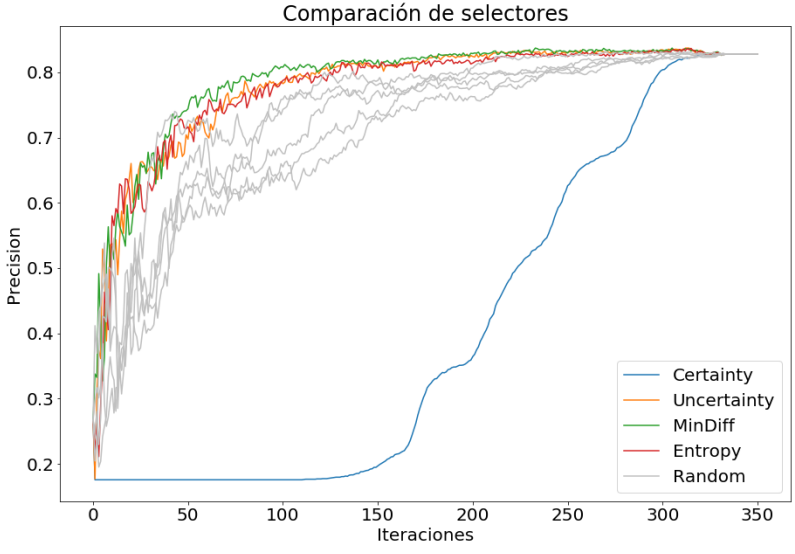



Figura 5.1: Comparación de selectores mediante simulación de oráculo para el 20Newsgroups, incluyendo 5 selectores aleatorios (en gris), usando *multinomial naïve bayes* como algoritmo de aprendizaje.

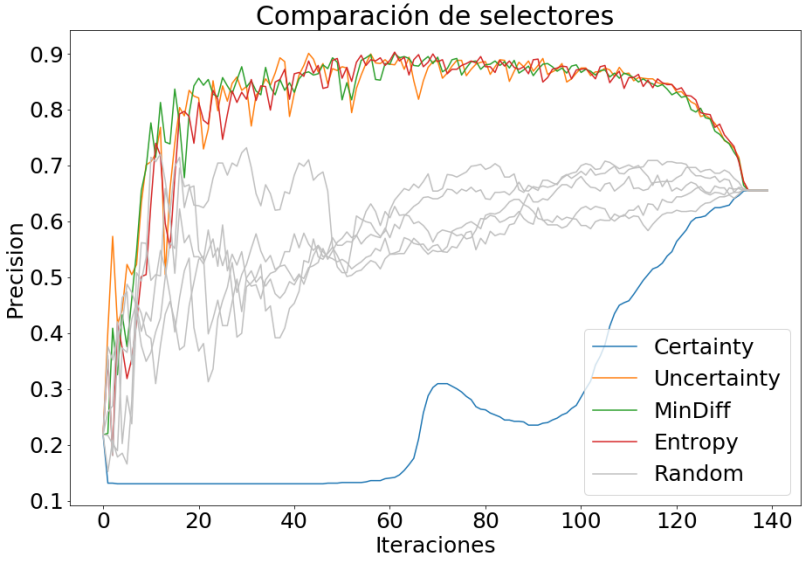



Figura 5.2: Comparación de selectores mediante simulación de oráculo para el 20Newsgroups con el preprocesado de limpieza descrito, incluyendo 5 selectores aleatorios (en gris), usando *multinomial naïve bayes* como algoritmo de aprendizaje.

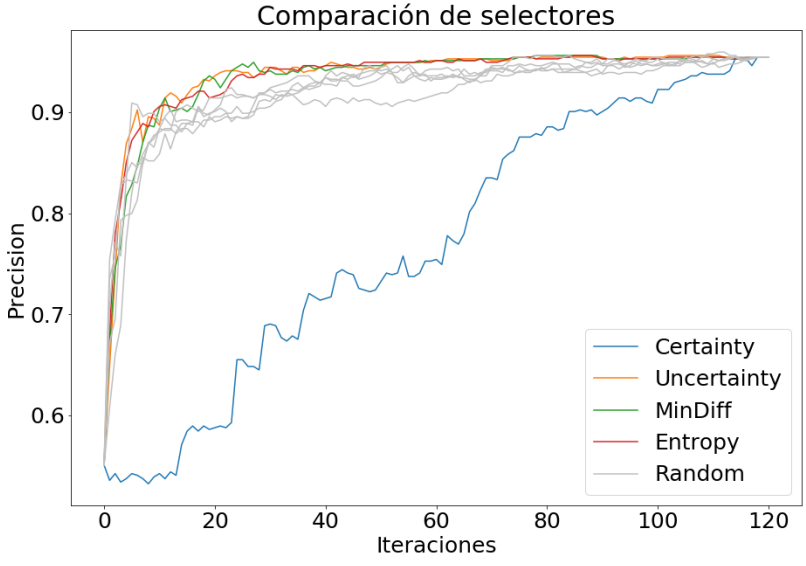



Figura 5.3: Comparación de selectores mediante simulación de oráculo para el MNIST, incluyendo 5 selectores aleatorios (en gris), usando *logistic regression* como algoritmo de aprendizaje.

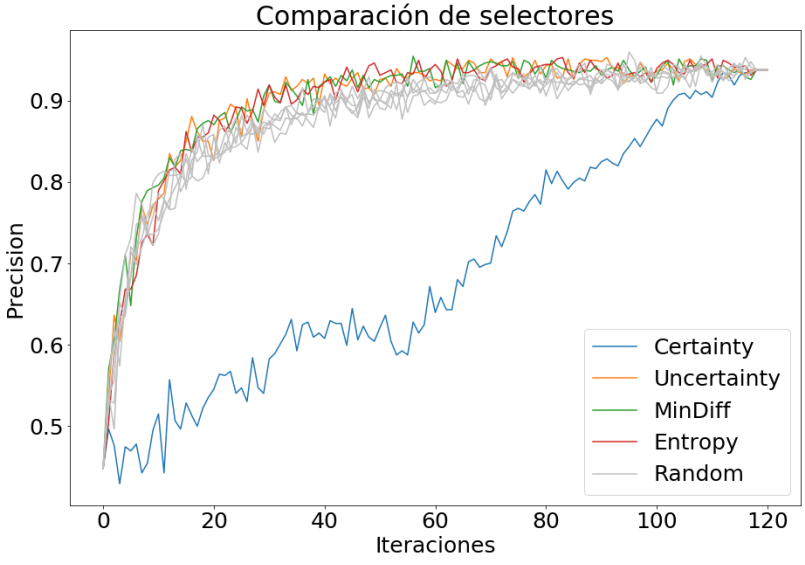



Figura 5.4: Comparación de selectores mediante simulación de oráculo para el MNIST, incluyendo 5 selectores aleatorios (en gris), usando *random forests* como algoritmo de aprendizaje.

diferencias, aunque el selector `MinDiff` parece rendir consistentemente mejor que los otros, quedando los selectores basados en entropía y certeza por detrás con resultados muy similares. Parece que este selector se complementa bien con la simple probabilidad en la que se basa el modelo de Naïve Bayes.

El selector basado en certidumbre (`CertaintySelector`) obtiene resultados mucho peores que el resto de selectores, incluso los aleatorios, y solamente llega al rendimiento del resto cuando se han etiquetado todas las instancias. Por lo tanto, no se recomienda el uso de este selector en el caso general. Sin embargo, este selector puede resultar de utilidad en proyectos que requieran específicamente consolidar los conceptos en el primer momento de anotación.

En el segundo experimento, con los datos "limpios", podemos ver una notable diferencia entre los selectores inteligentes y los aleatorios (nuevamente exceptuando el selector basado en certidumbre). Como era de esperar, con los datos más limpios se alcanzan precisiones más altas, llegando estas a casi el 90 %. Además podemos ver cómo los selectores inteligentes logran diferenciarse más de los aleatorios, esto se debe a que con el dataset más limpio logran selecciones más eficientes sobre los datos maximizando aún más el aprendizaje.

Podemos observar también que en un cierto momento añadir más ejemplos impacta negativamente en el aprendizaje. Mi hipótesis es que cuando se ha alcanzado la meseta de rendimiento añadir nuevos ejemplos solo produce sobre ajuste del modelo. Esto no se observa en los dataset con ejemplos ruidosos porque no se alcanza la meseta de performance y los nuevos ejemplos siguen aportando información valiosa.

También en el dataset MNIST se observa que todos los selectores menos el `CertaintySelector` obtienen mejores resultados que cualquiera de los selectores aleatorios (en gris), para ambos algoritmos de aprendizaje. Todos obtienen mejor rendimiento que cualquier aleatorio con la misma cantidad de ejemplos y convergen al mejor rendimiento que se puede obtener con menor número de ejemplos.

Tampoco se observan diferencias importantes entre los diferentes

selectores, ni siquiera la tendencia que se observaba en el 20Newsgroups para el `EntropySelector`, ya que en este dataset rinde de forma muy comparable al resto. Sí podemos observar que el `MinDiffSelector` suele quedar al mismo nivel o por encima del resto si usamos *logistic regression*, de forma consistente con los resultados para el 20Newsgroups. Parece, entonces, que este selector es una buena opción si no tenemos ningún criterio para preferir otro. Por lo tanto, parece una buena opción como selector por defecto para este hotspot.

Con respecto a los algoritmos de aprendizaje, observamos que *logistic regression* obtiene resultados más estables que *random forests*, por lo tanto, este parece también un buen valor como algoritmo por defecto para el hotspot `Model`.

Una vez analizados estos 4 experimentos, concluyo que al momento de seleccionar instancias para etiquetar es más eficiente hacerlo de forma inteligente que aleatoria. Cuanto más eficiente es esta forma, depende mucho del tipo de datos que se tengan y de que tan diferenciadas estén las clases objetivos entre sí. En un problema donde los clases no se diferencian bien entre ellas, los selectores inteligentes tendrán problemas para seleccionar instancias que maximicen el aprendizaje, rindiendo casi como un selector aleatorio. Pero mientras más se diferencien estas, más rápido se llegará (o con menos datos etiquetados) al máximo de rendimiento del modelo para los datos con los que se cuenta.

## 5.5. Extensiones de evaluación

De la misma forma que se ha presentado en este capítulo, se podrían comparar también los parámetros de otros hotspots del framework, como por ejemplo profundizar en la comparación de diferentes algoritmos de Aprendizaje Automático o diferentes estrategias de evaluación.

También podríamos experimentar con diferentes esquemas de clasificación, como por ejemplo una aproximación jerárquica o de cascada.

En esta aproximación, un primer modelo clasificaría las instancias en clases gruesas (por ejemplo, podríamos considerar los subconjuntos del 20 Newsgroups mencionados anteriormente). Una vez clasificados en una gran clase, se aplica un modelo específico por clase que determina a cuál de las subclases de la gran clase pertenece cada instancia.

También se podrían realizar experimentos promediando los resultados obtenidos con diferentes datasets iniciales.

# Capítulo 6

# Conclusiones y trabajo futuro

## 6.1. Aportes

En este trabajo he desarrollado un framework de software para facilitar el despliegue de proyectos basados en aprendizaje activo. El objetivo de estos proyectos es desarrollar un conjunto de datos (*dataset*) con anotación manual, optimizando el esfuerzo humano.

El framework desarrollado es abierto, disponible libremente (`https://github.com/agusdmb/Active-Learning-Framework`) y recurre a librerías y formatos estándar y está disponible mediante una notebook de Jupyter, todo lo cual facilita su integración a implementadores con conocimientos muy básicos de aprendizaje automático.

En experimentos con *datasets* estándar se ha visto que el rendimiento de las aplicaciones de Aprendizaje Activo mediante este software es el esperado, y que la eficiencia es aceptable.

En comparación con otros frameworks de software con la misma funcionalidad, este framework se diferencia por tener su documentación en castellano, usar como interfaz la Jupyter notebook y tener algunas parametrizaciones diferentes, que pueden acomodar mejor a

ciertos tipos de proyectos.

## 6.2. Trabajo futuro

En esta sección se describen posibles mejoras y funcionalidades interesantes que podrían agregarse al framework. Las mismas están agrupadas en subsecciones dependiendo de qué partes del sistema impactan.

### 6.2.1. Selector

Como dijimos al principio de este documento, los selectores son el corazón del Aprendizaje Automático. Por lo tanto, no podía quedar fuera del trabajo futuro seguir mejorando sobre este área. Los selectores se pueden agrupar en dos grandes grupos, los que deben predecir sobre los datos no etiquetados, y los que no. En esta tesis sólo se han implementado selectores del primer grupo. Existe una gran variedad de selectores dentro de este grupo, en particular me gustaría implementar el selector por comité. Esto es un selector que se basa en las predicciones de más de un modelo, y elije para enviar al oráculo aquellas donde estos estén más en desacuerdo.

Por otro lado, empezar a agregar selectores que no dependan de hacer predicciones sobre los datos no etiquetados para seleccionar instancias para etiquetar. Esto tiene sus ventajas y desventajas, por ejemplo, cuando se tienen muchos datos no etiquetados, y el modelo para predecir sobre los mismos tiene un costo en tiempo elevado, hace que cada iteración del framework lleve mucho tiempo de procesar ralentizando el trabajo de etiquetado manual. Una solución a este problema es, por ejemplo, hacer clustering sobre los datos no etiquetados. Luego, elegir los centros de los clusters como las instancias a etiquetar. La idea intuitiva detrás de esto es que las instancias que estén en los mismos clusters tienen mayor probabilidad de tener la misma etiqueta.

Otra mejora que se puede implementar, pensando en los usuarios

con pocos conocimientos técnicos, es "Learning by Learning" como hace Libact. Esto es, que el mismo framework vaya probando y haciendo competir a los selectores entre si, eligiendo al mejor para cada proyecto. Esto sería de utilidad para aquellos usuarios que no sepan que selector usar, ni las diferencias entre los mismos. Cada proyecto es un caso particular, y como vimos en el trabajo previo, hasta lo que puede sonar como el selector por defecto, el "Uncertainty", a veces puede tener un desempeño peor que uno aleatorio. Esta técnica ayudaría a usuarios sin mucho conocimiento a no caer en estos casos.

### 6.2.2. Oráculo

Un aspecto donde se puede mejorar mucho es el Oráculo. La herramienta de etiquetado es donde, probablemente, la mayor parte del tiempo se invierta, por lo tanto es de mucha importancia que esta sea lo más útil y amena de usar posible. A priori es difícil diseñar una única herramienta que se adecúe a todos los tipos de proyectos que queremos llegar. Sin embargo, podemos buscar aspectos comunes en todos ellos para definir qué debería tener dicha herramienta. Una de estas características puede ser una verificación de identidad, para que no cualquier persona pueda etiquetar instancias (por ejemplo en un sistema que se etiquete vía internet). Otro sería obtener redundancia en las etiquetas, haciendo que sólo pasen al conjunto de datos etiquetados si más de una persona las ha etiquetado y estas etiquetas han coincidido entre ellas.

### 6.2.3. Evaluación

Uno de los mayores problemas que se presentaron en este proyecto, fue cómo evaluar en un proyecto real (no simulado) la precisión del modelo que se entrena. Este problema se debe a que en principio se poseen muy pocas instancias etiquetadas, por lo que tomar una muestra de las mismas para dejar para evaluación no es muy representativo. A la vez, seleccionar nuevamente instancias para evaluación una vez

que ya hemos empezado a etiquetar instancias dentro del framework no genera muestreos aleatorios, y por lo tanto no son representativos, ya que los mismos no son un muestreo de la población, sino de los que el selector elige. Por lo que con esta técnica, Certainty pareciera siempre rendir mejor que los otros selectores, ya que al etiquetar suficiente cantidad de instancias con el mismo, y tomar una muestra aleatoria de estas, el modelo fácilmente predice sobre las mismas, lo que implica que generalice bien. De hecho, hemos visto que este selector es el que peor rinde en ambos datasets sobre los que hemos evaluado, incluso mucho peor que los selectores aleatorios. Una forma que se me ocurrió para solucionar este problema es solicitarle al usuario etiquetador que etiquete instancias seleccionadas aleatoriamente en conjunto con las del selector que el usuario programador haya determinado. Una vez que se obtienen las etiquetas, el framework sabe cuales son las que se han seleccionado aleatoriamente, y en lugar de agregar estas instancias a los datos de entrenamiento, agregarlas al conjunto de instancias de evaluación. De esta forma logramos que conforme crece el número de instancias etiquetadas, crece también el conjunto de evaluación, haciendo que la misma la vez sea cada vez más representativa de la población.

### 6.2.4. Rendimiento

Muchas veces cuando se trabaja con proyectos reales no se entrena a los algoritmos localmente. En su lugar se usan servidores con gran potencia de cálculo y grandes volúmenes de memoria. Esto se debe a que muchas veces estos proyectos requieren más memoria de la que se dispone en la computadora personal o del trabajo. O tal vez, el tiempo que demoraría en entrenarse en ese hardware sea de semanas o incluso meses, cuando en un servidor se pueden correr en un par de días u horas. Estos servidores tienen un alto costo económico y el mismo se paga por tiempo de uso. Por lo que no parece razonable tener una instancia corriendo en uno de estos servidores mientras se espera que el usuario etiquetador introduzca las etiquetas que el sistema le requie-

ra. Una forma de mitigar esto es montando el oráculo en un servidor separado mas económico (o localmente) y utilizando el otro servidor, con hardware más potente, sólo durante los entrenamientos. Además, si el entrenamiento demora mucho tiempo incluso en estos servidores potentes, también se puede paralelizar parte de este workflow, haciendo que mientras el usuario etiquetador está ingresando etiquetas, otro proceso esté entrenando con las etiquetas ingresadas anteriormente y seleccionando nuevas instancias para enviar al usuario una vez que este termine de etiquetar el batch que ya se le ha solicitado.

### 6.2.5. Extensión de dominio

Otras mejoras de aspecto más teórico que ingenieril sería soportar más tipos de proyectos. Por ejemplo, uno que me interesa mucho sería agregar soporte para problemas de multi etiqueta. Esto es, donde cada instancia no sólo pertenece a una clase, sino que puede pertenecer a varias, llamándole a estas "etiquetas". Otra funcionalidad interesante sería predicción estructurada. Por ejemplo, en una imágen, detectar qué objetos hay y dónde están, que es un problema multiclase y estructurado. Lo mismo se puede aplicar a extracción de información sobre textos, etc. Estos problemas son demasiado complejos para poder tratarlos con un framework genérico, pero lo que se puede sugerir al usuario es que los transforme en un problema de clasificación (simple o multiclase), dividiendo el input en fragmentos y luego combinando las predicciones con alguna heurística. De esta forma, aplicamos *separation of concerns*, y delegamos en el experto de dominio la representación del problema, y el framework puede permanecer agnóstico al problema.

# Bibliografía

Cardellino, C. and et. al. (2015). Information extraction with active learning: A case study in legal text. In *Proceedings of the 16th International Conference on Intelligence Text Processing and Computational Linguistics (CICLing 2015)*.

Settles, B. (2011a). Closing the loop: Fast, interactive semi-supervised annotation with queries on features and instances. In *EMNLP*, pp. 1467–1478. ACL.

Settles, B. (2011b, 16 May). From theories to queries: Active learning in practice. In I. Guyon, G. Cawley, G. Dror, V. Lemaire, and A. Statnikov (Eds.), *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, Volume 16 of *Proceedings of Machine Learning Research*, Sardinia, Italy, pp. 1–18. PMLR.

Shannon, C. E. (1948, 7). A mathematical theory of communication. *The Bell System Technical Journal*@(3), 379–423.

Teruel, M. (2015). Aprendizaje activo para clasificación de preguntas. Master's thesis, Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba, Córdoba, Argentina.

Tivadar Danka. modAL. `https://github.com/cosmic-cortex/modAL`.