Pybrain

1. Introducción

En este documento se realizará un análisis sobre las características y pormenores de la librería de Machine Learning para Python conocida como Pybrain.

El objetivo de esta librería es ofrecer una variedad de algoritmos fáciles de usar y potentes para la realización de tareas de Machine Learning, esto permite su uso tanto para estudiantes dando sus primeros pasos en la inteligencia artificial así como para investigadores de primer nivel.

2. Algoritmos

En este caso se analizarán los algoritmos de aprendizaje, tanto supervisado como no supervisado, a pesar de que Pybrain contiene una gran variedad de algoritmos para gradientes o métodos de exploración.

2.1. Aprendizaje supervisado

El aprendizaje supervisado es una técnica para deducir una función a partir de datos de entrenamiento. Los datos de entrenamiento consisten de pares de objetos (normalmente vectores): una componente del par son los datos de entrada y el otro, los resultados deseados. El objetivo del aprendizaje supervisado es el de crear una función capaz de predecir el valor correspondiente a cualquier objeto de entrada válida después de haber visto una serie de ejemplos, los datos de entrenamiento. Para ello, tiene que generalizar a partir de los datos presentados a las situaciones no vistas previamente.

2.1.1. Back-Propagation

La propagación hacia atrás de errores o retropropagación (del inglés backpropagation) es un algoritmo de aprendizaje supervisado que se usa para entrenar redes neuronales artificiales. El algoritmo emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas.

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.

2.1.2. R-Prop

R-Prop es el acrónimo de “Resilient Back-Propagation”. En este caso se toma en cuenta el signo de la derivada parcial para cada peso. Si ocurre un cambio de signo de la derivada parcial de la función de error total comparado con la anterior iteración el valor de actualización para ese peso se multiplicará por un factor $\eta^- < 1$. Si no existe un cambio de signo se multiplica el valor de actualización por un factor $\eta^+ > 1$. η^+ suele ser considerado empíricamente como 1,2 y η^- como 0,5.

2.1.3. Support-Vector-Machines

Una SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases por un espacio lo más amplio posible. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de su proximidad pueden ser clasificadas a una u otra clase.

Más formalmente, una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que puede ser utilizado en problemas de clasificación o regresión. Una buena separación entre las clases permitirá un clasificación correcta.

2.2. Aprendizaje no Supervisado

Un algoritmo de aprendizaje no supervisado se distingue del aprendizaje supervisado por el hecho de que no hay un conocimiento a priori. En este caso, un conjunto de datos de objetos de entrada es tratado como un conjunto de variables aleatorias, siendo construido un modelo de densidad para el conjunto de datos.

2.2.1. K-Means Clustering

El clustering o agrupamiento por K-means tiene como objetivo la partición de un conjunto de *n* observaciones en *k* grupos en el que cada observación pertenece al grupo más cercano a la media.

2.2.2. PCA/pPCA

El ‘Principal Component Analysis’ (Análisis de Componentes Principales en español) es una técnica estadística utilizada para reducir la dimensionalidad de un conjunto de datos. Técnicamente, busca la proyección según la cual los datos queden mejor representados en términos de mínimos cuadrados.

El pPCA es el enfoque probabilístico del PCA, usando como estimador de máxima verosimilitud un algoritmo EM (Expectation-Maximization).

2.2.3. LSH

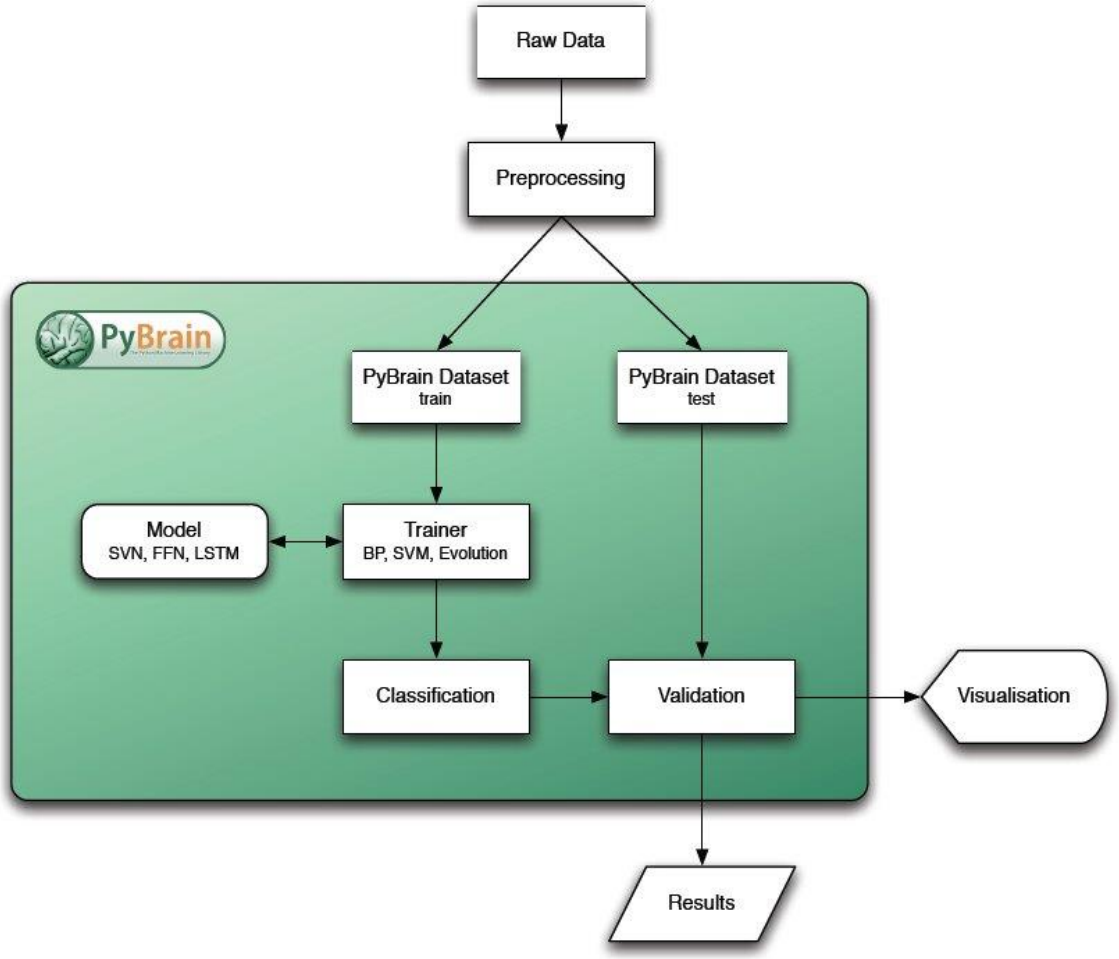
El ‘Locally-Sensitive Hashing’ es un algoritmo para la resolución del problema Nearest Neighbor Search en espacios de alta dimensionalidad separando las entradas de manera que cada una de ellas sea mapeada en una casilla con alta probabilidad, siento el numero de casillas mucho menor al universo de posibles entradas.

2.2.4. Deep Belief Network

Una ‘Deep Belief Network’ (Red de Creencia Profunda) es un tipo de red neuronal compuesto por múltiples capas de variables latentes, con conexiones entre las capas, pero no entre unidades dentro de cada capa. Una variable latente es una variable que no se observa directamente, sino que es inferida a partir de otras variables.

Si una DBN es entrenada de manera no supervisada con un conjunto de ejemplos de entrada puede aprender a reconstruir probabilísticamente dichas entradas. Puede ser entrenada de manera supervisada después de este paso para mejorar la fase de clasificación.

3. Funcionamiento de la herramienta



Como se puede ver en este diagrama de flujo la herramienta comienza convirtiendo unos datos ya preprocesados en ‘Datasets’, unos tipos de datos propios de la herramienta. A continuación se pasa ese Dataset a un Trainer creado con anterioridad que, como su nombre indica, se encargará de entrenar el modelo de Red Neuronal con los datos proporcionados. Por último se procede a la clasificación y a la validación de la red con el Dataset de testeo.

Ahora se mostrarán unos ejemplos de código en la consola de Python para la creación de redes, trainers, datasets, etc.

- patternDS = SupervisedDataSet(numcols-1, 10)

  Este ejemplo muestra la creación de un Dataset, donde los dos parámetros pasados al constructor son el número de parámetros distintos que se observarán y las distintas salidas que podría tener la red, por ese orden.

- net = buildNetwork(numcols-1, numhidden, 10, bias=True)

  Las redes se crean con la función "buildNetwork" al que se deben pasar el número de parámetros, el número de neuronas ocultas que se desean, las posibles salidas e indicar si se quiere usar un Bias.

- trainer = BackpropTrainer(net, trainDS, learningrate=myLearningRate, momentum=myMomentum)

  El trainer es bastante sencillo, solo es necesario que se le pasen la red, el dataset de entrenamiento y los parámetros de Learning Rate y Momentum, que suelen ser menos que 1 y mayores que cero.

- trainerror = trainer.trainUntilConvergence(verbose=True, trainingData=trainDS, validationData=validDS, maxEpochs=10)

  Por último se entrena la red a través de una de las funciones del trainer, en este caso se indica si se desea usar el modo Verbose, los dataset de entrenamiento y validación, así como el número máximo de iteraciones. En 'trainerror' se contendrán dos vectores con los errores de entrenamiento y validación respectivamente que podrán ser representados en una gráfica posteriormente para tener un resultado visual del resultado del entrenamiento.

4. ¿Cómo se llegó a elegir PyBrain?

Obviamente se barajaron varias alternativas antes de que la balanza se inclinara por utilizar PyBrain. La primera duda fue si escoger una herramienta con entorno gráfico o una de las muchas librerías para varios lenguajes que facilitan la programación de redes neuronales. Se decidió usar una librería por su flexibilidad, las herramientas con entorno gráfico, a pesar de su facilidad de uso, suelen no incluir todas las características posibles. Por poner un ejemplo, un programa podría no incluir un visor para las gráficas de aprendizaje, pero con una librería se pueden añadir funcionalidades con otras librerías y realizar dichas tareas.

Una vez elegido esto, era necesario elegir un lenguaje de programación en el que buscar una librería y se eligió Python debido a que es un lenguaje interpretado y no compilado y para realizar los cientos de pruebas que son necesarias implican cientos de ejecuciones y es mejor que sea en un script.

Las búsquedas de librerías de redes neuronales para Python arrojaron dos resultados principales: PyBrain y Neurolab. Esta última fue descartada por dos motivos:

- PyBrain funciona de manera que creas la red neuronal y posteriormente se le aplica el algoritmo de aprendizaje, pero Neurolab relaciona ambas cosas con una misma función del constructor, por lo que se hace imposible aplicar varias funciones de aprendizaje a la misma red sin crearla dos veces.
- PyBrain tiene su propio tipo de datos para gestionar las relaciones entre las entradas y los targets en el aprendizaje supervisado, los ‘Datasets’, algo que Neurolab no tiene, lo que dificulta la gestión de los datos.

5. Conclusiones

La librería parece bastante completa para lo que se desea realizar en este proyecto, contando con varios métodos de entrenamiento de redes neuronales supervisadas, métodos de visualización externos usando la librería ‘matplotlib’ y realiza todo esto bastante ágilmente, pero se echa en falta una interfaz gráfica de usuario como con la que cuenta el Microsoft Azure Machine Learning Studio que ayude a una mejor compresión de los datos, y el flujo del proceso.