
Generación procedural de contenido mediante aprendizaje automático



TRABAJO DE FIN DE GRADO

Víctor Emiliano Fernández Rubio

Gonzalo Guzmán del Río

Carlos Llames Arribas

Tutor: Samir Genaim

Facultad de Informática

Universidad Complutense de Madrid

29 de Febrero 2020

Documento maquetado con TEXIS v.1.0+.

Agradecimientos

Nos gustaría agradecer la realización y el desarrollo de este proyecto, en especial a nuestro tutor de TFG, Samir Genaim, el cuál nos ha ayudado en todo lo que le hemos ido preguntando, además de proponernos nuevas ideas sobre las que investigar. Además, agradecer también su predisposición a realizar reuniones virtuales en estos meses difíciles. También nos gustaría destacar a nuestros familiares, y compañeros y amigos que nos han ayudado a tomar ciertas decisiones, esenciales para poder desarrollar nuestro trabajo de manera correcta. Por último nos gustaría destacar nuestro trabajo tanto individual como colectivo, todas las charlas que hemos tenido y las propuestas de cada uno, lo que ha permitido llevar el trabajo siempre por el camino correcto.

Acknowledgment

We would like to thank the development and realization of this project to our TFG director, Samir Genaim, who has helped us on everything that has been asked to him, and also his proposals about new ideas to investigate on. Beside, we want to thank his predisposition to hold virtual meetings in all these difficult months. We would also like to highlight the support of all our families, colleagues and friends, who have helped us to take essential decisions in order to a correct development of the project. Finally, we want to stand out, our individual and collective work, all the meetings that we have had and each other's proposals, what have allowed us to go always the right way.

Resumen

Durante los últimos años, tanto la inteligencia artificial como el aprendizaje automático, se han convertido en un foco constante de investigación y enseñanza, así como de aprendizaje. Además cada vez más empresas, ven a estas técnicas como un punto de partida hacia su crecimiento tanto económico como tecnológico, permitiendo a estas, entrar en otros sectores. Por ejemplo Microsoft, empezó a adentrarse en el mundo de la inteligencia artificial y aprendizaje automático con Kinect, o Google desarrollando un algoritmo capaz de derrotar a los mejores jugadores del mundo de Dota. En otros sectores como en el de la agricultura, la inteligencia artificial está siendo utilizada, para mejorar la eficiencia en cuanto a producción, prediciendo los rendimientos de la cosecha. Además todo lo comentado anteriormente, nos lo encontramos hoy en día y vivimos con ello, destacando entre otros a asistentes personales como Alexa o Siri.

Debido a esto, hemos planteado nuestro trabajo de fin de tal manera, que se nos presenta una oportunidad única de aprender como funcionan los algoritmos de aprendizaje automático e inteligencia artificial, aplicándola al ámbito sobre el que hemos desarrollado nuestros estudios durante los últimos años, los videojuegos, y en concreto a la creación de mapas del videojuego SuperMario.

A lo largo de todo el proyecto, investigaremos acerca de cuales son las mejores técnicas tanto de aprendizaje automático como de inteligencia artificial, así como cuál es la forma más óptima de implementarlas. Desarrollaremos scripts para comprobar su funcionamiento, con la ayuda de diversas librerías entre las que se encuentran Tensorflow o NLTK, así como una aplicación en Unity, la cual nos servirá de base para poder mostrar los mapas que vayan siendo generados. Esta aplicación, permitirá la posibilidad de mostrar los 8 mapas originales de los que disponemos, los cuales han sido realizados a mano, así como crear nuevos mapas, con algunos de los algoritmos investigados e implementados. Estos mapas podrán ser monotema, que resultan a partir de un solo mapa, o multitema, que se crean a partir de la unión de diferentes mapas.

Palabras clave

Generacion procedural de contenido, aprendizaje automatico, inteligencia artificial, Tensorflow, nltk, Mario Bros, clasificación de texto, modelos del lenguaje, procesamiento de lenguajes naturales, n-gramas, redes neuronales.

Summary

During the last years, both the artificial intelligence and machine learning have become in a constant focus of research and teaching, and also of learning. Besides, more and more companies, see these techniques as an starting point to their economical and technological growing, by entering other sectors. For example Microsoft, stepped into in artificial intelligence and machine learning with Kinect, or Google developing an algorithm able to beat the best Dota players all over the world. In other sectors like the agricultural, the artificial intelligence is being used to improve the production efficiency by predicting the crop yields. Also, everything that has been commented above, can be found nowadays in our lifes, highlighting for example, voice assistants such as Alexa or Siri.

Because of all of this, we have raised our final degree project as a unique opportunity to learn how the machine learning and artificial intelligence algorithms work, by applying it on what we have been studying during the last years, the video games, and in particular the map generation based on SuperMario.

Throughout all the project, we will research about which are the best techniques of machine learning and artificial intelligence, and also which is the best way of implementing them. We will develop scripts in order to check the behaviour, with the help of some libraries such as Tensorflow or NLTK, and also a Unity project, which will serve us as a base to show all the maps that are being generated. This application will give the possibility of showing the first eight original maps, and also creating new ones, with some of the researched and implemented algorithms. These maps will be mono theme, from just one map, or multi theme, created from the join of different maps.

Keywords

Procedural content generation, machine learning, artificial intelligence, Tensorflow, nltk, Mario Bros, text classification, language models, natural language processing, n-grams, neural networks.

Índice

Agradecimientos	III
Acknowledgment	V
Resumen	VII
Summary	IX
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	1
1.3. Metodología	2
1.4. Plan de trabajo	3
1. Introduction	5
1.1. Motivation	5
1.2. Goals	5
1.3. Methodology	6
1.4. Working plan	7
2. Estado del Arte	9
2.1. Historia	9
2.2. Actualidad	13
2.2.1. Herramientas para el desarrollo software	13
2.2.2. Aplicación en los sectores económicos	15
2.2.3. Aplicaciones cotidianas	19
2.3. En desarrollo	21
3. Aprendizaje Automático	25
3.1. Definición	25
3.2. Tipos de Algoritmos	26
3.2.1. Algoritmos supervisados	26
3.2.2. Algoritmos no supervisados	38
3.2.3. Aprendizaje por refuerzo	41
3.3. Conclusión	44
4. Generación Procedural de Contenido	47

4.1. Definición	47
4.2. Origen y evolución	47
4.3. Procedural y aleatorio	48
4.4. Clasificación y taxonomía	49
4.4.1. PCG y aprendizaje automático	51
4.5. Desarrollo	52
4.6. Conclusión	52
5. Aplicación Práctica	55
5.1. Introducción	55
5.2. Nuestro Juego	56
5.3. Retos	60
5.4. Tipos de datos	61
5.5. Algoritmos	63
5.5.1. N-gramas	63
5.5.2. Redes neuronales recursivas	75
5.5.3. Redes neuronales convolucionales	80
5.6. Resultados	80
5.6.1. N-gramas	80
5.6.2. Redes neuronales recursivas	84
5.6.3. Redes neuronales convolucionales	84
5.6.4. Comparativa	84
6. Conclusiones	85
6.1. Conclusiones	85
6.2. Posibilidades	86
6.3. Limitaciones	87
6.4. Ampliaciones	87
7. Contribuciones	89
7.1. Víctor Emiliano Fernández Rubio	89
7.2. Gonzalo Guzmán del Río	89
7.3. Carlos Llames Arribas	90
7.4. Repositorios	91
Bibliografía	93

Índice de figuras

2.1.	Test de turing	9
2.2.	Carro de Stanford	10
2.3.	Kasparov jugando contra DeepBlue al ajedrez	10
2.4.	Kinect de Microsoft	11
2.5.	Laboratorios Google X	11
2.6.	OpenAI	12
2.7.	DeepMind jugando al juego de mesa chino Go	12
2.8.	Tensorflow	13
2.9.	Google Cloud ML Engine	14
2.10.	AWS Machine Learning	14
2.11.	Matplotlib	15
2.12.	InnerEye	16
2.13.	Aprendizaje automático de un coche autónomo	17
2.14.	Metal Gear Solid 5 y Red Dead Redemption 2	19
2.15.	Aprendizaje automático en asistentes personales	20
2.16.	Reconocimiento facial de Facebook	21
3.1.	Regresión Logísitca perceptrón	27
3.2.	Sigmoide	28
3.3.	Exactitud y precisión	29
3.4.	Redes Neuronales más comunes	31
3.5.	Arquitectura general de un algoritmo de conjuntos	32
3.6.	Esquema del método Averaging	33
3.7.	Esquema del método Boosting	33
3.8.	Representación hiperplano bidimensional y tridimensional	34
3.9.	Márgenes	35
3.10.	Representación de la regularización	36
3.11.	Representación de gamma	36
3.12.	Clustering jerárquico	39
3.13.	Basado en el centroide	39
3.14.	Basado en la densidad	40
3.15.	Basado en la distribución	40
3.16.	Aprendizaje por refuerzo	42
4.1.	Taxonomía según el contenido	49
5.1.	Fragmento del mapa 1 original del primer mundo	55

5.2. ESBR E (Todo el mundo)	57
5.3. PEGI 3 (Tres años y mayores)	57
5.4. CERO A (Todas las edades)	57
5.5. ACB L (General)	57
5.6. Mario	58
5.7. Monedas	59
5.8. Seta mágica	59
5.9. Mario	59
5.10. Super Mario	59
5.11. Primer mundo del Super Mario Bros	60
5.12. Slice de un nivel	61
5.13. División de las zonas de los niveles en tiled	62
5.14. Prefab de un tile	63
5.15. Conversión de texto a su representación en bolsas de palabras	68
5.16. Tabla del conteo de bigrams para ocho palabras en un corpus	73
5.17. Tabla de las probabilidades de bigrams para ocho palabras en un corpus	73
5.18. Tabla de las probabilidades, suavizadas con add-1, de bigrams para ocho palabras en un corpus	73
5.19. Cada tile (bloque) considerado como una palabra	81
5.20.	81
5.21.	81
5.22.	81
5.23. Niveles generados por bigramas con corpus de entrenamiento el nivel 1-1	82
5.24. Niveles generados por trigramas con corpus de entrenamiento el nivel 1-1	82
5.25. Niveles generados por cuatrigramas con corpus de entrenamiento el nivel 1-1	82
5.26. Nivel 1-1	83
5.27. Nivel 1-4	83
5.28. Nivel generado por bigramas con corpus de entrenamiento el nivel 1-1 y 1-4	83
5.29. Nivel generado por bigramas con corpus de entrenamiento el nivel 1-4 y 1-1	83
5.30. Niveles generados por trigramas con corpus de entrenamiento el nivel 1-1 y multiples niveles generados con n-gramas a partir del nivel 1-1 .	83

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación

En estos años, la inteligencia artificial y el aprendizaje automático se han convertido en una tecnología muy popular en todo el mundo, accesible por todos y con muchas proyecciones de cara al futuro. Además, grandes compañías como Google, Apple, Amazon o Microsoft están invirtiendo en investigación y desarrollo de proyectos de inteligencia artificial y aprendizaje automático, además de todas las nuevas empresas centradas únicamente en esto.

Esta tendencia solo aumentará en el futuro, ya que se estima que la industria de la inteligencia artificial alcanzará los 118 mil millones de dólares en 2025.

No obstante, hay otra parte más oscura acerca del aprendizaje de las máquinas. Stephen Hawking dijo: "la inteligencia artificial probablemente sea lo mejor o lo peor que le puede pasar a la humanidad". De momento no hemos visto lo peor y es muy difícil predecir cómo se desarrollará la inteligencia artificial en el futuro y si en algún momento se volverá plenamente consciente. Sin embargo, ya se conocen muchos de los problemas que podrían ocasionar tomar decisiones sin razonamientos humanos, basándose solamente en datos.

Pero aparte de los peligros, la inteligencia artificial tiene una gran cantidad de ventajas que nos pueden facilitar la vida a todos. Además, es aplicable a todos los sectores económicos. Actualmente, sectores como el automovilístico o el sanitario se están beneficiando de estos algoritmos de aprendizaje automático, y los videojuegos no están excluidos. Muchos videojuegos utilizan el aprendizaje automático durante su desarrollo para facilitar y ahorrar tiempo y costes a los desarrolladores e incluso durante el juego se utilizan inteligencias artificiales para dar una sensación más realista. Esto va a continuar evolucionando, sobretodo a la hora de jugar, para poder lograr una experiencia completamente inmersiva en el propio juego.

1.2. Objetivos

El objetivo general de este proyecto es la investigación y el análisis de las distintas formas para generar mapas o niveles y en última instancia, implementar una pequeña aplicación demo con la que probar los diferentes resultados de cada modelo.

Para poder alcanzar ese objetivo final se han establecido una serie de objetivos específicos:

1. Investigar sobre la clasificación de texto y sobre los modelos del lenguaje más usados, tales como Bolsa de Palabras (Bag of Words), N-gramas y Redes Neuronales.
2. Estudiar el modelo N-gramas e implementar nuestro propio modelo de este tipo para entender su funcionamiento en la práctica.
3. Comparar el comportamiento del modelo N-gram en situaciones con distintos tamaños de corpus, es decir, datos para entrenar.
4. Comparar el rendimiento de N-grams con un tamaño variable respecto a N-grams con un tamaño fijo.
5. Estudiar el modelo de Redes Neuronales para la clasificación y generación de texto.
6. Comparar las distintas posibles Redes Neuronales y así como los posibles parámetros variables.
7. Implementar distintos modelos anteriormente estudiados, así como su adaptación para la generación de los niveles.
8. Comparar todos los modelos nombrados anteriormente.
9. Crear una aplicación demo para mostrar los diferentes resultados generados con los modelos.

Para ello, utilizaremos como ejemplo el juego de Super Mario Bros, un juego de plataformas 2D horizontal, de la Nintendo Entertainment System, NES.

1.3. Metodología

Para realizar los objetivos citados anteriormente se investigarán fuentes en internet, artículos científicos, estudios previos y libros, todos ellos reflejados en la bibliografía y webgrafía. Estos recursos ayudaran a la hora de investigar entre los diferentes modelos existentes, así como la comparación entre ellos y su implementación para su posterior demostración.

Una vez completada la investigación teórica, se estudiarán las diferentes herramientas para el estudio de los modelos elegidos. Con el fin de ver los puntos fuertes y débiles de cada algoritmo en su implementación, ejecución y resultados obtenidos. A continuación, se pondrán a prueba dichos modelos. Se implementarán algoritmos para la generación de niveles y se compararán tanto los resultados como sus tiempos de ejecución o su complejidad de implementación.

Por último se desarrollará una aplicación donde poder explotar al máximo los modelos elegidos. De esta forma, obtener unos resultados y establecer unas conclusiones acerca de la generación procedural mediante aprendizaje automático respecto a la

generación de mapas en los videojuegos.

Para la creación de la aplicación, se utilizará como entorno de desarrollo Unity 2019.2.1f1, uno de los motores de videojuegos más conocidos y punteros en la industria. Este motor está presente en la facultad de informática de la Universidad Complutense de Madrid para el estudio a lo largo del grado de desarrollo de videojuegos. Gracias a esto, nos resultará más cómodo el desarrollo de nuestra aplicación.

A la hora de implementar los algoritmos de aprendizaje automático se utilizarán librerías en Python, debido a esto se utilizará el entorno de desarrollo de Jupyter NoteBook de Anaconda. Presenta una interfaz muy clara y sencilla de usar a la hora de probar scripts en Python.

A la hora de implementar los scripts definitivos se utilizará el editor de texto preferido por cada integrante, Sublime, Atom, Notepad++.

Para la construcción de los mapas modelo del videojuego se utilizará el programa Tiled. Muy sencillo de manejar, permitiendo exportar los mapas en varios formatos.

El sistema de control de versiones para el proyecto de Unity será Github. Las herramientas para comunicación entre el equipo serán Discord y Google Meets.

1.4. Plan de trabajo

La primera parte del trabajo consistirá en investigar que algoritmos se han usado hasta ahora para la generación de niveles en 2D. Tras tener unos modelos de referencia, empezaremos por implementar algoritmos básicos para ver su comportamiento ante pequeños ejemplos genéricos. Los modelos escogidos tras el estudio son el algoritmo de predicción de texto de N-grams y las redes neuronales recursivas.

Posteriormente se hará una implementación más específica a nuestro tema concreto. El modelo de N-gram y la primera implementación de la red neuronal recursiva se basaban en la predicción de texto debido a como hemos decidido abordar el problema de la generación de niveles. Debido a esto, las implementaciones sencillas se basarán en simples predicciones de palabras y la generación de pequeños fragmentos de texto.

Seguidamente se adecuarán a generar niveles de videojuegos. Y más tarde se plantearán resolver problemas e incompatibilidades de tal forma que se modificará el algoritmo para adaptarlo al campo de los videojuegos, manteniendo su esencia de predicción de texto.

Por último, una vez repetido este proceso por cada modelo estudiado, se compararán dichos modelos y se pondrán a prueba en una aplicación. Esta aplicación se basará en la generación de mapas y una pequeña interacción con ellos. Como hemos mencionado anteriormente el juego elegido para el estudio es el Super Mario Bros

de la consola NES.

Capítulo 1

Introduction

1.1. Motivation

In these years, artificial intelligence and machine learning have become a very popular technology worldwide, accessible to everyone and with many projections for the future. Moreover, large companies such as Google, Apple, Amazon or Microsoft are investing in research and development of artificial intelligence and machine learning projects, in addition to all the new companies focused solely on this.

This trend will only increase in the future, since it is estimated that the artificial intelligence industry will reach 118 billion dollars in 2025. However, there is a darker side about machine learning. Stephen Hawking once said: "artificial intelligence is probably the best or worst thing that can happen to mankind". So far we have not seen the worst and it is very difficult to predict how artificial intelligence will develop in the future and whether it will ever become fully conscious. However, many of the problems that could lead to making decisions without human reasoning, based on data alone, are already known.

But apart from the dangers, artificial intelligence has a lot of advantages that can make life easier for all of us. Moreover, it is applicable to all sectors of the economy. Currently, sectors such as the automotive and health sectors are taking advantage from these automatic learning algorithms, and video games are not excluded. Many video games use machine learning during their development to make it easier and save time and costs for developers. Even during the game, artificial intelligence is used to give a more realistic feeling. This will continue to evolve, especially when playing games, in order to achieve a fully immersive experience in the game itself.

1.2. Goals

The general objective of this project is the investigation and analysis of the different ways to generate maps or levels and ultimately implement a small demo application with which to test the different results of each model.

In order to achieve this final goal, a series of specific objectives have been established:

1. Research on text classification and on the most used language models, such as Bag of Words, N-grams and Neutral Networks.
2. Study the N-gram model and implement our own model of this type to understand how it works in practice.
3. Compare the behavior of the N-gram model in situations with different corpus sizes, that is, data for training.
4. Compare the performance of N-grams with a variable size with respect to N-grams with a fixed size.
5. Study the Neural Network model for classification and text generation.
6. Compare the different possible Neural Networks and the possible variable parameters.
7. To implement different models previously studied, as well as their adaptation for the generation of levels.
8. Compare all the models mentioned above.
9. Create a demo application to show the different results generated with the models.

For this, we will use as an example the game of Super Mario Bros, a 2D horizontal platform game, of the Nintendo Entertainment System, NES

1.3. Methodology

In order to achieve the objectives mentioned above, Internet sources, scientific articles, previous studies and books will be researched, all of which are reflected in the bibliography and webgraphy.

These resources will help in the research between the different existing models, as well as the comparison between them and their implementation for their later demonstration. Once the theoretical research is completed, the different tools for the study of the chosen models will be studied. In order to see the strengths and weaknesses of each algorithm in its implementation, execution and results obtained.

These models will then be tested. Algorithms for the generation of levels will be implemented and both the results and their execution times or implementation complexity will be compared. Finally, an application will be developed where the chosen models can be exploited to the full. In this way, to obtain some results and to establish some conclusions about the procedural generation by means of automatic learning with respect to the generation of maps in the video games.

For the creation of the application, Unity 2019.2.1f1, one of the best known and leading video game engines in the industry, will be used as the development environment. This engine is present in the faculty of computer science at the Universidad

Complutense de Madrid for the study along the degree of development of video games. Thanks to this, we will be more comfortable developing our application.

When implementing the automatic learning algorithms, Python libraries will be used, due to this, Anaconda's Jupyter NoteBook development environment will be used. It presents a very clear and easy to use interface when testing scripts in Python. When implementing the final scripts, the preferred text editor for each member will be used, Sublime, Atom, Notepad++.

For the construction of the model maps of the video game, the program Tiled will be used. Very easy to use, allowing the export of the maps in various formats. The version control system for the Unity project will be Github. The tools for communication between the team will be Discord and Google Meets.

1.4. Working plan

The first part of the work will be to investigate what algorithms have been used so far for the generation of 2D levels. After having some reference models, we will start by implementing basic algorithms to see how they behave when faced with small generic examples. The models chosen after the study are the N-grams text prediction algorithm and the recursive neural networks.

A more specific implementation will be made later on to our specific theme. The N-gram model and the first implementation of the recursive neural network were based on text prediction because of how we decided to address the problem of level generation. Because of this, simple implementations will be based on simple word predictions and the generation of small text fragments.

They will then be adapted to generate video game levels. And later on, problems and incompatibilities will be solved in such a way that the algorithm will be modified to adapt it to the field of video games, keeping its essence of text prediction.

Finally, once this process is repeated for each model studied, these models will be compared and tested in an application. This application will be based on the generation of maps and a small interaction with them. As we mentioned earlier, the game chosen for the study is the Super Mario Bros. for the NES system.

Capítulo 2

Estado del Arte

2.1. Historia

El primer paso a lo que hoy conocemos como aprendizaje automático fue dado por McCulloch, un neurofísico, y Walter Pitts, un matemático. Decidieron, en 1943, crear un modelo de aprendizaje automático basado en un circuito eléctrico a partir del cual nacieron las redes neuronales. Sin embargo, Alan Turing ya había comenzado a estudiar el cerebro como una forma de ver el mundo computacional en 1936.

Otro gran avance fue en 1949 propuesto por Donald Hebb que escribió un libro basándose en el aprendizaje psicológico. Fue el primero en explicar los procesos del aprendizaje desde un punto de vista psicológico. Hoy día sigue siendo el fundamento de muchas funciones de las redes neuronales.

Tras esto Alan Turing publica en 1950 un artículo titulado "Computación e Inteligencia", en donde planteaba lo que hoy conocemos como la Prueba de Turing. Esta prueba consiste en que una máquina consiga mostrar un comportamiento inteligente de la misma forma que un ser humano.

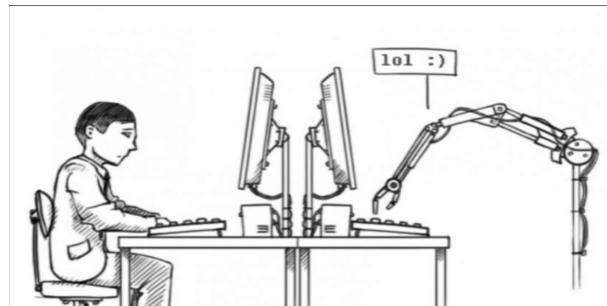


Figura 2.1: Test de turing

En 1952, Arthur Samuel desarrolla un programa capaz de aprender. El programa simplemente jugaba a las damas y era capaz de aprender de los errores cometidos en la partida anterior.

Frank Rosenblatt comenzó a desarrollar en 1957 el perceptrón. Una red neuronal que sigue utilizándose hoy en día. Este modelo es capaz de reconocer caracteres

y patrones. Sin embargo, tenía limitaciones como el problema de la función XOR, OR-exclusiva, incapaz de clasificar clases no separables linealmente.

En 1979, estudiantes de la diversidad de Stanford, diseñaron un carro capaz de cruzar una habitación llena de obstáculos moviéndose autónomamente. El carro tardó 5 horas, pero logró pasar la prueba con éxito.

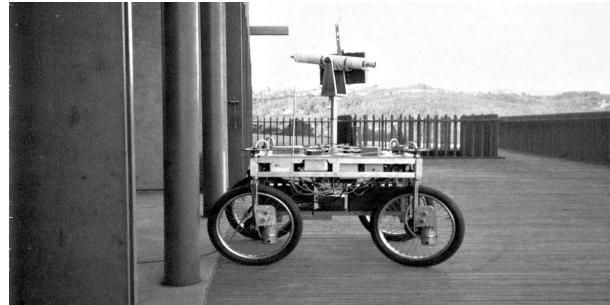


Figura 2.2: Carro de Stanford

En 1981, Gerald DeJong introduce el concepto de aprendizaje automático basado en la experiencia, haciendo que un computador analice información de entrenamiento y cree una regla general que le permita descartar información no importante.

Terry Sejnowski inventa NetTalk, un programa capaz de aprender y pronunciar palabras como lo haría un niño en 1985.

A finales de 1980 y principios de 1990 apenas hubo grandes descubrimientos en el campo del aprendizaje automático. Sin embargo, en 1996, el ordenador Deep Blue consigue ganar a Gary Kasparov una partida de ajedrez, aunque seguidamente Kasparov gano 3 partidas. En mayo de 1997, vuelven a enfrentarse Kasparov y una versión mejorada del ordenador, el Deep Blue. Esta vez el ganador fue la computadora tras 6 partidas.



Figura 2.3: Kasparov jugando contra DeepBlue al ajedrez

En el año 2006, Geoffrey Hinton presenta el concepto de Deep Learning o aprendizaje profundo. Gracias a esto se explicaron los algoritmos que permiten que los ordenadores distingan objetos y textos tanto en imágenes como en videos.

En 2010, el Kinect de Microsoft es capaz de reconocer 20 características corporales a una velocidad de 30 veces por segundo.



Figura 2.4: Kinect de Microsoft

En 2011 el ordenador Watson de IBM vence a dos inteligentes concursantes en la tercera ronda del concurso estadounidense de preguntas y respuestas Jeopardy.

Entre 2011 y 2012 se crea Google Brain por Jeff Dean de Google y Andrew profesor de la Universidad de Stanford. El proyecto consistía en utilizar toda la infraestructura de Google para detectar patrones en videos e imágenes.

En 2012 los laboratorios Google X pasan a llamarse X. Estos desarrollaron un algoritmo capaz de buscar e identificar gatos en los vídeos de YouTube. También AlexNet ganó la competición de ImageNet, lo que llevó al uso de GPUs y redes neuronales convolucionales en el aprendizaje automático. Además, crearon ReL., una función de activación que mejoraba en gran medida las CNN.



Figura 2.5: Laboratorios Google X

En 2014 un programa de ordenador logró convencer a más del 30 % de los jueces que era humano. Se trata de un chatbot que responde al nombre de Eugene Goostman, el programa fue capaz de convencer a los jueces que participaron en la prueba de que estaban chateando con un niño ucraniano de 13 años. Además, Facebook desarrolla DeepFace, un algoritmo de software que puede reconocer individuos en fotos al mismo nivel que los humanos.

En el 2015 ocurrieron una gran cantidad de avances en el campo del aprendizaje automático. Amazon lanza su propia plataforma de aprendizaje automático. Microsoft crea el kit de herramientas para el aprendizaje de máquinas, que permite la distribución eficiente de problemas de aprendizaje automático en múltiples computadoras. Google entrena un agente conversacional que no solo puede interactuar con

humanos como un servicio de soporte técnico, sino también discutir la moralidad, expresar opiniones y responder preguntas generales basadas en hechos. Es fundada OpenAI, compañía de investigación de inteligencia artificial sin fines de lucro que tiene como objetivo promover y desarrollar inteligencia artificial de manera que beneficie a la humanidad. Entre sus fundadores se encuentra Elon Musk. Debido a estos grandes avances obtenidos en el área de la inteligencia artificial, más de 3000 investigadores de estas áreas firman una carta abierta advirtiendo del peligro de las armas autónomas.



Figura 2.6: OpenAI

En el año 2016 el algoritmo desarrollado por Google, DeepMind, logró ganar cinco juegos de cinco a un jugador profesional en el juego de mesa chino Go, que es considerado el juego de mesa más complejo del mundo.



Figura 2.7: DeepMind jugando al juego de mesa chino Go

Un algoritmo desarrollado por OpenAI en 2017 derrota a los mejores jugadores de Dota 2 en partidos 1 contra 1.

Hoy día no hay prácticamente ningún sector que no se beneficie del aprendizaje automático por lo que existen una gran variedad de proyectos. Las tendencias en 2019 son olvidar datos mediante el aprendizaje automático o desaprendizaje automático, hay ocasiones en que es mucho más beneficioso que algunos datos sean olvidados por el sistema. Interpolariedad entre redes neuronales. La convergencia del Internet de las cosas y el aprendizaje automático.

2.2. Actualidad

2.2.1. Herramientas para el desarrollo software

En la actualidad el desarrollo del aprendizaje automático en los distintos sectores económicos ha aumentado enormemente, así como la gran cantidad de herramientas para sus desarrolladores.

Conocer las distintas herramientas y saber cuáles usar para el desarrollo puede ser de gran ayuda a la hora de desarrollar tu software. Las principales herramientas de aprendizaje automático hoy día son:

1. Tensorflow

Desarrollada por el equipo de Google, Tensorflow dispone de un esquema flexible de herramientas, bibliotecas y recursos que permite a los investigadores y desarrolladores crear aplicaciones de aprendizaje automático. Entre sus características principales destaca la ayuda a la hora de construir y entrenar sus modelos. Ofrece un ciclo completo del aprendizaje automático en todas sus fases. Es un software de código abierto y muy flexible que ofrece la posibilidad de ejecutarse en CPU y GPU.



Figura 2.8: Tensorflow

2. Google Cloud ML Engine

Es una plataforma alojada donde los desarrolladores de aplicaciones de aprendizaje automático ejecutan modelos de aprendizaje automático de calidad óptima. Sus principales características son proporcionar modelos de entrenamiento, construcción, aprendizaje profundo y modelos predictivos. Los servicios de predicción y entrenamiento pueden ser usados independientemente uno del otro si así se desea. Es un software utilizado por las empresas, destacando su uso en la respuesta a los emails de clientes.



Figura 2.9: Google Cloud ML Engine

3. Amazon Machine Learning

Es un software robusto de aprendizaje automático basado en la nube, que puede ser usado por cualquier desarrollador web o móvil. Sus principales características son los asistentes y herramientas visuales que proporciona. Admite tres tipos de modelos, es decir, clasificación multiclase, clasificación binaria y regresión. Permite a los usuarios crear objetos desde bases de datos MySQL y desde datos almacenados en Amazon Redshift.



Figura 2.10: AWS Machine Learning

4. Accord.Net

Aprendizaje automático .Net que se combina con bibliotecas de procesamiento de imágenes y audio escritas en C#. Consiste en un conjunto de bibliotecas para el reconocimiento de patrones, procesamiento de datos estadísticos, álgebra lineal.

Aparte de estas existen muchas más herramientas para el desarrollo de aplicaciones de aprendizaje automático. Estas herramientas de desarrollo suelen ir acompañadas de muchas otras para poder interpretar, analizar y comparar resultados. Algunos ejemplos de herramientas que ayudan al aprendizaje automático son Weka o matplotlib. Weka es un software de aprendizaje automático en Java que tiene una amplia gama de algoritmos de aprendizaje automático para tareas de minería de datos. Matplotlib es una biblioteca de aprendizaje automático basada en Python. Es útil para una visualización de calidad. Básicamente, es una biblioteca de trazado de Python.

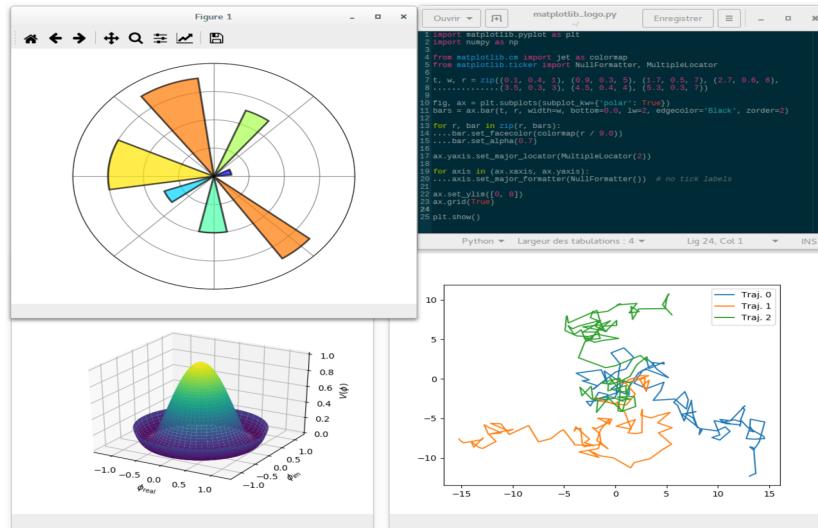


Figura 2.11: Matplotlib

2.2.2. Aplicación en los sectores económicos

La mayoría de las industrias que trabajan con una gran cantidad de datos han reconocido el valor del aprendizaje automático. Coleccionando ideas de esta gran cantidad de datos, la industria es capaz de trabajar de una forma más eficaz para obtener ventaja ante sus competidores. Algunos de los sectores que más se benefician del hecho de aplicar el aprendizaje automático a su sector son:

1. La industria de la salud

Desde principios del 2013 el aprendizaje automático se ha utilizado en el mundo de la medicina. Un ejemplo fue Google DeepMind, después de ganar la serie de juegos contra el mejor jugador del mundo de GO, el equipo de Google decidió apoyar proyectos de medicina con su tecnología. Muchas de las nuevas empresas de la industria del aprendizaje automático están ayudando a la atención médica. Un ejemplo es:

El diagnóstico a partir de imágenes médicas. Gracias al aprendizaje profundo automático y al tratamiento de imágenes Microsoft ha desarrollado InnerEye, una herramienta que está actualmente trabajando en dar diagnósticos a partir de imágenes. Sin embargo, se sabe que las aplicaciones de aprendizaje profundo tienen una capacidad explicativa limitada. En decir, este sistema de aprendizaje automático no puede explicar cómo llegó a sus predicciones, incluso cuando son correctas.

Otro ejemplo son consultas de tratamiento y sugerencias. El diagnóstico es un proceso muy complicado que involucra una gran cantidad de factores que las máquinas no pueden clasificar. Sin embargo, hay pocas dudas de que una máquina podría ayudar a los médicos a tomar las consideraciones correctas en el diagnóstico y el tratamiento.

El descubrimiento de medicamentos. Si bien gran parte de la industria de la salud es un cúmulo de leyes de varias partes interesadas, el descubrimiento de medicamentos se destaca como algo relativamente sencillo para el aprendizaje

automático.



Figura 2.12: InnerEye

2. La industria financiera

El aprendizaje automático en finanzas se está desarrollando rápidamente, hay docenas de opciones para su uso en el sector financiero, algunas de las más relevantes son:

Evaluación de solvencia de crédito. La Inteligencia Artificial ayuda a los bancos a emitir créditos con mayor confianza a quienes aprueban las verificaciones del sistema. Para esto, los programas y algoritmos analizan toda la información disponible, estudian su historial de crédito, los cambios en su nivel de salarios y, sobre esta base, determinan la confiabilidad del cliente y la seguridad del préstamo.

Toma de decisiones. Esta es una tarea global que se resuelve con éxito mediante la introducción de AI y ML en los servicios financieros. Cuando un algoritmo puede analizar todos los datos estructurados y no estructurados disponibles, tanto internos como externos, solicitudes de clientes y sus acciones en las redes sociales, una institución financiera puede descubrir tendencias útiles y peligrosas. Ayuda a evaluar los niveles de riesgo y permite a las personas tomar las decisiones más informadas.

Protección contra el fraude. Los bancos y los sistemas de pago ya han desarrollado modelos para identificar y bloquear la mayoría de las transacciones fraudulentas. Estos modelos se basan en el historial de transacciones del cliente, así como en el comportamiento del cliente en Internet. Los sistemas basados en inteligencia artificial que detectan fraudes en línea se han desarrollado a partir de tecnologías de Big Data.

3. La industria del transporte

Los casos de uso de IA en el transporte justifican por qué el mercado está

experimentando un aumento al alza y por qué las empresas deberían adoptar la tecnología.

Vehículos sin conductor. Una de las aplicaciones más innovadoras de la IA son los vehículos autónomos. Aunque las personas se mostraron escépticas sobre esta tecnología durante sus etapas de desarrollo, los vehículos sin conductor ya ingresaron al sector del transporte. Los taxis autónomos ya comenzaron a operar en Tokio. Sin embargo, por razones de seguridad el conductor se sienta en el automóvil para tomar el control del taxi en caso de emergencia. Del mismo modo, Estados Unidos está adoptando camiones autónomos para obtener beneficios. Por ahora, la mayoría de las compañías todavía están ejecutando sus proyectos piloto, esforzándose por hacer que los vehículos autónomos sean perfectos y seguros para los pasajeros. A medida que esta tecnología evoluciona, los vehículos autónomos ganarán una gran confianza y se convertirán en la corriente principal en el ámbito del consumidor.

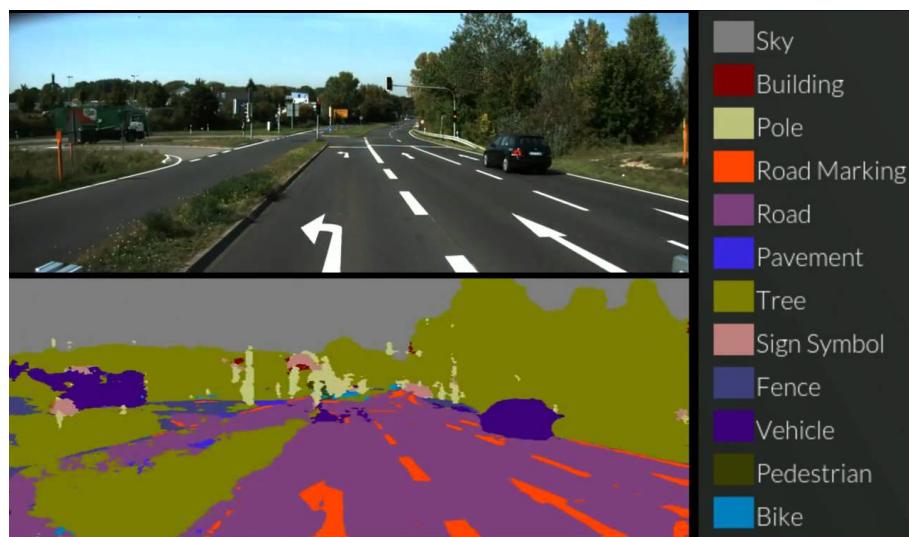


Figura 2.13: Aprendizaje automático de un coche autónomo

La gestión del tráfico. Otro problema de transporte al que las personas enfrentan a diario es la congestión del tráfico. La IA ahora está configurada para resolver este problema. Los sensores y las cámaras integradas en todas partes en las carreteras recogen la gran cantidad de información acerca del tráfico. Estos datos se analizan y se obtiene un patrón de tráfico. De esta forma, la inteligencia artificial se puede utilizar para no solo reducir el tráfico no deseado, sino también para mejorar la seguridad vial y reducir los tiempos de espera.

Predicciones de retraso en el transporte aéreo. Otro problema hoy en día son los retrasos en los vuelos. Los costos estimados debido a retrasos en los vuelos son enormes. Junto con la pérdida financiera, el retraso de los vuelos tiene un impacto negativo en la experiencia de vuelo de los pasajeros. La experiencia negativa durante el vuelo puede dar como resultado una mayor tasa de rotación de clientes. La inteligencia artificial puede predecir desde el mal tiempo hasta algún problema técnico que pueden retrasar los vuelos, es importante actualizar los detalles del vuelo a los pasajeros con anticipación para eliminar los tiempos de espera innecesarios. Con la ayuda de sistemas de inteligencia

artificial y aprendizaje automático procesarán datos de aviones en tiempo real, registros históricos y también la información meteorológica. El cálculo sobre el terreno ayudará a revelar patrones ocultos, lo que puede ayudar a la industria del transporte aéreo a obtener información útil sobre otras posibilidades que pueden causar retrasos y cancelaciones de vuelos.

4. La industria de agricultura

Gracias a la inteligencia artificial y el aprendizaje automático, los productores pueden acceder a datos y herramientas de análisis, lo que permite mejores decisiones, mejores eficiencias y menor desperdicio en la producción de alimentos y biocombustibles, todo mientras minimiza las consecuencias ambientales negativas.

La agricultura digital aporta una mayor precisión a la producción de cultivos al apoyar decisiones clave de gestión con conocimientos basados en datos. Los agricultores toman cada año cientos de decisiones complejas que afectan su riesgo, sostenibilidad y rentabilidad empresarial. Utilizando sensores en el campo junto con aplicaciones de aprendizaje automático se pueden predecir los rendimientos de cosecha, evaluar la calidad del cultivo, identificar especies de plantas.

Además, gracias al procesamiento de imágenes basado en aprendizaje automático permite a los agricultores confiar en herramientas digitales para reconocer especies de malezas y determinar qué cultivos son saludables y cuáles están infestados con enfermedades. La capacidad de identificar estas plagas hace posible entrenar dispositivos, como robots, para extraer malezas de los campos, protegiendo el medio ambiente del daño causado por el uso de pesticidas. Además, pueden evaluar los cultivos para detectar enfermedades, proporcionar un diagnóstico preciso de la enfermedad y recomendar un tratamiento óptimo.

5. La industria del videojuego

Personajes interactivos. Los personajes interactivos de los videojuegos están programados de tal forma que siguen un guión y responden a situaciones fijas, es decir, realmente está todo programado desde el principio y realizan unas acciones fijas. Con la incorporación del aprendizaje automático, estos personajes pueden adaptarse según el entorno y el estilo de juego del jugador. Por ejemplo, en el juego Metal Gear Solid 5, si un jugador usa continuamente la técnica de disparos a la cabeza en el juego, los personajes se adaptan a él y comienzan a usar cascos para evitar ser golpeados.

Modelado de sistemas complejos. Para crear un mundo más inmersivo, los desarrolladores usan algoritmos de aprendizaje automático para crear modelos predictivos. Interacciones realistas. Los juegos de mundo abierto requieren que el jugador interactúe con sus entornos. Con el auge del procesamiento del lenguaje natural, el jugador puede interactuar con otros personajes de una manera más realista. Por ejemplo, en Red Dead Redemption 2, los personajes del juego interactúan con el jugador de acuerdo con el nivel de honor de nuestro personaje.

Creación dinámica del universo. Otro caso de los juegos de mundo abierto

es la creación de estos mismos. Esta creación del universo toma mucho tiempo para ser perfecta y consiste en tareas repetitivas y pequeñas. Con ayuda del aprendizaje automático el tiempo que lleva este proceso se ha reducido.

Juegos móviles. Los juegos móviles contribuyen con aproximadamente el 50 % de los ingresos generados por los videojuegos. El alcance de estos juegos es limitado debido al hardware de los teléfonos. Pero esta situación ha comenzado a cambiar a medida que ahora se están instalando componentes hardware más adecuados al procesamiento del algoritmos de aprendizaje automático en relación al sector de los videojuegos.



Figura 2.14: Metal Gear Solid 5 y Red Dead Redemption 2

2.2.3. Aplicaciones cotidianas

1. Asistentes personales

Siri, Alexa, Google Now son algunos de los ejemplos populares de asistentes personales. Estos asistentes ayudan a encontrar información cuando se les pregunta por voz. Simplemente con realizar preguntas como por ejemplo "¿Qué tiempo hace hoy?", "¿Cómo tengo la agenda de mañana?". Para responder, su asistente personal busca la información, recuerda consultas relacionadas o usa otros recursos del móvil como aplicaciones para recopilar información. Incluso puede instruir a los asistentes para realizar ciertas tareas como configurar una alarma el día siguiente, añadir tareas o recordatorios en el calendario o la agenda e incluso pedir que llame a algún contacto.

El aprendizaje automático es una parte importante de estos asistentes personales, ya que recopilan la información de todas las anteriores veces que los has utilizado. Después, este conjunto de datos se utiliza para generar resultados que se adaptan a sus preferencias. Los asistentes virtuales están integrados en una variedad de plataformas como los smartphones, aplicaciones móviles, o los altavoces inteligentes.



Figura 2.15: Aprendizaje automático en asistentes personales

2. Aplicaciones de transporte

Cuando utilizamos servicios de navegación GPS, nuestras ubicaciones y velocidades actuales se guardan en un servidor central para administrar el tráfico. Estos datos se utilizan para construir un mapa del tráfico actual. Esto ayuda a prevenir el tráfico y hace un análisis de su congestión. Es aprovechado por las redes de transporte y sus aplicaciones. Al reservar un taxi o un VTC, la aplicación calcula el precio del viaje.

El aprendizaje automático ayuda a definir las horas de aumento de precios al predecir la demanda del piloto, o a calcular la ruta más eficiente a un lugar para evitar un consumo de combustible excesivo o la perdida de tiempo en atascos. En todo el aprendizaje automático está jugando un papel importante. Aplicaciones como Uber o Cabify son un ejemplo de esto.

3. Redes sociales

Tanto personalizar sus noticias como una mejor selección de anuncios que mostrarle, las redes sociales están utilizando el aprendizaje automático para sus propios beneficios. Algunos ejemplos de esto son:

Las recomendaciones de personas que quizás conozcas. Tanto Instagram, Twitter, Facebook y todas estas redes sociales están continuamente analizando los amigos con los que se conecta, los perfiles que visita con frecuencia, sus intereses, el lugar de trabajo, etc. En base a esta gran cantidad de información, se sugiere una lista de usuarios.

Reconocimiento facial. Subir una foto con un amigo a Facebook y este reconoce instantáneamente a ese amigo. Facebook verifica las poses y proyecciones en la imagen, observa las características únicas y luego las compara con las personas en tu lista de amigos. Todo este proceso lo realiza una aplicación de aprendizaje automático.

Pines similares. El análisis de imágenes y videos es un proceso basado en el aprendizaje automático. Pinterest utiliza esta forma de aprendizaje automático para identificar los objetos en las imágenes y recomendar similares.

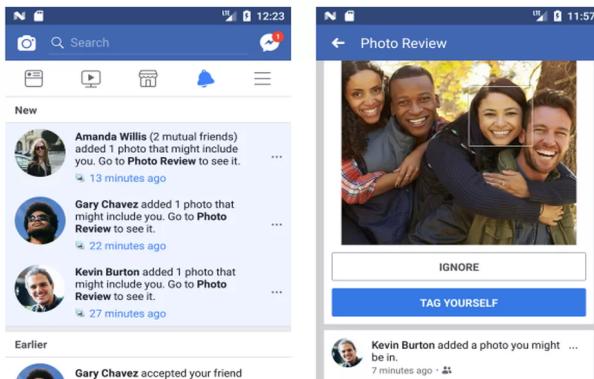


Figura 2.16: Reconocimiento facial de Facebook

4. Correos de spam y Malware

Existen distintos tipos de filtrado de spam que utilizan los correos electrónicos. Para asegurarse de que estos filtros de correo no deseado se actualicen continuamente, funcionan mediante aprendizaje automático. Ya que, si se realiza el filtrado de spam basado en reglas, este no puede rastrear los últimos trucos adoptados por los spammers. Estas herramientas antispam vienen con las aplicaciones de correo electrónico por defecto.

Una gran cantidad de malwares se detectan todos los días y muchas partes de código son entre un 90 y un 98 % similar a sus versiones anteriores. Los programas de seguridad que se ayudan del aprendizaje automático detectan fácilmente el nuevo programa maligno con una variación del 2 al 10 % y ofrecen protección contra ellos.

2.3. En desarrollo

Como hemos visto anteriormente la inteligencia artificial esta presente hoy día en todas las industrias que nos rodean. De hecho, alrededor del 77 % de las personas en el mundo ya usan IA de alguna forma. Además, la inteligencia artificial no solo afecta a la industria de la tecnología, sino a todos y cada uno de los sectores de la economía. Y con las principales compañías como Google, Facebook, Microsoft o Amazon trabajando en todas las aplicaciones posibles de inteligencia artificial, no hay duda de que habrá una gran evolución en los próximos años. Adobe incluso predice que el 20 % de todas las tecnologías emergentes tendrán algunas bases de inteligencia artificial para 2021. Algunos ejemplos de proyectos y tendencias que ocurren actualmente son:

1. Inteligencia artificial e internet de las cosas

Estas dos áreas de la tecnología se combinan bastante bien debido a que los dispositivos IoT, internet de las cosas(Internet of Things), crean una gran cantidad de información y, por otro lado, los algoritmos de inteligencia artificial requieren los datos antes de sacar conclusiones. Por lo tanto, los datos recopilados por IoT pueden ser utilizados por los algoritmos de aprendizaje automático para crear resultados.

La mayoría de estos dispositivos son electrodomésticos del hogar, estos electrodomésticos inteligentes para el hogar se están volviendo cada vez más populares. De hecho, un gran porcentaje de todos los hogares en los EE. UU. podrían convertirse en hogares inteligentes para 2021. No solo los hogares se benefician de estos aparatos, las empresas también están adoptando cada vez más dispositivos inteligentes, ya que reducen los costos. Nest, propiedad de Google, es una de las más conocidas en este mercado, ya que produce una gran variedad de productos inteligentes como termostatos, sistemas de alarma, timbres, etc. La integración de la inteligencia artificial e internet de las cosas también ha llevado a las ciudades como Nueva York a ser cada vez más inteligentes. Hay instalaciones para monitorear el uso del agua y contenedores inteligentes que funcionan con energía solar que pueden controlar los niveles de basura y programar la recolección de desechos.

2. Aprendizaje personalizado para los estudiantes

Muchos de los objetivos del aprendizaje automático y de la inteligencia artificial es la personalización para cada persona. Desde recomendaciones de Netflix o YouTube hasta la publicidad en aplicaciones, se basan en nuestras preferencias personales. Con esta misma idea, la inteligencia artificial podría ayudar a los profesores a abordar las necesidades individuales de cada estudiante desde cero, en lugar de identificarlos a lo largo de un semestre, un curso, o peor aún, al final de su etapa educativa. Esto les da a los profesores suficiente tiempo para ayudar a los estudiantes. Brightspace Insights predice y pronostica a los estudiantes en riesgo, para que los instructores puedan ayudarlos mientras aprenden. Al tener acceso a los datos de los estudiantes, los maestros pueden diseñar planes de aprendizaje personalizados para sus alumnos. En lugar de utilizar un enfoque único y común para toda una clase, de esta forma se trabajarían las fortalezas individuales de cada estudiante. El aprendizaje automático es capaz de identificar patrones que normalmente no reconocemos, esto podría ayudar a determinar qué estudiantes se sienten cómodos estudiando las distintas materias, así como su forma de aprenderlas, desde una forma puramente teórica a una más visual.

3. Inteligencia artificial y computación en la nube

La inteligencia artificial y la computación en la nube pueden revolucionar totalmente el mercado actual. Es obvio que la inteligencia artificial tiene una gran potencia, pero su integración también requiere empleados experimentados y una gran infraestructura. Cloud Computing puede proporcionar una ayuda inmensa. Incluso si las empresas no tienen una fuerte base informática o acceso a muchos datos, aún pueden aprovecharse de sus beneficios.

La inteligencia artificial también se puede usar para controlar y solucionar problemas en la nube. Se puede usar desde para automatizar el flujo de trabajo básico de los sistemas de computación en la nube pública, hasta para crear formas de trabajo mucho más eficientes. Actualmente, los líderes del sector tecnológico incorporan inteligencia artificial en sus servicios en la nube como

son Amazon Web Service (AWS), Google, IBM, Alibaba o Oracle. Se espera que crezcan aún más en el futuro gracias a la popularidad y el crecimiento que están teniendo estas áreas de la tecnología.

4. Inteligencia artificial en ciberseguridad

Hablamos de la mezcla de dos campos de la tecnología posiblemente en uno de sus mejores momentos respectivamente. Si bien la ciberseguridad es el dominio de la industria de tecnología, la inteligencia artificial tampoco se queda muy atrás. La adición de Inteligencia Artificial puede mejorar el análisis, la comprensión y la prevención de los ciberataques. También puede mejorar las medidas de seguridad cibernética de las empresas para que estén seguras y protegidas. Sin embargo, es muy difícil de implementar en todas las aplicaciones. Además, la inteligencia artificial es un arma de doble filo ya que puede usarse para mejorar los ciberataques. A pesar de todo esto, la inteligencia artificial será un elemento crítico de ciberseguridad en el futuro.

Por lo tanto, las empresas pueden comenzar con la integración de algoritmos de aprendizaje automático en sus protocolos de Ciberseguridad existentes. Esto se puede hacer utilizando análisis de datos para detectar amenazas o actividades maliciosas.

5. Conducción autónoma

Como hemos mencionado anteriormente la industria del transporte y de la automoción apuesta por la conducción autónoma de los vehículos, además ésta, se está popularizando en la sociedad. Algunos países de como EE. UU. o Tokio han puesto en marcha algunos modelos. Sin embargo, es una realidad que aún existen reticencias y con motivos justificados. Google está tratando de crear automóviles sin conductor usando Waymo , lo que podría reducir las muertes en la carretera, disminuir la congestión del tráfico y proporcionar una solución si no puede conducir.

Los automóviles Waymo tienen LiDAR, radar y cámaras colocados a su alrededor que les permite detectar todos los objetos en 360 grados alrededor del automóvil, incluso si es de noche. Google comenzó el proyecto de conducción autónoma de Waymo en 2009 cuando modificaron un Toyota Prius y practicaron la conducción autónoma a más de 100 000 millas en la vía pública. Después de eso, desarrollaron un prototipo de automóvil totalmente autónomo llamado Firefly que no tenía volante ni pedales. Actualmente, Waymo tiene un juicio público en curso en Phoenix con el objetivo final de hacer que la conducción sea segura con cero muertes en el futuro.

Capítulo 3

Aprendizaje Automático

3.1. Definición

El aprendizaje automático o aprendizaje automatizado (Machine Learning) es una disciplina del campo de la inteligencia artificial. Consiste en el desarrollo de algoritmos¹ “inteligentes” capaces de aprender a partir de datos y gracias a estos poder hacer predicciones. Lo que se denomina aprendizaje no es más que la capacidad del sistema para identificar una serie de patrones dentro de los datos recibidos y gracias a esto adquirir unas mejoras con la experiencia. Es decir, la máquina no aprende por si sola, sino gracias a sus algoritmos y heurísticas², que modifican los datos recibidos dando lugar a escenarios futuros o a decisiones.

En muchas ocasiones el aprendizaje automático se solapa con el campo de la estadística ya que las dos disciplinas se basan en analizar datos.

Los algoritmos usados en el aprendizaje automático realizan muchas tareas por su propia cuenta. Obtienen datos propios a partir de cálculos y cuantos más datos obtienen más “aprenden”, es decir, más precisos serán sus resultados. Cada dato puede modificar en mayor o menor medida dicho algoritmo, por lo tanto, a mayor cantidad de datos, mayor complejidad y efectividad del cálculo. La clave reside en tomar decisiones en base a los datos recibidos. Un sistema de aprendizaje automático necesita contar con datos relevantes suficientes para poder suministrar repuestas coherentes y válidas. El aprendizaje automático tira de una amplia gama de aplicaciones en distintos campos del mundo como pueden ser diagnósticos médicos, detección de fraudes, clasificación de secuencias de ADN, estudios de mercado o videojuegos.

El aprendizaje automático tiene como resultado la creación de un modelo para resolver un problema. Estos modelos se pueden distinguir en:

-Modelos lógicos: usan una lógica para dividir una parte del espacio en segmentos y así construir grupos. Esta lógica es una función que devuelve un valor bool. Una vez usada esta lógica, los datos quedan divididos en distintos grupos y se intenta resolver el problema.

¹Conjunto de instrucciones definidas, ordenadas y finitas que permiten solucionar un problema, realizar un cómputo y procesar datos.

²Métodos para aumentar el conocimiento.

-Modelos geométricos: en los modelos lógicos el espacio se divide en distintos grupos, sin embargo, dos partes de dicha división pueden ser similares, por ello existen modelos que tienen en cuenta esta similitud considerando la geometría de una parte del espacio. Las características de los modelos pueden ser tratadas como puntos en un plano (X, Y) o incluso en un espacio tridimensional (X, Y, Z). Esto es lo que se conoce como modelos lineales. Por otra parte, podemos usar la geometría para representar la distancia entre la similitud, es decir, si dos puntos están muy cerca es que guardan valores muy similares, esto se conoce como modelos basados en la distancia.

-Modelos probabilísticas: usan las probabilidades para la clasificación de nuevas entidades. Los modelos probabilísticos ven a los datos y las variables resultantes como valores aleatorios. El proceso de manipular el nivel de incertidumbre de estos datos. Hay dos tipos de modelos probabilísticos. Los predictivos, que usan la idea de una probabilidad condicional $P(Y|X)$ en la que Y puede ser predicha a partir de X . Y los generativos, que estiman una probabilidad conjunta $P(Y, X)$. Una vez se conoce la distribución conjunta podemos derivar cualquier distribución condicional que involucre a las variables.

3.2. Tipos de Algoritmos

Los algoritmos de aprendizaje automático se dividen en tres categorías, aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. De las cuales las más utilizadas son la supervisada y la no supervisada.

3.2.1. Algoritmos supervisados

Los algoritmos supervisados cuentan con una información previa basada en etiquetas asociadas a unos datos. Se entrena al sistema proporcionándole estos datos. Una vez que se le ha entrenado lo suficiente, es decir, se le han proporcionado suficientes datos, podremos introducirle nuevos datos sin necesidad de etiquetas. Este sistema se conoce como clasificación. Otro método es predecir un valor continuo, utilizando distintos parámetros que, combinados de distintas formas, predicen el resultado. Es lo que se conoce como regresión.

1. Clasificación y Regresión

Clasificación es la acción de identificar a qué categoría de un conjunto discreto pertenece un elemento de un conjunto de ejemplos, mientras que la regresión predice un número. Un ejemplo de clasificación es el correo, los correos pueden clasificarse como “spam” o como “legítimos”, mientras que un ejemplo de regresión puede ser el precio de una casa en función de sus características.

La regresión es un modelo estadístico que permite establecer un patrón entre los datos de entrada con los resultados. Hay algoritmos muy comunes asociados respectivamente a la clasificación y la regresión, la Regresión Logística y la Regresión Lineal.

En estadística la regresión lineal se utiliza para representar la relación que existe entre un conjunto de datos explicativos X (variables de entrada) con un conjunto de datos dependiente Y (resultados). En su versión más sencilla

lo que se hará será dibujar una recta que nos indicará la tendencia de ese conjunto de datos.

Para obtener automáticamente esa recta de tendencia lo que hacemos es medir el error con respecto a los datos de entrada y el resultado. El algoritmo debe minimizar el coste de la función y los coeficientes corresponden a la recta óptima. Hay distintos métodos para minimizar el coste, el más común es la llamada Ecuación Normal, que nos dará un resultado directo.

Dependiendo de la cantidad de valores de entradas podemos hablar de regresión lineal simple, como hemos estado haciendo anteriormente, o de regresión lineal múltiple. La regresión lineal múltiple sigue los mismos patrones que la simple, buscar la tendencia de los valores. Sin embargo, al tener más cantidad de valores de entrada el resultado no será una recta. Por ejemplo, en caso de tener tres variables de entrada el resultado será un plano bidimensional.

Al igual que en la regresión lineal, en la regresión logística podemos tener dos posibles estados, SI o NO, binario, o un número finito de etiquetas como pueden ser el 1, 2, 3, 4, ..., en el reconocimiento de números, múltiple. De hecho, la regresión logística es una red neuronal de una sola neurona.

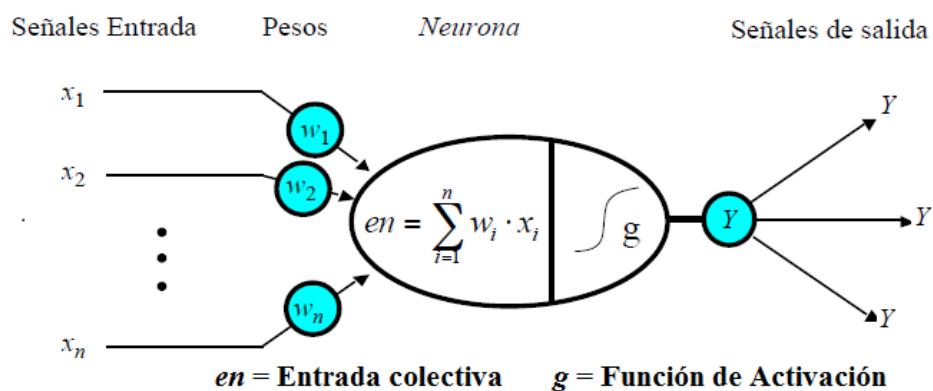


Figura 3.1: Regresión Logística perceptrón

Tenemos distintos datos de entrada a nuestro problema y queremos ver la probabilidad de que ocurra una cosa u otra. Para ello la regresión logística tiene dos partes, una combinación lineal y después una aplicación de la función logística. De esta forma todas las entradas de datos se combinan linealmente y posteriormente se le aplica la función logística o sigmoide al resultado.

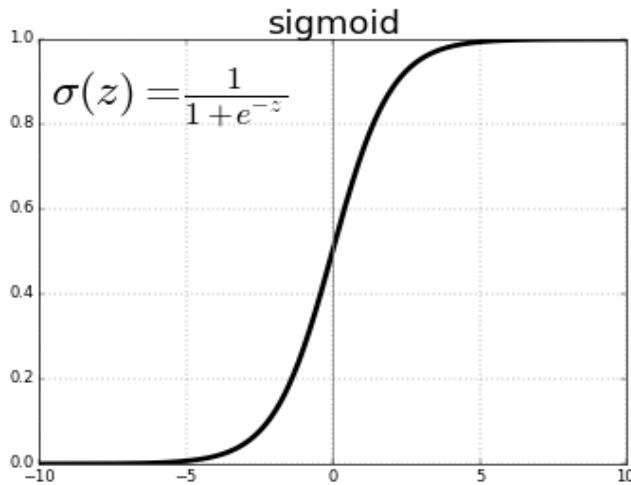


Figura 3.2: Sigmoid

Las características de la función sigmoide son:

- Esta acotada entre 0 y 1
- Podemos interpretar sus resultados como probabilidades
- Para problemas de clasificación binarios, podemos establecer un valor a partir del cuál los resultados corresponde al 0 o al 1.

En el aprendizaje automático existen algoritmos clasificadores que pueden trabajar con múltiples clases, existen otros que no pueden como es el caso de la regresión logística. Sin embargo, existen técnicas que se pueden utilizar para la regresión logística múltiple, como, por ejemplo:

- Uno contra todos. En esta técnica se ha de entrenar tantos clasificadores binarios como clases existan. Cada modelo predice la probabilidad de que el resultado pertenezca a una clase o no. A la hora de la predicción se ejecutan todos los clasificadores y se elige el que tenga mayor probabilidad.
- Uno contra uno. En esta técnica se crean tantos modelos con pares de posibles resultados existan. Un clasificador decidirá solamente entre dos posibles resultados. Al igual que la técnica anterior se ejecutan todos los clasificadores y se selecciona el de mayor probabilidad.

2. Evaluación de exactitud

Normalmente los algoritmos de aprendizaje supervisando entranan modelos de clasificación o de regresión usando ejemplos de entrenamiento. Estos ejemplos de entrenamiento se basan en parejas de datos de entrada y resultados a esos datos de entrada. La evaluación de exactitud es una forma de medir la precisión que existe en el sistema que nosotros hemos entrenado. La exactitud de un sistema es la proximidad de los resultados de una medición al valor correcto. La precisión de un sistema es el grado en el que las mediciones repetidas en condiciones sin cambios muestran los mismos resultados.

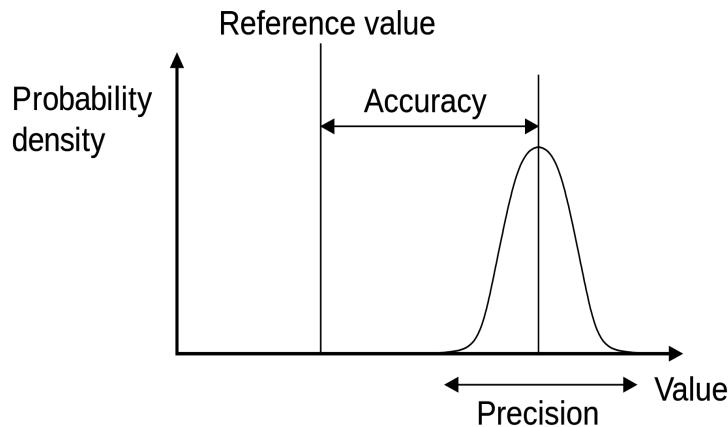


Figura 3.3: Exactitud y precisión

Como hemos mencionado antes, el aprendizaje automático esté ligado al campo de la estadística en muchos casos. En este caso, la estadística se refiere a los términos de sesgo (bias) y varianza (variability) en vez de exactitud y precisión.

Un sistema puede combinar estas dos cualidades de sus cuatro formas posibles. Un sistema puede ser preciso e inexacto, impreciso pero exacto, ambas o ninguna.

Sin embargo, la evaluación de exactitud en muchos casos no se acerca al resultado real. En muchas ocasiones el modelo ha sido sobreentrenado con los casos de ejemplo. Esto significa que será capaz de evaluar correctamente los casos de ejemplo y casos similares a estos, pero nunca podrá ir más allá de estos casos y generalizar el proceso. Por ejemplo, si un conjunto de 100 datos de entrenamiento tiene 95 resultados A, y 5 resultados B, nuestro sistema clasificará con bastante precisión los ejemplos futuros A, pero será nada fiable en los ejemplos B.

Para evitar esto nuestro conjunto de datos de entrenamiento debe ser lo más variado posible, incluyendo, en la mayor medida de lo posible, datos de todos los tipos. Sin embargo, esto no soluciona el problema del sobreentrenamiento. Una forma de solucionarlo es la división de nuestro conjunto de entrenamiento.

Existen varias formas de dividirlo, la más simple es repartir manualmente un 80 % para entrenamiento y un 20 % para validación. Sin embargo, si el conjunto de entrenamientos es lo suficientemente grande, se recomienda dividirlo en tres subconjuntos, uno de entrenamiento, uno de validación y uno final de comprobación o testeo. Con un porcentaje de 60 %, 24 %, 16 % respectivamente del conjunto original. (Los porcentajes no es obligatorio que sean estos, pero dan una ligera orientación de en que subconjuntos tiene que haber mas datos). Para mayor fiabilidad en muchos casos se suele desordenar el conjunto original para evitar que todos los ejemplos con un mismo resultado caigan en el mismo subconjunto. Ésto se conoce como validación cruzada.

3. Redes neuronales

Las redes neuronales artificiales es otro modelo de aprendizaje supervisado inspirado en el comportamiento de las neuronas biológicas. Sin embargo, las neuronas naturales son muchísimo mas complejas que las neuronas artificiales que tenemos hoy en día. Y es por esto por lo que cuanto mejores sean nuestros conocimientos acerca de las neuronas mejores réplicas artificiales podremos hacer de ellas. Actualmente el éxito de las redes neuronales no es la recreación exacta de las naturales, sino que los investigadores han buscado la forma en la que las personas resuelven problemas que tradicionalmente no han podido resolverse mediante la informática tradicional. Para ello, una neurona artificial está formada por la simulación de cuatro funciones de las neuronas naturales.

Una neurona tiene una serie de datos de entrada, normalmente representados por $x(n)$. Cada uno de estos datos de entrada son multiplicados por el peso de la conexión $w(n)$. En el caso más simple de todos estos productos, simplemente son tratados por una función de suma para ver el dato de entrada total y posteriormente pasarr por una función de transferencia generando un nuevo dato. Después, este dato pasará por una función de salida en la que se procesará su información para que pueda ser entendible.

En casos más complejos entre la función de transferencia y la función de salida puede haber una función intermedia de escalado y limitación (normalización). Este proceso de escalado simplemente multiplica un factor de escalado por el dato. El proceso de limitación sirve para asegurar que el resultado del producto anterior no exceda un límite, tanto superior como inferior.

En la mayoría de los sistemas, existe otra función que se encarga de calcular la diferencia entre el resultado actual y el deseado. Este error es después transformado para la arquitectura de la red neuronal.

La función de aprendizaje se centra en modificar los pesos de las variables de entrada en cada proceso según un algoritmo. Este algoritmo es el que clasifica a las redes neuronales dividiéndolas en si pertenecen al aprendizaje supervisado o no.

En su mayoría, las redes neuronales pertenecen al aprendizaje automático supervisado, sin embargo, existen algunos ejemplos que pertenecen al no. Estas neuronas están agrupadas en capas entre las que podemos distinguir la capa de entrada, la oculta y la de salida. Siguiendo esta estructura algunas neuronas están en contacto con los datos de entrada del mundo real. Otras suministran datos al mundo real. Todo el resto de las neuronas están en la capa oculta. Pero una red neuronal no es solamente una gran cantidad de neuronas artificiales conectadas aleatoriamente entre sí. Es aquí donde existen una gran cantidad de arquitecturas de las redes neuronales, dependiendo de la forma en que se conectan las neuronas artificiales y todas las posibles funciones descritas anteriormente.

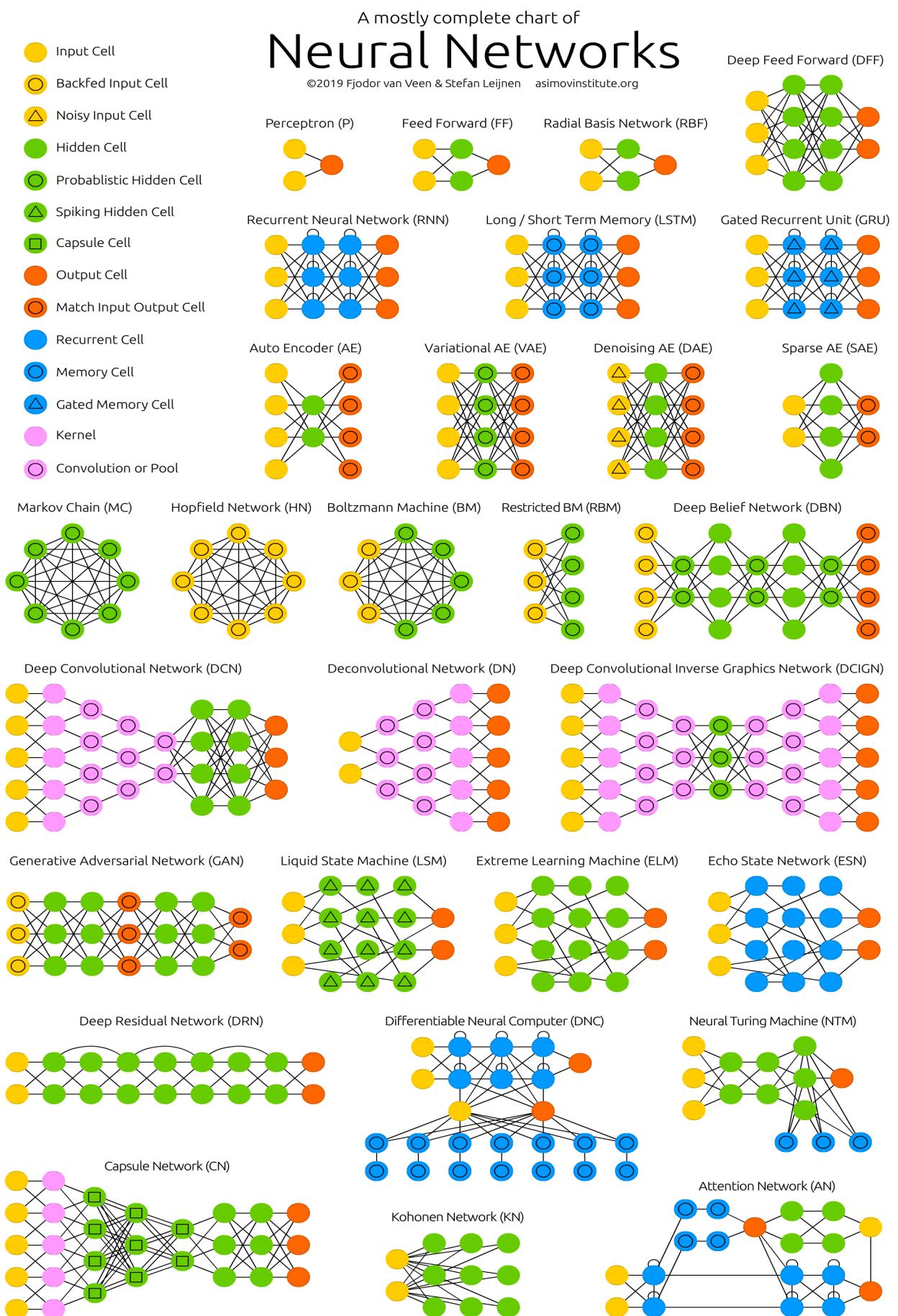


Figura 3.4: Redes Neuronales más comunes

4. Conjuntos

Un conjunto es un algoritmo supervisado ya que puede ser entrenado y, después, usado para realizar predicciones. El objetivo de los algoritmos de conjuntos es el de combinar las predicciones de varios modelos para mejorar la generalización y robustez respecto a un único modelo.

Este algoritmo utiliza distintos modelos para obtener mejores predicciones de las que se podrían obtener de los modelos que lo constituyen por separado, es decir, hace una predicción total basada en las predicciones de cada modelo. En muchas ocasiones existen muchas hipótesis que se adaptan muy bien a un problema o que todas ellas son muy prometedoras a la hora de resolver dicho problema. Los conjuntos nos permiten agrupar dichas hipótesis para poder generar una que resuma todas las anteriores.

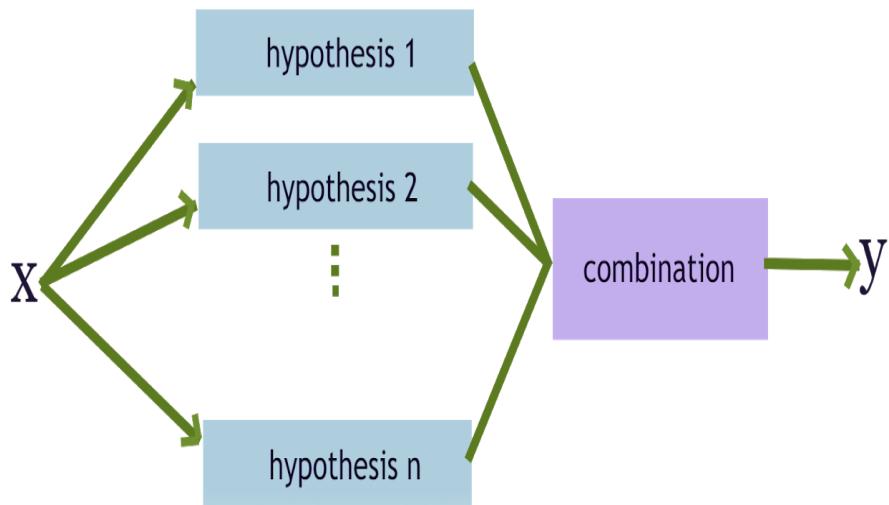


Figura 3.5: Arquitectura general de un algoritmo de conjuntos

Existen dos tipos distintos de métodos de conjuntos, los promedio o Averaging y los que se van mejorando o boosting. Los métodos averaging se basan en la creación de varios modelos de predicción de manera independiente y, posteriormente, realizar una media de las predicciones de dichos modelos. La mayoría de las veces el conjunto de estos modelos es mejor que por separado puesto que su varianza es menor. Ejemplos de estos tipos de métodos son Bagging y Bosques de árboles aleatorios.

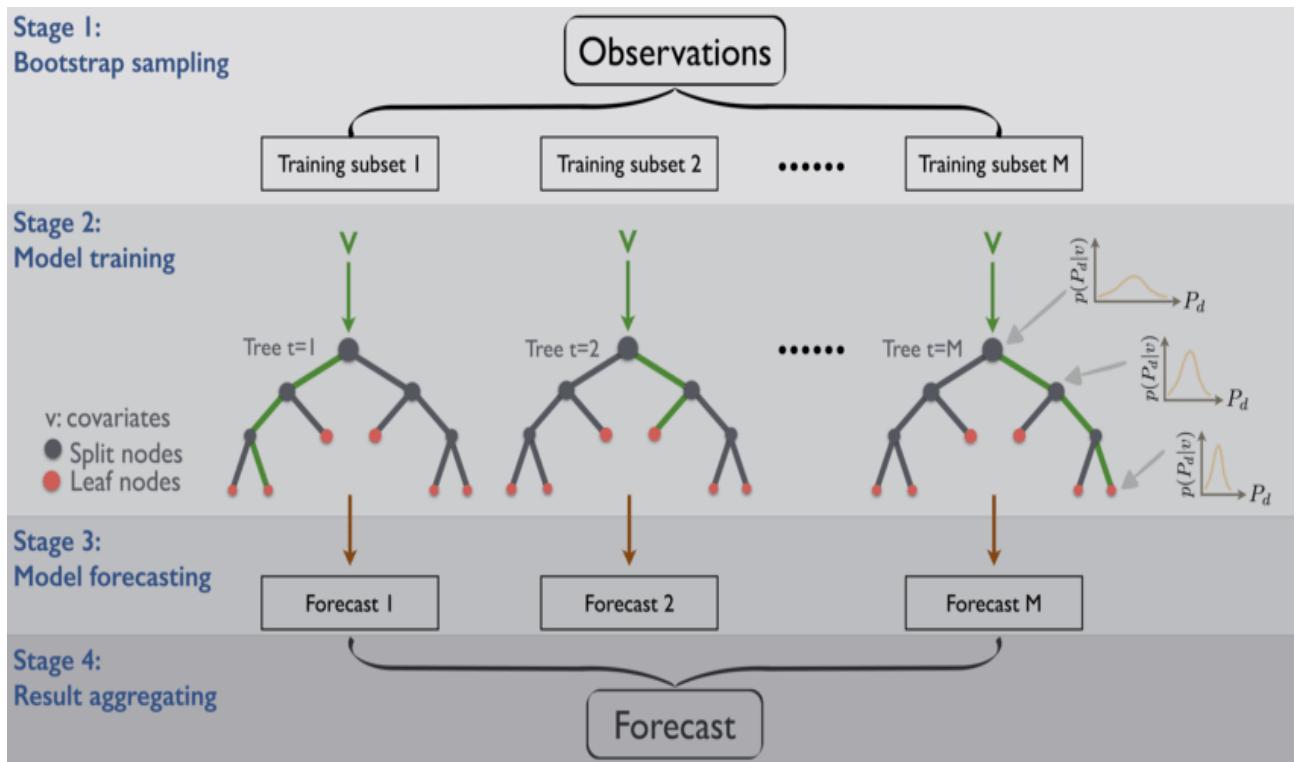


Figura 3.6: Esquema del método Averaging

Por otro lado el objetivo de los métodos Boosting es la construcción de un conjunto de modelos construidos secuencialmente en el que cada modelo sucesivo realiza una nueva predicción tratando de solucionar los errores de su predecesor. De esta manera, se combinan modelos menos robustos para formar uno mucho más robusto. Ejemplos de estos métodos son AdaBoost, Gradient Tree Boosting, XGBoost y Light GBM.

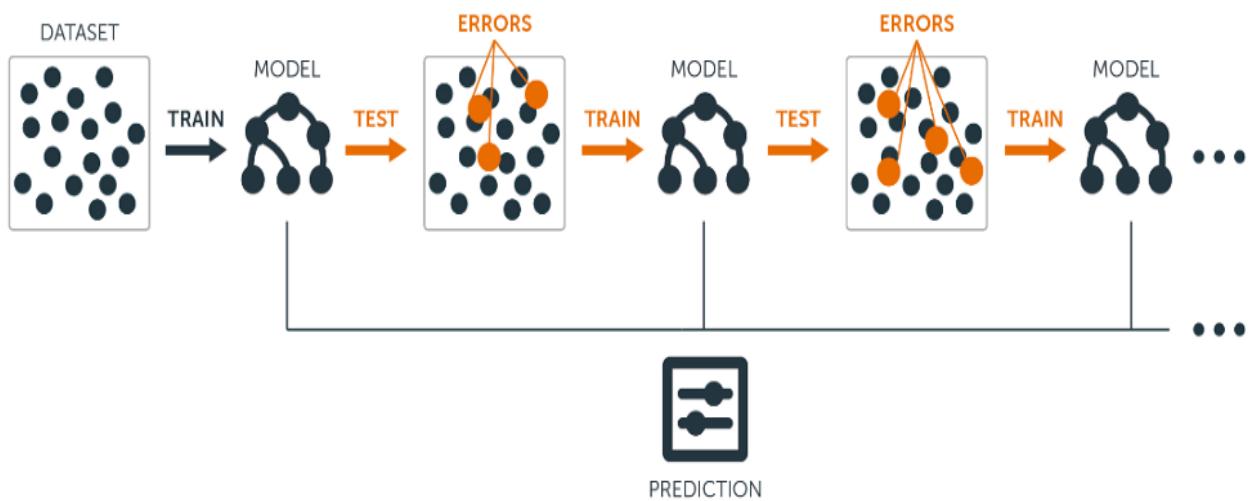


Figura 3.7: Esquema del método Boosting

Este tipo de algoritmos cuenta con una serie de ventajas y desventajas.

Ventajas de los algoritmos de conjuntos:

- Mejoran la exactitud del modelo y funciona correctamente la gran mayoría de las veces.
- Hacen al modelo general más robusto y estable.

Desventajas de los algoritmos de conjuntos:

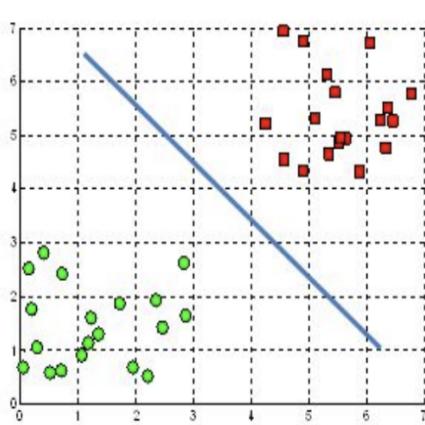
- Necesitan tiempo para desarrollarse.
- La creación del conjunto en cuanto a la elección de los modelos es muy difícil de realizar y se necesita experiencia en ello.

5. Soportes de máquinas de vectores

En el ámbito del aprendizaje automático, las support vector machine (SVM, también conocidas como support vector networks) son modelos de aprendizaje supervisado. Éstas están relacionadas con métodos de aprendizaje encargado de analizar datos utilizados para problemas de clasificación y análisis de la regresión. En términos formales, su objetivo es el de encontrar un hiperplano en un espacio N-dimensional ($N = \text{número de características}$) que se encargue de clasificar los diferentes puntos de información. Dado un conjunto de ejemplos de entrenamiento, cada uno perteneciente a una clase etiquetada, un algoritmo de entrenamiento SVM construye un modelo que se encarga de predecir la clase de una nueva muestra. Un modelo SVM es la representación de los ejemplos de la muestra como puntos en el espacio, situados de tal forma que se distinga la separación de clases con un espacio intermedio que sea lo más ancho posible. Los nuevos ejemplos serán mapeados en ese mismo espacio y se les asignará una categoría basándose en el lado del espacio en el que hayan caído.

Refiriéndonos a los hiperplanos, éstos son áreas de decisión que nos ayudan a clasificar los puntos de información. La dimensión del mismo depende del número de características. Por ejemplo si tenemos 2 características, el hiperplano será una línea, si tenemos 3, será un plano tridimensional.

A hyperplane in \mathbb{R}^2 is a line



A hyperplane in \mathbb{R}^3 is a plane

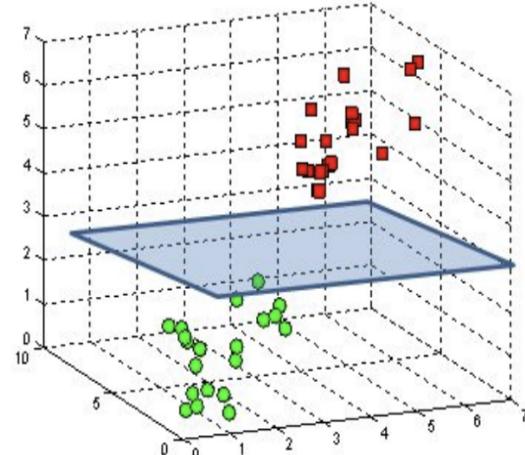


Figura 3.8: Representación hiperplano bidimensional y tridimensional

Para separar dos clases, hay muchos hiperplanos que se pueden escoger. Lo más óptimo es encontrar un hiperplano con el margen máximo (máxima distancia entre puntos de información de ambas clases). Maximizar la distancia del margen nos puede ayudar en el futuro a clasificar con mayor seguridad nuevos ejemplos.

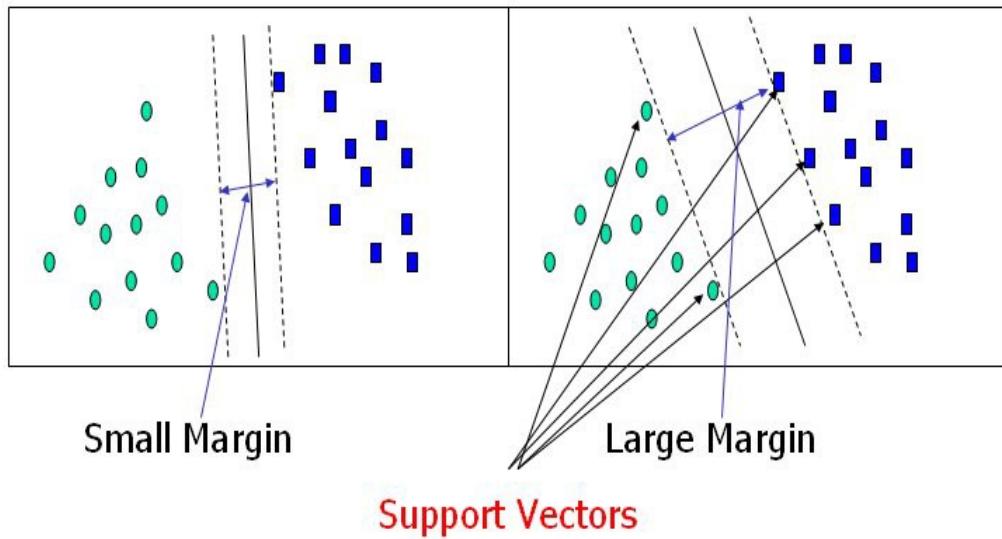


Figura 3.9: Márgenes

En referencia a la imagen de arriba (Fig 3.4: Márgenes) los support vectors son los puntos de información más cercanos al hiperplano que influencian en la posición y orientación del mismo. Mediante estos support vectors conseguimos maximizar el margen del clasificador. Un algoritmo SVM no solo trabajará con planos bidimensionales. Tendrá que poder trabajar también con más de dos variables predictoras, conjuntos de datos en los que la separación entre clases no es tan clara, o clasificaciones con más categorías. Para ello utilizamos las funciones kernel.

Finalmente hay dos características claves, a parte del margen, a la hora de ajustar el funcionamiento de una SVM:

- Regularización: Específica a la SVM cuantos error se asume a la hora de clasificar cada ejemplo de entrenamiento



Left: low regularization value, right: high regularization value

Figura 3.10: Representación de la regularización

-Gamma: Define cuánto puede llegar a influenciar un único ejemplo de entrenamiento. Un valor bajo para gama indica que los puntos lejanos al margen también se tienen en cuenta para los cálculos, mientras que si este valor es alto, solo se tendrán en cuenta los valores más cercanos.

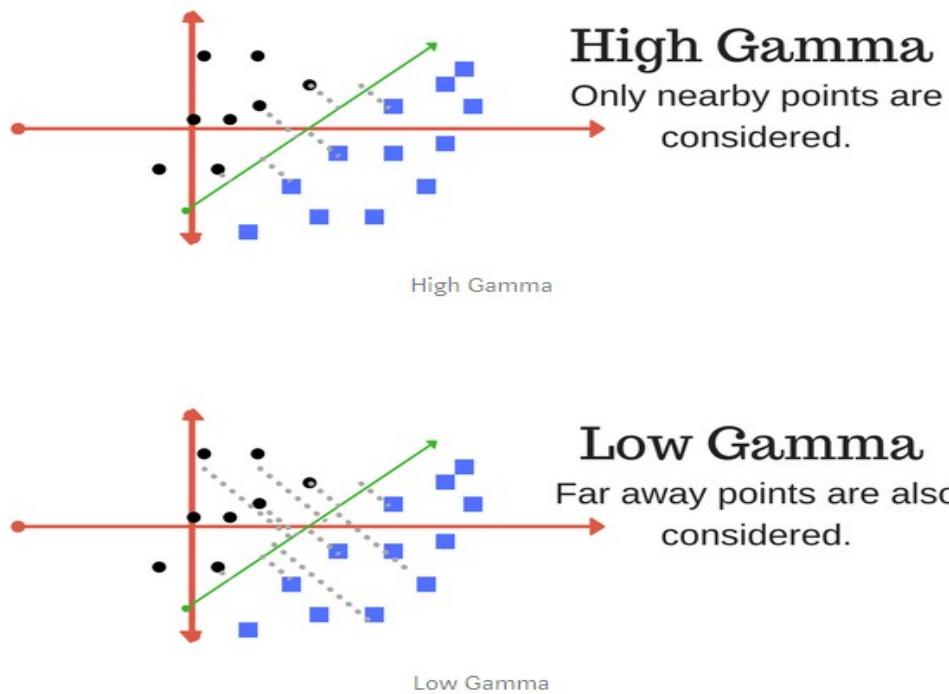


Figura 3.11: Representación de gamma

6. Árboles de decisión

Los árboles de decisión es un modelo de predicción utilizado tanto en el aprendizaje automático como en el mundo de la probabilidad, estadística y la economía. Este modelo de aprendizaje automático es la representación gráfica de posibles soluciones basándose en si los datos cumplen o no ciertas condiciones. Es uno de los algoritmos más utilizados del aprendizaje automático supervisado. Los árboles de decisión pueden ser de dos tipos principalmente, de regresión o de clasificación. Es decir, pueden realizar tareas de regresión o clasificación (Classification And Regression Tree (CART)).

Pueden tomar como resultado una clase discreta a la que pertenecen los datos o un número real. La comprensión de este tipo de algoritmos es muy simple, ya que utilizamos muy a menudo en nuestra vida cotidiana sin darnos cuenta y a la vez son muy potentes, como, por ejemplo: "¿Hace frío? Sí, me pongo un abrigo. No, no me pongo un abrigo."

Los árboles de decisión están formados por:

- Los nodos es el lugar en el que se toma una decisión entre varias posibles. Cuantos más nodos más posibilidades tienes a la hora de tomar una solución a la pregunta, es decir más finales a los que se pueden llegar.
- Los vectores de números serían la solución final a la que se llega a través de los distintos nodos visitados.
- Las flechas son las uniones entre nodos y representan la opción tomada tras evaluar la pregunta del nodo.
- Las etiquetas se encuentran en cada nodo y cada flecha y dan nombre a la acción.

Tratándose de árboles debemos tener en mente ciertos conceptos y reglas característicos de los árboles.

En cuanto a los criterios:

- Costo, se refiere al coste de determinadas propiedades como puede ser la inserción de un elemento, su borrado u operaciones como saber la profundidad del árbol.
- Poda. Evitan el procesamiento de subárboles que no afectan a la decisión, consiste en eliminar una rama de un nodo transformándola en una hoja Terminal (Podas alfa y beta, usado en árboles de juego).

En cuanto a las reglas, los árboles de decisión han de cumplir:

- Al comienzo del árbol se da un nodo inicial que no es apuntado por ningún otro y es el único con esta característica.
- El resto de los nodos son apuntados por otro a través de una única flecha. Lo que se conocen como nodos padre e hijo.
- Derivado de lo anterior se deduce que solo hay un camino para llegar del nodo inicial al resultado final.

Para obtener un árbol óptimo y valorar que subárbol escoger, el algoritmo deberá medir las predicciones para poder compararlas y obtener la mejor. Para ello utiliza diversas funciones basadas en la entropía. El concepto de entropía mide la impureza de un conjunto de entrada. En física y matemáticas es conocido como la aleatoriedad o impureza del sistema. Para entenderlo podemos imaginar un conjunto de 100 bolas verdes, se podría decir que es puro. En términos de entropía diríamos su entropía es 0. Imaginemos que 30 de esas bolas son azules y 20 rojas, ahora el conjunto será más impuro y por tanto su valor de entropía aumentará.

Las funciones más conocidas basadas en la entropía son: la ganancia de información, la impureza de Gini y la reducción de varianza.

3.2.2. Algoritmos no supervisados

Los algoritmos no supervisados a diferencia de los supervisados no cuentan con una información previa. Este tipo de algoritmos no usa etiquetas, su finalidad es la de encontrar patrones de todos los datos recibidos directamente.

Los algoritmos de aprendizaje no supervisado permiten trabajar con tareas mucho más complejas en comparación con los algoritmos supervisados.

A parte de para ejecutar tareas complejas, el aprendizaje no supervisado se puede usar también para encontrar todo tipo de patrones dentro de los datos o encontrar características útiles para categorización.

Sin embargo, los resultados serán menos consistentes e impredecibles debido a que los datos de entrada no están ni reconocidos ni etiquetados previamente. Además el usuario necesitará invertir más tiempo en interpretar y etiquetar las clases que necesitará en la posterior clasificación.

Hay 2 varios tipos de aprendizaje no supervisado agrupados en:

Clustering

Procedimiento de agrupación de una serie de vectores con un criterio, por lo general, distancia o similitud. Gracias a esto se forman grupos que comparten valores comunes o muy similares entre sí llamados clusters. El clustering puede ser utilizado para estudiar los terremotos. Basándose en áreas golpeadas por los mismos, puede ayudar a analizar cuál será la siguiente zona a la que podría afectar.. Dentro del clustering encontramos k-means. Este algoritmo en cada iteración se encarga de asignar a cada ejemplo, el centroide más cercano. (Un centroide es el punto con el valor más alto, el centro de un cluster. El algoritmo k-means encuentra k centroides) generando como salida un conjunto de etiquetas con los valores más óptimos. Dentro del clustering encontramos diferentes tipos de algoritmos:

- Clustering jerárquico: Es un algoritmo que se encarga de construir una jerarquía de clusters.

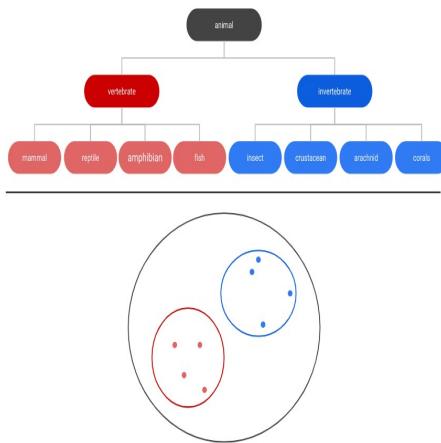


Figura 3.12: Clustering jerárquico

- Basado en el centroide: Organiza los datos en clusters no jerárquicos. Destaca k-means. Son algoritmos muy eficientes y efectivos pero sensibles a condiciones iniciales.

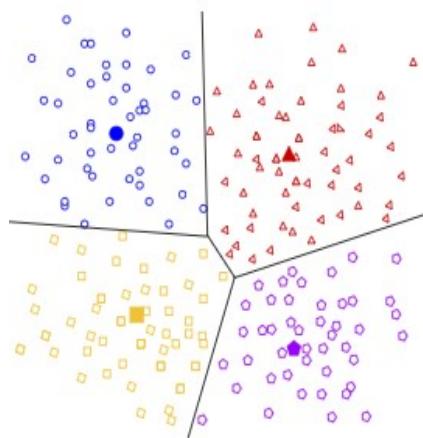


Figura 3.13: Basado en el centroide

- Basado en la densidad: Se encarga de conectar áreas de gran densidad dentro de clusters. Estos algoritmos tienen dificultades con datos en los que las densidades varían y cuyas dimensiones son muy grandes.

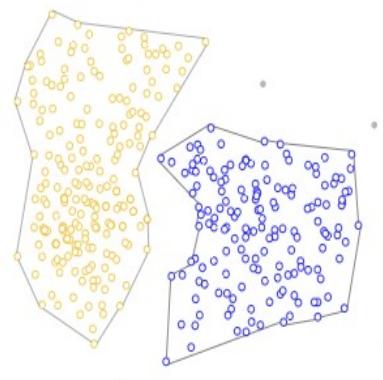


Figura 3.14: Basado en la densidad

- Basado en la distribución: Asume que el conjunto de datos está formado por distribuciones. En la imagen, podemos observar que el algoritmo ha dividido los datos en tres distribuciones gaussianas. Cuanto más lejos está un punto del centro de la distribución, su probabilidad de pertenecer a la misma disminuye.

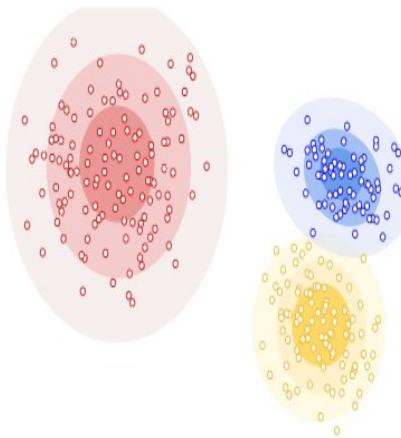


Figura 3.15: Basado en la distribución

Asociación

Esta técnica permite descubrir relaciones de interés entre las diferentes variables de un conjunto de datos. Por ejemplo, si un cliente de una cadena de supermercados compra cebollas y patatas, muy probablemente comprará carne de hamburguesas.

1. k-medias y k-medoides

- K-means: Es un algoritmo iterativo que intenta dividir el conjunto de datos en k clusters (grupos) predefinidos de manera que cada punto de información pertenezca a un único grupo. Se encarga de asignar a cada cluster aquellos puntos de información en los que la suma de la distancia al cuadrado entre el cluster y el mismo es mínima. Cuanta menos variación tengamos, más homogeneidad habrá entre los puntos dentro de un mismo cluster. El algoritmo se

compone de varios pasos. 1. Especifica el número de clusters K. 2. Inicializa los centroides mezclando el conjunto de datos y seleccionando K puntos de información como centroides. 3. Itera hasta que haya convergencia, es decir, no hay cambios en los centroides. En este último paso, calcula la suma de las distancias al cuadrado entre el centroide y los puntos de información, asigna cada punto al centroide más cercano y recalcula el centroide para cada cluster tomando la media de la distancia de cada punto de información que pertenecen a ese cluster.

- K-medoids: Este algoritmo es un acercamiento al clustering a través de k-mean, dividiendo el conjunto de datos en k clusters. Cada cluster es representado por uno de los puntos de información dentro del mismo. Estos puntos se llaman medioides. Un medioide se refiere a un objeto dentro del cluster cuya disimilaridad media a todos los objetos dentro del grupo es mínima. Es una alternativa fiable al clustering con k-mean. K-medoides es más robusta frente al ruido y los valores atípicos que k-means ya que intenta minimizar la suma de las diferencias entre los puntos etiquetados dentro de un cluster y el mediode en vez de sumar todas las distancias euclídeas al cuadrado. La forma más común de llevar a cabo el clustering a través de k-medoides es con el algoritmo de Partición sobre los medioides (PAM). 1. Con él seleccionamos aleatoriamente k medioides de los n puntos de información. 2. Asociamos cada punto al medioide más cercano. 3. Para cada medioide “m” y cada punto de información “o” asociado a “m”, intercambiamos ambos puntos y calculamos de nuevo el coste de esta nueva configuración (diferencia media desde “o” a todos los puntos “m” del cluster) y seleccionamos el medioide con el menor coste de configuración. Los pasos 2 y 3 se repetirán hasta que haya convergencia, es decir, que el resultado no cambie.

2. Análisis de componentes principales

Es un método que permite simplificar la complejidad de espacios de muestra de grandes dimensiones, condensando la información en pocas componentes. Es una técnica utilizada para la extracción de características. Combina cada una de las características de tal manera que nos podamos deshacer de las variables menos importantes, mientras que nos quedamos con las más relevantes. Esto es óptimo ya que los supuestos de un modelo lineal requiere que nuestras variables sean independientes las unas de las otras. PCA debe utilizarse cuando queremos reducir el número de variables pero no somos capaces de identificar que variables borrar completamente y cuando nos queremos asegurar que nuestras variables son completamente independientes.

3.2.3. Aprendizaje por refuerzo

Los algoritmos por refuerzo se basan en el aprendizaje a partir de experiencias anteriores. La información que recibe el sistema son las respuestas que recibe del entorno que le rodea a sus acciones. Y poco a poco, mediante un sistema de premios y castigos, el sistema va aprendiendo a modificar sus acciones para acercarse más a la meta mediante dichos premios. Un ejemplo de esto es la clasificación de secuencias de ADN o los coches autónomos.

El aprendizaje por refuerzo puede entenderse con los siguientes conceptos:

- Agente es quien realiza la acción.
- Acción (A) es el conjunto de todas las opciones posibles que un agente puede hacer.
- Factor de descuento, es el que se multiplica por recompensas futuras descubiertas por el agente para equilibrar el efecto de estas recompensas en la elección del agente, es decir que las posibles recompensas futuras tengan menos valor que las actuales.
- Medio ambiente, el mundo en el que se mueve el agente y al que responde. El entorno toma el estado y la acción del agente como entrada y devuelve una recompensa como salida además de su siguiente estado.
- Estado (S) es la situación concreta e inmediata en la que se encuentra el agente.
- Recompensa (R) es la retroalimentación por la cual medimos el éxito o el fracaso de las acciones de un agente en un estado.
- Política (π) es la estrategia a seguir que el agente emplea para determinar la siguiente acción basándose en el estado actual.
- Valor (V) es el rendimiento esperado a largo plazo, en lugar de la recompensa a corto plazo. $V\pi(s)$ es el rendimiento esperado según la política π .
- Valor Q o valor de acción es similar al valor excepto porque requiere al acción actual. $Q\pi(s, a)$ es el retorno a largo plazo de una acción que toma una política π en el estado s.
- Trayectoria es la secuencia de estados y acciones que originan esos estados.

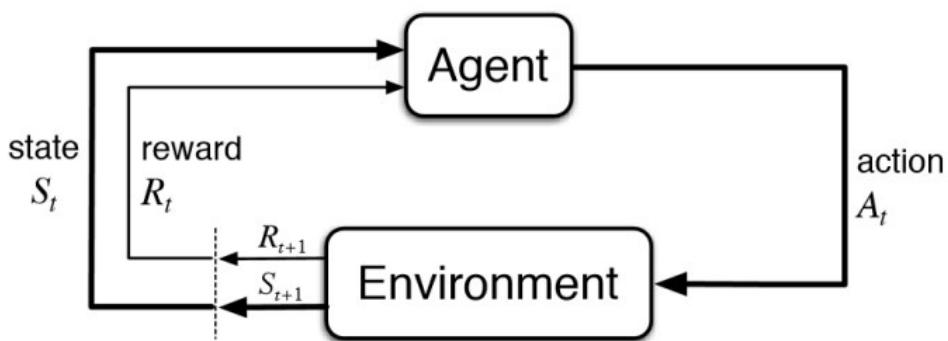


Figura 3.16: Aprendizaje por refuerzo

Hay tres enfoques a la hora de implementar un algoritmo de aprendizaje por refuerzo. -Basándose en el valor. Un método basado en el valor debe intentar maximizar el valor devuelto a largo plazo siguiendo una política $V\pi(s)$. -Basándose en la política. Un método basado en la política intenta llegar a una política que lo ayude a obtener la máxima recompensa en el futuro. Pueden ser deterministas, para cualquier estado la política π produce la misma acción, o estocástica, cada acción tiene

una probabilidad que viene determinada. -Basado en el modelo. Un método basado en el modelo debe crear un modelo virtual para cada entorno y el agente aprende a actuar en ese entorno específico.

1. Proceso de decisión de Márkov

La propiedad de Márkov establece que “El futuro es independiente del pasado dado el presente”. Matemáticamente podemos establecer esta afirmación como:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \dots, S_t] \quad (3.1)$$

Donde S_t denota el estado actual del agente y S_{t+1} el siguiente. Lo que significa que una transición del estado t al estado $t+1$ es completamente independiente de las transiciones pasadas. Es decir, nuestro estado actual ya captura la información de los estados pasados.

El proceso de decisión de Márkov o cadenas de Márkov es un proceso aleatorio con una secuencia de estados siguiendo la propiedad de Márkov. Se puede definir con un conjunto de estados y una matriz de probabilidades de transición.

Para maximizar la recompensa se sigue el proceso de recompensa de Markov en el que se obtiene un valor de cada estado en el que se encuentra nuestro agente.

$$Rs = E[R_{t+1}|S_t] \quad (3.2)$$

Cuanta recompensa Rs obtenemos de un estado particular S_t inmediatamente.

Ahora podemos añadir una toma de decisiones a estas ecuaciones para obtener el proceso de decisión de Markov. Donde S es un conjunto de estados, A es el conjunto de acciones se pueden elegir, P es la matriz de probabilidad de transición, R es la recompensa acumulada por las acciones del agente y γ es el factor de descuento.

$$P_{ss'}^a = P[S_{t+1} = s' | S_t = s, A_t = a] \quad (3.3)$$

Matriz de probabilidad de transición

$$R_s^a = E[R_{t+1}|S_t = s, A_t = a] \quad (3.4)$$

Función de recompensa

Hasta ahora obteníamos una recompensa (r) cuando pasabamos por un conjunto de estados siguiendo una política π . Realmente en el proceso de decisión de Markov la política es el mecanismo para realizar una acción. De esta forma ahora nosotros tenemos un mecanismo que tomará una acción.

2. **Aprendizaje Q** El aprendizaje Q es un método basado en valores Q, también llamados valores de acción, para mejorar iterativamente el comportamiento del agente de aprendizaje.

3. Valores Q o valores de acción. Los valores Q se definen para estados y acciones. $Q(s, a)$ es una estimación de qué tan bueno es tomar la acción a en el estado s . Esta estimación se calculará de forma iterativa utilizando la regla de actualización TD.
4. Recompensas y episodios. El agente realiza una serie de transiciones desde su estado actual al siguiente dependiendo de la acción tomada y del entorno en el que el agente la realiza.
5. Diferencia temporal o actualización de TD. La regla de diferencia temporal o actualización de TD se puede representar de la siguiente manera,

$$Q(s, a) \leftarrow Q(s, a) + \alpha (R + \gamma Q(s', a') - Q(s, a))$$

Esta regla de actualización para estimar el valor de Q se aplica en cada paso de la interacción de los agentes con el entorno. Donde:

$\gamma (>0 \text{ y } \leq 1)$: Factor de descuento para recompensas futuras. Las futuras recompensas son menos valiosas que las recompensas actuales, por lo que deben descontarse.

α : Longitud del paso tomada para actualizar la estimación de Q (s, a).

Elegir la acción a tomar usando la ϵ política de greedy:

Consiste en elegir acciones utilizando las estimaciones actuales del valor Q. Va de la siguiente manera:

Con probabilidad $1 - \epsilon$, elija la acción que tenga el valor Q más alto.

Con probabilidad ϵ , elija cualquier acción al azar.

3.3. Conclusión

Después de ver el estado actual y estudiar más a fondo el aprendizaje automático y sus distintos tipos de algoritmos y algunos ejemplos de estos, podemos decir que es una técnica que replica muchas tareas que el cerebro humano puede hacer. En muchas ocasiones estas tareas son realizadas en menor tiempo y en muchas ocasiones con mejores resultados que el propio trabajo de un ser humano. Como hemos visto anteriormente las máquinas pueden vencer a los campeones de muchos juegos como el ajedrez o el AlphaGO. Además, hemos visto que las máquinas pueden aprender a realizar actividades en una gran cantidad de sectores, ayudando de esta forma a los seres humanos en sus trabajos, en sus casas o incluso a la hora de divertirse, es decir, en su vida diaria.

Hemos visto que existen una gran variedad de tipos de algoritmos de aprendizaje automático. Dentro de cada tipo existen a su vez diferentes algoritmos para usar. Cada uno de ellos se adapta mejor a un cierto tipo de problemas, por ello es necesario conocer el problema que se quiere resolver. Saber que se pide, pero también de que datos dispones.

Finalmente, cuando se trata del desarrollo de modelos propios de aprendizaje automático se estudia las distintas opciones de lenguajes de desarrollo, entonces de

desarrollo y plataformas. Lo siguiente es conocer y saber aplicar cada técnica de aprendizaje de automático a cada tipo de problema, es decir, por ejemplo si se necesita un algoritmo supervisado o no. Una vez elegido el tipo de algoritmo ver cual de todos ellos se adapta mejor a la resolución del problema.

El tema es muy amplio y ofrece muchas posibilidades, pero analizándolo detenidamente, cada técnica es diferente y específica para resolver un tipo de problema determinado.

Capítulo 4

Generación Procedural de Contenido

4.1. Definición

La generación procedural o generación por procedimientos de contenido es la producción de recursos de manera aleatoria en base a unos algoritmos preestablecidos por los desarrolladores, es decir son recursos que no han sido creados de ante manos por personas. De esta forma, dichos recursos que se ven por pantalla son únicos, prácticamente irreproducibles otra vez.

Esto abre una gran cantidad de opciones y posibilidades creando factores impredecibles. No obstante, no todo son ventajas, esto implica un sacrificio en el diseño debido a la impredecibilidad del propio contenido creado.

4.2. Origen y evolución

El mundo de los videojuegos explota este concepto al máximo en todos los procesos de desarrollo de este mismo. Además, al otorgar cierta aleatoriedad a los videojuegos esto genera una sensación que sumerge mucho más al jugador dentro del mundo del juego. Es por estos motivos por lo que ha ganado estos últimos años mucha fama el uso de generación procedural de contenido a la hora de desarrollar un videojuego. Sin embargo, a pesar de que el término pueda parecer muy moderno, la realidad es que este método de creación de recursos se lleva usando bastante tiempo. Famosos títulos como ‘Rogue’ de 1980 padre del género roguelike usaban estos mecanismos. Rogue utilizaba estos métodos para generar niveles aleatorios, si bien es cierto que no fue el primer juego en usar algoritmos semialeatorios para la creación de contenido, si que lo hizo con muy buenos resultados. Otros buenos ejemplos de que esta práctica se llevan usando tiempo pueden ser ‘Elite’ de 1984 o ‘The Elder Scrolls II: Daggerfall’ de 1996.

Sin embargo, la tecnología ha evolucionado y gracias a esto los algoritmos han conseguido llegar más allá de simplemente un par de procesos semialeatorios y basarse incluso en experiencias de jugadores para la creación de contenido.

La generación procedural de contenido se presenta como una solución a la producción de contenido para videojuegos, mediante la ayuda o automatización del proceso de generación. Haciendo uso de estas técnicas, grandes cantidades de contenido se

pueden generar de forma algorítmica sin la necesidad de trabajar en su creación. En otras palabras, la idea de la generación procedural de contenido es que parte del videojuego se genere computacionalmente a través de un procedimiento y algoritmos bien definidos, en lugar de ser creado a mano.

Como hemos mencionado anteriormente no es un descubrimiento actual, lleva usándose desde hace bastante tiempo y a lo largo de estos años ha habido un gran numero de videojuegos que lo han realizado de forma muy efectiva. Un buen ejemplo es el Minecraft publicado el 2009 es fácilmente el videojuego más popular hasta la fecha que haga empleo de estas técnicas.

Desde entonces, la cantidad de videojuegos que usan esta técnica de creación de contenido ha aumentado, y es un factor muy importante para los videojuegos producidos por pequeñas y medianas compañías. Algunos títulos posteriores son No Man's Sky de 2016 que implementaban generación procedural de galaxias, con planetas completamente

4.3. Procedural y aleatorio

Hay que diferenciar entre generación procedural y aleatoriedad. Por mucho que a simple vista puedan parecer sinónimos o incluso iguales, realmente guardan bastantes diferencias. La generación aleatoria es algo basado en al azar, en probabilidades y estadísticas.

La generación procedural se basa en procedimientos y seguir unos algoritmos que usualmente tienen en cuenta una gran variedad de factores. Estos factores pueden ser externos a la propia generación o pueden ser resultados anteriores, por tanto, los contenidos generados están supeditados a un mayor numero de datos a evaluar.

Pongamos un ejemplo de un videojuego RPG, role-playing game o juego de rol, en el que derrotas a un monstruo y este deja caer algunos objetos. Dentro de la lista de objetos que puede soltar hay oro, un arma cuerpo a cuerpo y un arma a distancia. Si la generación fuese aleatoria lo más rápido sería asignar una probabilidad a cada elemento, por ejemplo, repartiendo una cantidad de números y escogiendo uno al azar. Las probabilidades de cada objeto irían en función de la cantidad de números que posean.

En cambio, si la generación fuese procedural, el algoritmo podría tener en cuenta factores como el enemigo ha sido derrotado por un arma, ¿Qué clase de arma era?, ¿cuerpo a cuerpo o a distancia?, ¿tiene mucho oro el personaje? Y en función de estos factores alterar la generación de uno u otro objeto. Elementos propios de un juego como la dificultad elegida puede influir en este tipo de situaciones, tanto procedural como aleatoriamente.

Ahora surge la pregunta de cual es mejor. En cuanto a diseño y calidad, decidir entre contenido aleatorio o procedimental se basa en la cantidad de tiempo de que se dispone. Crear un algoritmo para generar contenido otorgara al videojuego la posibilidad jugarlo una y otra vez, sin embargo, hacerlo bien lleva tiempo. Con contenido aleatorio, puedes tener resultados en poco tiempo gracias a que habrá elementos codificados que luego se barajan para crear una experiencia aleatoria.

4.4. Clasificación y taxonomía

La generación procedural de contenido no supone un problema trivial, ya que estas técnicas se pueden aplicar a muchos de los factores que componen un videojuego desde un nivel muy bajo como sonidos o texturas, hasta un nivel más abstracto como sus propias reglas. Podemos diferenciar cinco clases principales de contenido que pueden generarse mediante generación procedural dedicadas exclusivamente al videojuego, y una más orientada a atraer nuevos jugadores.

Hay una gran variedad de algoritmos de PCG, dentro de ésta gran cantidad de algoritmos hay algunos que comparten ciertas características en algunos aspectos. Por lo que agruparlos en distintos grupos según estas características comunes es algo interesante. Hay dos formas distintas de agrupar estos algoritmos.

La primera es dada por el contenido que se puede generar. La segunda, deriva de la primera y se centra más en cuestiones técnicas.

Siguiendo la primera organización podemos distinguir el siguiente esquema:



Figura 4.1: Taxonomía según el contenido

1. En la capa 1 podemos encontrar lo que se conoce como game bits, son los componentes de más bajo nivel de un videojuego como pueden ser texturas, sonidos, objetos que componen la escena.
2. En la capa 2 vemos el espacio del videojuego como podría ser un nivel de juego o el entorno que puede rodear un juego de mundo libre.
3. En la capa 3 están presentes los sistemas de juego. Estos se refieren a esos componentes que hacen que el juego parezca más realista. Aquellos detalles que hacen que te puedas sumergir en su entorno. Un ejemplo de estos pueden ser los personajes no jugables y sus comportamientos.
4. En la capa 4 vemos los escenarios o fases del juego. Básicamente está compuesta por el orden en el que ocurren los eventos dentro del juego como puede

ser la narrativa de este mismo. El reto es generar nuevas historias automáticamente, cuya progresión se base en las decisiones del jugador. Otros ejemplos son rompecabezas, guiones gráficos, historias y niveles.

5. En la capa 5 encontramos el diseño del juego. Se componen básicamente de las reglas y patrones para jugar al juego.
6. En la capa 6 es lo que se conoce como contenido derivado o contenido que se crea a parte del videojuego en sí. Incluye imágenes y noticias sobre el nuevo juego. Interacción entre distintos jugadores si es multijugador el videojuego. Publicaciones en medios sobre las noticias recientes del juego.

Por otro lado, la clasificación más técnica es la siguiente:

1. Online versus Offline: Diferencia entre el contenido generado en tiempo de ejecución del juego contra el contenido preestablecido anteriormente que se crea antes del comienzo de un juego. Por ejemplo, un juego en el que los niveles se van creando conforme se van superando, de esta forma el estilo de juego varía siendo el mismo juego. Por otro lado, por ejemplo, un entorno generado proceduralmente que se utiliza como escenario constante del juego.
2. Necesario versus Opcional: Distingue el contenido que es necesario u obligatorio para alcanzar un objetivo frente a un contenido que puede ser simplemente decorativo. Por ejemplo, la llave que conduce a un enemigo de una mazmorra frente a una textura de una pared de la mazmorra.
3. Grado y dimensiones del control: Añade el control sobre el nivel de generación de contenido a través de unos parámetros modificables por el usuario. Un ejemplo de esto puede ser la temperatura de una red neuronal, variable utilizada para dar aleatoriedad a los resultados generados.
4. Genérico versus Adaptativo: El contenido genérico es aquel que siempre va a ser generado de la misma forma, mientras que en el adaptable alguno de sus parámetros va supeditado a diferentes hechos. Por ejemplo, los niveles de los personajes de dos jugadores pueden ser distintos en una misma zona del juego.
5. Estocástico versus Determinístico: Diferencia entre el contenido creado mediante algoritmos deterministas que producen el mismo resultado siempre que se den los mismos parámetros frente a algoritmos estocásticos que crean un contenido diferente cada vez.
6. Constructivo versus Generar y probar: Crear contenido en una sola pasada frente a generar un contenido y probarlo continuamente de tal forma que se va mejorando.
7. Generación automática versus autoría mixta: Generación de contenido totalmente producido por un algoritmo frente a un algoritmo que tenga en cuenta ciertos parámetros de entrada elegidos por el jugador. De esta forma puede cambiar el comportamiento del proceso de diseño.

Además de estas taxonomías se pueden agrupar según el tipo de algoritmo usado para la generación de dicho contenido.

1. Generación de números pseudoaleatorios.
2. Gramáticas generativas.
3. Filtrado de imágenes, morfología binaria y filtros de convolución.
4. Algoritmos espaciales, mosaico y estratificación, subdivisión de cuadrícula, vectorización, fractales y diagramas de Voronoi.
5. Modelado y simulación de sistemas complejos.
6. Inteligencia artificial como algoritmos genéticos o redes neuronales artificiales.

4.4.1. PCG y aprendizaje automático

En los tipos de algoritmos que hay para la generación procedural de contenido encontramos uno que es el que nos lleva a nuestro trabajo. Es el uso de aprendizaje automático para la generación de contenido, PCGML o procedural content generation via machine learning.

Es una técnica relativamente nueva a la hora de generar distintos tipos de contenido para videojuegos. Y la mayoría se basa en replicar los diseños ya existentes para que el jugador obtenga niveles infinitos de un juego con variaciones únicas entre ellos, pero manteniendo la impresión de que han sido creados por una persona. Otro enfoque que tiene la generación procedural mediante aprendizaje automático es la creación de mecánicas para un juego haciendo que el propio sistema PCGML sea un adversario contra el que jugar o una ayuda con la que jugar.

Como hemos mencionado el PCGML puede ofrecer una gran cantidad de contenido tanto en entidades del juego como en las propias mecánicas de este. Generalmente, la mecánica de PCG ofrece una repetición del juego. Sin embargo, esto va a variar significativamente con la ayuda del aprendizaje automático, por ejemplo, un jugador podría volver a jugar un juego con un comportamiento diferente debido a que los eventos del juego han variado y por tanto sus acciones han sido distintas. Otra ventaja del aprendizaje automático es la adaptación de dificultad entre un nivel y el jugador.

Un ejemplo de este caso sería que el sistema podría funcionar a favor o en contra la preferencia del jugador, utilizando un tipo u otro de arma en función de cómo evolucione el jugador y por tanto poder aumentar o disminuir la dificultad al momento. Otra ventaja es la conexión emocional con el jugador. Un ejemplo de esto es el efecto Tamagotchi que surgió del famoso juego de incubación de mascotas. Básicamente un sistema PCGML que necesita ser atendido a lo largo del juego generando así recuerdos positivos en los jugadores.

4.5. Desarrollo

Es recomendable atenerse a los objetivos que se quieren lograr, así como a sus propiedades requeridas cuando se desarrollan algoritmos sobretodo si están destinados a la generación de contenido. Se puede perder fácilmente el objetivo y los factores cruciales durante el desarrollo y esto puede provocar una mala experiencia en los jugadores.

Algunos de los factores más críticos son:

1. Velocidad: Tanto si un algoritmo PCG produce contenido durante el juego como si lo generó antes del propio juego, nunca debe excederse con la cantidad de tiempo que necesita para la generación del contenido ya que podría afectar a la experiencia del jugador.
2. Fiabilidad: Algunos algoritmos crean contenido desde cero sin saber lo que están creando mientras que otros son capaces de crear y evaluar su contenido. Esto es muy crucial sobretodo si lo que se está generando es el nivel que se va a jugar. Quizás la fiabilidad no es tan necesaria si el contenido a generar es meramente estético.
3. Controlabilidad: Es una propiedad básica en cuanto al PCG ya que otorga una gran ventaja tener el control para poder definir ciertos aspectos del contenido generado. Si que es cierto que esta propiedad puede interferir en ciertas ocasiones con la fiabilidad ya que el usuario está modificando a su gusto ciertos parámetros que a lo mejor no deberían de moverse de los valores por defecto.
4. Expresividad y diversidad: Es necesario desarrollar algoritmos que creen contenido con expresividad y diversidad tal y como lo haría una persona.
5. Creatividad y credibilidad: Al igual que la expresividad y diversidad, es necesario que el algoritmo produzca un contenido creíble que parezca diseñado por el hombre. El objetivo es que los jugadores no puedan distinguir entre un contenido generado por algoritmos y uno completamente diseñado por personas. Al fin y al cabo, es lo que se quiere conseguir, imitar el proceso creativo humano en un algoritmo.

Además de las anteriores características es recomendable que los algoritmos de PCG sean simples y centrados en generar un contenido específico. No tratar de con un algoritmo generar elementos muy dispares.

La idea es tener distintos algoritmos, una para cada contenido que se desee, y después mezclar los resultados. Tampoco es conveniente que los jugadores sean abrumados por muchos algoritmos PCG interactivos.

4.6. Conclusión

Como hemos podido observar a lo largo del desarrollo de este tema, cuando pensamos en generación procedural, nos referimos casi siempre al mundo de los videojuegos, ya que es algo muy visual y atractivo para el usuario. Sin embargo, la

generación procedural puede aplicarse en otros campos como el cine, donde se utiliza para crear rápidamente espacios atractivos y precisos. A parte del cine, también nos encontramos un acercamiento a la generación procedural en el mundo de la música, con la denominada música generativa. Este término fue popularizado por Brian Eno, y se refiere a ese tipo de música que está en constante cambio y es siempre diferente, la cuál es creada por un sistema. Sin embargo, a parte de todo esto, la generación procedural podría ser un término que podría ser aplicable a cualquier ámbito. Por ejemplo, en el ámbito de la arquitectura, un arquitecto podría servirse de la generación procedural, para crear edificios, y concluir sobre ciertas ideas para el posterior diseño de una nueva estructura, o para la planificación de una ciudad, etc.

El uso de contenido generado proceduralmente en un juego ofrece muchas posibilidades, como hemos estado viendo. El hecho es que el uso de modelos de PCG en los juegos es una forma de que los jugadores puedan experimentar el juego de una forma nueva cada vez que se juega.

Además, tiene el potencial de reducir significativamente los costos y tiempo de desarrollo, ya sea ayudando a los diseñadores o generando automáticamente el contenido. Esto es particularmente marcado en el desarrollo de secuelas o reskins. Un juego reskin es básicamente un juego donde el desarrollador ha cogido un juego anterior y le ha cambiado aspectos como los gráficos o la temática, pero el juego sigue siendo el mismo. Las mecánicas y el núcleo del videojuego es el mismo que el anterior. Es una práctica muy habitual en el desarrollo de juegos para móviles, donde la compañía lanzará varios juegos que parecen diferentes pero que comparten el mismo código. Por tanto, la parte que cambia como pueden ser texturas, mapas o niveles se generan automáticamente lo que supone la producción de un nuevo juego con costes prácticamente gratuitos. Además, el valor del nuevo producto se incrementa debido a tener un volumen de contenido mayor.

Una habilidad que nunca puede ser imitada por los diseñadores humanos, es la capacidad de generar contenido adaptado a cada jugador específico, y poder hacer en tiempo de juego. El contenido personalizado puede llegar a ser un cambio en la industria del videojuego como lo fue en su momento los juegos en línea o como esta siendo actualmente la realidad virtual.

Capítulo 5

Aplicación Práctica

5.1. Introducción

En el capítulo anterior, hemos estado hablando sobre qué es la generación procedural de contenido, qué técnicas utiliza y qué aplicaciones tiene en un mundo tan tecnológico y avanzado como en el que vivimos. Debido a todo ello, consideramos que este concepto era algo muy interesante sobre lo que investigar y trabajar. En un primer momento tuvimos la idea clara de que queríamos relacionarlo con el mundo de los videojuegos, y sobre todo con el Super Mario, uno de los juegos más aclamados de toda la historia de los videojuegos. Actualmente existen herramientas como MarioMaker para la creación de mapas basados en este juego. Sin embargo, nosotros quisimos realizar una aplicación en Unity por la que el ordenador aprendiera como eran los mapas originales de Mario para así poder crear modelos o estructuras a partir de las cuales crear nuevos mapas del Mario tanto monotemáticos (Ej: mapa 1 del mundo 1) como multitemáticos (Ej: unión del mapa 1 del mundo 1 con el mapa 2 del mundo 2), consiguiendo así la mezcla de artes entre los diferentes mapas (zonas subterráneas con zonas acuáticas).

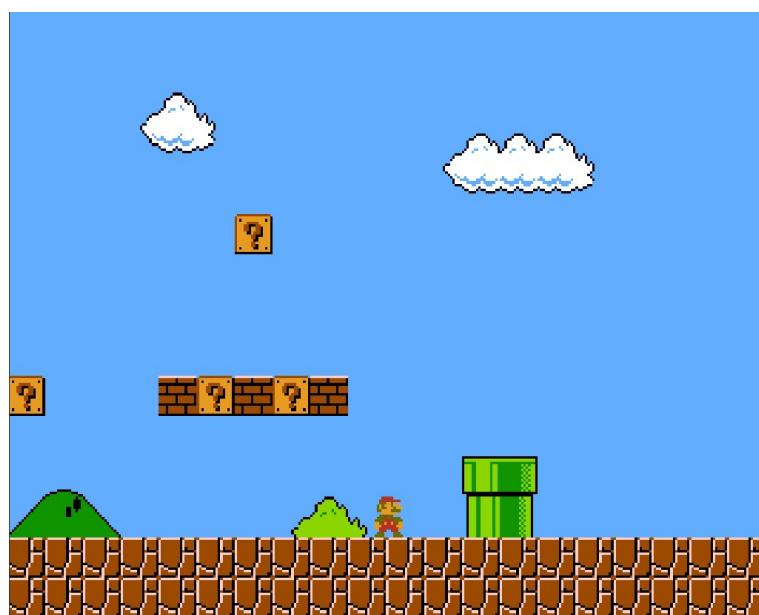


Figura 5.1: Fragmento del mapa 1 original del primer mundo

En la sección 5.2 explicaremos más a fondo el juego Super Mario, su historia, personajes jugables, items o transformaciones. En la sección 5.3 hablaremos sobre los retos que lleva consigo haber realizado esta aplicación y cuales son los resultados esperado. Finalmente, en la sección 5.4 comentaremos como hemos interpretado los datos para poder representar cada tile del mapa en unity y para su uso en los algoritmos de aprendizaje. Finalmente, la sección 5.5 corresponde a los algoritmos utilizados para la generación de mapas: NGrams y RNN.

5.2. Nuestro Juego

Super Mario Bros

Resumen

Super Mario Bros es un videojuego publicado para la Nintendo Entertainment System (NES) en 1985. Este juego cambio la forma de jugar de todos sus juegos arcade predecesores, e instauró los juegos de plataformas laterales. No es el primer juego de la franquicia de Mario, sin embargo, es el más icónico. Introdujo una gran diversidad de elementos, desde power-ups hasta diversos tipos de enemigos con la premisa de rescatar a la Princesa Toadstool (Peach) del rey Koopa (Bowser).

En nuestro proyecto hemos recreado un prototipo del Super Mario Bros, no todas las características del juego original han sido implementadas.

Historia

El siguiente texto esta cogido directamente del manual de instrucciones:

One day the kingdom of the peaceful mushroom people was invaded by the Koopa, a tribe of turtles famous for their black magic. The quiet, peace-loving Mushroom People were turned into mere stones, bricks and even field horse-hair plants, and the Mushroom Kingdom fell into ruin.

The only one who can undo the magic spell on the Mushroom People and return them to their normal selves is the Princess Toadstool, the daughter of the Mushroom King. Unfortunately, she is presently in the hands of the great Koopa turtle king.

Mario, the hero of the story (maybe) hears about the Mushroom People's plight and sets out on a quest to free the Mushroom Princess from the evil Koopa and restore the fallen kingdom of the Mushroom People.

You are Mario! It's up to you to save the Mushroom People from the black magic of the Koopa!

Un día el Reino Champiñón fue invadido por los Koopa, una tribu de tortugas que podían utilizar magia. Utilizaban su magia para transformar a las personas del Reino Champiñón en objetos inanimados como consecuencia el reino cayó. Solamente la princesa Peach puede deshacer la maldición. Y restaurar la normalidad pero Bowser la mantiene cautiva. Mario escucha de la situación de la princesa y se embarca en una aventura para derrotar a la tribu Tortuga y devolver la paz al reino.

Géneros

Plataformas 2D lateral

Modos

Modo 1 jugador

Público objetivo

Cualquier edad:



Figura 5.2: ESRB E (Todo el mundo)



Figura 5.3: PEGI 3 (Tres años y mayores)



Figura 5.4: CERO A (Todas las edades)



Figura 5.5: ACB L (General)

Plataformas

PC: Aplicación

Descripción

Nuestro Super Mario Bros esta dividido en dos mundos, cada uno de ellos contiene cuatro niveles. Sin embargo puedes generar infinitos mundos a partir de los ocho originales.

Mario tiene que conseguir llegar hasta el final del nivel saltando agujeros y evitando a los enemigos de su camino. También hay tuberías a lo largo del camino, en alguna de las cuales puedes entrar y visitar varias habitaciones secretas con monedas y volver más adelantado al nivel. Los enemigos son Goombas, Koopa Troopas. Estos enemigos pueden ser derrotados cuando Mario salta encima de ellos. Los Koopa Troopas se esconden en sus caparazones cuando saltas en ellos, después puedes

patearlos. Hay un nivel que tienen lugar debajo del agua en el cual Mario puede bucear. Si Mario normal recibe un golpe, cae por un hoyo se vuelve a empezar el nivel. Mario puede obtener power-ups de los bloques ?. Con la Seta Mágica Mario se transforma en Super Mario, con lo que puede destruir los ladrillos.

Al final de cada nivel hay un castillo con un banderín. Cuando Mario alcanza el banderín baja la bandera enemiga y entra al castillo completando el nivel.

Los ocho niveles originales a partir de los cuales puedes generar infinitos mundos son tres principalmente sobre tierra, uno subterráneo, dos de castillo, uno submarino, y uno que mezcla sobre tierra, subterráneo y en el cielo.

Mecanismos e interfaz de juego

a, d, flechas derecha e izquierda: Moverse y cambiar de posición en una beanstalk, desplazarte horizontalmente en modo vista.

s, flecha abajo: Entrar en una tubería y escalar en una beanstalk, desplazarte verticalmente en modo vista.

w, flecha arriba: Saltar, nadar hacia arriba y escalar en una beanstalk.

l: Entrar en modo vista.

r: Sales del modo vista.

o: Crecer como si consiguieras una seta mágica.

p: Disminuir como si perdieras una vida.

ruleta del ratón: Hacer zoom en el modo vista.

Personajes jugables

Mario: Es el personaje principal y protagonista del videojuego. También es la mascota principal de Nintendo.



Figura 5.6: Mario

Enemigos

Cheep-cheep: Un pez que encuentras nadando en el agua.

Fire-Bar: Varias bolas de fuego unidas moviéndose en círculos.

Koopa Troopa: Un soldado del imperio Tortuga que avanza por el mapa. Si los pisas se esconden en su cocha, pudiendo patearla para derrotar otros enemigos. Los verdes se mueven hacia adelante y atrás, los rojos se dan la vuelta al encontrar un agujero.

Little Goomba: Una seta traidora que camina hacia adelante y atrás. Son los enemigos más débiles del juego.

Piranha Plant: Una planta carnívora que vive en las tuberías. Sale para intentar morder a Mario y después se esconde. No sale si Mario está cerca.

Bowser falso: Enemigo de los castillos de cada mundo. En caso de quemarlo sale

quién es el impostor.

Bowser: Conocido también como el Rey Koopa, es el mayor enemigo de Mario.
IMG

Objetos

Monedas.



Figura 5.7: Monedas

Seta mágica, Otorga la transformación de Super Mario.



Figura 5.8: Seta mágica

Transformaciones

Mario: Forma inicial del juego. Es la forma más débil.



Figura 5.9: Mario

Super Mario: Requiere de una Seta mágica. Si es golpeado por un enemigo vuelve a la forma normal.



Figura 5.10: Super Mario

Niveles

Hay dos mundos y en cada uno contiene cuatro niveles de forma original, sin embargo puedes generar cuantos quieras.



Figura 5.11: Primer mundo del Super Mario Bros

5.3. Retos

El objetivo principal como hemos comentado anteriormente en la introducción, es el de generar mapas del SuperMario para un diseñador y no para un jugador casual de videojuegos, ya que incluye una jugabilidad muy limitada. La aplicación permite por un lado mostrar los mapas originales del mundo 1 y del mundo 2, y por otro lado, generar nuevos modelos y entrenamientos para la creación de nuevos mapas, o utilizar modelos ya existentes, para la generación también de nuevos mapas. Estos entrenamientos, pueden estar basados en un solo mapa, o en varios mapas, lo que implicará que los mapas generados a partir de ese "training", serán una combinación de los mapas seleccionados en el primer paso. Por todo ello, pensamos que nuestra aplicación podría ser utilizada en juegos basados en el SuperMario, cuyo propósito fuera la creación automática de un nuevo mapa cuando el jugador superara el nivel actual, o para diseñadores que quisieran adoptar ideas para la generación de nuevos mapas.

Hacer esto ha supuesto un gran reto, ya que hemos tenido que generar de cero esos 8 mapas mapas originales del SuperMario a través de Tile2Unity, colocando cada tile mano a mano. Tras ello otro reto fue la lectura de cada mapa tile a tile y colocándolo correctamente en el mapa. Sin embargo, estos problemas fueron algo menor. El principal surgió, cuando quisimos conectar los scripts de python, los cuales contienen los algoritmos de machine learning comentados en la sección 5.5. Para ello, finalmente, conseguimos conectar ejecutar el script de Python a través de linea de comandos desde el terminal de windows.

En cuanto a la jugabilidad, dentro de la existente en la aplicación, el reto más destacable fue el de conseguir que Mario interaccionara con las tuberías para poder acceder a las zonas subterráneas/acuáticas. Para ello, lo que hicimos fue fijarnos en aquellas tuberías que en su vertical tenían una tubería de salida de una zona secreta, asignando a la inmediatamente anterior, la función de acceso a la zona secreta.

Finalmente, en lo que respecta a los retos planteados y cumplidos con los algoritmos de aprendizaje automático, con Ngrams el principal reto que nos surgió fue el del cálculo de probabilidades correctamente, para así poder saber cual sería la siguiente parte del mapa que correspondería. Con redes neuronales, el problema más destacable fue la interpretación correcta de los datos. Estos datos serán explicados a fondo en la sección 5.5

En la siguiente sección hablaremos como hemos interpretado cada mapa, y qué es un slice

5.4. Tipos de datos

En nuestro proyecto creamos niveles para el juego Super Mario Bros de la NES. Para lograr esto seguimos un proceso en el que utilizamos varios tipos de datos a nuestro favor a partir de imágenes de los niveles originales.

A partir de una imagen completa del nivel original sacamos un .csv con los distintos bloques. Para ello cargamos la imagen en Tiled a modo de plantilla y una hoja de sprites para su creación.

Una vez completado el mapa tile a tile se exporta en formato csv para que sea leído por el algoritmo elegido por el usuario.

Tanto el algoritmo N-gram, de predicción de texto, como las redes neuronales recursivas tratan el .csv de la misma manera. Cada columna de sprites del nivel es lo que más tarde en el apartado de algoritmos trataremos como una palabra, y a su vez el nivel como una frase o texto completo.

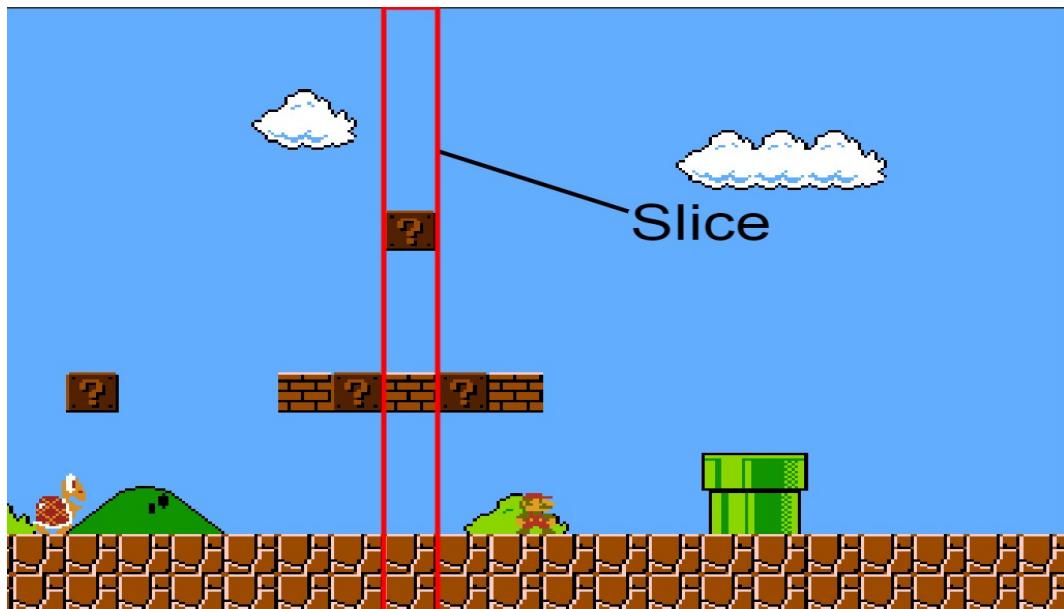


Figura 5.12: Slice de un nivel

Al principio los mapas eran de 15 bloques de altura por el largo que tuviesen según el mapa original. Sin embargo, tras aplicar la idea que definimos como multitema, los mapas pasan a tener 60 bloques de altura por la longitud correspondiente de ancho. Esto es debido a la oportunidad de incluir distintos temas en distintas alturas del mapa para conseguir su mezcla. Este concepto sera explicado más detenidamente más adelante.



Figura 5.13: División de las zonas de los niveles en tiled

Ademas, al cargar el .csv este es transformado a una matriz de números y esta a su vez es traspuesta para mayor facilidad de obtención de las frases, slices, del nivel. Tras la aplicación del algoritmo se genera una nueva matriz con el nuevo nivel generado. Esta matriz deberá de ser traspuesta de nuevo para generar un nuevo fichero .csv que mantenga las mismas características que el original. Salvo por longitud en anchura que es la especificada por el usuario, es decir la longitud del nivel deseada y el nombre del fichero. Tras esto el fichero se deja en un directorio de Unity donde están el resto de ficheros de mapas.

Por otro lado otros tipos de datos que merece la pena nombrar son los prefabs de los tiles que utilizamos para crear esos mapas en Unity. Los cuales dependiendo del tipo de bloque tienen unos u otros componentes.

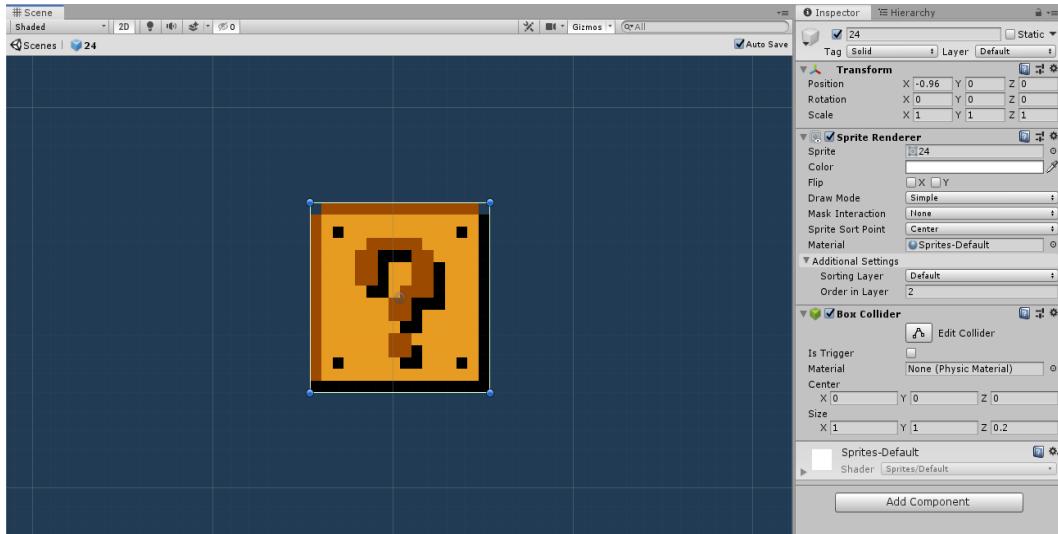


Figura 5.14: Prefab de un tile

5.5. Algoritmos

A continuación, desarrollaremos como hemos conseguido la generación de mapas de manera automática a partir de una serie de algoritmos.

Los algoritmos utilizados principalmente en nuestro proyecto han sido dos: Los N-gramas y las redes neuronales recursivas. Como breve introducción de ambos, podríamos decir que los N-gramas, nos permiten saber cuál es la siguiente subsecuencia a elegir dentro de una secuencia de n datos y a partir de ciertas probabilidades. Por otro lado, las redes neuronales recursivas, permiten un análisis muy potente de secuencias, con una serie de conexiones arbitrarias entre las neuronas, lo cuál permite crear ciclos entre las propias neuronas, dotando así a toda la red de "memoria".

5.5.1. N-gramas

Introducción

Predecir unas pocas palabras es mucho más fácil que predecir un texto entero, por ello ir prediciendo pequeñas secuencias de palabras progresivamente ayuda a obtener el texto final. Esta tarea se puede realizar utilizando modelos que asignan probabilidades a cada posible siguiente palabra, también pueden servir para asignar probabilidades a frases enteras.

Las probabilidades son esenciales en tareas en las cuales hay que identificar palabras en textos. Estas probabilidades sirven para identificar si una combinación de palabras es más probable de aparecer que una otra combinación. La asignación de probabilidades es fundamental en tareas como el reconocimiento de voz, corrimiento ortográfico o la traducción automática.

Aquellos modelos que asignan probabilidades a las secuencias de palabras son llamados modelos del lenguaje o LMs (su acrónimo en inglés). El LM más simple es el n-grama.

Un n-grama es una subsecuencia de n elementos escogidos a partir de un texto dado. Esta subsecuencia puede estar formada por caracteres o palabras. Debido a la ambigüedad que pueda surgir, hay que considerar que podemos usar el término n-grama para referirnos al LM o la propia secuencia de palabras.

Se puede decir que un n-grama es un token formado a partir del movimiento de una “ventana” a través de un texto dado, o bien llamado “corpus”. La ventana se mueve creando subsecuencias dependiendo del tamaño N del token, de este modo, por ejemplo, si $N=2$ se van formando tokens de 2 palabras o caracteres. Dependiendo del movimiento de la ventana, los n-gramas pueden clasificarse en dos tipos: los basados en caracteres y los basados en palabras. También pueden considerarse las frases, como nosotros hicimos en una primera implementación para obtener los mapas del Super Mario Bros. Sin embargo, este último tipo carece de sentido y coherencia según nuestra experiencia. Más adelante, en el apartado 5.6.1 se expondrán estos resultados y la correspondiente explicación.

Daniel Jurafsky y James H. Martin definen, en su libro “Speech and Language Processing”, a los n-gramas como una secuencia de N palabras. De este modo, se puede obtener un 1-grama (unígrafo) como “hola”, un 2-grama (bigrama) que es una secuencia de dos palabras como “hola qué”, un 3-grama (trígrafo) que es una secuencia de tres palabras como “hola qué tal”, etc.

El n-grama más simple es el unígrafo, donde $N=1$ y es el token formado por un único carácter o palabra. Normalmente, el valor de N es un número fijo que depende del corpus dado. Cada n-grama es un par que representa el token estudiado y la frecuencia con la que ese token aparece. En el modelo unígrafo se seleccionan palabras de una en una, esto hace que se elimine la noción de orden entre las palabras y da la misma probabilidad a cualquier permutación de un texto. Aquellos n-gramas más grandes si que mantienen un cierto orden entre las palabras.

Los n-gramas son el método más simple de categorizar textos, donde la N es el número de palabras usado para dividir el corpus. Inicialmente se lleva a cabo un preprocesamiento general y, después, se divide el texto en n-gramas.

Historia de los N-gramas

El modelo n-grama es un modelo probabilístico ideado por Andrey Markov y más tarde fue introducido por Claude Elwood Shannon en 1948, en su teoría de la comunicación (Information Theory).

En ella, expresaba su preocupación por el conocimiento estadístico sobre la fuente de la información. Por ejemplo, en telegrafía, se transmiten mensajes que consisten en secuencias de palabras que no son aleatorias.

El conjunto de estas palabras forma frases y mantienen un orden estadístico. Esto se puede demostrar observando que hay letras que son más frecuentes que otras. En español, la letra E tiene un porcentaje de aparición del 13,68 % , mientras que la Z tiene un 0,52 %. También hay secuencias de letras que son más frecuentes que otras.

Estas condiciones ofrecen la posibilidad de ahorrar tiempo si se codifican los mensajes adecuadamente en señales. En 1838, Samuel Morse optimizó, usando esta estructura estadística de letras y palabras, el lenguaje máquina que desarrolló en 1830. El len-

guaje desarrollado por Morse consistía en la codificación de letras y números como la combinación de puntos, guiones y espacios. La optimización que llevó a cabo se basó en la frecuencia de las letras en inglés, es decir, las letras más frecuentes se representan con combinaciones más cortas de esos símbolos.

Los modelos que producen secuencias de caracteres estructuradas por probabilidades son llamados procesos estocásticos. En teoría de la probabilidad un proceso estocástico o proceso aleatorio es un objeto matemático definido normalmente como una familia de variables aleatorias.

En estos procesos se consideran los casos en los que caracteres seguidos secuencialmente de otros son elegidos por probabilidades dependientes en los caracteres anteriores.

Claude Elwood Shannon uso los N-gramas para analizar y predecir texto en inglés y sirvió de precedente para su uso en otros temas.

Google ofrece una gran cantidad de información hacer de los n-gramas más frecuentes. Esta información ha sido obtenida a través del análisis de millones de libros, más de cinco millones, publicados desde hace quinientos años.

Esa información está disponible públicamente en Google Ngram Viewer.

Este consiste en la búsqueda de la aparición de frases, elegidas por el usuario, en un corpus de libros (por ejemplo, “Inglés Británico”, “Francés”) durante los años elegidos. Una vez elegidos estos parámetros se muestra un gráfico con los resultados. El gráfico representa el porcentaje de aparición de un n-grama elegido por el usuario respecto a todos los n-gramas contenidos en los libros de ejemplo. La información recolectada por Google es usada para implementar sus sistema de recomendación de consultas. Se puede visitar la página web de Google Ngram Viewer: <https://books.google.com/ngrams/info> para obtener más información acerca de esta aplicación.

En esta página <https://books.google.com/ngrams/graph?content=leadership> se puede probar de manera práctica la aplicación y puede resultar interesante para analizar el uso del lenguaje durante el avance de los años.

PARA CITAR A GOOGLE: I'm writing a paper based on your results. How can I cite your work? If you're going to use this data for an academic publication, please cite the original paper: Jean-Baptiste Michel*, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K. Gray, William Brockman, The Google Books Team, Joseph P. Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, Steven Pinker, Martin A. Nowak, and Erez Lieberman Aiden*. Quantitative Analysis of Culture Using Millions of Digitized Books. *Science* (Published online ahead of print: 12/16/2010)

Uso de los N-gramas

Los N-gramas ofrecen una clasificación de texto eficiente y efectiva que puede ser usada en aplicaciones de corrección o detección de errores ortográficos, identificación de idioma, etc. El uso de modelos N-grama es una idea simple, pero de gran utilidad y efectiva en muchas ocasiones.

1. Identificación de lenguajes de programación:

Así como los N-gramas sirven para identificar idiomas en textos, también pueden ser usados para identificar lenguajes de programación.

A pesar de que los n-gramas no son un concepto nuevo, como ya se ha visto en el apartado sobre su historia, estos han surgido como una alternativa viable frente a las librerías para la detección de lenguaje natural. Este tipo de librerías están implementadas sobre una gran cantidad de extensos diccionarios en los que realizan comprobaciones de coincidencias. Debido a esta implementación, necesitan de un mayor tiempo para compilar. Además, cuanto más extensos sean estos diccionarios, mayor será el tamaño de la librería.

También puede darse la posibilidad que se reciban palabras que no están incluidas en la librería. Por otro lado, los n-gramas son más eficaces, además de poder usar archivos más pequeños en tamaño como entrenamiento.

La identificación de lenguajes de software es un problema que consiste en determinar en qué lenguaje de programación un fragmento de código está escrito. Las técnicas de identificación de lenguajes de software son aplicables a muchas situaciones. Desde el soporte universal de los IDE (Integrated Development Environments) hasta el análisis de código de legado (van Dam Zaytsev, 2016).

La mayoría de las heurísticas usadas se basan en metada del software, como las extensiones de los archivos, o en el análisis de la gramática de un texto como la búsqueda de palabras claves. Sin embargo, van Dam y Zaytsev proponen el uso de modelos estadísticos de lenguaje como n-gramas, skip-gramas, multinomial naïve Bayes y distancia normalizada de compresión.

2. Traducción automática:

La traducción de texto de un idioma a otro es uno de los principales objetivos del procesamiento del lenguaje natural. Sin embargo, esta tarea no es tan sencilla como asumir que una palabra se puede traducir tal cual a otra de un idioma diferente. No siempre se corresponde una palabra de un idioma con una de otro idioma, por lo que la traducción automática no se puede conseguir únicamente con el análisis sintáctico y léxico. También hay tener en cuenta la ambigüedad del texto y de los mensajes a transmitir.

El proceso de traducción puede definirse en dos pasos. Estos son la decodificación del significado del texto y la codificación de este significado en el idioma deseado.

El primer traductor automático fue CANDIDE creado por IBM. El traductor de Google y los programas similares basados en traducción estadística funcionan detectando patrones en millones de documentos que han sido previamente

traducidos por humanos. Cuanto mayor sean los documentos traducidos por humanos en un idioma, mejor será la traducción del texto.

3. Filtrado de contenido:

El filtrado de contenido es un método muy útil para extraer características estructuradas de texto desestructurado. En esencia, permite la obtención de palabras que nos aportan una gran información y contexto. Para ello, este proceso se puede llevar a cabo mediante diferentes subprocesos, tales como la eliminación de palabras vacías (en inglés conocidas como stop words), la eliminación de palabras raras (en inglés rare words) y la reducción de una palabra a su raíz (en inglés este método es conocido como stemming).

Las palabras vacías o stop words son aquellas palabras sin significado que no aportan mucho contexto como, por ejemplo, los artículos, pronombres, preposiciones, etc. El uso de este término se le atribuye a Hans Peter Luhn, informático alemán que trabajó en IBM siendo el primero en emplear la estadística en los análisis textuales en Recuperación de Información y fue el creador del indexado KWIC (Key Words In Context).

Las palabras raras o rare words son palabras que no aparecen con gran frecuencia en un texto, por lo que pueden ser palabras mal escritas, palabras inventadas... Por ello, casi todas estas palabras con conteos bajos suelen ser palabras que no aportan información alguna.

También pueden ser palabras que simplemente aparecen muy pocas veces, por lo que al modelo estadístico le pueden parecer ruido y no ser nada útiles.

Este tipo de palabras se pueden eliminar empleando expresiones regulares, es decir, eliminar aquellas palabras que coincidan con un patrón. Por ejemplo, se pueden borrar palabras que contengan números, letras repetidas más de tres veces o palabras de una sola letra.

Otra, opción para borrar las palabras raras es no tener en cuenta aquellas que repiten menos X veces. Esta frecuencia mínima para tener en cuenta palabras se debe analizar y recalcular dependiendo de los textos de entrenamiento. Por otro lado, se pueden mantener las palabras infrecuentes y agruparlas a todas como infrecuentes.

Muchas veces tenemos palabras que se escriben de forma un poco diferente, pero mantienen la misma raíz y expresan un mismo significado. Es decir, aquellas pertenecientes a la misma familia de palabras. Un ejemplo sería el siguiente: flor, flores, florecilla, floricultura, florería, florista, florecita, floreado... Todas se escribe de una manera distinta, pero están relacionadas con las flores. La reducción a su raíz o stemming consigue cortar las palabras y obtener su raíz.

Existe el debate sobre la utilidad de este método que normalmente es usado para comparar el rendimiento características no reducidas a su raíz frente a aquellas sí reducidas.

Con el término características nos referimos a los datos que se pueden obtener de un texto y que servirán de utilidad posteriormente.

La bolsa de palabras o Bag of Words (BOW) es el método más simple para convertir un texto en características estructuradas.

	it	is	puppy	cat	pen	a	this
it is a puppy	1	1	1	0	0	1	0
it is a kitten	1	1	0	0	0	1	0
it is a cat	1	1	0	1	0	1	0
that is a dog and this is a pen	0	2	0	0	1	2	1
it is a matrix	1	1	0	0	0	1	0

Figura 5.15: Conversión de texto a su representación en bolsas de palabras

4. Detección de plagio:

Los n-gramas también sirven para detectar un posible plagio realizado sobre un texto. De este modo, el texto sospechoso es comparado con ciertos textos usados como corpus. Según un estudio realizado por Alberto Barrón-Cedeño y Paolo Rosso (Universidad Politécnica de Valencia) se determina que los mejores resultados para detectar casos de duplicados son obtenidos mediante bigramas y trigramas.

Esta tarea es más sencilla de llevar a cabo cuando el plagio se realiza de manera exacta, sin embargo, resulta más compleja cuando se producen cambios en el texto original. Por ello, se lleva a cabo una comparación mediante n-gramas. Textos independientes tienen una pequeña cantidad de n-gramas en común, lo que facilita la detección de plagios puesto que esta cantidad de n-gramas comunes será mayor. La probabilidad de encontrar n-gramas comunes en textos

diferentes disminuye cuando la n aumenta (Alberto Barrón-Cedeño y Paolo Rosso).

Para conseguir obtener la similitud entre dos documentos se utiliza el coeficiente de Jaccard que se expresa como en la siguiente ecuación:

$$\text{sim}(d1, d2) = |\text{Sn}(d1) \cap \text{Sn}(d2)| / |\text{Sn}(d1) \cup \text{Sn}(d2)|$$

Donde $\text{Sn}(d1)$ es el conjunto de n -gramas en el documento uno y $\text{Sn}(d2)$ el conjunto de n -gramas en el documento dos. Por último, se establece un umbral para determinar si el índice obtenido de la ecuación indica que se ha producido plagio.

5. Corrector automático:

El modelo n -grama puede ser empleado para la construcción de aplicaciones de detección y corrección de errores ortográficos, para el autocompletado de palabras y para ofrecer sugerencias como continuación a una palabra.

El proceso para realizar estas tareas se denomina corrección ortográfica o spell checker. Se detectan y proporcionan sugerencias para palabras incorrectas. Los algoritmos para implementar estos procesos se basan en diccionarios que contienen vocabulario con el que comprobar las palabras a corregir. Cuanto mayor sea este vocabulario mayor será el porcentaje de detección de errores. Sin embargo, un problema común es la falta de vocabulario y datos que contienen los diccionarios usados por estos algoritmos.

Existen múltiples experimentos y aplicaciones que llevan a cabo estas tareas. Por ejemplo, cabe destacar el experimento realizado por Youssef Bassil en el que usa un corpus de n -gramas de Yahoo! que comprime trillones de secuencias de palabras y n -gramas.

El resultado de su algoritmo propuesto demuestra una eficacia en la corrección de errores del 94 %, distribuido en un 99 % en errores de palabras no existentes (non-words) y en un 65 % en errores de palabras reales. La razón de estos porcentajes tan altos, en comparación con otros estudios, es el uso del corpus de n -gramas de Yahoo!, puesto que usado como diccionario ofrece una gran variedad de vocabulario como jerga técnica o juvenil o de otros tipos, nombres propios, acrónimos, etc.

Modelo del lenguaje n -grama

El n -grama es el modelo del lenguaje más simple. Para recordar, se denomina modelo del lenguaje a aquellos modelos que asignan probabilidades a secuencias de palabras o caracteres de tamaño n .

Permite clasificar texto desconocido, es decir, no usado para entrenar previamente, con gran efectividad y certeza.

Este modelo es capaz de predecir la siguiente palabra a una secuencia de palabras previas.

Para lograr predecir palabras se debe calcular la probabilidad de una palabra dada una secuencia de palabras anteriores, es decir, dada una historia. La manera de conseguir estimar esta probabilidad es mediante el conteo de frecuencias relativas que consiste en contar las veces que una palabra u otras (historia), dependiendo del valor de n para el n-grama, es seguida por otra. Un ejemplo, sería contar en un corpus las veces que la frase “me gusta el” es seguida por “café”. La ecuación para esta probabilidad es:

$$P(\text{cafe}|\text{me gusta el}) = \frac{C(\text{me gusta el cafe})}{C(\text{me gusta el})} \quad (5.1)$$

Probabilidad condicional a la historia

Esta forma de estimar probabilidades a través del conteo de las apariciones de una palabra respecto a otras es eficaz en muchas ocasiones, sin embargo, no contamos con corpus lo suficientemente grandes como para estimar de manera correcta la gran mayoría de las veces. También se podría descomponer la probabilidad de una secuencia de palabras usando la regla de la cadena de probabilidad:

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1}) \quad (5.2)$$

Regla de la cadena de la probabilidad

Se multiplican las probabilidades condicionales de cada palabra de la secuencia, pero por la misma razón anterior no se puede calcular probabilidad exacta de una palabra respecto a una secuencia. Por ello, los n-gramas calculan la probabilidad aproximada de una palabra respecto a unas pocas palabras anteriores. De este modo, siguiendo el ejemplo anterior en lugar de calcular la probabilidad de que a “me gusta el” le siga “café” un bigram calcularía la probabilidad de que a “el” le siga “café”: $P(\text{café} | \text{el})$

Las probabilidades se estiman usando el método maximum likelihood estimation (MLE), por el cual se realiza el conteo de todos los n-gramas (secuencias de palabras de tamaño n) y se normaliza por la suma de todos los n-gramas que comienzan por esa secuencia de longitud n-1. La ecuación es la siguiente:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})} \quad (5.3)$$

Maximum likelihood estimation (MLE)

Debido a que los datos de entrenamiento no comprenden a todos los n-gramas posibles en un idioma puede producirse el hecho de encontrar n-gramas que no han aparecido previamente en los datos de entrenamiento y asignándoles una probabilidad de cero. Una probabilidad de cero en un n-grama produce que toda la frase tenga una probabilidad de cero. Para solucionar este problema y ser capaz de calcular una probabilidad realista se usa el smoothing.

Existen diversos métodos de suavizado como el add-1, add-k, stupid backoff y Kneser-Ney.

Estos modelos tienen que ser entrenados con grandes corpus para permitir realizar una mejor predicción.

Proceso de desarrollo

En primer lugar, empezamos investigando acerca de este modelo del lenguaje, en qué consistía, su utilidad, modo de empleo para una posible implementación, características, etc. Y acto seguido estudiamos la teoría sobre los n-gramas con el objetivo de obtener el conocimiento necesario para poder desarrollar el modelo.

Utilizamos como base de estudio el capítulo *N-gram Language Model* del libro *Speech and Language Processing* de Dan Jurafsky y James H.Martin, profesores de la Universidad de Stanford y de la Universidad de Colorado. Este capítulo explica de una manera amplia el modelo n-grama, así como métodos de mejora, métodos de evaluación de la validez del modelo, posibles problemas...

Ya en el lado práctico, a partir de la ecuación general calculamos las ecuaciones para calcular las probabilidades de los distintos n-gramas dependiendo de su tamaño n. De este modo, conseguimos entender los distintos parámetros y entender la ecuación para cualquier n-grama.

Una vez hecho esto, en un pequeño texto de ejemplo calculamos las probabilidades de cada bigram extraído del texto y, así familiarizarnos con la fórmula para este cálculo.

Para obtener la probabilidad de un n-grama se realiza un conteo de la frecuencia con la que aparece el n-grama en un corpus y se divide por las veces que se repite la secuencia, de tamaño n-1, previa de ese n-gram.

Para el caso general, como ya se ha mostrado previamente, la ecuación es la (5.3).

Más tarde, trajimos la ecuación para los trigramas y obtuvimos las probabilidades para los trigramas distintos de cero en un corpus de ejemplo. El corpus era el siguiente:

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

Las etiquetas `< s >` y `</ s >` se emplean para indicar el inicio y fin de una frase. También sirve para cuando se empiezan a construir los n-gramas desde el inicio de un texto y dado el tamaño n determinado no se puede recorrer atrás en la historia. Así, como no existen más palabras hacia atrás en el inicio del texto, pues, como es lógico, antes de la primera no puede haber más palabras, se rellenan con las etiquetas de inicio `< s >`. Por ejemplo, el cálculo para un trígrafo al principio del texto anterior sería $P(I|< s >< s >)$.

Otro ejercicio realizado con el objetivo dominar las fórmulas y cálculos de n-gramas fue el de calcular la probabilidad de la frase “I want chinese food” de dos formas distintas:

1. Usando las probabilidades útiles, sin modificaciones:

$$\begin{aligned} P(< s > i \text{ want chinese food} </ s >) = \\ P(i|< s >)P(\text{want}|i)P(\text{chinese}|\text{want})P(\text{food}|\text{chinese})P(</ s >|\text{food}) = \\ 0.25 \times 0.33 \times 0.0065 \times 0.52 \times 0.68 = 0.0002 \end{aligned}$$

2. Usando el método add-1 de suavizado: este método determina que se debe sumar 1 al conteo de todos los n-gramas para evitar probabilidades de 0 y conseguir unas probabilidades más realistas más allá del corpus de entrenamiento. La fórmula sería:

$$P_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV} \quad (5.4)$$

Suavizado add-k

donde V es el vocabulario del corpus, es decir, el número de palabras únicas que aparecen, sin contar repeticiones. Para el caso add-1 (Laplace smoothing) se sustituiría k por 1.

De este modo la probabilidad suavizada usando el método add-1 sería

$$\begin{aligned} P(< s > i \text{ want chinese food} </ s >) = \\ P(i|< s >)P(\text{want}|i)P(\text{chinese}|\text{want})P(\text{food}|\text{chinese})P(</ s >|\text{food}) = \\ 0.19 \times 0.21 \times 0.0029 \times 0.052 \times 0.4 = 0.000024 \end{aligned}$$

Como se puede observar las probabilidades sin suavizar son mayores, puesto que para suavizar se da cierto porcentaje aquellos bigramas que tenían un probabilidad de cero lo que produce que las probabilidades restantes se reduzcan.

Los datos utilizados para la realización del ejercicio son proporcionados en el pdf del libro comentado anteriormente. Asumiendo algunas probabilidades tal como indican en el ejercicio. Se asume que las probabilidades de $P(i|< s >)$ = 0.25 y de $P(</ s >|\text{food})$ = 0.68 y que las de add-1 de $P(i|< s >)$ = 0.19 y para $P(</ s >|\text{food})$ = 0.4

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figura 5.16: Tabla del conteo de bigrams para ocho palabras en un corpus

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figura 5.17: Tabla de las probabilidades de bigrams para ocho palabras en un corpus

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figura 5.18: Tabla de las probabilidades, suavizadas con add-1, de bigrams para ocho palabras en un corpus

Continuamos haciendo los ejercicios del capítulo para afianzar el conocimiento acerca de los n-gramas y, una vez terminados, escribimos un primer programa para calcular unigramas y bigramas.

En el que leemos un archivo de texto y guardamos en un diccionario un par con clave la palabra y el valor la frecuencia de aparición de esa palabra. Además, añadimos las etiquetas de inicio (`<s>`) y final de frase (`</s>`). Después, extraemos los n-gramas y calculamos una matriz de probabilidades a la que accedemos usando índices que se corresponde con cada palabra del bigrama. Rellenamos un diccionario "dictngrams" con clave la unión de cada palabra que forma el bigrama y valor la probabilidad calculada previamente y guardada en la matriz "matrixProb". "matrixProb" funcionaría igual que la foto de la tabla justo arriba, la columna es una palabra del bigrama y la fila es la otra palabra que lo constituye y el valor la probabilidad para ese bigrama.

```
dictngrams[(str(listDictKeys[i]) + " " + str(listDictKeys[j]))] = matrixProb[i][j]
```

En un primer lugar, sacamos las probabilidades sin suavizar, es decir, pueden aparecer bigrams en la matrix con probabilidad 0. Más tarde, añadimos la posibilidad de usar el método de suavizado add-k. Con esta primera implementación generamos una serie de textos, que se mostrarán más adelante en el apartado de resultados 5.6.1.

Más tarde, adaptamos esta implementación a la generación de mapas del videojuego Super Mario Bros. Para ello, los datos de entrenamiento se cambiaron siendo ahora utilizados archivos csv. Estos son creados mediante Tiled y exportados en formato csv.

En nuestro caso respecto a la generación de los niveles no tenemos el problema de la aparición de palabras desconocidas o unknown words, ya que entrenamos el modelo con los diferentes mapas originales del videojuego y, posteriormente, se genera el mapa escogiendo una primera palabra y a partir de ahí seleccionando palabras de los datos entrenados.

Este caso es conocido como vocabulario cerrado o closed vocabulary, el modelo sólo trabaja con las palabras utilizadas en los entrenamientos.

Por otro lado, podríamos tener un vocabulario abierto u open vocabulary en el que pueden aparecer palabras desconocidas y que todas serán tratadas como pseudopalabras, con la etiqueta `<UNK>`.

No usamos la perplexidad para evaluar el modelo, pues esto se puede hacer estudiando los mapas generados. La primera modificación que hicimos fue usar la librería NLTK (Natural Language Toolkit) que nos permite crear las secuencias de n-gramas y nos evita crearlas a mano.

En lugar de calcular todas las probabilidades a mano en una matriz, generamos un diccionario donde la clave es la secuencia de palabras de tamaño n y el valor es una lista de las siguientes posibles palabras, de tal forma que la probabilidad está implícita en la lista, es decir si las siguientes posibles palabras son [azul, rojo, azul,

azul, verde, rojo] al hacer un random sobre esa lista de palabras, directamente se elige la palabra resultante en base a las probabilidades implícitas según el numero de veces que aparece una palabra en la lista.

Gracias a este cambio pudimos generar de una manera más fácil y eficiente cualquier n-grama, no únicamente bigramas, pues no dependemos de la matriz de probabilidades que para cuatrigramas sería gigante, por ejemplo.

Con el objetivo de tratar los niveles del videojuego como texto decidimos tratar los bloques o tiles como letras, los slices verticales como palabras (son un conjunto de tiles, es decir, letras) y el nivel entero será el conjunto de palabras (frases o texto). Escogemos la primera secuencia a partir de la cual generar los mapas y de este modo no usamos las etiquetas $\langle s \rangle$.

A la hora de la generación de los mapas no empleamos métodos de suavizado, puesto que al utilizar un vocabulario cerrado no nos vamos a encontrar con probabilidades de cero. Y al añadir uno al conteo de n-gramas las probabilidades van a seguir siendo iguales en el caso del método de Laplace (conocido previamente también como add-1).

5.5.2. Redes neuronales recursivas

Introducción

Las redes neuronales recursivas son un tipo de red neuronal cuya aplicación principal es el análisis de secuencias. Este tipo de redes pueden ser consideradas muy útiles ya que nos dejan tener un input y un output con longitudes de secuencia variables. Son modelos muy populares y utilizados en numerosas tareas relacionadas con el procesamiento de lenguaje natural.

La principal idea detrás de las RNN es el uso de información secuencial. En una red neuronal tradicional, se asume que cada uno de los inputs y outputs es independiente del resto. Por ejemplo, si queremos predecir la siguiente palabra de una secuencia, lo comentado anteriormente sería una mala idea, ya que nosotros lo que queremos es que cada output dependa del resto, no de sí mismo. Por tanto, podríamos considerar que una red neuronal recurrente tiene “memoria”, capturando así información sobre lo que se ha calculado anteriormente. Con todo esto, podemos predecir que una red neuronal es recurrente porque para cada elemento de una secuencia, realiza la misma acción.

Las RNN pueden aparecer de diferentes formas como pueden ser:

Completamente recurrentes:

Los nodos están organizados en diferentes capas. Cada nodo de una capa dada, es conectado directamente con otro nodo de la siguiente capa. Además cada nodo guarda un factor de activación y un peso dentro de la red. Un nodo puede ser de input(reciben datos de fuera de la red), outputs (generan resultados), o hidden nodes

(que van modificando los resultados desde el input hasta el output).

Redes recurrentes simples:

Son redes de únicamente tres capas, con la suma de un conjunto de unidades de contexto. La capa de enmedio, está conectada con estas unidades a través de sus respectivos pesos. En cada actualización, el input se transfiere hacia delante, y se crea una regla de aprendizaje.

Redes Hopfield:

Las redes Hopfield son un tipo de RNN en las que todas sus conexiones son simétricas. Requiere inputs cuya probabilidad no cambie a lo largo del tiempo. No puede ser considerada una red neuronal recursiva ya que no procesa patrones de secuencias.

Redes Bi-direccionales:

Las RNN bidireccionales utilizan una secuencia finita para predecir o etiquetar cada elemento de la secuencia, basándose en los elementos pasados y futuros. Esto se consigue concatenando el input de dos RNN, en donde una se encarga de procesar la secuencia de izquierda a derecha, y la otra, de derecha a izquierda. Esta técnica es especialmente útil cuando se combinan LSTM y RNN.

Entre otras muchas como redes jerárquicas, de segundo orden, independientes o multicapa.

Historia

Las redes neuronales recursivas se basaron en el trabajo llevado a cabo por David Rumelhart en 1986. Las redes Hopfield, comentadas anteriormente fueron descubiertas por John Hopfield en 1982. En 1993, un sistema de compresor de historia neural, fue capaz de resolver una tarea que requería de un gran aprendizaje, formada por más de 1000 capas en una RNN desdoblada durante “ejecución”.

En 1997, Hochreiter y Schmidhuber inventaron las redes LSTM (Long short-term memory) y establecieron récords de precisión, en aplicaciones de múltiples ámbitos. Alrededor de 2007, las redes LSTM empezaron a revolucionar el reconocimiento de voz, superando a modelos tradicionales en ciertas aplicaciones de reconocimiento de voz.

En torno a 2009, una CTC-trained LSTM (Connectionist Temporal Classification) fue la primera red neuronal recursiva en ganar concursos de reconocimiento de patrones, en pruebas como el reconocimiento de escritura. En 2014, el gigante Chino, Baidu, un navegador, utilizó CTC-trained RNN para romper el punto de referencia de reconocimiento de voz del Switchboard Hub5'00 sin usar ningún método tradicional de procesamiento de voz.

Las redes LSTM también mejoraron para el reconocimiento de grandes vocabularios así como la síntesis de texto a voz, siendo así utilizada por Android. En el año 2015, el método de Google para el reconocimiento de voz, experimentó una caída de rendimiento del 49 % a través de las redes CTC-trained LSTM.

Finalmente, las redes LSTM rompieron récords, una vez mejoraron para la traducción de texto, el modelado de lenguaje, y el procesamiento de lenguaje multilingüístico. Las redes LSTM combinadas con redes neuronales convolucionales, permitieron mejorar la captura automática de imágenes.

Uso de las RNN

Generar texto con redes neuronales recurrentes es quizá la forma más directa de aplicar las redes neuronales recursivas.

Desde el punto de vista del negocio, la generación de texto se utiliza como un método para agilizar el proceso del trabajo, y minimizar la rutina.

La generación de lenguaje natural se apoya en algoritmos de predicción de redes neuronales recurrentes. Dado que el lenguaje está organizado secuencialmente, es relativamente fácil entrenar un modelo para la generación de documentos de texto genérico.

Las formas más comunes son:

Resumen de textos: El proceso consiste en condensar el texto original. El resumen se utiliza en la gestión de proyectos para incorporar rápidamente a los trabajadores al flujo de trabajo. También se puede utilizar para resumir las noticias y agilizar la generación de artículos de noticias.

Generación de documentos: Usado en banca y seguros para crear cuestionarios adaptados a las necesidades de un cliente concreto.

Generación de reportes: En este caso, la generación de texto se utiliza como una forma para la visualización de datos.

Otra forma habitual son las interfaces de conversación y los chatbots.

Las redes neuronales recurrentes se pueden aplicar también en la traducción automática debido a su capacidad para determinar el contexto del mensaje, o en la localización de contenido. Desde un punto de vista técnico, la traducción automática no es más que una simple sustitución de palabras que representan ciertos conceptos, con sus equivalentes en otro idioma. Hoy en día la aplicación más utilizada es Google Translate. También podemos encontrar numerosas aplicaciones de las RNN en la localización de contenido. Por ejemplo en el comercio electrónico, Amazon, AliExpress, etc, usan la traducción automática para adaptar el contenido, como las fichas de producto, y mejorar la eficiencia de los resultados de la búsqueda.

Por otro lado, el reconocimiento de imágenes, es una de las formas más accesibles para explicar las RNN. El algoritmo está diseñado para reconocer una unidad de input, en este caso la imagen, en múltiples grupos para el output (la descripción de la imagen). Para ello se pueden utilizar redes neuronales convolucionales, que procesan la imagen y reconocen las características de la misma. Las industrias más beneficiadas por el reconocimiento de imágenes son el comercio electrónico (recomendación de productos y personalización del producto), los motores de búsqueda (buscar imágenes similares, a partir de una dada) o las redes sociales (para el reconocimiento de caras en las publicaciones).

Algunos asistentes virtuales como Alexa o Siri, se están volviendo algo habitual en nuestro día a día, ayudando a los usuarios en sus hábitos cotidianos, a partir de frases previamente formuladas. La tecnología que permite esto, es el reconocimiento de voz con las redes neuronales recurrentes. Desde el punto de vista técnico, tanto el reconocimiento de voz, como el de imágenes tienen mucho en común. Sin embargo, para poder reconocer voz, se utiliza una capa de rendimiento extra. El reconocimiento de voz, se utiliza sobre todo en interfaces conversacionales (atención al cliente), chatbots, o aplicaciones que convierten audio en texto (micrófono de búsqueda de Google).

Por último, las redes neuronales recurrentes, pueden ser una herramienta muy útil frente a la detección de fraudes, spam o bots. Sobre todo, la prevención de fraude se apoya en algoritmos predictivos que se encargan de exponer las actividades

ilegales. En el caso de los fraudes publicitarios, las RNN se usan para determinar patrones sospechosos o anormales. También pueden ser utilizadas para la detección de spam, aplicando NLP (Natural Language Processing), para exponer patrones y seguidamente bloquear el mensaje.

Finalmente, aparte de todo lo comentado, las RNN también pueden ser utilizadas para análisis predictivos (predecir la caída de la bolsa) o para el análisis del feedback de los clientes desde el punto de vista de los negocios.

Modelo del lenguaje neuronal

Las redes neuronales están construidas con unidades neuronales, inspiradas en las neuronas humanas. Cada unidad neuronal multiplica los valores introducidos por un vector de pesos, añade el término bias y aplica una función de activación no lineal como las funciones sigmoides, tanh o ReLU (lineal rectificada). La ecuación para el resultado de una unidad neuronal sería:

$$z = w \cdot x + b \quad (5.5)$$

Ecuación de la salida de una unidad neuronal

Donde x es el vector de entrada, w el vector de pesos por los que se multiplica la entrada y b el término escalar bias. Por lo que z será un número. Sin embargo, las unidades neuronales aplican una función no lineal a z , llamada función de activación. Como se ha mencionado arriba hay varias funciones de no lineales de activación (sigmoides, tanh y RELU), las cuales son de las más usadas. La función sigmoide no es tan usada como la función tanh o ReLU. Cada una presenta diferentes propiedades por lo que cada una es más útil en diferentes aplicaciones del lenguaje.

En una red totalmente conectada de tipo feedforward, es decir, aquellas redes neuronales en las que el output se usa como input en las siguientes capas, cada unidad en la capa i está conectada con cada unidad en la capa $i+1$ y, además no existen ciclos. Se usa una matriz de pesos para cada capa oculta para hacer los cálculos de cada capa más eficientes. Se emplea una función para normalizar los outputs de vectores de números en vectores de probabilidades. Esta función es llamada softmax:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^d e^{z_j}} \quad 1 \leq i \leq d \quad (5.6)$$

Ecuación de la salida de una unidad neuronal

La utilidad de las redes neuronales es obtenida gracias a que las primeras capas aprenden representaciones que pueden ser utilizadas por capas más profundas en la

red. El objetivo del entrenamiento es aprender los parámetros W y b , es decir, los pesos y el término bias para cada capa para que el resultado de la red sea lo más aproximado al valor esperado.

Para lograr este objetivo, se usa la función de pérdida que mide la diferencia entre el resultado esperado y el obtenido.

Las redes neuronales se entranan mediante algoritmos de optimización como el gradiente descendente que se usa para minimizar la función de pérdida. El método de la propagación del error hacia atrás es usado para calcular los gradientes de la función de pérdida de una red neuronal.

Los modelos del lenguaje neuronales usan redes neuronales como clasificadores probabilísticos para calcular la probabilidad de la siguiente palabra dadas n palabras previas.

Estos modelos del lenguaje basados en redes neuronales tienen ciertas ventajas sobre los de modelos del lenguaje de n-gramas como, por ejemplo, no necesitan realizar suavizado (smoothing) porque pueden generalizar en contextos con palabras similares. Sin embargo, su mayor desventaja es que son mucho más lentas entrenaendo.

Los modelos del lenguaje neuronales pueden usar incrustaciones o embeddings ya entrenados o pueden aprenderlos desde cero en el proceso de modelado del lenguaje. En estos modelos neuronales la historia se representa por embeddings de las palabras anteriores. Los embeddings hacen referencia a la vinculación de palabras o frases de un vocabulario en vectores de números reales. Esta representación de la historia en lugar del uso de las palabras exactas permite generalizar mejor los datos no vistos.

5.5.3. Redes neuronales convolucionales

...

5.6. Resultados

...

5.6.1. N-gramas

En una primera implementación se decidió usar los slices como si fueran frase, es decir cada fila vertical de tiles era una frase y cada tile una palabra.

Los n-gramas se formaban por tiles de los que al final se componían los slices, como los bloques pueden estar seguidos prácticamente de cualquier otro se forman slices incoherentes. Este problema es equivalente a si cogiésemos de un texto, por

ejemplo, bigramas formados por letras. Al final, contaríamos como muchísimos bigramas compuestos por casi todas las combinaciones de letras posibles y sería extremadamente difícil formar nuevas palabras, que de verdad existan, con esos bigramas. Sin embargo, de esta forma se dieron malos resultados, ya que cualquier n-grama de tiles podía estar seguido de otro, en caso de una n pequeña. Carecía de coherencia y consistencia.

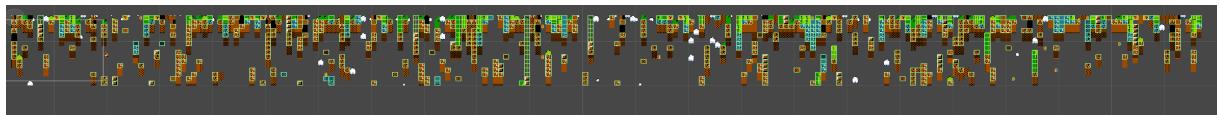


Figura 5.19: Cada tile (bloque) considerado como una palabra

Por lo que decidimos usar cada slice como una palabra y cada tile como una letra.

Luego, nos surgió un problema por el cual muchos resultados no tenían la longitud esperada. Esto era debido a que se llegaba antes de tiempo a una secuencia final y el modelo ya no podía continuar.

Para solucionar este problema, decidimos llenar con ceros (tile del suelo) hasta completar la longitud correspondiente, pero no fue la mejor solución. Queda realmente feo ver el mapa con los mismos tiles hasta el final.

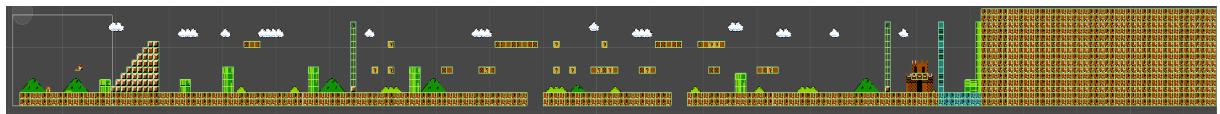


Figura 5.20

La siguiente posible solución a este problema fue resetear la secuencia actual por la secuencia con la que se inicio la generación del texto. Aunque, seguimos teniendo problemas en los resultados de los niveles debido a errores en el código. Los niveles continuaban con las mismas slices.

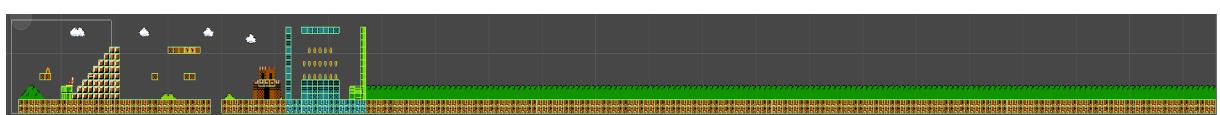


Figura 5.21

Definitivamente arreglamos el problema y al llegar a la secuencia final antes de tiempo se reiniciaba con la primera secuencia.

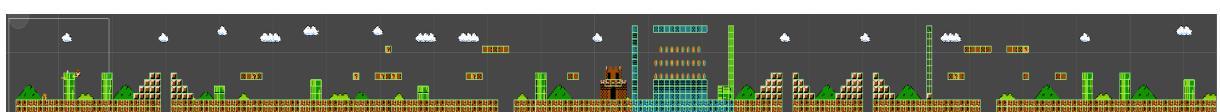


Figura 5.22

El siguiente paso fue permitir la generación de niveles multi temas. Usamos el término multi tema para referirnos a las múltiples opciones temáticas que ofrecen los niveles del Super Mario Bros, pues existen niveles submarinos, zonas secretas, cielo, terreno normal...

Para conseguir este objetivo, se aumentó el tamaño de los slices. De 15 bloques de altura se pasó a 60 bloques (15 para el cielo, 15 para el terreno normal, 15 submarino y 15 subterráneo, en este orden de arriba abajo). En tiled creamos los mapas de entrenamiento con la nueva altura 60 y se llenan los 15 tiles de altura correspondientes a cada temática.

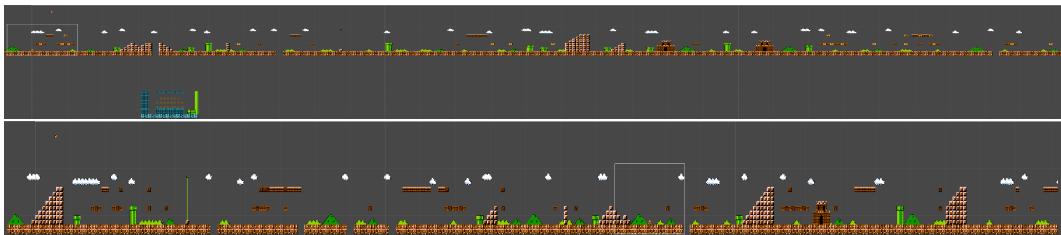


Figura 5.23: Niveles generados por bigramas con corpus de entrenamiento el nivel 1-1

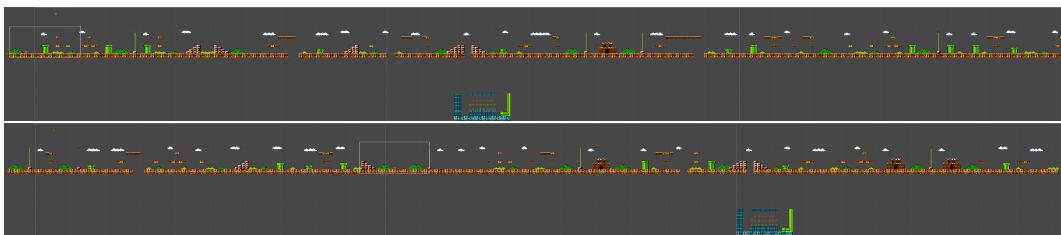


Figura 5.24: Niveles generados por trigramas con corpus de entrenamiento el nivel 1-1

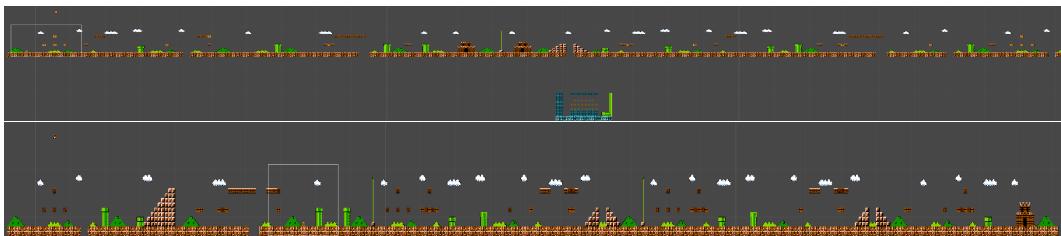


Figura 5.25: Niveles generados por cuatrigramas con corpus de entrenamiento el nivel 1-1

Antes de esto, teníamos el problema que las zonas secretas (tiles azules y tubería) quedaban seguidas, como se puede observar en las imágenes de arriba.

Damos la opción de entrenar el modelo con múltiples corpus, es decir, niveles. Leemos cada nivel y vamos ampliando el diccionario que contiene los n-gramas con los n-gramas del nuevo nivel leído. El mapa para generar comienza con la primera

secuencia del primer mapa usado como entrenamiento, por lo que el mapa resultante es muy posible que se asemeje más al primero y, más si no comparten slices comunes. Esto se muestra en las siguientes imágenes:



Figura 5.26: Nive 1-1



Figura 5.27: Nivel 1-4



Figura 5.28: Nivel generado por bigramas con corpus de entrenamiento el nivel 1-1 y 1-4



Figura 5.29: Nivel generado por bigramas con corpus de entrenamiento el nivel 1-4 y 1-1



Figura 5.30: Niveles generados por trigramas con corpus de entrenamiento el nivel 1-1 y multiples niveles generados con n-gramas a partir del nivel 1-1

FALTA PONER MULTI TEMAS Y EJEMPLO CON VARIOS CORPUS DEL NIVEL 1-1 Y VARIOS NIVELES GENERADOS A PARTIR DEL 1-1

IMG

IMG

...

Una ventaja observable de los n-gramas es que no descarta palabras o slices poco frecuentes, como se hace en otros modelos de clasificación. Esto ofrece unos resultados con una cierta mayor variedad y no ajusta los niveles únicamente aquellas slices más repetidas.

Como se ha podido observar en los resultados, con el empleo de trigramas se producen mejores niveles y más similares a los del corpus de entrenamiento que usando

bigramas y, bastante mejores que los resultados de los unigramas.

En general los resultados son buenos y podrían usarse para el juego. Tanto como para tener un mundo más amplio y añadir, de por sí, más horas de jugabilidad a Super Mario Bros, como una mecánica del propio juego mediante la cual al morir el jugador se reinicia el nivel, pero generando otro distinto en base al original. Esto permite que el jugador no vaya aprendiendo de memoria con la práctica los sitios o situaciones en las que muere y aumentar la dificultad para aquellos jugadores más experimentados.

También observamos que los niveles generados en base a un solo nivel son bastante parecidos y mantienen la estética.

IMG
IMG

Puede ser extendido a muchos otros juegos con cierta facilidad, otorgando una gran utilidad a este modelo como herramienta para crear niveles en videojuegos.

5.6.2. Redes neuronales recursivas

...

5.6.3. Redes neuronales convolucionales

...

5.6.4. Comparativa

...

Capítulo 6

Conclusiones

6.1. Conclusiones

Hemos estudiado el aprendizaje automático y muchos de sus tipos, algoritmos, posibilidades o aplicaciones. Hemos visto aprendizaje supervisado, no supervisado y por refuerzo. Dentro de estos distintos tipos de algoritmos, algunos mas comunes y prácticos y otros menos. También hemos visto la generación procedural de contenido. Su clasificación desde distintos puntos de vista y según los algoritmos a utilizar. Además, a lo largo de esto hemos ido viendo las diferencias entre algunos conceptos que a priori pueden parecer sinónimos, pero que no lo son como podría ser el caso de aleatorio y procedural. Una vez visto de forma genérica muchos de los caminos que existen planteamos el estudio de aplicar la generación procedural de contenido en el mundo de los videojuegos, un campo que lo explota bastante bien.

Además, decidimos comparar dos de sus vertientes, una de ellas orientada a modelos probabilísticos basados en probabilidades, estadísticas y valores pseudoaleatorios. Por otro lado, un modelo mas complejo usando el aprendizaje automático.

Como hemos visto los modelos elegidos han sido ngrams y redes neuronales recursivas. Sin embargo, ambos recibían la misma forma de input, pero este es tratado de distinta forma antes de su procesamiento. En ambos casos los mapas eran tratados como palabras y la función de ambos algoritmos era predecir la siguiente es decir generar texto o en nuestro caso el nivel.

Como hemos visto ngrams es utilizado en labores referidas al texto, sirve tanto para su predicción como para la detección de plagio. Es un modelo bastante rápido, una vez tienes la información del mapa, saca las posibles secuencias, dependiendo del N que quieras de un tamaño u otro, y opera con ellas rápidamente. A partir de estas secuencias de palabras anteriores saca las distintas posibles palabras siguientes. Otra de sus grandes ventajas es que apenas necesita información de entrada. Como hemos podido observar con mapas de entre 150 a 250 slices de longitud los resultados son buenos. Además, el tamaño de las secuencias permite acercarse mas al modelo de ejemplo o alejarse de el hasta obtener resultados malos. Esto permite acercarse a mapas muy aleatorios en los que no puedes esperarte nada ya que son muy cambiantes, o, por el contrario, regirte mas por las normas del mapa modelo diseñado por una persona. Ambos manteniendo la coherencia del nivel.

Por otro lado, las redes neuronales recursivas. Existen distintos tipos, las simples, las LSTM y las GRU, cada una de ellas con sus propiedades. Es un modelo lento, requiere de un proceso de aprendizaje que requiere un esfuerzo y un tiempo. No es realizar una serie de operaciones sobre unos números para saca una secuencia u otra. Requiere de saber que tipo de red es mas idónea para cada problema. Sin embargo, ambas fueron creadas con el fin de evitar problemas de memoria gracias a su estructura. El problema de entrenamiento no queda solo en el tipo y en su posible intercalación en una red neuronal. Existen una gran cantidad de parámetros para configurar una red neuronal. Algunos de estos pueden ser tanto el número como el orden de sus capas, el número de neuronas de la capa de entrada, de las capas ocultas o de la capa de salida. Distintos algoritmos optimizados. Además de algunos parámetros muy parecidos como la longitud de las secuencias a tratar. Incluso parámetros más complejos como pueden ser la representación de las palabras a la hora de procesarlas, estas transformaciones que reciben el input se llama vectorización. En cuanto al tema de información necesaria, necesitan una gran cantidad de información. Hemos observado que, en ejemplos para la predicción de texto, a la hora de entrenar cogían secuencias muy largas y conjuntos con muchas de estas secuencias de estas. Como hemos dicho requiere un esfuerzo, pero además requiere de su tiempo de entrenamiento, obviamente no es lo mismo dos épocas de entrenamiento que cien. Sin embargo, una vez escogidos buenos parámetros y entrenado el tiempo necesario, las redes neuronales ofrecen muy buenos resultados.

Comparando con ngrams, se puede apreciar como ngrams aumentando mucho la N, por ejemplo, cuatrígram, los resultados son patrones de nivel original copiados y pegados con pequeñas uniones entre ellos, que pueden ser otros patrones. Es decir, una copia del nivel original, pero en distinto orden. Y si nos vamos al otro extremo valores de N bajos como unigram, se aprecian resultados muy típicos de escoger una dentro de muchas opciones.

Sin embargo, en los resultados de redes, se ve como se mantiene la idea del mapa sin llegar a notarse una gran cantidad de fragmentos copiados. Se podría decir que la red se acerca a seguir el mismo proceso creativo que siguió el diseñador en su momento.

Esto no quiere decir que los resultados de uno sean mejores o peores que los del otro. Simplemente cada uno cumple unas características y ofrece unas cosas a cambio de otras.

6.2. Posibilidades

Las posibilidades de la generación procedural de contenido en el mundo de los videojuegos es algo bastante relevante que las grandes empresas se han dado cuenta. Y es que cada vez es mas usado para tareas menos relevantes a la hora de crear grandes títulos. Las compañías se han dado cuenta de que con un buen modelo pueden ahorrarse mucho tiempo, esfuerzo y dinero obteniendo resultados los suficientemente buenos como para que no nos demos cuenta de que eso no lo ha diseñado una persona.

Tras haberlo estudiado bastante mas a fondo podemos ver como esta práctica va a ir cada vez a mas y mas, sobretodo gracias a su mezcal con el aprendizaje automático.

Algo sorprendente es que el aprendizaje automático se utilice en todos los sectores del mundo, desde ayudar en granjas a producir óptimamente hasta conducir vehículos. Sin embargo, la generación procedural apenas tiene el público de los videojuegos y algún pequeño impacto en el mundo del cine y de la música. Se podría decir que es explotada por los sectores más artísticos de la sociedad.

6.3. Limitaciones

Las limitaciones de la generación procedural vienen limitadas por el tipo de algoritmo utilizado. No es lo mismo basarse en un modelo que genera valores pseudoaleatorios cuyas limitaciones son muchas más que las de un modelo basado en algoritmos de aprendizaje automático. Como hemos visto el aprendizaje automático se utiliza en una gran cantidad de aplicaciones distintas, lo que se traduce en que sus limitaciones están bastante lejos. No obstante, también las tiene, no porque sean difíciles de alcanzar signifique que no las tenga. Y es que uno de los mayores problemas del aprendizaje automático es una de sus ventajas, la información y los datos. Hay una gran cantidad de estudios que demuestran que todavía se puede engañar con relativa facilidad a las máquinas.

En uno de estos experimentos trataron con una de las mejores redes neuronales, VGG-19. Mostraron a esta red imágenes en color de animales y objetos modificadas. Por ejemplo, una mostraba una tetera, pero con textura de pelota de golf, otra, por ejemplo, las típicas rayas de las cebras, pero en un camello. VGG-19 sólo pudo hallar 5 de 40 objetos en la primera ronda. Básicamente sobreentrenaban la red con unos datos sin enseñarle posibles modificaciones, aunque no reales, de esos datos. De esta forma algo que el ser humano distinguiría rápidamente, la máquina no pudo.

Otro experimento fue una inteligencia artificial basada en datos de pacientes de un hospital. Esta decidía que pacientes podían entrar al hospital para ser atendidos y cuáles no dependiendo de la enfermedad y distintas características como edad, peso, etc. El algoritmo evaluaba muy bien los casos que oscilaban entre las edades más comunes de la población. Sin embargo, en edades como por ejemplo los 100 años no te debían atender en el hospital tuvieses lo que tuvieses, aunque fuese un simple lo que sea. Esto era debido a la poca información acerca de ese tipo de casos.

6.4. Ampliaciones

Se pueden probar los algoritmos desarrollados en muchos juegos y ver los resultados. Nosotros los hemos aplicado al juego Super Mario Bros, no obstante, estaría bien estudiar sus posibles aplicaciones en otros juegos del mismo estilo. Juegos de plataformas 2D como pueden ser el Donkey Kong, el Castlevania o el Metroid. Incluso el algoritmo de ngrams puede ser aplicable a juegos con vista top down en 2D, solamente sería encontrar la división óptima para ese tipo de juegos. Un ejemplo de juegos top down pueden ser los Zelda o los escenarios de los Pokémon. En cuanto a las redes neuronales habría que probar y ver los resultados.

Otra posible paliación sería el uso de redes neuronales convolucionales para la crea-

ción de niveles. Utilizar imágenes de los mapas generados para crear nuevos mapas. O para mezclar los ya existentes.

Otra posible ampliación sería la implementación de redes neuronales convolucionales para el análisis de la situación dentro del juego y desarrollar un algoritmo para que él juegue al juego. Para ello la red neuronal analizaría la situación dentro del juego, analizando el mapa y posibles enemigos y gracias a un proceso de aprendizaje automático aprender como moverse dentro del juego. Estas mismas redes convolucionales podrían servir para estudiar los mapas generados y si existen diferencias por ejemplo en el numero de bloques normales generados por un modelo como ngrams y un modelo de red neuronal. Si existe algún patrón que sigan una red neuronal LSTM.

A la hora de entrenar la IA que juega los niveles de Mario probar distintos tipos de aprendizajes por refuerzo. Por ejemplo, uno de los mas famosos es el proceso de decisión de Márkov, del que hablamos un poco en el capítulo 3.

Las aplicaciones de este proyecto son muchas y todas ellas pueden intentar aplicarse a otros videojuegos de las mismas características para ver su reacción. Esto es gracias a la gran cantidad que presenta el aprendizaje automático y la facilidad que tiene para mezclarse con cualquier tipo de tecnología.

Capítulo 7

Contribuciones

7.1. Víctor Emiliano Fernández Rubio

...

7.2. Gonzalo Guzmán del Rio

Como primera tarea, al empezar el desarrollo e investigación del TFG, y con el objetivo final ya fijado, fue empezar a investigar sobre qué técnicas de aprendizaje automático podrán ser más utiles o más cómodas"para la generación de mapas. Para ello empecé leyendo diferentes tesis y ensayos de la universidad de Stanford que trataban el tema de la propia generación de mapas con Ngrams interpretando los datos de una manera concreta (por slices, como hemos hecho finalmente) y sobre el procesado de secuencias con redes neuronales recursivas. Una vez que cada uno de nosotros terminamos de leernos estos ensayos, nos encargamos de realizar de manera individual una serie de ejercicios que venían adjuntados para así poder comprender de manera más concreta todo lo que habíamos leído anteriormente. Tras ello, un primer acercamiento, fue el de realizar un pequeño prototipo para la generación de texto a través de Ngrams.

Durante este proceso, de manera paralela, fuimos realizando individualmente diversas tareas dentro de la aplicación de Unity, como podían ser el movimiento básico de Mario, así como el salto. En mi caso, además de lo anterior, yo me encargué de realizar el seguimiento de la cámara estilo Super Mario. Esta cámara solo sigue hacia delante con cierto offset(distancia), mientras que para atrás no puede seguir a Mario. Para complementar a la cámara, también me encargué de realizar un modo vista. Este modo vista, al cuál se accede con la tecla L, permite ver el mapa generado o cargado de manera libre, sin tener que mover al jugador, moviéndote por él con las teclas de dirección y haciendo zoom con la rueda del ratón.

Tras ello, entre los tres nos encargamos de seguir intentando implementar de manera correcta el algoritmo de NGrams. Tras tener una versión previa del algoritmo, Victor y Carlos siguieron trabajando en ello. Yo me encargué de realizar entonces de manera más concreta todo lo relacionado con la aplicación de Unity. En un primer momento, cree un menú inicial que permitiría elegir entre todos los mapas disponi-

bles para cargarlos individualmente o generarlos (pero sin añadir la funcionalidad). Una vez tenía todo esto funcionando me encargué de, con ayuda de Carlos y Victor, pues cada uno buscamos diferentes maneras de conectar Unity con python, de la ejecución del proceso para lanzar el script de Python. Para lanzar este proceso se crearon diversos InputFields, cuyos valores eran recogidos para luego ser plasmados en el comando requerido por el script de NGrams. Este menu permitiría la generación de un único mapa (monotemático). Además de todo ello, una vez Victor había terminado de implementar el multitema en NGrams, con la ayuda de Carlos me encargué de hacer la entrada y salida de las zonas subterráneas. Tras ello, y tras la implementación de Victor, me encargué de permitir que Unity cambiara los parámetros para poder introducir las nuevas variables requeridas por los procesos. Tras ello, los 3 empezamos a leer acerca de tensorflow y de redes neuronales recursivas. Una vez tuvimos una primera versión me encargué de simular todo lo relacionado con Ngrams en la aplicación de Unity, pero esta vez con redes neuronales recursivas. Con un primer script de RNN, cada uno, nos encargamos individualmente, de generar múltiples mapas con diferentes parámetros para tener una primera impresión de RNN. Con todo ello realizado, continue con la aplicación de Unity, implementando nuevas mecánicas como la de nadar, o mejorando ciertos aspectos que no estaban bien realizados como la entrada a tuberías. También me encargué de añadir comportamiento a los tiles del mapa que lo requerían. Explicándolo a grossó modo, me encargué de seguir mejorando todo lo relacionado con Unity, haciendo la aplicación más interactiva y visual, y arreglando jugabilidad y más mecánicas que eran incorrectas. Aparte, me encargué de hacer una nueva escena con todo lo necesario para poder ejecutar la aplicación de manera correcta con imágenes descriptivas.

Además de todo lo comentado anteriormente, me he estado documentando, aparte de lo comentado más arriba, acerca de otros tipos de redes neuronales como las convolucionales, o y de ciertos detalles relacionados con NGrams como el smoothing o interpolación.

7.3. Carlos Llames Arribas

Una vez decidimos el tema del proyecto, empecé buscando bibliografía e información disponible relacionada con la generación automática de niveles/mapas en videojuegos.

En una de las primeras reuniones con nuestro director Samir, nos indicó un artículo respecto a la generación procedural de contenido a través del machine Learning (PCGML), mediante el cual estudiamos los distintos métodos y soluciones para generar contenido de videojuegos usando modelos entrenados mediante contenido existente. También estudiamos los problemas más comunes en esta clase de procesos de generación, tales como la falta en muchas ocasiones de datos de entrenamiento, corpus de entrenamiento pequeños, ajuste de parámetros...

Inmediatamente, después acordamos emplear el modelo n-grama como primer método a desarrollar para la generación de nuestros niveles para el juego. Decidimos utilizar este modelo debido a que es un modelo más o menos simple, pero que

resulta muy útil y eficaz. Para conseguir este objetivo, realicé de manera paralela a mis compañeros una investigación acerca de este método, tanto para aprender en qué consiste, sus posibles usos (que son bastantes y comunes como se ha explicado en el capítulo correspondiente a n-gramas), el empleo y procesamiento que realiza sobre los datos de entrenamiento, etc.

Paralelamente a este proceso de investigación y estudio, fuimos creando la aplicación demo destinada a probar los niveles resultantes generados por los modelos a implementar. Así, como en mi caso, la implementación del movimiento, salto, algunos enemigos (que, aunque no se usan, podríamos ampliar fácilmente la generación de los niveles con los enemigos), la entrada y salida de tuberías. Muchas de estas tareas las realizamos conjuntamente.

Retomando tema del modelo n-grama, hice, como mis compañeros, de manera individual unos ejercicios prácticos sobre la teoría de los n-gramas para afianzar el conocimiento en esta área. Seguidamente, comenzamos a realizar la implementación de este método. Para ello, realizamos implementaciones individuales y, más tarde, pusimos en común. Y tras una versión inicial, Víctor y yo seguimos trabajando en la mejora de la implementación de este modelo. Tanto buscando librerías útiles, como seguir investigando en métodos de mejora y refinamiento para el modelo (como el smoothing). Primero para textos y después adaptado al tipo de datos usado por nosotros para representar los niveles.

Más tarde, me encargué de realizar pruebas y obtener resultados de niveles.

A continuación, cada uno buscó formas de conectar Python con Unity y, ejecutar el script de Python.

Por último, comenzamos la investigación y estudio de las redes neuronales como modelo para la clasificación de texto, aprendizaje y generación. Sobre todo Víctor trabajó en la implementación del modelo, sin embargo, yo seguí investigando sobre la teoría de las redes neuronales, tales como sus tipos, funcionamiento, problemas, etc., así como sobre la API de TensorFlow. Por ejemplo, sobre las distintas funciones de activación para las redes neuronales, sus ventajas y desventajas. De este modo, tener una idea del ajuste de los distintos parámetros para mejorar la generación de los resultados. Para acabar realicé pruebas variando parámetros para ver los distintos resultados.

En resumen, me he encargado sobre todo en la investigación y estudio para tener una base y, en el desarrollo de los modelos. También, he trabajado en la demo de Unity, pero en menor medida.

7.4. Repositorios

...

Bibliografía

*Y así, del mucho leer y del poco dormir, se
le secó el celebro de manera que vino a
perder el juicio.*

Miguel de Cervantes Saavedra