

Aprendiendo a pronunciar texto en Inglés

Jeremías Rodríguez

July 22, 2019

1 INTRODUCCIÓN

La pronunciación del lenguaje inglés ha sido estudiada extensivamente por lingüistas, y mucho es sabido sobre las correspondencias entre letras y los sonidos del habla inglesa, llamados *fonemas*. El inglés es un lenguaje particularmente difícil de dominar por sus irregularidades en la pronunciación. Por ejemplo, la letra "a" en la palabra "gave" es una vocal larga, pero no en "have" o "read".

En este trabajo se busca aplicar varios de los contenidos estudiados en este curso (y el anterior) al problema de convertir texto en inglés a su correspondiente transcripción en fonemas¹.

single vowels				diphthongs			
I	i:	ʊ	u:	eɪ	ɔɪ	aɪ	
ship	sheep	book	shoot	wait	coin	like	
e	ɜ:	ə	ɔ:	eə	ɪə	ʊə	
left	her	teacher	door	hair	here	tourist	
æ	ʌ	ɒ	ɑ:	əʊ	aʊ	/	
hat	up	on	far	show	mouth		
unvoiced consonants							
p	f	θ	t	s	ʃ	tʃ	k
pea	free	thing	tree	see	sheep	cheese	coin
voiced consonants							
b	v	ð	d	z	ʒ	dʒ	g
boat	video	this	dog	zoo	television	joke	go
m	n	ŋ	h	w	l	r	j
mouse	now	thing	hope	we	love	run	you

Figure 1.1: Algunos de los 51 fonemas del inglés oral.

¹El Alfabeto Fonético Internacional es un sistema de notación fonética creado por lingüistas. Su propósito es otorgar, de forma regularizada, precisa y única, la representación de los sonidos de cualquier lenguaje oral. Tiene aproximadamente 107 símbolos básicos, y cada idioma hace uso de un subconjunto particular.

El dataset utilizado se llama netTalk², fue usado por primera vez en el paper "Parallel Networks that Learn to Pronounce English Text"[1]. Los autores entrenaron redes neuronales que convierten strings de texto en inglés (e.g. *calculator*) en strings de fonemas (*kælkjwleIt@*).

En este trabajo recrearé y analizaré varios experimentos llevados a cabo en el citado paper (con ANNs). Posteriormente aplicaré otros clasificadores (Random Forest, Gaussian Process, SVM y KNN) y compararé resultados.

2 DATASET NETTALK

2.1 PRESENTACIÓN DEL DATASET

El dataset netTalk contiene una fila por cada una de las 20000 palabras del *Miriam Webster's Pocket Dictionary*. Por cada fila, el dataset contiene sólo dos columnas:

[written_representation]	[phonemic_representation]
argue	argY-
argumentation	argYmxnteS-xn
argumentative	argYmEntxtIv-
aright	xrA--t
arise	xrAz-
aristocracy	@rxstakrxsi
aristocrat	xrIstxkr@t
aristocratic	xrIstxkr@tIk
arithmetic	xrIT-mxtIk
arithmetical	@rIT-mEtIk-L
arithmetician	xrIT-mxtIS-xn

Ambas columnas son de tipo string, la primera columna sólo contiene caracteres del alfabeto inglés (abc..z), mientras que la segunda columna contiene caracteres que representan fonemas. Los autores del paper original eligieron un conjunto de 51 fonemas. (ver rawDs/netTalk.names para mas detalle).

En el extracto del dataset que he mostrado podemos ver que se ha forzado una correspondencia 1-1 entre letras de la primera columna y fonemas de la segunda: cada fonema corresponde al sonido producido por una de las letras . Por ejemplo, la palabra "arithmetical":

a>@ ; r>r ; i>I ; t>T ; h>- ; m>m ; e>E ; t>t; i>I ; c>k ; a>- ; l>l

Los autores del paper original incluyen un fonema "vacío", representado con el símbolo '-', para indicar la ausencia de sonido. Utilizando este simbolo, se completa las transcripciones foneticas para que tengan la misma cantidad de caracteres que la palabra original (lo cual es computacionalmente conveniente).

2.2 PREPROCESAMIENTO

Todos los métodos que hemos estudiado en Machine Learning y Data Mining requieren que los datos input estén expresados como una matriz numerica. El dataset netTalk que se acaba de introducir es una representación a muy alto nivel de la información que queremos utilizar, y por ello los autores del paper original realizaron una codificación y transformación bastante compleja para obtener datos numéricos con los que trabajar.

²[https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+\(Nettalk+Corpus\)](https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Nettalk+Corpus))

Comencemos por definir claramente el problema a tratar. Deseamos hallar una función f que convierta strings con palabras en inglés, a strings con su correspondiente transcripción fonética:

$$f: Words \rightarrow Phonetics$$

$$f("hypotenuse") = "hApAt - NYs - ".$$

Utilizando el hecho de que cada letra del input se corresponde a un fonema en el output, podemos reducir el problema a hallar el fonema correspondiente de cada letra. Es decir, deseamos obtener una función g que se focalice en predecir el fonema de una sólo de las letras a la vez, obviamente teniendo en cuenta el entorno de ella. Asumimos que para predecir el sonido de una letra, sólo necesitamos saber las 3 letras anteriores, la letra misma y las 3 siguientes (si las hay):

$$g: L \times L \times L \times L \times L \times L \times L \rightarrow Phonemes$$

$$g(-, -, -, \textcolor{red}{h}, y, p, o) = h$$

$$g(-, -, h, \textcolor{red}{y}, p, o, t) = A$$

$$g(-, h, y, \textcolor{red}{p}, o, t, e) = p$$

$$g(h, y, p, \textcolor{red}{o}, t, e, n) = a$$

$$g(y, p, o, \textcolor{red}{t}, e, n, u) = t$$

$$g(p, o, t, \textcolor{red}{e}, n, u, s) = -$$

$$g(o, t, e, \textcolor{red}{n}, u, s, e) = N$$

$$g(t, e, n, \textcolor{red}{u}, s, e, -) = Y$$

$$g(e, n, u, \textcolor{red}{s}, e, -, -) = s$$

Por supuesto, esta ventana de 7 letras puede no ser suficiente para algunas palabras donde se necesita considerar una porción más amplia para generar una correcta pronunciación; pero en general funciona bien. Finalmente, si obtenemos esta función g , podemos derivar f aplicando sucesivamente g a cada letra del input de f .

$$f("hypotenuse") = g(" - - - \textcolor{red}{h}ypo")g(" - - \textcolor{red}{h}ypot")g(" - \textcolor{red}{h}ypote")g("\textcolor{red}{h}ypoten")...$$

Qué ventajas tiene g respecto a f ? Al tener un input fijo de 7 caracteres y un output fijo de 1 fonema; podemos aplicar los algoritmos de machine learning que hemos estudiado (tener inputs/outputs de longitud variable complicaría el problema). Además, aprender la función g es sustancialmente más fácil porque sólo se predice un caracter a la vez.

Por lo tanto, se convirtió el dataset netTalk a un nuevo dataset de 8 columnas, donde dividimos cada palabra en varias filas como se ilustró arriba con "hypotenuse". En nuestro caso, de tener 10000 palabras pasamos a tener 146934 filas. Los scripts utilizados en esta fase pueden verse en /src/encodedDs.

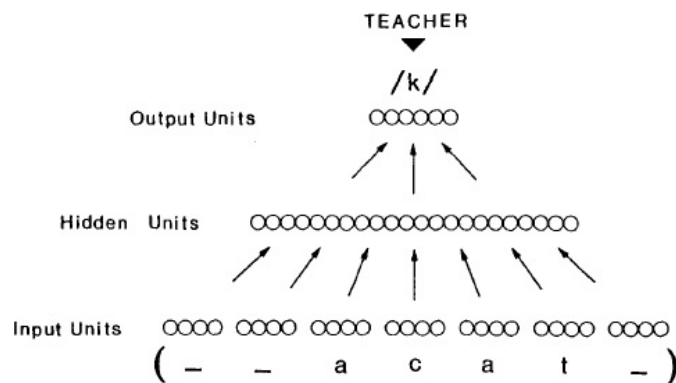
El paso restante es codificar todos los caracteres (letras y fonemas) de las 146934 filas como números. Para codificar las **letras del input** usaremos una representación muy simple: Dado que tenemos 26 letras en el alfabeto inglés, cada letra por 26 columnas binarias 1-0. Si la primera letra era una 'j', entonces la columna 10 tendrá un 1 y las otras 25 columnas un cero. Los **fonemas output**, por otra parte, se codificarán en 20 columnas binarias (ver anexo I).³

Por lo tanto, el dataset final de 146934 filas tendrá $26 * 7$ columnas binarias representando las 7 letras input; y otras 18 columnas binarias representando el fonema output. A este dataset resultante le aplicaré distintas técnicas de aprendizaje que estudiamos en el curso.

³Los 50 fonemas se codifican en 18 columnas que representan "articulatory features" que los definen, como por ejemplo si tienen un tono alto, medio o bajo; si el ruido procede de la garganta, del paladar, etc; o si el ruido es con o sin voz.

3 REDES NEURONALES

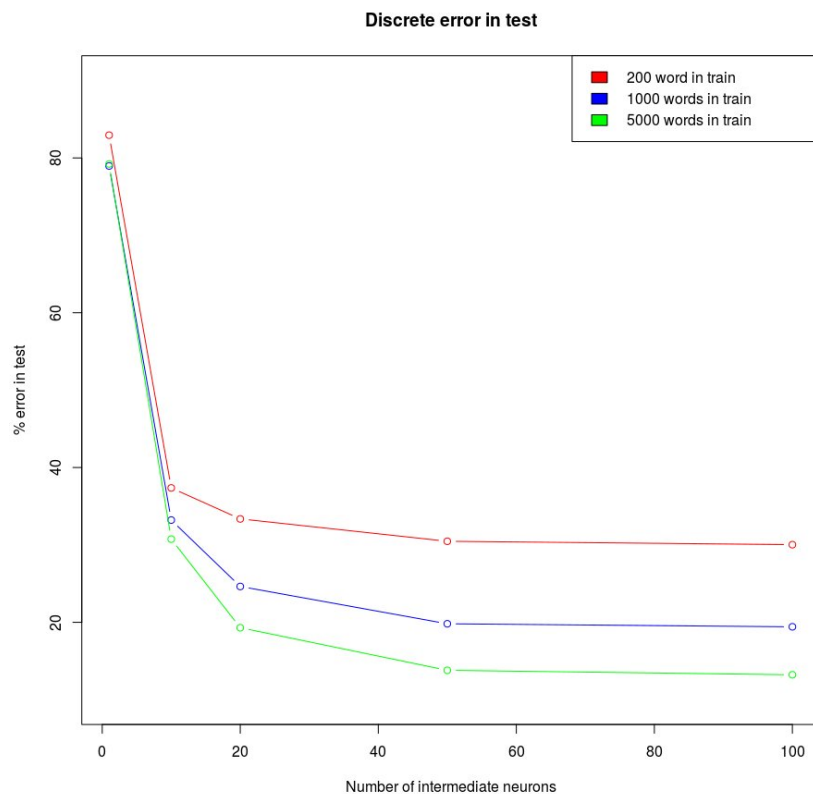
El primer paso es repetir los experimentos que se realizaron usando redes neuronales en [1]. Para ello, utilizaré una red neuronal con $26 * 7$ neuronas en la capa de entrada, y 18 en la capa de salida.



3.1 OPTIMIZACIÓN DE PARÁMETROS

Para optimizar parametros se usaron 1000 de las 20000 palabras del dataset original; divididas en 750 training y 250 validacion. Realicé la búsqueda en grid search para learning rate (η) y momentum (μ) entre 10^{-1} y 10^{-5} (25 combinaciones). Utilicé el algoritmo backpropagation, en particular una version modificada del codigo en C que utilizamos en el curso de machine learning. Los parámetros óptimos fueron $\mu=0.0001$ y $\eta=0.01$ y el código de toda esta sección puede verse en /src/ANNs

3.2 RESUMEN DE RESULTADOS USANDO ANNs



Una vez seleccionados los parámetros óptimos, el siguiente paso fue entrenar redes con distintas cantidades de neuronas intermedias y con distintas cantidades de puntos en training, y los resultados pueden verse en la figura anterior.

Algunas apreciaciones:

- Como vemos, usando una red con 20 o más neuronas podemos llegar a un accuracy realmente alto, donde más del 80% de los fonemas son generados correctamente.
- El mejor error en test se obtiene con 5000 palabras en training y 100 neuronas, donde el 86.78% de los fonemas son generados correctamente.
- A partir de 50 neuronas en adelante, continuar agregando más neuronas no mejora el resultado. Apartentemente no queda más información que pueda ser aprendida por una ANN llegado a ese punto, independientemente de la complejidad de la red. (Para comprobar esto, entrene una red con 150 neuronas intermedias; y el error no mejoro aun entrenando con muchisimas epocas)
- Similarmente, la ganancia de seguir agregando más palabras al dataset de entrenamiento se reduce cada vez más a medida que llegamos a unas 5000 palabras.

El "estancamiento" en la reduccion del error (descrito en los ultimos dos items) probablemente se deba a que:

- Hay irregularidades y excepciones del lenguaje que probablemente no puedan ser aprendidas de otra forma que memorizando los casos puntuales.
- Estamos usando una ventana de 7 caracteres para pronunciar, y probablemente hay palabras cuyos patrones de pronunciacion podrian ser aprendidos considerando una ventana mayor.

Me parece asombroso llegar a predecir con tanta exactitud los fonemas, dado que este problema es muy complejo, altamente dimensional y posee una gran cantidad de reglas y excepciones.

Lo que mas me sorprende de estos resultados es que con sólo 200 palabras de ejemplo del idioma somos capaces de predecir la pronunciación del resto del idioma con casi 80% de precisión (habilidad que a un ser humano puede llevarle realmente mucho tiempo y ejemplos adquirir).

3.3 EJEMPLOS Y CONCLUSIÓN

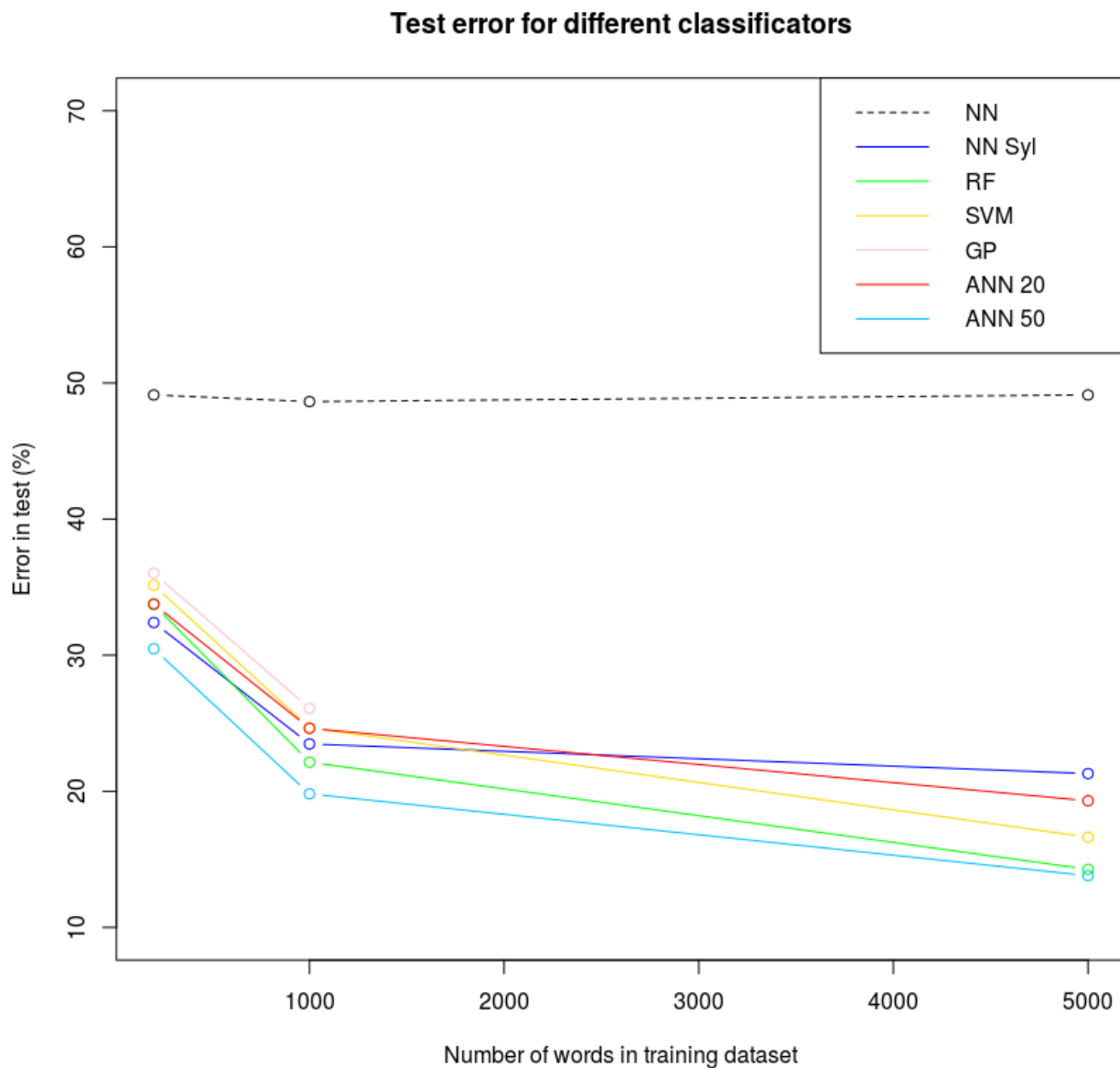
Esta tarea exhibe una gran cantidad de regularidades globales junto con reglas más especializadas y casos excepcionales. Los resultados fueron muy buenos y las redes aprendieron exitosamente a generar una pronunciación cercana a la real. Veamos algunos ejemplos:

Written Representation	predicted Phonetics	real Phonetics
machine	m@C-en-	mxS-in-
learning	lI-rnIG-	l--RnIG-
unexpected	xnEXpEktxd	xnIXpEktxd
beautiful	bcc-tIfcl	bY--tIf^l
telescope	tIlEskxp-	tElxskop-
virus	varxs	vArxs

Como se puede apreciar, la red se confunde en varios fonemas, pero los errores son muy similares a los que cometemos las personas, pues la confusión se da generalmente entre vocales con sonidos similares.

4 OTROS METODOS DE MACHINE LEARNING

Me interesa analizar ahora la performance de otros algoritmos que hemos estudiado en el curso, y compararla con los resultados vistos en la seccion anterior. Los metodos que decidi aplicar a este problema de clasificacion son Nearest Neighbor, Random Forest, SVM y Gaussian Processes. A continuacion los resultados, y luego comentarios de implementacion y conclusiones:



- **Nearest Neighbor:** Decidi usar este metodo simple para obtener un "baseline" del error, calculando Nearest Neighbor usando solo la letra central. Por ejemplo, si deseamos predecir el fonema generado por el input (y,p,o,t,e,n,u), se descartan las columnas de contexto y simplemente se busca en el dataset de training otro input donde la letra a predecir sea tambien **t**. Encontrado este nearest neighbor en train (por ejemplo, (w,a,n,t,-,-,-) , se retorna el fonema asociado **t**.

Como era de esperar, aproximadamente la mitad de los fonemas son predichos correctamente. Estos fonemas corresponden principalmente a consonantes cuya pronunciacion es generalmente la

misma. Este metodo tiene la gran desventaja de que, ademas, el tiempo de prediccion es muy alto pues hay que recorrer todo el dataset.

Vemos que al quintuplicar el tamanyo del dataset de train (1000 a 5000), no se ve un gran incremento en la reduccion del error. Mi hipotesis es que esto sucede porque este metodo es incapaz de hacer algun aprendizaje real, solo busca silabas similares, y esta forma de operar es muy limitada. Pronunciaciones de vocales que sigan patrones mas sofisticados, o fonemas que requieran de un contexto mayor, no seran predichos correctamente.

- **Nearest Neighbor silaba:** Similar al anterior. En este caso, si deseamos predecir el fonema generado por el input (y,p,o,t,e,n,u), buscamos el nearest neighbor en train usando las tres columnas centrales (o,t,e). De esta forma, se busca una silaba en train que sea parecida a la silaba que queremos predecir. Me sorprendio descubrir que el error obtenido por este metodo rivaliza a los otros metodos, siendo los demas mucho mas sofisticados. El salto de accuracy entre NN y NN-silabico demuestra la importancia de tener en cuenta el contexto de una letra a la hora de generar su fonema.

La principal desventaja de este approach es el tiempo que se tarda para realizar una prediccion, lo cual lo hace inadecuado para aplicaciones que generen sonidos en tiempo real.

- **Random Forest:** Dado que hay 18 columnas binarias de output, entrené 18 clasificadores RF, cada uno especializado en predecir la columna de output i-ésima. Utilicé 500 árboles por ensemble.

En mi opinion este es el mejor metodo de los que he experimentado. Si bien ANN con 50 neuronas es un poco mas preciso, el tiempo consumido para entrenar los 20 clasificadores random forest es menor que el de entrenar la red neuronal. Ademas, RF fue extremadamente sencillo de usar, pues no requirio optimizar hiperparametros.

Cada nodo en cada arbol de decision supongo que significara una regla especial para una determinada letra, dado que cada input tiene valor 0 o 1 (no hay mas variacion que esa). Por lo tanto, cuando los arboles de decision intenten hacer "cortes", representaran decisiones a tomar de acuerdo a particulares letras del input.

- **SVM:** Al igual que con RF, entrene 18 clasificadores, cada uno especializado en predecir la columna de output i-ésima. Use un kernel Gaussiano, lo cual involucra la dificultad extra de optimizar parametros. Para cada una de las 18 columnas, optimice C y σ en grid search.

El problema es altamente dimensional y no linear, por lo que la posibilidad de usar el kernel trick es muy valiosa. Optimizar los parametros a medida que el dataset crece se vuelve muy costoso computacionalmente, pero se reflejo en mejoras de aproximadamente 5%.

- **Gaussian Processes:** Si bien no estudiamos este metodo en el curso, me parecio interesante incluirlo[2]. Al igual que para SVM y RF, este metodo sirve para clasificacion y regresion. Tuve que entrenar un clasificador por columna.

Otra similitud de este metodo con SVM es la posibilidad de aplicar el kernel trick. El kernel usado es Gaussiano, aunque la optimizacion de parametros es realizada automaticamente por kernlab.

Una ventaja de este metodo es que cada prediccion tiene una varianza asociada, que nos permite medir cuan confiable es la prediccion. Otra ventaja es que los hiperparametros del kernel pueden ser aprendidos maximizando un likelihood, a diferencia de SVM.

Sin embargo, este metodo probó no ser nada adecuado para este problema. En primer lugar, dado que se necesita mantener una matriz de n^2 puntos en memoria donde n es el valor de training, el consumo de memoria resultó tan prohibitivo que no pude realizar los calculos para $n = 40000$ (5000 palabras). Una de las principales ventajas de GPs es que son muy buenos interpolando datos continuos en areas donde no hay mucha informacion, pero esta ventaja no es aprovechada en este problema.

- **ANNs** Las redes neuronales dieron los mejores resultados. Esta tarea de aprendizaje complejo, en donde el output es claramente una funcion no linear de los inputs, es adecuada para la potencia

de redes neuronales. Las redes se benefician de que tenemos una gran cantidad de datos de entrenamiento, y el uso de un conjunto de validación es muy útil para evitar overfitting. A diferencia de los otros métodos, hubo que entrenar una sola red con 18 outputs; mientras que todos los demás clasificadores (excepto KNN) requirieron entrenar 18 clasificadores diferentes.

5 IMPORTANCIA DE VARIABLES

No pude aplicar técnicas de clustering/reducción de la dimensionalidad debido a que no tienen sentido para la codificación numérica del dataset. Sin embargo, dado que Random Forest es uno de los algoritmos que dio mejores resultados, me resultó interesante aprovechar su capacidad de asignar una importancia a los inputs.

Para hacer este pequeño experimento, use el dataset de training más pequeño (200 palabras) y entrene 18 clasificadores RF (uno por cada feature del output, ver anexo 1).

Luego, para cada output, consulte cuáles eran las 26 columnas input más importantes de acuerdo a random forest. Como es de esperar, de las variables más importantes a la hora de generar fonemas, la mayoría pertenecen a las 26 columnas que codifican la letra central.

En promedio, si tomamos las 26 variables más importantes para RF, el 51.25% de ellas es uno de los inputs codificando la letra central. Además, el 82.91% de las 26 variables más importantes pertenece a los inputs que codifican las tres letras centrales. Esto se debe a que, naturalmente, las letras más importantes para generar un fonema son las más cercanas a la letra central.

Además, pude observar otras regularidades mirando outputs singulares. Tomemos por ejemplo la columna de output 8, que significa "fonema nasal" (el fonema predicho tendrá un sonido nasal). Sabemos, por la tabla del anexo 1, que los fonemas asociados a sonidos nasales son "m", "n", "G", "J", "M", "N". Letras que usualmente generan estos sonidos son "m" y "n".

De acuerdo a random forest, las cinco variables input más importantes para detectar si un fonema es nasal son:

[1] 92 91 83 85 87

Que corresponden a las letras n, m, e, g, i. Esto nos demuestra que random forest es suficientemente inteligente para descubrir estas conexiones entre letras del input generando particulares outputs.

6 REFLEXION FINAL

Los distintos métodos de machine learning que estudiamos en estos dos cursos dieron resultados muy buenos en este problema, con errores bastante similares. En el caso particular de lenguaje Inglés, la aplicación práctica de estos métodos puede no ser muy amplia dado que es un lenguaje extensamente estudiado y probablemente un sistema con reglas pueda tener mayor performance.

Sin embargo, imagino que este mismo procedimiento funcionaría para aprender cualquier idioma, provisto un dataset de palabras y sus transcripciones fonéticas. Esto es muy interesante, porque partiendo de una pequeña muestra de palabras, podríamos aproximar bastante bien la pronunciación del lenguaje.

Otra posible aplicación que se me ocurre es en el estudio de lenguajes antiguos o semi-extinguidos. Si solo se conoce la pronunciación de un subconjunto de palabras del lenguaje, podría ser posible aplicar estos métodos para predecir como sería la pronunciación del lenguaje completo.

7 ANEXO I: ARTICULATORY FEATURES

Phoneme	Sound	Articulatory Features
/a/	father	Low, Tensed, Central2
/b/	bet	Voiced, Labial, Stop
/c/	bought	Medium, Velar
/d/	deb	Voiced, Alveolar, Stop
/e/	bake	Medium, Tensed, Front2
/f/	fin	Unvoiced, Labial, Fricative
/g/	guess	Voiced, Velar, Stop
/h/	head	Unvoiced, Glottal, Glide
/i/	Pete	High, Tensed, Front1
/k/	Ken	Unvoiced, Velar, Stop
/l/	let	Voiced, Dental, Liquid
/m/	met	Voiced, Labial, Nasal
/n/	net	Voiced, Alveolar, Nasal
/o/	boat	Medium, Tensed, Back2
/p/	pet	Unvoiced, Labial, Stop
/r/	red	Voiced, Palatal, Liquid
/s/	sit	Unvoiced, Alveolar, Fricative
/t/	test	Unvoiced, Alveolar, Stop
/u/	lute	High, Tensed, Back2
/v/	vest	Voiced, Labial, Fricative
/w/	wet	Voiced, Labial, Glide
/x/	about	Medium, Central2
/y/	yet	Voiced, Palatal, Glide
/z/	zoo	Voiced, Alveolar, Fricative
/ʌ/	bite	Medium, Tensed, Front2 + Central1
/ç/	chin	Unvoiced, Palatal, Affricative
/ð/	this	Voiced, Dental, Fricative
/ɛ/	bet	Medium, Front1 + Front2
/G/	sing	Voiced, Velar, Nasal
/I/	bit	High, Front1
/J/	gin	Voiced, Velar, Nasal
/K/	sexual	Unvoiced, Palatal, Fricative + Velar, Affricative
/L/	bottle	Voiced, Alveolar, Liquid
/M/	absym	Voiced, Dental, Nasal
/N/	button	Voiced, Palatal, Nasal
/O/	boy	Medium, Tensed, Central1 + Central2
/Q/	quest	Voiced, Labial + Velar, Affricative, Stop
/R/	bird	Voiced, Velar, Liquid

Phoneme	Sound	Articulatory Features
/S/	shin	Unvoiced, Palatal, Fricative
/T/	thin	Unvoiced, Dental, Fricative
/U/	book	High, Back1
/W/	bout	High + Medium, Tensed, Central2 + Back1
/X/	excess	Unvoiced, Affricative, Front2 + Central1
/Y/	cute	High, Tensed, Front1 + Front2 + Central1
/Z/	leisure	Voiced, Palatal, Fricative
/ə/	bat	Low, Front2
/!/	Nazi	Unvoiced, Labial + Dental, Affricative
/#/	examine	Voiced, Palatal + Velar, Affricative
/*/	one	Voiced, Glide, Front1 + Low, Central1
/!/	logic	High, Front1 + Front2
/~/	but	Low, Central1

REFERENCES

- [1] Terrence J. Sejnowski and Charles R. Rosenberg *Parallel Networks that Learn to Pronounce English Text*. Complex Systems 1 (1987)
- [2] Book by Carl Edward Rasmussen and Christopher K. I. Williams *Gaussian Processes for Machine Learning*.