Aprendizaje Automático
Trabajo práctico Nro 2

GRUPO 7
ANTONI, FERNANDO, SUSTER, MATEO,
VARELA, HERNÁN, VAZQUEZ BROQUÁ, JUAN IGNACIO

MAESTRÍA EN EXPLOTACIÓN DE DATOS Y
DESCUBRIMIENTO DEL CONOCIMIENTO
DEPTO. DE COMPUTACIÓN – FCEYN - UBA

12 DE JULIO DEL 2020

Resumen

El presente trabajo evalúa una serie de modelos de reconocimiento del habla sobre audios en los que se pronuncian dígitos. Utilizaremos un conjunto de datos provisto por Google Speech Commands para entrenar cuatro modelos diferentes, con Naïve Bayes, Random Forest, Gradient Boosting y Redes Neuronales. Los modelos entrenados serán analizados en audios de test de la fuente original, así como en otros que presentan diversos tipos de ruido y en algunos grabados específicamente para este trabajo. El modelo de mejor performance fue el de Redes Neuronales que obtuvo un accuracy de 0,78 en el conjunto de testeo. En todos los casos, se vio una baja significativa al evaluarlos en datos con ruido o con datos grabados en distintas condiciones, siendo Redes Neuronales el que se mostró más robusto ante los datos modificados.

1. Introducción

El reconocimiento del habla es un tópico muy estudiando en las investigaciones sobre Aprendizaje Automático. En este trabajo utilizaremos distintos algoritmos como Naïve Bayes, Random Forest y Gradient Boosting Machine y Redes Neuronales para construir modelos que permitan reconocer audios con dígitos del cero al nueve. El objetivo del mismo es un problema de clasificación de 10 clases que intentará predecir qué dígito se dijo a partir del habla.

El informe consta de 4 secciones. En la sección de Datos, se especifican los datos utilizados y el cálculo que se realizó de los atributos para poder construir los modelos. Además, se tendrán en cuenta los atributos de los audios realizados, para evaluar las fortalezas y debilidades de cada uno de los modelos. Se analizará también el balance de las clases y la métrica utilizada. En la sección Metodología, se describirá la forma utilizada de partición de los datos en entrenamiento, validación y testeo, así como los algoritmos utilizados en cada modelo y las características e hiperparámetros evaluados de cada uno. En Resultados, se mostrarán las métricas obtenidas para cada modelo tanto en el set de datos de evaluación, como en el mismo modificado por distintos tipo de ruido. También, se analizarán los resultados de cada modelo en audios producidos para este trabajo. Asimismo, nos detendremos en analizar algunos de los errores cometidos. Finalmente, se expondrán las principales conclusiones obtenidas.

2. Datos

Para construir el modelo utilizaremos un conjunto de datos provisto por Google Speech Commands Dataset. Seleccionaremos los audios que correspondan al pronunciamiento de números del cero al nueve. Así, construimos un dataset con un total de 23.666 archivos. Los audios suelen tener una duración de 1 segundo y la cantidad de audios por dígito es similar, por lo que podemos afirmar que los datos se encuentran balanceados.

Para entrenar los modelos obtuvimos atributos de cada uno. Para ello, realizamos un procesamiento de señales, de forma tal de obtener los coeficientes cepstrales en frecuencia mel (MFCCs). Estos concentran mucha información del espectro en pocos coeficientes y poco correlacionados.

Al aplicar las transformaciones necesarias, nos quedamos con la media y el desvío estándar de los primeros 12 MFCCs (que contienen información sobre los formantes) y la energía. A su vez, se calcula sus diferencias de primer y segundo orden de estos 13 atributos. Esto nos da como resultado un total de 78 atributos.

Para la implementación del modelo de perceptrón multicapa se estandarizaron todos los features con media 0 y desvío estándar 1, así como *onehotencoding* para los targets.

Finalmente, para la evaluación de la performance de los distintos modelos se agregaron distintos tipos de ruido ambiente y gaussiano con una intensidad entre 0,01 a 0,1 al conjunto de test, así como se produjeron nuevos audios grabados para el presente trabajo, que también fueron evaluados por los modelos con modificaciones de ruido y sin ellas. Para todos los audios se extrajeron los mismos 78 atributos, que habían sido obtenidos de los datos originales de Google Speech Commands.

3. Metodología

3.1. Métrica elegida

Al ser un problema de clasificación con varias clases, pretendemos valorar las predicciones correctas, sin tener en cuenta el tipo de error en que pudiera incurrirse, por lo que elegimos utilizar como métrica de performance *accuracy*. Asimismo, los datos están balanceados, por lo que el modelo no debería dar preeminencia a casos de una clase por sobre la otra, sobrestimando la previsión del modelo. En ese sentido, consideramos que *accuracy* es la medida de performance que mejor se ajusta a nuestro problema.[1]

Asimismo, para realizar una análisis más detallado de los errores cometidos por los modelos, observaremos en algunos casos *precision* y *recall*, pero sin detenernos en sus valores, sino sólo para entender los números mal clasificados.

3.2. Entrenamiento, validación y testeo

Se utilizará la partición ya hecha de los datos originales para entrenamiento, validación y testeo. De esta forma nuestro set de datos consta de 18.620 audios para entrenamiento, 2.552 para validación y 2.494 para testeo. En la división ya realizada del dataset, las clases de cada uno se encuentran balanceadas.

[1] En el reconocimiento de voz, existe también una medida de performance muy reconocida llamada Word Error Recognition: "For more than three decades, the speech recognition community has coalesced around a single performance metric: word error rate or WER"[1]. A los fines del estudio realizado y por los motivos indicados, la métrica elegida es la que nos permitirá evaluar mejor la performance de cada modelo: "Accuracy has the advantage of offering a simple measure of the ability of the system to take the correct decision; it is the most-used metric in sound scene and sound event classification, being straightforward to calculate and easy to understand. It has, however, a critical disadvantage of being influenced by the class balance: for rare classes, a system can have a high proportion of true negatives even if it makes no correct predictions, leading to a paradoxically high accuracy value. Accuracy does not provide any information about the error types (i.e., the balance of FP and FN); yet, in many cases these different types of error have very different implications."[1]

Debido a que distintos audios fueron realizados por el mismo hablante, no se realizará cross-validation para analizar la performance de los entrenamientos del modelo. Esto permitirá que no esté la misma persona en los datos de entrenamiento, validación o test y cada modelo se evaluará con los datos de validación ya separados para comparar su desempeño. De esta manera, se evita que el modelo sobreajuste por recordar algún tipo de hablante.

Por otra parte, se construirán nuevos datos para poder evaluar el comportamiento de los modelos elegidos para cada algoritmo. En primer lugar, se agregará ruido gaussiano con distintas intensidades y diferentes tipos de ruido ambiente (tales como lavado de platos, una canilla abierta o una bicicleta fija) a los audios del set de datos de testeo. Por otro, se analizará cómo performa cada modelo con audios propios, producidos para el trabajo práctico, algunos grabados en distintas condiciones ambientales (grabados en un baño y con micrófono alejado) y otros con diversos ruidos ambiente agregados (de lavarropa, ducha, trenes, etc.).

3.3. Naïve Bayes

Naïve Bayes es un clasificador probabilístico, que intenta predecir la distribución de probabilidades sobre un conjunto de clases. En este caso, para clasificar una nueva instancia, se asignará la clase más probable, dados los atributos que describen a dicha instancia[2]. El mismo, se basa en la asunción de que las probabilidades de los atributos son independientes dada determinada clase. El mecanismo de aprendizaje se basa en la estimación de probabilidades de cada clase, y probabilidades de cada atributo dada la clase, según su frecuencia en los datos de entrenamiento. Dado que estamos trabajando con datos continuos, se utilizará la implementación de scikit-learn GaussianNB[3]. No se utilizarán priors, dado que las clases se encontraban balanceadas. Como se explicó en la sección anterior los datos son evaluados en el set de validación.

3.4. Random Forest

Random Forest es un algoritmo basado en bagging, lo cual consiste en utilizar distintos subsets de datos de entrenamiento con los cuales se entrenan árboles de decisión de manera simultánea. Este algoritmo combina una cantidad n de árboles, pero a diferencia de un árbol individual, que considera todas las variables para realizar cada split, Random Forest permite que sea considerado sólo un set aleatorio de features en cada nodo ("subset splitting")[4]. Esto busca eliminar la correlación entre los árboles, lo que permite reducir la varianza. Cuando un árbol cometa errores, probablemente otros no lo hagan y cuánto más independientes sean los árboles entre sí, más podrá reducirse la varianza.

Para la búsqueda del modelo de mejor performance, se evaluaránn distintos hiperparámetros en función del desempeño en los datos de validación, incluyendo la realización de bagging o no, sin realizar cross-validaiton por los motivos analizados en el apartado 3.2. La implementación se realizó también con scikit-learn[3]. Entre los hipermparámetros evaluados se encuentran los siguientes:

- Número de estimadores: número de árboles usados en el modelo (secuencia entre 150 y 250, en tandas de 5).
- Máxima Profundidad del árbol: de ser igual a 'None', los nodos se expanden hasta obtener hojas puras (rango entre 10 y 25).
- Bootstrap: indica si se elaboran muestras con boostraping al construir los árboles (True o False). Esta es una técnica de resampleo a partir de generar nuevos conjuntos de datos con reemplazos.
- Máximos features: cantidad de atributos a considerar cuando busca el mejor split (usamos el valor por default, el cual es igual a la raíz cuadrada de los features).
- Criterion: se analiza si se utiliza Gini Gain ("gini") o Information Gain ("entropy") para cada uno de los splits.

3.5. Gradient Boosting

Este es un algoritmo basado en boosting, que en lugar de entrenar en forma paralela los árboles con los distintos set de datos, el proceso se realiza en forma secuencial. De esta manera, a diferencia de bagging, cada modelo se construye usando información de los anteriores. De esta manera, intenta combinar weak learners, que tienen alto sesgo y baja varianza, para realizar un strong learner que reduzca el sesgo. En el caso particular del Gradient Boosting, se utiliza una función de costo, de manera que cada nuevo árbol se realiza de forma que se minimice dicha función, yendo en dirección opuesta al gradiente de dicha función, lo que se denomina descenso por gradiente. De esta forma, cada árbol es entrenado para predecir el residuo entre las observaciones y las predicciones anteriores, intentando mejorar los casos en los que las predicciones anteriores estaban equivocadas[5]. Tampoco se realizó cross-validation y para la implementación se utilizó scikit-learn[3]. Los hipermparámetros evaluados son:

- Número de estimadores (valores al azar entre 20 y 200).
- Máxima Profundidad del árbol (valores al azar entre 1 y 7).
- Learning rate: ponderador por el cual se reduce la importancia o contribución de cada nuevo árbol entrenado (valores al azar entre 0,01 y 1).

3.6. Modelo de Perceptrón Multicapa

El cuarto modelo que se implementó es el de perceptrón multicapa. Las Redes Neuronales están inspiradas en la biología y tienen unidades de procesamiento sencillas, que operan en paralelo y suelen ser robustas ante fallas. Cada neurona recibe datos de entrada y tiene pesos asociados, para ser sumadas luego en una combinación lineal de dichos inputs. Tienen además una función de activación y un umbral que permite variar su actividad y una tasa de aprendizaje.

En nuestro caso, se utilizará una capa oculta que recibirá como entrada un vector y la capa de salida tendrá 10 neuronas, que son la cantidad de clases que se intentan predecir en nuestro

problema (los dígitos del cero al nueve). Al ser clases mutuamente excluyentes, se utilizará la activación $softmax$ para forzar a que las probabilidades sumen 1.

Como función de costo, se utilizará ”categorical crossentropy” porque nuestro problema de clasificación admite una sola opción correcta para cada caso y una tasa de aprendizaje de 0,001 y 50 épocas. Para evitar el sobreentrenamiento, se dejará de entrenar cuando el error de validación deje de disminuir después de 5 épocas.

El modelo se implementó utilizando keras[6]. En el mismo se evaluaron distintos hiperparámetros, como el número de neuronas y la función de activación para la capa oculta en forma aleatoria. Para esto se utilizará la implementación de scikit-learn[3] $ParameterSampler$. La misma se utilizó además para ir modificando la cantidad de ejemplos de entrenamiento utilizados en cada iteración (batch-size) e ir evaluando de esta manera la performance en el conjunto de validación.

4. Resultados

4.1. Evaluación de los modelos y performance en datos limpios

La implementación del modelo Naïve Bayes de scikit-learn dio un valor de accuracy de 0,54 en los datos de validación.

El modelo Random Forest mejoró ese valor, obteniendo para accuracy 0,71 en el set de validación. Esto se consiguió evaluando en los hiperparámetros anteriormente mencionados, de los cuales los mejores resultaron 230 estimadores, 17 de profundidad, sin realizar Bootstrap y seleccionando entropía como criterio.

Por su parte, el modelo de Gradient Boosting mejoró el accuracy, obteniendo 0,75 en los datos de validación. El mismo se obtuvo haciendo la combinación de hiperparámetros aleatoria explicitada anteriormente. Para este modelo, los de mejor performance resultaron ser 161 estimadores, 4 de profundidad y un $learningrate$ de 0,257.

Por último, el modelo de Redes Neuronales resultó el que mejor accuracy obtuvo en los datos de validación con 0,79. Esto se consiguió con 40 iteraciones al azar, obteniendo el mejor valor con 750 neuronas en la capa oculta y la función de activación $relu$, con un batch-size de 256.

Al analizar las predicciones de estos modelos en los datos de evaluación las métricas fueron las siguientes: Naïve Bayes obtuvo 0,53, Random Forest 0,69, Gradient Boosting 0,73 y Redes Neuronales 0,78. Los resultados se presentan en la Tabla 1.

Con respecto a los errores en la evaluación con los datos de test, si bien a simple vista no se identifica algún sesgo a predecir algún número determinado por los modelos, existen algunos errores o confusiones a destacar. En todos los modelos mencionados se tiende a predecir, casi en la misma medida, el uno en casos que son cuatro o el nueve. Esto se observa por ejemplo en la matriz de confusión para el modelo de Gradient Boosting de la Figura 1. En Naïve Bayes esto no sólo es así, sino que también se presenta la tendencia inversa (posee una inclinación a predecir nueve, casos que son cuatro o uno), lo que podría deberse a similitudes en la pronunciación o en la fonética de los dígitos.

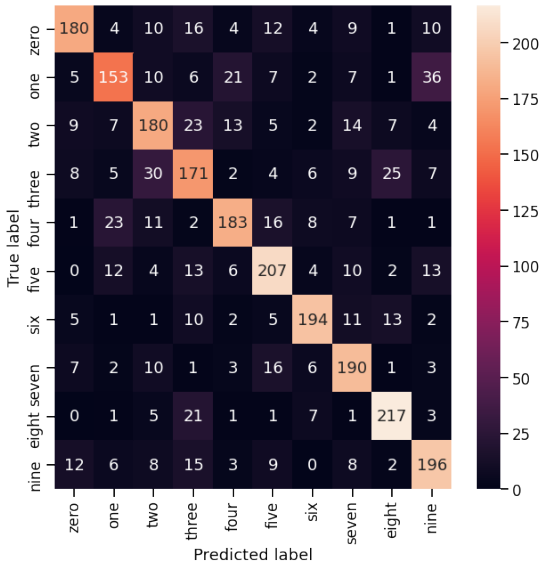

Figura 1: *Matriz de confusión de modelo de Gradient Boosting en conjunto de testeo*

Otro error que es común en todos los modelos, es que en algunos casos predicen el número tres, audios que corresponden al ocho o al dos, en especial en Random Forest y Gradient Boosting. En Naïve Bayes esto también ocurre, aunque se presentan otros errores de igual importancia (como la predicción de tres en lugar de dos o de siete en lugar de seis), lo que pudimos comprobar al escuchar algunos audios.

4.2. Evaluación de modelos en datos con ruido

Al evaluar en datos con ruido, las métricas disminuyen considerablemente, tal como se observa en la Tabla 1. En la misma, se presenta el accuracy para todos los modelos evaluados con los datos de validación, test y con los ruidos agregados sintéticamente.

En Naïve Bayes encontramos las métricas más bajas, oscilando entre 0,13 con ruido de *water tap* y 0,20 con ruido gaussiano con intensidad de 0.01. En general, encontramos que este modelo presenta un sesgo con respecto al dígito dos, el cual se inclina a predecir bajo cualquier efecto de ruido. Esto hace que el recall de la clase dos sea alto en todos los casos, dado que siempre que el número verdadero era dos, el modelo tendió a predecirlo correctamente, aunque el modelo performa de mala manera mirado de conjunto.

Algo llamativo en este caso es que existen algunos números que el modelo no eligió nunca en sus predicciones (o eligió muy poco), tales como el seis y el tres bajo todos los tipos de ruido y el uno, el siete y el dos en algunos casos particulares.

	Naive Bayes	Random Forest	Gradient Boosting	Red Neuronal
Validation	0,54	0,71	0,75	0,79
Test	0,53	0,69	0,73	0,78
Gaussian noise * 0,01	0,20	0,42	0,40	0,66
Bike exercise noise	0,16	0,33	0,39	0,61
Water tap noise	0,13	0,20	0,21	0,48
Dishes noise	0,20	0,28	0,34	0,52

Tabla 1: *Accuracy de cada modelo en datos de validation, test y con ruidos sintéticos*

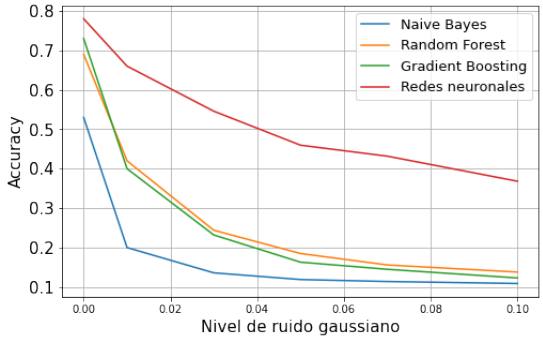

Figura 2: *Performance de los modelos ante diferentes niveles de ruido gaussiano*

Con respecto a Random Forest, el modelo posee un accuracy entre 0,42 con el ruido gaussiano con intensidad de 0,01 y 0,20 con el ruido de *water tap*. En la mayoría de los casos con ruido, este modelo tiende a predecir excesivamente el número seis, el dos y el cuatro, variando un poco según se trate de algún tipo de ruido en específico. Estas tendencias hacen que, por ejemplo, el precision para el número seis y dos sean siempre los más bajos (salvo para el caso con ruido *dishes*).

Al igual que en los modelos anteriores, con Gradient Boosting, la mejor métrica de esta sección se obtiene con el ruido gaussiano con intensidad de 0,01 (0,40) y la peor con el ruido *water tap* (0,21). De la misma manera que con Random Forest, salvo para los casos de ruido *water tap*, este modelo tiende a confundir cuando predice el número seis, el cual elige mayoritariamente. En cambio, con ruido de *water tap* tiene un sesgo a predecir los números dos, cero y nueve, independientemente de qué clase verdadera sea el audio introducido. Por tal motivo, el precision para tales casos es el más bajo.

En el caso de las Redes Neuronales, las métricas son claramente superiores a la de los otros modelos. Estas oscilan entre 0,66 para el caso del ruido gaussiano a 0,01 y 0,48 con el ruido de *water tap*. En el primer caso, si bien este modelo tiene algunos errores en algunos números particulares (por ejemplo, presenta la confusión de predecir como dos un número que en realidad era un tres) estos son problemas aislados, sin dar cuenta de algún sesgo significativo. En el resto de los casos, Redes Neuronales se presenta con mayor robustez frente a los ruidos. Aunque repite algunos patrones de errores que poseen los modelos anteriores (por ejemplo, mantiene un leve sesgo a predecir el número dos con el ruido de *water tap* o *dishes*), Redes Neuronales presenta una mejora sustancial en las predicciones.

Con la inclusión de ruido gaussiano los tres primeros modelos cometieron errores en las clases predichas: en Random Forest y Gradient Boosting sobrepredijeron el seis, mientras que Naïve Bayes sobrepredijo la clase dos. Con respecto a este tipo de ruido hicimos el ejercicio de testear cómo se ven afectados los modelos cuando ponderamos distintos niveles de ruido gaussiano (entre 0 y 0,1) en el data set de evaluación. Tal como vemos en la Figura 2, en los modelos Naïve Bayes, Random Forest y Gradient Boosting el accurracy disminuye notablemente. En cambio, en Redes Neuronales, observamos que, si bien su métrica de performance empeora, esta no disminuye abruptamente, sino que es más atenuada.

En presencia de ruidos ambientales sintéticos se vio una concentración de las predicciones en unas pocas clases. Dependiendo del ruido ambiental y el modelo, la concentración de clases fue cambiando. Por mencionar unos ejemplos, el caso del ruido de *bike exercise* acentuó los patrones observados con ruido gaussiano. En cambio, Redes Neuronales no presenta ningún sesgo a predecir algún número en particular, aunque sí tiene confusiones entre algunos números. En el caso del ruido *water tap*, se ve una mayor presencia a predecir la clase dos en todos los modelos, además de que en Naïve Bayes y Random Forest tienen un sesgo hacia la clase ocho y Gradient Boosting se repartió en cero y nueve.

Como se observa, los distintos tipos de ruido afectan a todos los modelos, haciendo que sus métricas disminuyan en proporciones similares. Por lo cual, las performances relativas entre modelos se mantienen, salvo para el caso del ruido gaussiano, en donde Random Forest mejora a Gradient Boosting.

4.3. Evaluación de modelos en audios propios

En los audios producidos para este trabajo se observa una reducción notable en la performance de cada modelo. Al evaluarlos sobre los datos generados sin modificaciones, el accuracy con Naïve Bayes es de 0,14, en Random Forest 0,40, en Gradient Boosting 0,30 y en Redes Neuronales 0,60. Los tres primeros modelos mantuvieron algunos de los sesgos mencionados anteriormente: Naive Bayes y Random Forest se inclinan a predecir la clase nueve, mientras que Gradient Boosting tiene problemas con la clase uno, tres y nueve. En cambio, Redes Neuronales no posee algún sesgo notable, sino que sus errores se relacionan a alguno de los problemas encontrados anteriormente (por ejemplo, la predicción como dos o tres, casos que en realidad eran seis u ocho).

La introducción de ruidos ambiente sintéticos sobre los audios propios no afecta a todos los modelos por igual. Podemos observar como Random Forest es más robusto al ruido que Gradient Boosting, algo que no habíamos obtenido anteriormente con la mayoría de los ruidos sintéticos sobre el dataset original. Asimismo, sobre estos conjuntos de test, Redes Neuronales posee también la mejor performance, superando notablemente a los modelos anteriores. El resumen de estas métricas se puede observar en la Tabla 2.

	Naive Bayes	Random Forest	Gradient Boosting	Redes Neuronales
Audios limpios	0.14	0.39	0.30	0.60
Ruido jardinero	0.13	0.40	0.20	0.38
Ruido ducha	0.14	0.39	0.30	0.47
Ruido tren	0.14	0.26	0.20	0.47
Ruido lavarropas	0.12	0.32	0.25	0.42
Ruido aeropuerto	0.13	0.32	0.26	0.35
En baño	0.26	0.57	0.52	0.74
Micrófono lejos	0.11	0.51	0.31	0.52

Tabla 2: *Accuracy de cada modelo en audios propios (limpios y con ruido)*

Cuando evaluamos los modelos con audios propios grabados en distintas condiciones ambientales la performance relativa entre modelos no sufre grandes cambios. En el caso de los realizados en un baño, Redes Neuronales obtuvo el accuracy más alto (0,74), seguido de Random Forest (0,57), Gradient Boosting (0,52) y Naive Bayes (0,26). Por su parte, como se observa en la Tabla 2, al evaluar con audios grabados con el micrófono lejos, la performance disminuye con respecto al caso anterior. Esto podría explicarse debido a que poseen menor volumen, lo cual dificulta la identificación de dígitos por parte de los modelos. Por otro lado, los grabados en el baño poseen una mejor métrica, en este caso seria con volumen moderado y mayor reflexión del sonido. Cabe destacar que estos dos conjuntos de audios poseen una menor cantidad de registros y hablantes, por lo que su performance absoluta no es del todo comparable con los analizados anteriormente en este apartado con ruido.

Una característica que notamos de los modelos evaluados es que baja notablemente el accuracy cuando los audios introducidos no se encuentran centrados. En primera instancia hicimos un recorte a través de un software de edición, en el cual todos los que producimos tenían duración exacta de 1 segundo, pero el audio no estaba precisamente centrado. Al evaluarlos, la performance en los distintos modelos fue menor al 15 %. Ante esto, se recortaron a través del método visto en clase (libreria Librosa[7]). Los audios así generados tienen duración menor a 1 segundo, pero si se encuentran precisamente centrados. En estas condiciones, conseguimos los resultados reportados. Los audios del set de entrenamiento, en general tienen una duración de un segundo y desviaciones con respecto al centro en general moderadas, por lo que se podría evaluar si esto impacta en la performance de los modelos entrenados con ellos. Como hipótesis, se podría continuar esta línea de investigación y entrenar los modelos introduciendo ruido y perturbaciones con respecto al centro a los audios de entrenamiento para evaluar la performance de los mismos prediciendo sobre nuevas grabaciones. Esta idea de ”perturbar” el set de entrenamiento para reducir el sesgo es conocida en la literatura como ”data augmentation” [1].

5. Conclusiones

A lo largo del presente trabajo desarrollamos una serie de modelos de reconocimiento de habla sobre audios con dígitos. En cada uno de los casos, Naïve Bayes, Random Forest, Gradient Boosting y Redes Neuronales, evaluamos la performance sobre el conjunto de evaluación y sobre otros audios grabados específicamente para este trabajo, incluyendo pruebas con diversos tipos de ruido.

En los datos de testeo, el modelo de Redes Neuronales fue el que mejor se desempeñó. La performance predictiva de todos los modelos cayó notablemente al evaluarlos con audios que poseían distintas intensidades de ruido gaussianos y otros tipos de ruidos ambientales, siendo el de Naïve Bayes el de peor accuracy. Esto puede deberse a que fueron entrenados con datos sin ruido y predicen peor cuando se modifican las condiciones en que se graban los audios. Asimismo, vimos como las métricas de performance disminuían notablemente al intentar predecir audios propios. Eso podría deberse también a que estos no fueron realizados por hablantes nativos, las condiciones de grabación o a los dispositivos que se utilizaron para la grabación.

Finalmente, puede destacarse que el modelo de Redes Neuronales demostró ser el más robusto en los distintas evaluaciones. Aunque con diferencias, este se mostró como el mejor para las predicciones realizadas tanto con los data set de test originales y modificados como para los audios propios, con y sin modificaciones.

Referencias

[1] T. Virtanen, M. D. Plumbley, and D. Ellis, *Computational analysis of sound scenes and events*, ch. 5 y 6, pp. 139, 148, 170. Springer, 2018.

[2] T. M. Mitchell, *Machine Learning*, ch. 6, pp. 154 – 190. New York: McGraw-Hill, 1997.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] G. Seni and J. Elder, *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*, ch. 4, pp. 54 – 62. Morgan and Claypool Publishers, 2010.

[5] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, ch. 8, pp. 319 – 323. Springer, 2013.

[6] F. Chollet *et al.*, "Keras," 2015.

[7] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015.