Aprendizaje de Máquinas - MA5204: Propuesta Proyecto Final
Clasificación de Posturas de Mano Mediante Actividad Muscular

Matias Alegria [1] **Eduardo Jorquera** [1] **Vicente Cabezas** [2]

1. Abstract

En el presente documento, se plantea la motivación, desarrollo, resultados y conclusiones del proyecto final del curso MA5204. El problema escogido consiste en predecir la postura de la mano, dentro de 4 posibles alternativas, según los datos obtenidos mediante electromiografías. Se cuenta con una base de datos con las mediciones de 8 puntos del brazo, en 8 instantes de tiempo durante la postura. Este trabajo tiene como motivación aportar al área de diseño de prótesis ortopédicas robóticas para asistir a quienes les falta una extremidad.

Se estudia y describe la base de datos, concluyendo que es adecuada para ser presentada a modelos de clasificación, por lo que se implementan varios de estos, siendo los mejores Random Forest y Una red neuronal con dos capas fully-connected, cada una con dropout. Tras optimizar los modelos, se obtiene un accuracy máximo de 94% con la red neuronal.

Se concluye que los resultados obtenidos son satisfactorios y se cumplen los objetivos del trabajo experimental, esperando que sirvan para a futuro enfrentar problemas más complejos de la misma índole, como por ejemplo lidiar con la clasificación de gestos de una base de datos más amplia, con más clases, lo cual potencialmente ayudaría a la funcionalidad de las prótesis robóticas para pacientes que les falta una mano.

2. Introducción

La relación entre humano y máquina es una característica fundamental de la era moderna, donde las personas interactúan a diario con una infinidad de dispositivos electrónicos. En particular, un área de la ciencia de la salud corresponde al desarrollo de prótesis ortopédicas para aquellos que carecen de una o más extremidades. Esta área se ha relacionado en el último tiempo con la robótica, creando el área de la robótica protésica, que se dedica al diseño y prueba de diversas soluciones para los pacientes que requieren de estos artefactos.

En este contexto, se genera el interés de reconocer entre los distintos posibles movimientos de una extremidad, mediante los impulsos eléctricos generados por el sistema nervioso. Estos se pueden medir en los músculos mediante una electromiografía (de ahora en adelante EMG), que consiste en utilizar electrodos en la piel, que miden la tensión eléctrica en un punto del cuerpo a través del tiempo.

El objetivo de este trabajo es desarrollar modelos de clasificación que logren distinguir entre distintas posturas de la mano, según los datos obtenidos mediante EMG. Esto podría ser de utilidad en el desarrollo y uso de prótesis robóticas para pacientes que carezcan de una mano o un brazo.

La base de datos utilizada corresponde a la medición de EMG de 4 poses distintas de la mano, tomando 8 mediciones en 8 instantes de tiempo para cada una. Las mediciones tomadas comenzaban y terminaban con el brazo en la misma postura. Las posturas fueron las 3 jugadas del piedra papel o tijera, junto al gesto de "OK", que corresponde a juntar las yemas del índice y el pulgar, mientras el resto de los dedos están estirados.

3. Base de Datos

En primera instancia, se crea un jupyter notebook, donde se importan las librerías necesarias. Estas incluyen las usuales como numpy, pandas, etc, junto a las funciones asociadas clasificación que son extraídas desde sklearn.

La base de datos [1] consiste de cuatro archivos en formato CSV, donde cada uno son los datos tomados para cada una de las 4 posturas de mano explicadas anteriormente. Estos archivos son subidos a un jupyer notebook en Google Colab, donde se convierten en objetos *Dataframe* de la librería pandas. Se juntan las 4 tablas en un único Dataframe y se le asigna nombre a los headers, para que sea más fácil entender a qué corresponde cada columna.

Se observa que la nueva tabla no contiene valores nulos ni NaNs, lo cual significa que no hay que realizar una limpieza de datos. La tabla resultante posee 11678 datos (filas) y 65 features (columnas). De los features, 64 son de tipo float y la label de clase es del tipo entero. Notar que los 64 valores en

[1] A recording of human hand muscle activity producing four different hand gestures

punto flotante son las mediciones de voltaje en mili-voltios. En cuanto al balance de clases, se observa que hay 2910 datos para la clase 0, 2903 para la clase 1, 2943 para la clase 2 y 2922 para la clase 3. Se observa un conjunto de datos balanceado, por lo que no se requiere utilizar estrategias como OverSampling ni UnderSampling.

Se calculó una matriz de correlación para los atributos, pero no se pudo concluir una relación suficiente entre variables para eliminar features o extraer información útil para los modelos de clasificación.

Visualización de la base de datos

Representar gráficamente la data de las bases de datos resulta bastante útil, dado que permite distinguir visualmente tendencias, valores atípicos, patrones, clusters, entre otras características, que se deben considerar al momento de tomar decisiones. Por ello, se decidió utilizar tres herramientas de visualización, PCA, T-SNE y UMAP, que consisten principalmente en herramientas multiuso, de tratamiento de data, reducción de dimensiones y visualización de data, para transformar la base de datos a 2 y 3 dimensiones, y con ello visualizar las características resultantes. A continuación se mostraran los resultados de dichas transformaciones y visualizaciones.

PCA

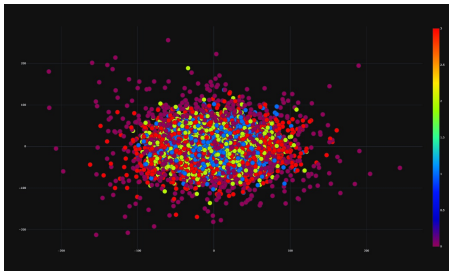

Figure 1. PCA con 2 componentes

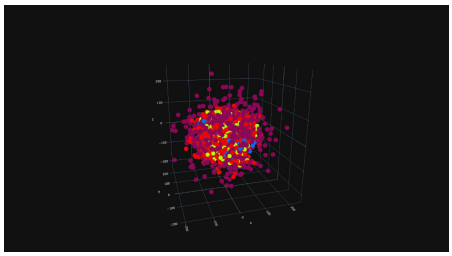

Figure 2. PCA con 3 componentes

T-SNE

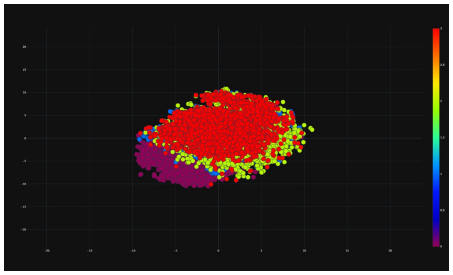

Figure 3. T-SNE con 2 componentes

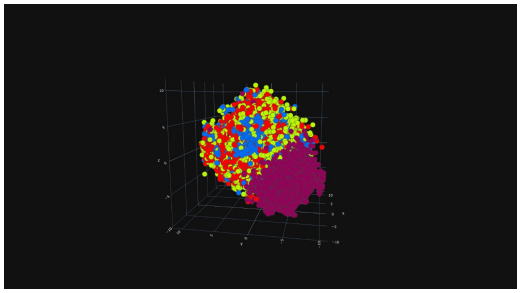

Figure 4. T-SNE con 3 componentes

UMAP

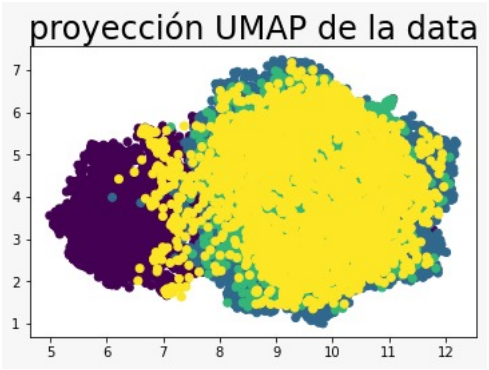

Figure 5. UMAP con 2 componentes

A pesar de que las 3 herramientas, favorecen la separación de elementos para facilitar la visualización de clusters, se puede observar tanto en 2 dimensiones como en 3 que la data yace muy concentrada, lo que implica que los elementos son relativamente similares y no hay regla directa para separar los conjuntos por clase. También se puede destacar que de las 4 clases, una es visualmente más separable que las otras 3, en particular, la de color morado (clase 0). Finalmente, se puede mencionar que en general, no hay un gran número de outliers visibles en los datos. En particular en el grafo

2D del PCa se observan puntos alejados del centro, pero la mayoría son de la clase 0, habiendo pocos puntos de otra clase alejándose del centro de masa de la nube de puntos.

4. Desarrollo

Creación del conjunto de Entrenamiento, validación y Test

En primera instancia, se separó la base de datos en 2 archivos de misma proporción de clases, donde uno de ellos presentaba un 10% de los datos originales y el otro 90%. Este primer conjunto sería el conjunto de prueba o test y no sería tomado en cuenta para el diseño e implementación de ninguno de los modelos.

Adicionalmente, mediante la función "train_test_split" de sklearn, se separó al implementar los distintos modelos entre conjunto de entrenamiento y validación. El conjunto de entrenamiento sería el que usen los modelos para aprender a clasificar los datos y el de validación sería el que se utilizaría para evaluar el desempeño de cada clasificador y tomar desiciones respecto a sus hiperparámetros o arquitectura según el caso correspondiente.

Modelos de Clasificación

Todos los modelos fueron implementados en python 3.6, mediante un jupyter notebook [2].

Clasificador Dummy

Como primer acercamiento al problema, se decidió implementar algoritmos de clasificación simples, para establecer un baseline sobre el cual mejorar en futuras entregas. El primero de estos fue un clasificador "Dummy"(4), el cual decide la clase según reglas heurísticas simples, como por ejemplo elegir la clase mayoritaria o incluso al azar. Esto se hizo con el fin de poder tener un valor de accuracy del "peor" modelo, con el cual se pueda comparar el resto de los modelos, siempre buscando mejoras en métricas sobre este "piso" mínimo.

K-vecinos cercanos

El primer modelo "real" que se implementó fue un algorítmo de K-vecinos más cercanos o KNN(5) por sus cifras en inglés. Este modelo se basa en comparar el dato a clasificar con un conjunto de puntos conocidos (datos de entrenamiento), mediante sus distancias. Se buscan los K puntos más cercanos al punto de estudio, y la clase predominante en esos puntos será la predicción del modelo. Una variante a este modelo es el KNN con pesos, o WKNN por sus cifras en inglés. Su diferencia con el KNN tradicional yace en que la votación entre los K vecinos ahora está ponderada al inverso de su distancia con el punto de estudio, lo que significa que los puntos más cercanos influyen más en la clase predicha por el modelo.

Máquina de soporte vectorial

El modelo SVC(6) o Support Vector Machine busca separar las clases creando hiperplanos separadores que maximizan el margen de distancia entre el hiperplano y los puntos más cercanos en el espacio de datos. Adicionalmente, se pueden proyectar los puntos a espacios de mayores dimensiones mediante el uso de funciones kernel, que permiten simplificar la separación de clases comparado al espacio original.

La implementación se realizó con la clase SVC de sklearn en python, la cual permite usar diferentes kernels y funciones de decisiones, en este caso se probaron núcleos lineal, polinomial, sigmoide, y RBF. También se experimentó cambiar la regla de decisión entre one vs all, one vs one, donde finalmente se tomaron resultados con la combinación que tuviera mejor desempeño.

Regresión Logística

Luego, se ocupó Regresión Logistica(7), un método de regresión que permite estimar la probabilidad de una variable cualitativa o categórica binaria en función de una variable cuantitativa. Su principal enfoque yace en la clasificación binaria de elementos o en que las variables dependientes toman valores dentro de un conjunto finito de valores.

La implementación se realizó con la clase LogisticRegression de sklearn en python, la cual permite usar diferentes penalizaciones, métodos de optimización y estrategias multiclase: one v/s one y one v/s all. Se escogió la combinación que maximizara el accuracy.

Random Forest

Se sigue con Random Forest(8), modelo que usa árboles de decisión para predecir, de tal forma que cada árbol depende de un vector aleatorio independiente y con la misma distribución. Este modelo utiliza una selección aleatoria de atributos, lo cual permite construir árboles de decisión con una variación controlada. En este caso se separaron los datos para luego normalizarlos con la función tf.keras.utils.normalize. Finalmente se ocupó un número de estimadores igual a 100, en donde el número máximo de features para cada árbol se obtendría con la raiz cuadrada del número de features totales, es decir 10.

Red Neuronal

El ultimo modelo en ser implementado fue una red neuronal(9), el cual consiste en un conjunto de unidades, llamadas neuronas artificiales, o perceptrones, conectadas

[2] Notebook Proyecto

entre sí para transmitirse señales. La información de entrada atraviesa la red neuronal donde luego se propaga una función de error hacia atrás, modificando los pesos de cada conexión entre unidades. La estructura de la red neuronal implementada fue de dos capas ocultas de 1024 neuronas cada una y una capa de salida de 4 neuronas. Las neuronas de capa oculta utilizaron función de activación ReLU mientras que las de salida utilizaban la función Softmax. Adicionalmente se implementaron capas de dropout del 20%, esto significa que durante el entrenamiento hay una probabilidad del 20% de que la neurona se desconecte y no participe de esa iteración, lo que aumenta la robustez del modelo, pues no depende siempre de unas pocas unidades. Su elaboración fue realizada sobre la arquitectura de Tensorflow y Keras (10) y se optimizó con el algoritmo "Adam".

Uso de Grid Search

Finalmente, cabe mencionar que todos los métodos empleados fueron optimizados por un GridSearch Manual o de las librerías de sklearn. Un grid search es un algoritmo exhaustivo que recorre las distintas posibles combinaciones de parámetros de un modelo, lo que se realiza con objetivo de maximizar la calidad de clasificación del modelo.

5. Resultados, Análisis y Discusión

En la tabla 1, se observan los resultados de los distintos modelos implementados, tanto en conjunto de validación como en el conjunto de test. Se ordenan los resultados de peor a mejor.

Table 1. Accuracy obtenido por distintos modelos de clasificación

Clasificación	Acc. Val. [%]	Acc. test [%]
Dummy	25.21 ± 0.00	25.25 ± 0.0
LogisticRegresion	34.33 ± 1.47	34.96 ± 0.57
KNN	68.09 ± 0.01	68.24 ± 0.92
WKNN	68.04 ± 0.01	66.67 ± 0.26
SVC	86.72 ± 0.67	87.31 ± 0.34
RandomForest	92.48 ± 0.69	93.66 ± 0.32
Red Neuronal	94.38 ± 0.51	94.24 ± 0.52

Analizando la tabla 1, es posible notar que hay un gran salto en desempeño entre el clasificador *LogisticRegression* y el modelo *KNN*, de aprox. 33%. La siguiente mejora notoria fue del *SVC* que presentó una mejora de desempeño con respecto al *WKNN* de aprox. 21%. Los dos mejores modelos fueron *RandomForest* y la *Red Neuronal*, llegando a un máximo de accuracy de 94.24%.

Se muestran a continuación las matrices de confusión en el conjunto de test de los dos mejores modelos obtenidos:

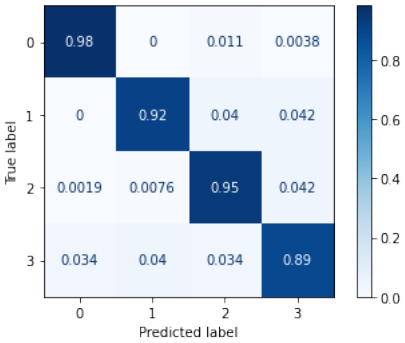

Figure 6. Matriz de confusión para RF

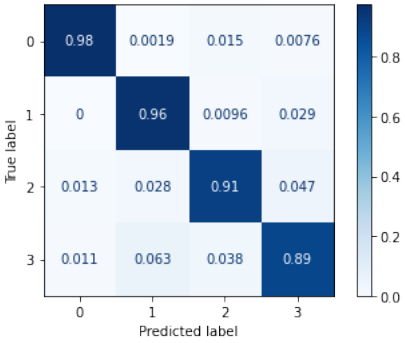

Figure 7. Matriz de confusión para la Red Neuronal

Se observa en estos resultados que está fuertemente marcada la diagonal, que es lo que se busca obtener en problemas de clasificación. A pesar de que la métrica de desempeño es el accuracy general y no toma en cuenta el desempeño particular por clase, se obtienen buenos resultados, esto es debido a que los datos estaban balanceados. Destacar que la clasificación de la clase 0 es la con mejor desempeño, lo cual tiene sentido con lo visto en las visualizaciones de datos, pues era la que estaba más notoriamente separada del resto.

Los resultados mostrados anteriormente son razonablemente esperables, pues *DUMMY classifier* es un clasificador básico que utiliza reglas simples para clasificar, mientas que *KNN* y *WKNN* no aprenden a reconocer patrones ni a generalizar la data, sino que utilizan la ubicación de los puntos en R^n y sus distancias para clasificar. *LogisticRegresion*, en cambio, tiene una arquitectura de clasificación binaria, aplicando los enfoques One v/s One y One v/s All, que tiende a fallar en problemas multiclase.

Por otro lado, *SVC*, es un clasificador supervisado de arquitectura binaria, que posee un mejor desempeño pues aplica las mismas estrategias de *LogisticRegresion*, pero intentando trazar el hiper-plano que genere el mejor límite o distancia entre clases, aplicando una función de kernel para proyectar

los datos en un hiperespacio que los haga más separables.

En cuanto a *Random Forest*, se tiene que, al utilizar los múltiples árboles de decisión creados aleatoriamente, maximizará el conjunto de decisiones para obtener el mejor posible resultado, lo cual funciona bastante bien para este problema multiclase.

Por último, la *Red Neuronal*, es el modelo o el clasificador más complejo, dado que se compone de múltiples neuronas o capas ajustables, que se encargan de encontrar patrones en los datos, minimizando el error. El número grande de neuronas utilizadas (2048), junto a las capas de dropout, permitieron generar un modelo capaz de reconocer razonablemente bien las distintas clases, dentro de un espacio complejo y dificil para otros modelos.

Es interesante notar que los 2 mejores modelos obtenidos presentan enfoques bastante distintos para procesar y clasificar los datos, pero aun así lograron resultados similares.

6. Conclusiones

Aprendizajes

Tras implementar todos los modelos de interés, optimizarlos y obtener sus métricas, se puede concluir que se ganó experiencia en el uso del lenguaje de programación python orientado a la aplicación de modelos y librerías relacionadas al aprendizaje de máquinas. Adicionalmente se aprendió a planificar un pipeline de trabajo y los distintos aspectos que se deben cubrir al realizar una investigación en esta área.

Alcances y trabajos futuros

El mayor alcance del proyecto es el haber podido implementar un método de clasificación razonablemente exitoso, buenas métricas para las mediciones registradas en la base de datos, alrededor de un 94%. Un aspecto a futuro que se puede trabajar es una toma de muestras con un mayor número de gestos, pues la actual solo cuenta con 4 distintas. De esta forma, se podría aumentar la posible funcionalidad de una prótesis ortopédica que utilice estos datos para funcionar.

Dificultades

La primera dificultad fue no poder extraer visualmente muchas características, patrones, outliers, clusters, entre otros, de las herramientas de visualización. Estas podrían haber ayudado a tener ideas más claras sobre la base de datos, entendiendo mejor la forma en que se agrupan los datos.

Otra dificultad fue tener que realizar el modelo de clasificador de red neuronal "a mano". Es decir, se enfrentó a un problema de diseño, pues se debió probar distintas configuraciones con distinto número de capas y unidades, siempre buscando el mejor desempeño posible.

References

[1] G. (2020, 22 de Noviembre). *Hand Gesture Prediction*. Kaggle. Recuperado el día 29 de Julio del 2021. https://www.kaggle.com/gcdatkin/hand-gesture-prediction

[2] G. (2021, 08 de Marzo). *Hand Gesture Rícognition with Python*. Kaggle. Recuperado el dia 29 de Julio del 2021. https://www.kaggle.com/gauravduttakiit/hand-gesture-recognition-with-python

[3] C. (2019, 1 Diciembre). *GitHub - cyber-punk-me/emg-nn: A classifier for EMG data that can run directly or as an API in Hivemind*. GitHub. Recuperado el día 29 de Julio del 2021. https://github.com/cyber-punk-me/emg-nn

[4] scikit-learn developers. (2011a). *sklearn.dummy.DummyCassifier — scikit-learn 0.24.2 documentation*. Scikit-Learn. Recuperado el día 29 de Julio del 2021. https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html

[5] scikit-learn developers. (2011). *sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.24.2 documentation*. Scikit-Learn. Recuperado el día 29 de Julio del 2021. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[6] scikit-learn developers. (2011). *sklearn.svm.SVC — scikit-learn 0.24.2 documentation*. Scikit-Learn. Recuperado el día 29 de Julio del 2021. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[7] scikit-learn developers. (2011). *sklearn.linear_model.LogisticRegression — scikit-learn 0.24.2 documentation*. Scikit-Learn. Recuperado el día 29 de Julio del 2021. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[8] scikit-learn developers. (2011). sklearn.ensemble.RandomForestClassifier — scikit-learn 0.24.2 documentation. Scikit-Learn. Recuperado el día 29 de Julio del 2021. https://scikit-learn.org/stable/

modules/generated/sklearn.ensemble.RandomForestClassifier.html

[9] H. Kinsley (2018, 11 de Agosto). Python Programming Tutorials. pythonprogramming. Recuperado el día 29 de Julio del 2021. https://pythonprogramming.net

[10] The Sequential model — TensorFlow Core. (2021, 25 marzo). TensorFlow. Recuperado el día 29 de Julio del 2021. https://www.tensorflow.org/guide/keras/sequential_model