Tarea 1 - Aprendizaje Automático: Clasificación de EMNIST-Letters con Support Vector Machines

Nombre: Daniel Psijas
Profesora: Violeta Chang
Departamento de Ingeniería Informática
Universidad de Santiago de Chile, Santiago, Chile

***Resumen*—El siguiente documento describe el proceso de experimentación con modelos de aprendizaje automático de Support Vector Machines para clasificar imagenes de letras manuscritas. Se desarrolla el código en Python y se experimenta con distintos parámetros y modos de representar los datos, como lo sería con hiperparámetros por defecto, busqueda de hiperparámetros optimos, reducción de dimensionalidad y embedding, resultanto este último como el modelo más robusto testeado con pruebas adicionales como K Cross Validation. Mientras que los modelos con hiperparámetros ajustados suelen dar malos resultados para este tipo de datos. Se discuten los resultados y se evalúan disitntas métricas como el accuracy total, la precisión, recall y F1-Score por cada clase.**

***Palabras claves*—Aprendizaje Autommatico, clasificador, SVM, accuracy, precision, recall, F1-Score, PCA, embedding, K-Cross Validation, Bagging**

I. Introducción

A continuación se realizarán experimentos de Aprendizaje Automático supervisado, clasificando el conjunto de datos de EMNIST-letters (imagenes de letras) [1] con Support Vector Machines (SVM), que es un modelo que divide el espacio de datos con hiperplanos que separan de mejor modo las clases, en este caso cada letra. En este caso se usará Python con la biblioteca de Scikit-Learn, para entrenar distintos modelos de SVM con diferentes parámetros, modos de entrenamiento y formas de representar los datos con tal de determinar cuál es el que obtiene mejores resultados clasificando las imagenes en las 26 clases de letras. Se mostrarán los resultados y se discutirán sus causas, dando recomendaciones y determinando el mejor modelo.

II. Desarrollo

Para desarrollar este trabajo se empezó leyendo los datos descargados de EMNIST Letters [1] tanto del conjunto de entrenamiento (train) y conjunto de prueba (test), para a partir de ahí generar una muestra propia de entrenamiento de y de prueba donde hay 1000 ejemplos de cada clase para entrenamiento y 100 ejemplos de cada clase para prueba. Se estableció una semilla fija para manetener los resultados aleatorios al muestrear. Los datos consisten en los valores numéricos de los pixeles en escala de grises (0 a 255) de la imagen 16x16 que representa una letra manuscrita que se debe clasificar en las 26 clases representando las 26 letras del alfabeto inglés, consistiendo en 256 características por cada dato. A continuación se describe como se construyó cada modelo SVM para entrenar con estos datos.

II-A. SVM inicial

El modelo inicial de SVM será con los datos anteriormente descritos (26000 de entrenamiento y 2600 de prueba, con 1000 y 100 ejemplos de cada clase respectivamente) con los parámetros C y gamma por defecto y un kernel de base radial (RBF). Para esto se creó la función *trainSVM* que entrena los modelos en base al conjunto train y test, también se creo la función *metrics* para evaluar los modelos y graficar medidas como accuracy, precisión, recall y F1-Score por cada clase.

También se implementó una función para entrenar un modelo con parámetros dados y se entrenó esta SVM con distintos valores para ellos, para eso se creó la función *best_model* que usa la función anterior y recibe un arreglo de parámetros C y gamma y decide cuál es mejor en base al accuracy generado por cada uno, se probó con valores de C entre 0.2 a 1000 y de gamma desde 0.0001 a 100. En la Figura 1 se muestra el código de esta parte junto con los valores probados.

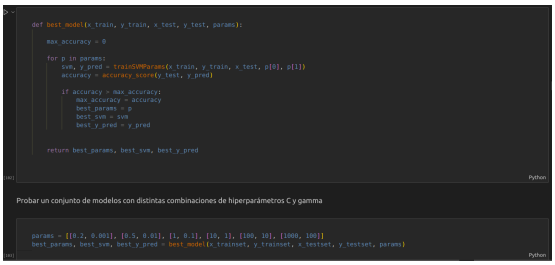

Figura 1: Función best model

II-B. SVM reducida con PCA

El segundo modelo a evaluar es una SVM entrenado con datos a los que su dimesionalidad se redujo a casi la mitad con PCA, de 256 a 128. Se usaron los mismos conjuntos de entrenamiento y prueba muestreados para el experimiento anterior, pero con la dimensionalidad reducida. En la Figura 2 se muestra el código que implementa esta parte.

```
pca = PCA(n_components=128)

# Ajustar el PCA al conjunto de datos
pca.fit(x_trainset)

# Obtener las componentes principales y transformar el conjunto de dat
x_pcaTrain = pca.transform(x_trainset)
x_pcaTest = pca.transform(x_testset)


svmPCA, y_predPCA = trainSVM(x_pcaTrain, y_trainset, x_pcaTest)
```

Figura 2: Aplicación de PCA

II-C. SVM con embeddings

Para el tercer experimento se usó el conjunto de datos con las características como salidas de redes neuronales para simplificar la representación y capturar a la vez patrones significativos (embeddings). Para eso se leen los datos que provee EMNIST-letters tanto de entrenamiento como de prueba y se seleccionan los datos con los mismos conjuntos de índices que se seleccionaron para los entrenamientos anteriores. Entonces se llama a la función para entrenar la SVM con parámetros por defecto y también se probó con los mejores parámetros encontrados en la sección anterior.

II-D. Pruebas adicionales

A partir de los resultados anteriores se hicieron pruebas adicionales para intentar mejorar o verificar la confiabilidad de los modelos, para eso se hizo una función de validación cruzada de k pliegues con k=10 en donde se evalúa el mejor modelo resultante de los experimentos anteriores (en la sección siguiente se muestran), se juntan los datos de entrenamiento y prueba anteriores en un solo dataset, ya que en este algoritmo se van rotando los conjuntos de entrenamiento y prueba. También se hizo un ensamblado con bagging del mismo modelo en base a 10 de ellos para ver si se podía mejorar la capacidad predictiva de esta técnica.

III. Resultados experimentales y discusión

A continuación se muestran los resultados de cada modelo mostrado en la sección anterior, se compararán y se discutirán.

III-A. SVM inicial

Para la SVM inicial con parámetros por defecto se tiene una accuracy de 0.86 para el conjunto de prueba, si bien tiene un desempeño aceptable se puede mejorar aún más. En la Figura 3 se muestra un gráfico de la precisión, recall y F1-Score (que combina las dos primeras en una medida) por clase. Tuvo mejores resultados en la clase 12 y 18, que representan las letras **M** y **R** respectivamente. En la Figura 4 se muestra la matriz de confusión que asocia cada clase predicha con cada clase real. La matriz diagonal verde indica que la mayoría de casos predichos se concentraron en las etiquetas correctas, sin embargo también hay algunos valores predichos fuera de la diagonal con un máximo de 22 ejemplos mal clasificados.

Para mejorar esto se probaron distintos parámetros C y gamma sobre dicho modelo, como se puede ver en la Figura 1, dando por resultado que para ese conjunto de

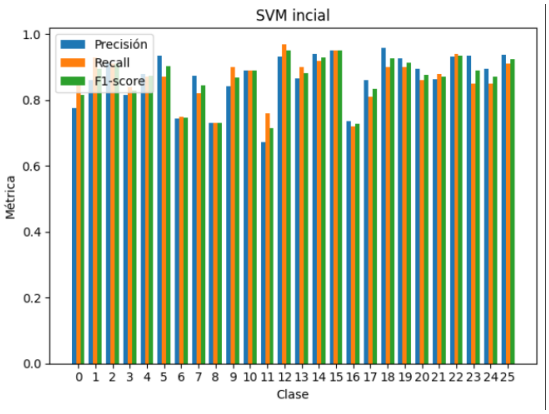

Figura 3: Medidas - SVM Inicial

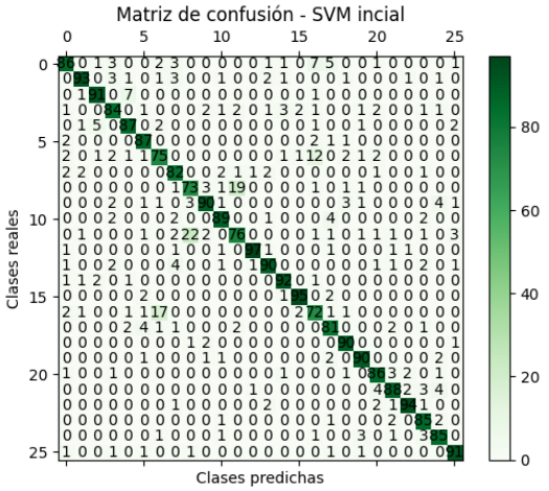

Figura 4: Matriz de Confusión - SVM inicial

hiperparámetros el mejor fue de C=0.2 y gamma=0.001, sin embargo para ser el mejor el modelo resultante es bastante pobre con un accuracy total de 0.1. En la Figura 5 se puede ver que ni siquiera predijo todas las clases ya que hay unas que no tienen casos. Hay medidas altas de precisión concentradas en algunas clases, sin embargo con un recall (y consiguiente F1-Score) deficiente. Recordar que la diferencia entre precisión y recall es que la precisión mide los resultados clasificados como positivos respecto a todos los ejemplos mientras que el recall mide los resultados clasificados como positivos respecto a los que verdaderamente lo son, esto quiere decir que clasificó relativamente bien muchos ejemplos en relación a todos los datos pero respecto a los datos que realmente lo son, le faltó bastante por acertar. Con respecto al empeoramiento del rendimiento respecto del experimento anterior, se puede deducir que para este caso ajustar hiperparámetros empeora los resultados (considerar que este es el mejor caso con hiperparámetros). El parámetro C introduce un margen de error para tolerar ciertos ejemplos mal clasificados en el entrenamiento con tal de que el modelo resultante sea más generalizable y no se sobreajuste, como C es el costo del error que penaliza la función a optimizar, un C bajo como es

el caso introduce un margen grande. Mientras que el valor de gamma regula cuánto afectan ciertos ejemplos de entrenamiento a una distancia del hiperplano, un valor bajo evita el sobreajuste. En este caso valores altos tanto para gamma y C empeoran el modelo, sugiriendo que es sensible a los cambios en los hiperparámetros y esto puede tener implicaciones en la varianza de los modelos. Por lo tanto, esto sumado a que introducir hiperparámetros agrega un gran costo computacional, para los siguientes experimentos se usarán los parámetros por defecto, ya que son los que mejor resultados producen.

En la matriz de confusión de la Figura 6 se pude ver algo interesante, y es que la mayoría de clases predichas se concentran en la última (letra Z), lo que introduce una clase de bias, provocando así que debido a los cambios en los hiperparámetros se vuelva indistinguible una clase en particular versus las demás.

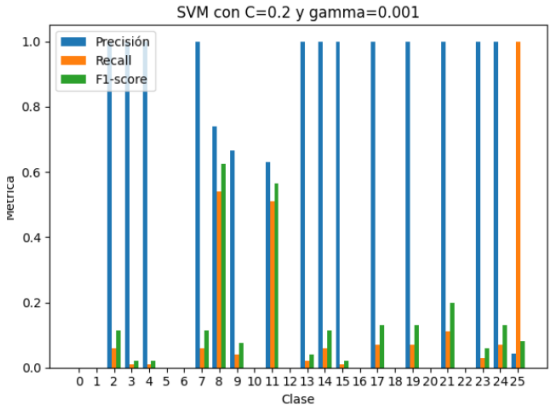

Figura 5: Medidas SVM con hiperparámetros

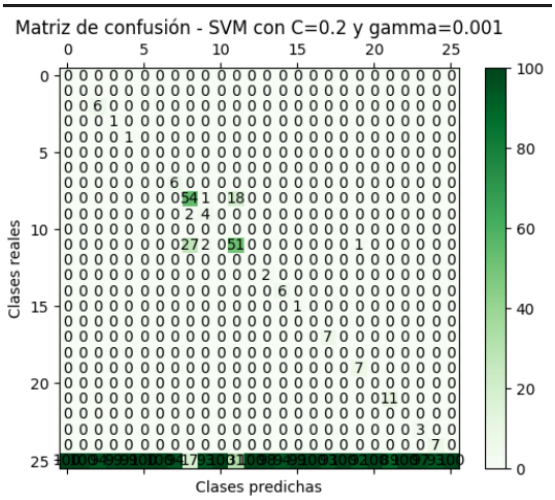

Figura 6: Matriz de Confusión - SVM con hiperparámetros

III-B. SVM con PCA

Al aplicar PCA y entrenar un modelo SVM con parámetros por defecto se obtiene una precisión de 0.88, un poco mejor que la precisión de 0.86 del modelo anterior. En la Figura 7 se pueden ver las medidas por cada clase, obteniendo resultados similares al modelo con hiperparámetros por defecto anterior, destacando también la clase M.

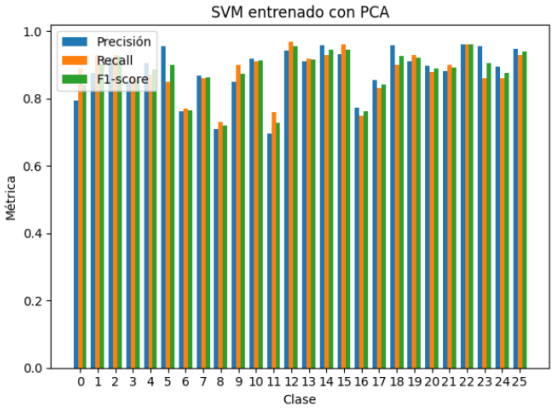

Figura 7: Medidas SVM con PCA

Estos resultados sugieren que reducir la dimensionalidad puede ser buena idea, sin embargo hay que tener cuidado con la eliminación de información que puede ser importante, en este caso al parecer los primeros 128 ejes tenían suficiente variabilidad como para no reducir la información y además se captaron patrones en la distribución de los datos, esto puede explicar la mejora en los resultados. Esto sumado a que ahorra costo computacional en el entrenamiento del modelo, es recomendable usar técnicas de reducción de dimensionalidad para este tipo de datos que representan figuras mediante píxeles.

III-C. SVM con embeddings

Al aplicar los embeddings descritos en la sección anterior, se llega a mejores resultados aún, con un accuracy total de 0.95 (lo que refuerza la tesis de que para este tipo de datos se debe intentar reducir la dimensionalidad). En la Figura 8 se pueden ver los resultados por cada clase y mejoran bastante, teniendo una distribución bastante pareja entre los rendimientos de tanto precisión como recall en cada clase, sin embargo hubieron más complicaciones para las clases 8 y 11, que representan las letras I y L, sin embargo al mirar los resultados de los modelos anteriores también se puede apreciar que en esta clase hubo un rendimiento bajo, sugiriendo que son las letras más difíciles de identificar, esto puede ser debido a que tienen una distribución de patrones parecida a el promedio de las demás letras, por ejemplo el conjunto de píxeles que representa una línea larga desde arriba a abajo está presente en muchas letras y esto puede llevar a confusión en el modelo.

Sin embargo, los embeddings mejoran los resultados más que PCA, esto sugiere que hacen mejor los datos identificando patrones, además hay que considerar que vienen de una red neuronal y por lo tanto se minimiza una función de error.

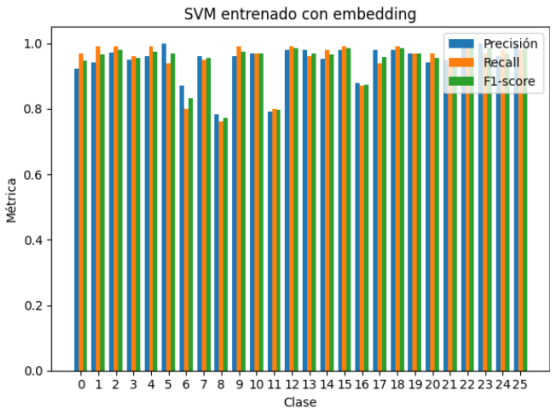

Figura 8: Medidas SVM con embeddings

III-D. Pruebas adicionales

A continuación se describen las pruebas adicionales en base a los resultados obtenidos con los modelos anteriores.

III-D1. Validación Cruzada: El mejor modelo resultante fue el producto de entrenar los datos de embeddings, por lo tanto se evaluó más a fondo este modelo con validación cruzada con 10 pliegues. El accuracy resultante promedio para cada conjunto de entrenamiento es de 0.96, un poco mayor que el accuracy total para el caso que se usó un conjunto de train y test, sugiriendo que el modelo es bueno y con cierto conjunto de datos puede mejorar aún más, y generalizar bien. El accuracy total por cada pliegue de entrenamiento, truncado al segundo decimal se muestra a continuación.

1. 0.97
2. 0.96
3. 0.96
4. 0.96
5. 0.97
6. 0.97
7. 0.96
8. 0.96
9. 0.96
10. 0.94

Los resultados son muy parejos, lo que indica que este modelo no tiende a generar clasificadores con alta varianza, además que el bias es muy bajo pues en promedio clasifica bien.

III-D2. Ensamblado con Bagging: Se usó el mismo modelo para hacer Bagging (dividir los datos de entrenamiento aleatoriamente para entrenar muchos modelos base del mismo tipo y combinar sus respuestas). Sin embargo hacer uso de esta técnica empeora los resultados de un accuracy total de 0.95 a 0.94, la diferencia no es muy grande pero sugiere que para este tipo de problema no es muy recomendable usar este tipo de ensamblado. Esto puede ser porque el modelo base rinde bien por sí mismo y combinar las soluciones de entrenarlo con distintos datos no representa una mejora, además que el conjunto de datos puede influir al no tener tantos outliers que sea importante tratar. La recomendación final sería usar esta técnica cuando el modelo base tenga no muy buenos resultados y alta varianza, este no es el caso.

IV. Conclusiones

Como resultado final de este experimento se obtuvieron varios modelos SVM con resultados distintos, se destaca el modelo entrenado con embeddings que produce un accuracy total de 0.95, frente al accuracy de 0.86 y 0.88 de los modelos anteriores. El primero de 0.86 es el modelo con hiperparámetros por defecto que de por sí clasifica relativamente bien pero teniendo errores marcados en varias clases. Al ajustar hiperparámetros se empeoró bastante el rendimiento, sugiriendo que el conjunto de datos es muy sensible a cambios en los parámetros y esto se puede deber a muchas cosas, como por ejemplo que la distribución de las variables cambie un poco la frontera de decisión de los hiperplanos o la influencia de los datos mediante los valores de C y gamma, y así cambien completamente los resultados. También se concluye que reducir la dimensionalidad mejora los resultados, tanto con PCA como con embeddings, que ilustra que estos datos tienen patrones que son identificables por estos algoritmos y no se pierde información.

Con respecto al mejor modelo se hizo un análisis adicional con K Cross Validation, reforzando su robustez al tener una buen accuracy promedio. Y debido a esto mismo se verificó también que técnicas de ensamblado como Bagging (que usalmente se usa cuando los modelos base son del mismo tipo, como este caso) no representan una mejora, cuando ya de por sí el modelo es robusto.

Como conclusión final se pudo aprender sobre el entrenamiento de modelos de aprenizaje automático supervisado, en este caso SVM, y cómo la forma de representar los datos, entrenar el modelo y la definición de hiperparámetros son importantes para el rendimiento de éste, conduciendo a experimentos con muchas pruebas para verificar el mejor modo de entrenar determinado clasificador.

Referencias

[1] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," *arXiv preprint arXiv:1702.05373*, Mar. 2017. [Online]. Available: https://arxiv.org/abs/1702.05373