Aprendizaje Automático - Tarea n°1

Vicente Mieres

***Resumen*—El presente informe busca analizar el rendimiento de modelos de SVM (Supporte Vector Machine) utilizando el subconjunto de datos QuickDraw-10, el cual contiene imágenes de dibujos hechos a mano, para 10 clases distintas. Se entrenaron dos modelos SVM utilizando diferentes tipos de kernels: uno con kernel RBF y otro con un kernel polinomial, y se experimentó con tres técnicas de generación de vectores de características: imágenes directamente convertidas en vectores, vectores obtenidos tras aplicar analisis de componentes principales (PCA), y vectores generados mediante UMAP. Los resultados se analizaron mediante métricas de precisión por clase y precisión total, mostrando las diferencias en el rendimiento de los modelos en función de la estrategia de reducción de dimensionalidad y tipo de kernel. Finalmente, se discuten las implicaciones de estos resultados para mejorar la precisión de los modelos SVM en tareas de clasificación de imágenes.**

***Index Terms*—Aprendizaje Automatico, Kernel, PCA, UMAP, SVM.**

I. Introducción

EL análisis de imágenes es fundamental en el área de aprendizaje automático, espec;ificamente en lo que respecta a la clasificación de estas. Así, en el presente informe tiene como objetivo la construcción de diferentes modelos SVM (Support Vector Machine) con ayuda de la librería Scikit-learn [1], utilizando el subconjunto de datos QuickDraw-10, el cual contiene cerca de diez mil dibujos hechos a mano, compuesto por 10 clases y representada por imágenes de 256x256 píxeles.

Particularmente, en lo que refiere a la construcción de modelos, se utilizarán dos tipos de kernel, RBF (Radial Basis Function) y Polinomial, y debido a la alta dimensionalidad de las imágenes, se implementarán tres enfoques en lo que respecta a vectores de características, en primer lugar se utilizarán las imágenes (2D) como un único vector, es decir aplanadas, y luego una reducción de dimensionalidad utilizando PCA (Análisis de Componentes Principales) y UMAP (Uniform Manifold Approximation and Projection). Obteniendo así, seis modelos a estudiar.

Finalmente, tras el entrenamiento y la evaluación de los modelos, se busca medir y comparar el rendimiento de esto utilizando metras de precisión, tanto del modelo como de cada una de las clases a predecir.

II. Desarrollo

En la siguiente sección se detalla el procedimiento utilizado para la correcta implementación del programa.

II-A. Preprocesamiento de Datos

En primer lugar, como paso fundamental en la construcción de los modelos, se realizó la carga de datos y el procesamiento inicial correspondiente. Este proceso comenzó con la lectura de los archivos **train.txt** y **test.txt**, con el objetivo de obtener cuatro listas separadas. De estas, dos contienen las rutas a las imágenes, mientras que las otras dos almacenan las etiquetas correspondientes. Esto se hizo mediante la función **load_image_paths_and_labels()**, la cual, dada la ruta a un archivo, retorna tanto las rutas a las imágenes, como las etiquetas de cada una en arreglos separados.

Luego utilizando la funcion **load_images()**, que recibe como parámetro el listado de *paths* a las imágenes, se realizó la lectura y carga de las mismas, con su respectivo *reshape(-1)*, es decir aplanadas a vectores de largo 65.536. Además, se utilizó la función **StandarScaler()** sobre las imágenes aplanadas, esto para trabajar con datos normalizados y mejorar el rendimiento [2].

Con lo anterior se tiene los datos para entrenar los modelos que utilizan las imágenes directamente (aplanadas), sin embargo, es necesario reducir la dimensionalidad de estás a 256 componentes, mediante dos técnicas diferentes, en particular con Análisis de Componentes Principales y *Uniform Manifold Approximation and Projection*, es decir, utilizando las funciones **PCA()** y **UMAP()**.

De esta forma, los conjuntos obtenidos se muestran en la Tabla I.

Tabla I: Conjuntos y Dimensiones

Nombre del Conjunto	Dimensión de los Arreglos
train_images_scaled	65536
test_images_scaled	65536
train_images_pca	256
test_images_pca	256
train_images_umap	256
test_images_umap	256

II-B. Búsqueda de Hiperparámetros

Dado que se deben construir y entrenar modelos SVM, es fundamental determinar los hiperparámetros adecuados, asegurándose de que sean apropiados para cada conjunto de datos. En particular se buscarán 2, *C* y *gamma*, donde C es el encargado de controlar el equilibrio entre maximizar el margen y la minimización del error, es decir, mientras mas pequeño el valor de C, el modelo permitirá más errores en los datos de entrenamiento. Mientras que *gamma* controla la influencia de cada punto a la hora de formar el hiperplano, por ejemplo, para un kernel RBF determina que tan cerca debe estar un punto para influenciar la decisión del modelo [3].

Ahora, para determinar que valores son los más adecuados en cada modelo, se hizo uso de la **GridSearchCV()**, la cual recibe como parámetros un estidor, en este caso SVM, una grilla con todos los valores probar de los distintos hiperparámetros, numero de *folds* para validación cruzada, y núcleos de CPU a utilizar para el proceso. Para el caso de *C*, y dado que debe ser estrictamente positivo, se utilizará una escala logarítmica para

cubrir una amplio rango de valores de manera mas eficiente. Por otro lado, el valor de *gamma* ofrece dos alternativas.

$$scale = \frac{1}{(n\_features * X.var())}$$

$$auto = \frac{1}{n\_features}$$

Sin embargo, para ampliar la búsqueda se considerarán dos valores extra, siendo estos: 0.01 y 0.001.

En lo que respecta a la elección del segundo kernel, se optó por un kernel polinomial (poly), esto para aprovechar el uso del parámetro *gamma*, puesto que tanto RBF como Poly se ven influidos por este. El resumen de valores utilizados en cada *grid search* se muestra en la Tabla II.

Tabla II: Parámetros GridSearch

Hiperparámetro	Valores
Kernel	RBF, Poly
C	0.1, 1, 10, 100
gamma	scale, auto, 0.01, 0.001

Finalmente, dado las magnitudes del problema (aproximadamente diez mil imágenes de entrenamiento) se optó por realizar el *grid search* para el 10 % de los datos, para cada uno de los modelos a construir (Imágenes raw, PCA y UMAP), y de esta forma agilizar el proceso de búsqueda. La elección del 10 % de los datos se hizo de forma aleatoria y equitativa por clase, esto mediante una función personalizada que retorna un porcentaje de los datos de cada clase, de tal forma que, utiliza la función **train_data_split()** para mantener la consistencia entre clases. Para más detalles revisar la función **sample_data_porcentage()** en el código adjunto. Los resultados de la busqueda de hiperparámetros se muestran en la Tabla III.

Tabla III: Resultados GridSearch

Kernel	Conjunto	C	gamma
RBF	RAW	10	scale
Poly	RAW	0.1	scale
RBF	PCA	10	scale
Poly	PCA	0.1	auto
RBF	UMAP	10	scale
Poly	UMAP	0.1	auto

II-C. Entrenamiento, Evaluación y obtención de Métricas

Una vez obtenidos los mejores hiperparámetros para cada modelo, estos fueron construidos. Estos mediante las funciones **fit()** y **predict()** para entrenamiento y evaluación respectivamente, mientras que para las métricas de rendimiento se hizo uso de la función **accuracy_score()** para obtener la precisión global del modelo y **classification_report()** para visualizar el *accuracy* por clase. Por último, con la función **confusion_matrix()** se dibujó la matríz de confusión asociada al modelo. Los resultados obtenidos se presentan en la siguiente sección.

III. Análisis de resultados

En primer lugar, cabe destacar que para el proceso en su totalidad se utilizó el *dataset* completo, es decir, las aproximadamente diez mil imágenes de entrenamiento y mil imágenes de evaluación. Esto bajo un procesador AMD Ryzen7 de la serie 5000, y 16GB de memoria RAM.

De esta forma, se presentan los resultado obtenidos para cada uno de los modelos. La Tabla IV muestra la precisión general de estos. Destacando un buen rendimiento para los modelos 1, 3 y 4, con un precisión entre 70 y 72 %. Mientras que el modelo 2, es decir kernel polinomial, con todas las imágenes sin procesar, posee un *accuracy* del 11 %, un resultado demasiado bajo. Por último, los modelos 5 y 6, también presentan un rendimiento bajo, este con aproximadamente 28 %. Las posibles razones de estos resultados son discutidas en las subsecciones siguientes.

Tabla IV: Resultados precisión por modelo

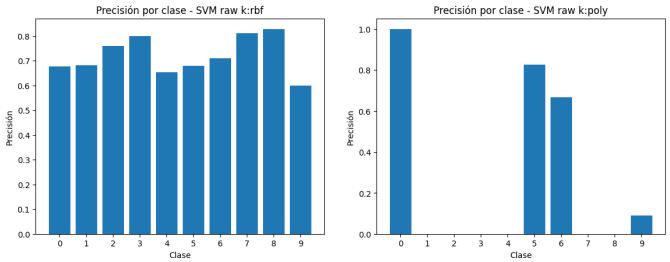
Modelo	Kernel	Conjunto	Precisión
1	RBF	RAW	0.7195540308747856
2	Poly	RAW	0.1097770154373928
3	RBF	PCA	0.7075471698113207
4	Poly	PCA	0.7161234991423671
5	RBF	UMAP	0.2770154373927959
6	Poly	UMAP	0.2684391080617496

III-A. Modelos con imágenes RAW

Reiteradas veces a lo largo del informe se menciona las palabras *imágenes RAW*, esto hace referencia a que se utilizan las imágenes directamente como vectores aplanados, sin reducción de dimensionalidad, lo cual para el caso particular de esta tarea, implica tener vectores de largo 65.356. De esta forma, el entrenamiento de ambos fue el más extenso, alcanzando tiempos de ejecución de un poco mas de cinco horas cada uno. De esta forma, podemos notar que el modelo que utiliza el kernel RBF es superior en todos los aspectos, tanto en precisión general como para cada clase. Para corroborar esta información, la Figura 1 muestra la precisión para cada una de las clases mediante gráficos de barra y la Figura 2 la matriz de confusión asociada a cada modelo. Con esto es posible afirmar que el modelo 1 presenta mejor generalización y rendimiento en todas las clases, mientras que el modelo 2, en su mayoría tiene clases que no puede predecir. Tras esto, se plantean dos mejoras posibles, la elección de un kernel más adecuado o una mejor búsqueda de hiperparámetros (*grid search*), considerando un rango mas amplio para *C* y/o *gamma*, e incluir el parámetro *degree*, correspondiente al grado de la función polinómica.

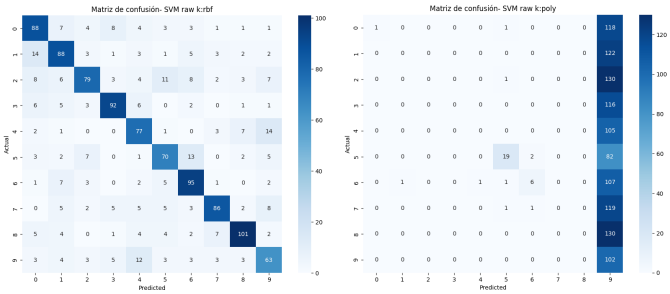
III-B. Modelos con imágenes PCA

Tal como se mencionó en la Sección II-A, los modelos 3 y 4 fueron entrenados con imágenes a las que se les aplicó reducción de dimensionalidad, específicamente mediante Análisis de Componentes Principales (PCA), presentándose así dos de los tres mejores resultados obtenidos en toda la experiencia, con precisiones de 71 y 72 % respectivamente. Además, un detalle no mencionado en la Sección III-A, es que las clases 4 y 9



(a) SVM Raw RBF (b) SVM Raw Poly

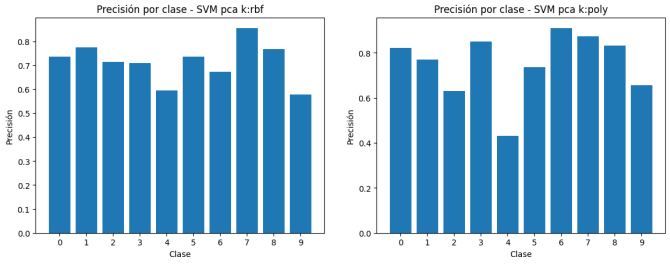
Figura 1: Precisión por Clase Modelos RAW



(a) Matriz de Confusión SVM Raw RBF (b) Matriz de Confusión SVM Raw Poly

Figura 2: Matriz de Confusión Modelos RAW

presentan los menores valores en cuando a precisión, lo cual se vuelve reiterado para los modelos con PCA, dando cuenta que el problema es intrínseco de la clase, y no de los modelos como tal, Las Figuras 3 y 4 muestran esta información. Por último, si bien se presenta una mejora de rendimiento, especialmente en el modelo con kernel polinomial, los resultados no son excesivamente altos, lo cual sugiere que, por un lado PCA ayuda a simplificar el problema, pero también puede estar eliminando información importante de los datos, lo cual es algo a tener en consideración para futuras experiencias, específicamente el número de componentes.

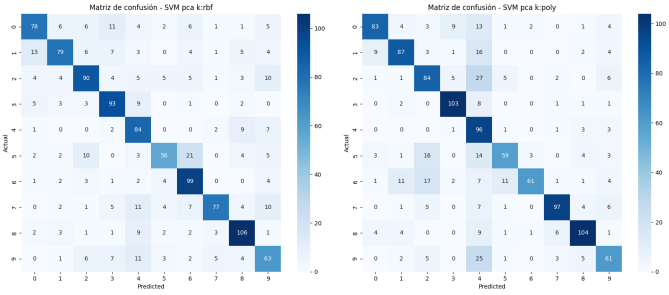


(a) SVM PCA RBF (b) SVM PCA Poly

Figura 3: Precisión por Clase Modelos PCA

III-C. Modelos con imágenes UMAP

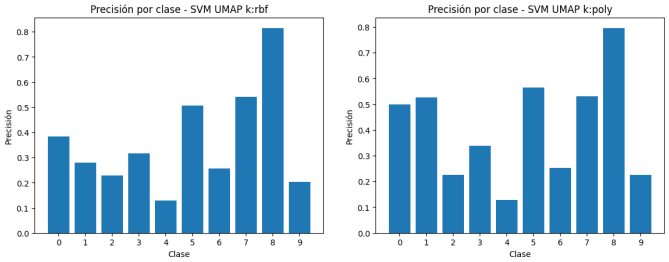
En lo que respecta a los modelos 5 y 6, las Figuras 5 y 6 presentan los correspondientes, nuevamente, a la precisión por clase y la matriz de confusión. En este caso, con resultados realmente desalentadores en cuanto al funcionamiento



(a) Matriz de Confusión SVM PCA RBF (b) Matriz de Confusión SVM PCA Poly

Figura 4: Matriz de Confusión Modelos PCA

de aplicar UMAP a los datos, según la Tabla IV, se tiene resultados de 28 y 27 % en *accuracy*, por lo que es posible afirmar que se está perdiendo información relevante al reducir la dimensionalidad. Además, es posible verificar que tanto la clase 4 como la 9 poseen un desempeño peor en comparación a las demás, esto puede ser por diferentes factores, entre ellos, poca variabilidad dentro de la misma clase, es decir que se tenga imágenes muy parecidas entre si, o desbalance entre clases, sin embargo este último se descarta, pues todas las clases tiene la misma cantidad de ejemplares. Finalmente, se recomienda realizar un ajuste a los hiperparámetros del modelo, principalmente a *n_neighbors*, para así tener mayor control sobre el numero de vecinos que UMAP considera a la hora de reducir las dimensiones, teniendo en cuenta que un valor bajo podría enfocar demasiado el modelo en detalles locales y un valor alto perder demasiada información.



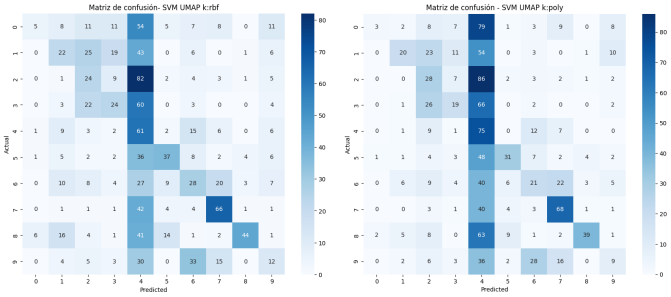
(a) SVM UMAP RBF (b) SVM UMAP Poly

Figura 5: Precisión por Clase Modelos UMAP

IV. Conclusión

En este trabajo se evaluó el rendimiento de modelos SVM con distintos kernels y técnicas de preprocesamiento para la clasificación de imágenes. El análisis de los resultados reveló que los modelos con kernel RBF aplicados a imágenes sin procesar, así como ambos kernels combinados con PCA, alcanzaron precisiones destacables entre 70 % y 72 %. Por el contrario, el modelo con kernel polinomial sobre imágenes sin procesar mostró un desempeño notablemente deficiente (11 %), mientras que los modelos basados en UMAP presentaron resultados insatisfactorios (aproximadamente 28 %).

La técnica PCA demostró ser efectiva al mantener un rendimiento comparable al modelo RAW con kernel RBF,



(a) Matriz de Confusión SVM UMAP RBF

(b) Matriz de Confusión SVM UMAP Poly

Figura 6: Matriz de Confusión Modelos UMAP

pero reduciendo drásticamente la dimensionalidad de 65.536 a 256 componentes. Esto sugiere que PCA preservó información esencial de las imágenes, aunque posiblemente eliminó características que podrían haber mejorado aún más el desempeño. Por otro lado, UMAP no logró conservar la estructura necesaria para una clasificación efectiva en este contexto, evidenciando la importancia de un ajuste meticuloso de sus hiperparámetros.

Para futuras experiencias, es recomendado trabajar rangos más amplios de hiperparámetros, especialmente para el kernel polinomial, experimentar con diferentes configuraciones de PCA y buscar otras alternativas en cuanto a kernel se refiere.

Por ultimo, este estudio demuestra que los modelos SVM con kernel RBF, tanto con imágenes sin procesar como con preprocesamiento mediante PCA, ofrecen resultados prometedores para la clasificación de dibujos del conjunto QuickDraw-10. La selección adecuada del kernel y la técnica de reducción de dimensionalidad resultan determinantes para obtener modelos con rendimiento computacional eficiente y precisión aceptable.

REFERENCIAS

[1] S. developers, “sklearn.svm.svc,” 2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[2] Scikit-learn, “sklearn.preprocessing.standardscaler,” 2025. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[3] C. Sampaio, “Understanding svm hyperparameters,” 2023. [Online]. Available: https://stackabuse.com/understanding-svm-hyperparameters/