Tecnológico de Monterrey

Instituto Tecnológico de Estudios Superiores de Monterrey

Campus Estado de México

***Inteligencia Artificial Avanzada para la Ciencia***

# Análisis sobre el desempeño del modelo: Support Vector Machine

**Profesor**

Jorge Adolfo Ramírez Uresti

**Alumna**

Natalia Duarte Corzo

**Matrícula**

A01745482

Septiembre 2023

## I. Introducción

Hoy en día, el aprendizaje automático es una rama de la inteligencia artificial que ha tenido un rápido crecimiento gracias a la amplia aplicación que tiene en distintas áreas de nuestra vida. Así pues, una de sus aplicaciones es la creación de modelos de clasificación, los cuales, permiten que a partir de una base de datos las máquinas puedan identificar patrones y, con estos descubrimientos sean capaces de realizar predicciones con nuevos datos.

A lo largo de la clase Inteligencia Artificial Avanzada para la Ciencia, se han visto diversos modelos de clasificación como lo son, la regresión logística, los árboles de decisión, las redes neuronales, entre otras. Así mismo, con el fin de poner en práctica los temas vistos en clase, se estará trabajando con una base de datos, para generar un modelo de clasificación capaz de realizar predicciones con buena precisión y, en caso de ser necesario, utilizar técnicas para mejorar la calidad de los resultados dados por dicho modelo.

## II. Justificación de la selección del dataset

La base de datos seleccionada para esta actividad es de pacientes con y sin hipertensión; esta cuenta con características de cada paciente, como su sexo, edad, niveles de colesterol, su presión arterial, entre otras. Primeramente, la selección de los datos se hizo buscando una aplicación de la inteligencia artificial a un tema de salud, como lo es la hipertensión, la cual, es uno de los principales factores de riesgo para enfermedades cardiovasculares, por lo que me parece un tema en el que un modelo de clasificación puede llegar a ser de gran ayuda para identificar esta condición en pacientes.

La base de datos se consiguió de la plataforma Kaggle, la cual cuenta con miles de bases de datos de diversos temas, incluyendo temas de salud, las cuales se pueden acceder de forma gratuita. Una vez descargada la base de datos, se aseguró que esta contara con datos suficientes y, relevantes para realizar el modelo. Para ello, se identificó que esta contaba con 26083 filas de pacientes registrados, lo cual es una buena cantidad de datos para que el modelo aprenda; de igual manera, se revisaron los datos faltantes del dataset, siendo este solamente uno y, finalmente se identificó que todas las características de los pacientes, o bien, los valores de las columnas de la base

son numéricas, por lo que no es necesario convertir ninguna de categórica a numérica para el uso del modelo.

## III. Separación de datos

Contando con la base de datos de hipertensión ya limpia y, sin ningún dato faltante, se realizó la separación de datos en tres grupos. Primeramente los datos de entrenamiento, siendo estos un 80% de nuestro dataset, después los datos de prueba siendo un 10% y, finalmente el último 10% fueron los datos de validación. Dicha separación se realizó con la función *train_test_split*, como se muestra en la Figura 1, dónde se divide primeramente el dataset en 80% de entrenamiento y 20% de prueba, y, después el set de prueba se divide en dos, la primera mitad para ser de prueba y la otra para valoración.

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5, random_state=42)
```

Figura 1. Separación de datos de entrenamiento (80%), de prueba (10%) y valoración (10%).

Con el fin de identificar cómo es que se realiza la separación de los datos, se realizó un gráfico de barras, como se muestra en la Figura 2 y, un gráfico de dispersión, como se muestra en la Figura 3. En el primer gráfico, se puede identificar que los datos de prueba y de valoración tienen el mismo tamaño, mientras que los de entrenamiento son el 80% del dataset, lo cual se puede visualizar ya que la última barra graficada son todos los datos de hipertensión. Por otro lado, el gráfico de dispersión representa la columna de 'Age' del dataset, la cual, como todo el conjunto tiene el mismo porcentaje dividido en los tres grupos, lo cual se puede visualizar identificando que los puntos azules con los de entrenamiento, los verdes son los de prueba y, los amarillos los de validación, siendo en total 4000 datos como se muestra en el eje y, que indica el número todal de pacientes (con el ID de cada uno).

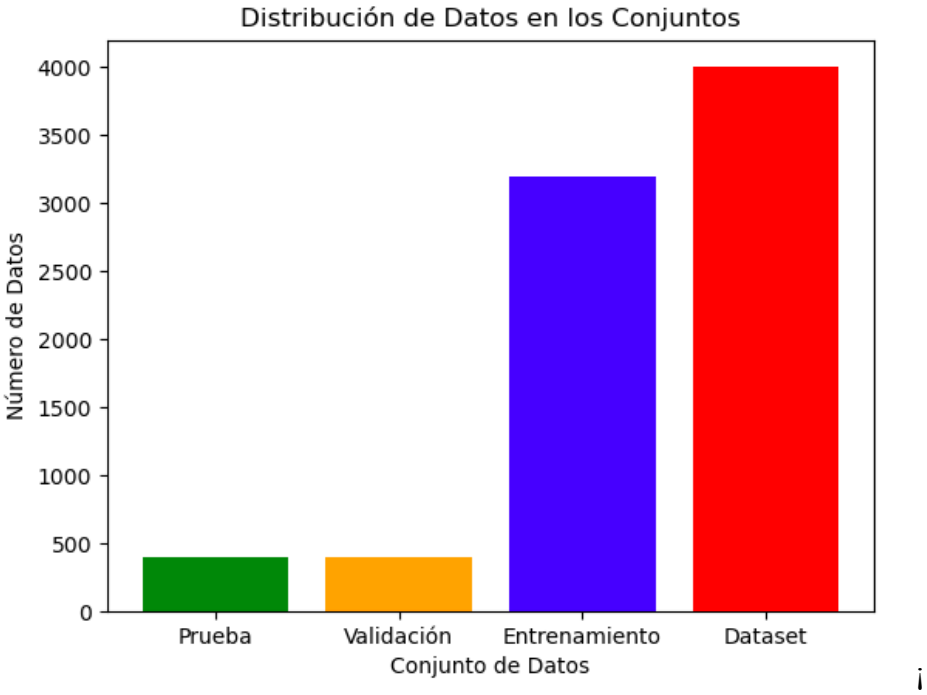


Figura 2. Gráfico de barras de separación de datos.

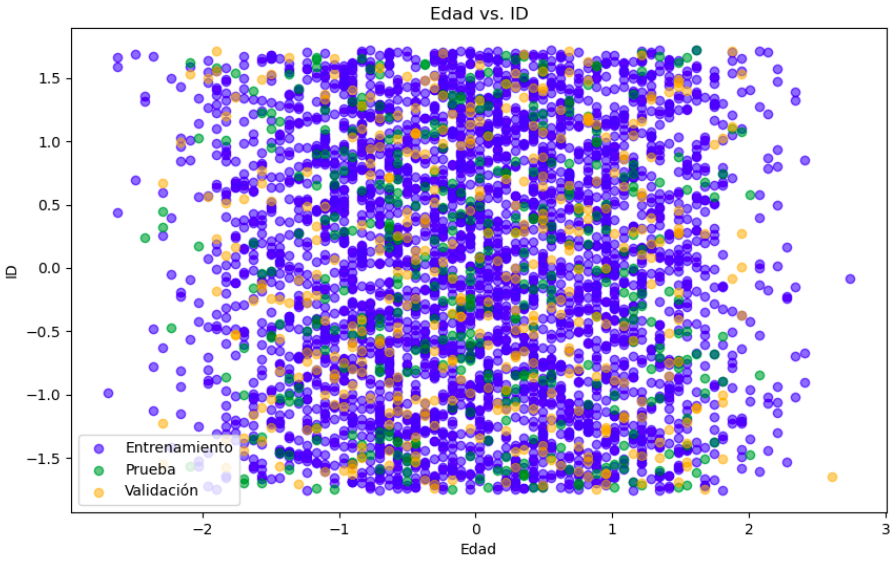


Figura 3. Gráfico de dispersión de separación de datos.

## IV. Modelo

Ahora bien, el modelo de clasificación que se eligió utilizar para nuestros datos fue el de Support Vector Machine. Este modelo se eligió debido a que es ampliamente

reconocido por su eficacia en clasificación binaria, la cual es precisamente nuestro caso, pues se quiere lograr clasificar a los pacientes en pacientes hipertensos o no hipertensos. De igual manera, el SVM tiene un parámetro de regularización C que permite que se pueda controlar el sobreajuste de los datos y así se pueda mejorar la generalización del modelo. Así mismo, este modelo puede manejar datos que no son lineales y tiene un resistencia a la dimensionalidad alta, la cual no está tan presente en nuestro dataset, debido a que se cuenta con tal solo 14 columnas, sin embargo, ha sido una de las razones por las que el SVM se ha utilizado tanto para aplicaciones en salud, como lo es nuestro caso.

## V. Diagnóstico del modelo

Así pues, como se muestra en la Figura 4 utilizando el modelo de SVM con sus valores default, siendo estos la función de kernel lineal y, 1 el parámetro de regularización (C) que controla la cantidad de error tolerada por el modelo se realizó el entrenamiento con los datos y, se realizaron predicciones tanto para los datos de prueba como para los datos de validación.

```python
#Puedes ajustar el tipo de kernel y otros hiperparámetros
svm_model = SVC(kernel='linear', C=1.0)

#Se realiza el entrenamiento con los datos correspondientes
svm_model.fit(X_train, y_train)

#Se hacen predicciones en X_test (nuestros datos de prueba)
y_pred_test = svm_model.predict(X_test)

#Se hacen predicciones en X_test (nuestros datos de validación)
y_pred_val = svm_model.predict(X_val)
```

Figura 4. Código para implementación del modelo SVM.

A continuación, se mostrarán las medidas y gráficas que se utilizaron para evaluar este modelo de SVM, con el fin de ver qué tan óptimos fueron sus resultados y, comparar la precisión en los datos de prueba y de validación.

### a. Métricas

Las métricas que se obtuvieron con el modelo default de Support Vector Machine, se muestran en la Tabla I. Tomando en cuenta los resultados, podemos concluir que, con base al accuracy el 82.25% de las muestras se clasificaron de forma correcta, lo cual nos dice que el modelo tuvo un buen rendimiento. Al mismo tiempo, la precisión nos dice que el 80.16% de las predicciones de la clase positiva fueron correctas. Continuando, el recall nos hace ver que el 91.15% de los pacientes con hipertensión, tuvieron un resultado correcto. Finalmente, el F1 score nos indica que al ser de 85.30% se tiene un buen equilibrio entre la precisión y el recall. Gracias a todo ello podemos concluir que nuestro modelo es bastante bueno, sin embargo, tiene oportunidades de mejora, lo cual se estará analizando más adelante.

Tabla I. Resultados de SVM Original.

| **Métrica** | **Valor** |
|---|---|
| Accuracy | 0.8225 |
| Precision | 0.8016 |
| Recall | 0.9115 |
| F1-score | 0.8530 |

b. Matriz de confusión

Ahora bien, con lo que nos dice la matriz de confusión sabemos que con nuestro modelo se clasificaron correctamente 123 muestras como clase negativa, sin embargo 51 se clasificaron incorrectamente como positivas. De igual forma se clasificaron correctamente 206 muestras como clase positiva, pero 20 muestras de la clase positiva se clasificaron incorrectamente como clase negativa. Tomando en cuenta estos resultados, se puede decir que el modelo es bastante bueno, ya que, a pesar de que 71 muestras se clasificaron incorrectamente, solamente 20 de estas fueron clasificadas como pacientes sin hipertensión, cuando en realidad si tenían, lo cual pone en riesgo la salud de los pacientes; no obstante estos pacientes son únicamente el 5% del 100% de las muestras, lo cual es un porcentaje bastante bajo.

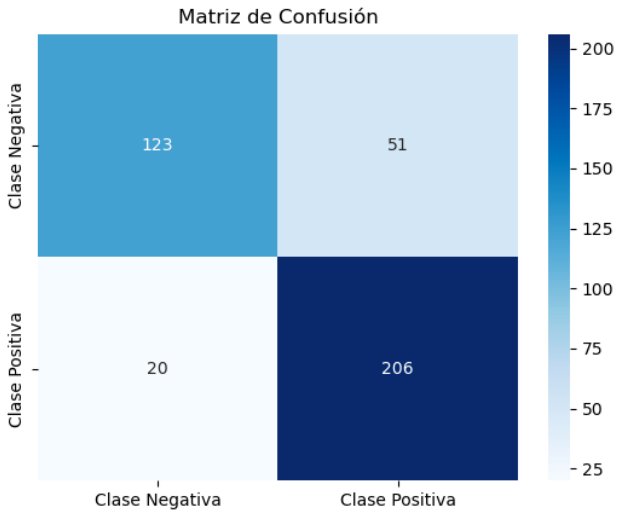


Figura 5. Matriz de confusión.

c. Grado de bias y varianza

A continuación, con el código de la Figura 6 se muestra cómo se calcularon el sesgo y la varianza utilizando la función *bias_variance_decomp*. El cálculo del sesgo, con el error cuadrático medio (MSE), nos permite conocer la diferencia que existe entre las predicciones realizadas con el modelo y los valores reales en el dataset, mientras que la varianza nos dice la sensibilidad de nuestro modelo ante las fluctuaciones en los datos de entrenamiento.

```python
_, bias, var = bias_variance_decomp(svm_model, X_train1, y_train1, X_test1,
y_test1, loss='mse', num_rounds=20, random_seed=1)
```

Figura 6. Código para cálculo de varianza y sesgo.

Tabla II. Resultados de sesgo y varianza del SVM Original en los datos de prueba

| Métrica | Valor |
|---|---|
| Bias | 0.1308 |
| Variance | 0.0189 |

Tabla III. Resultados de sesgo y varianza del SVM Original en los datos de validación.

| Métrica | Valor |
|---|---|
| Bias | 0.1696 |
| Variance | 0.0124 |

A continuación, tomando en cuenta los valores de sesgo y de varianza en los datos de prueba y de validación y, realizando un gráfico de barras con estos, es posible identificar que son bastantes cercanos unos a los otros, por lo que el modelo nos indica lo mismo en ambos casos, pues los cuatro valores están cercanos a 0, por lo que son bajos. En resumen, existe un buen equilibrio entre bias y varianza, ya que un bias bajo, sugiere que nuestro modelo hace predicciones precisas en los datos de prueba, mientras que una varianza también es baja indica que se generalizan bien los datos nuevos. Así pues, al igual que las métricas anteriores, el sesgo y la varianza nos ayudan a saber que el modelo tiene un buen rendimiento y un buen ajuste.

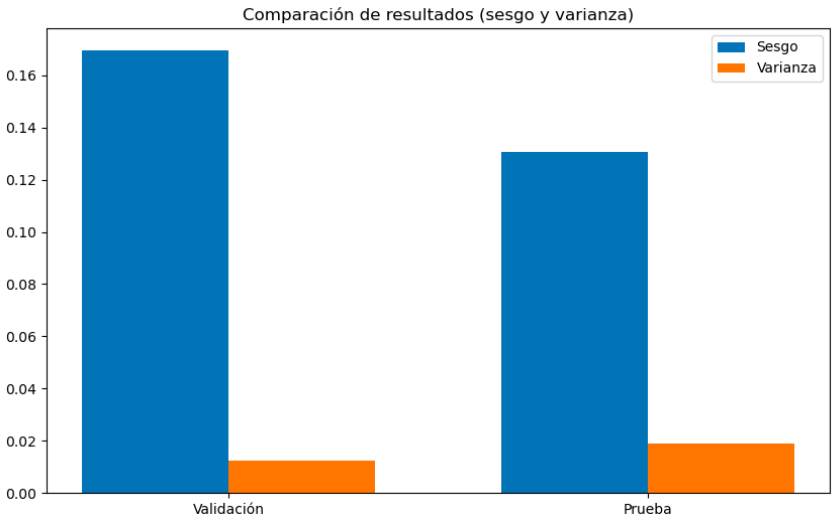


Figura 7. Comparativo de varianza y sesgo de los datos de validación y de prueba.

d. Nivel de ajuste

Otra función que viene con el cálculo de la varianza y del sesgo es el conocer si nuestro modelo tiene overfitting, underfitting, o esta bien ajustado. Se conoce que, un

cuando se obtienen un bias alto y una varianza baja el modelo tiende a presentar un *underfitting*, mientras que un modelo cuando se calculan un bias bajo y una varianza alta se dice que hay *overfitting*; por esta razón un buen modelo, con un equilibrio y ajuste apropiado presenta un sesgo y una varianza bajos, por lo que, con base a los cuatro valores calculados anteriormente, sabemos que el modelo tiene un buen rendimiento con datos desconocidos.

Con el fin de poder visualizar este concepto, se realizó una gráfica de la curva de aprendizaje, la cual se puede observar en la Figura 9 y muestra cómo el rendimiento del modelo va mejorando conforme se incrementa la cantidad de datos de entrenamiento. Esta se realiza con el código que se de la Figura 8, que usa la función learning_curve.

```python
from sklearn.model_selection import learning_curve

# Calculamos las curvas de aprendizaje utilizando learning_curve con nuestro modelo entrenado svm_model, X_train y y_train y, la lista de tamaños de entrenamiento
train_sizes, train_scores, test_scores = learning_curve(svm_model, X_train, y_train, train_sizes=[250,500,750,1000,1250,1500,1750,2000,2500,2560])
#Se grafican las curvas de aprendizaje, la de entrenamiento en rojo y la de prueba en azul
plt.plot(train_sizes, np.mean(train_scores, axis=1), color='red', label='train')
plt.plot(train_sizes, np.mean(test_scores, axis=1), color='blue', label='test')
#Se agrega una leyenda al gráfico
plt.legend()
#Damos los títulos para los ejes x y y
plt.xlabel('Tamaño del Conjunto de Entrenamiento')
plt.ylabel('Puntuación Media')
#Se define el título del gráfico
plt.title('Curva de Aprendizaje')
#Se muestra el gráfico
plt.show()
```

Figura 8. Código para graficar la curva de aprendizaje.

La curva roja que es la de entrenamiento muestra el rendimiento del modelo a medida que aumenta el tamaño del conjunto. Inicialmente, cuando se entrena con menos datos hay un overfitting ya que la curva de entrenamiento es alta y la de prueba baja. No obstante, a medida que se agregan más datos el modelo tiene más información para

generalizar.En cuanto a la curva de prueba vemos que cuando el tamaño del conjunto de entrenamiento es pequeño, el modelo no generaliza bien y, por lo tanto, el rendimiento es bajo, sin embargo, a medida que agregamos más datos, el rendimiento mejora, ya que él tiene más información para aprender y generalizar. Como vemos en la gráfica ya que las dos curvas convergen a un rendimiento aceptable a medida que se aumenta el tamaño del conjunto, lo cual, nuevamente, indica un buen equilibrio entre sesgo y varianza, y que el modelo está funcionando bien

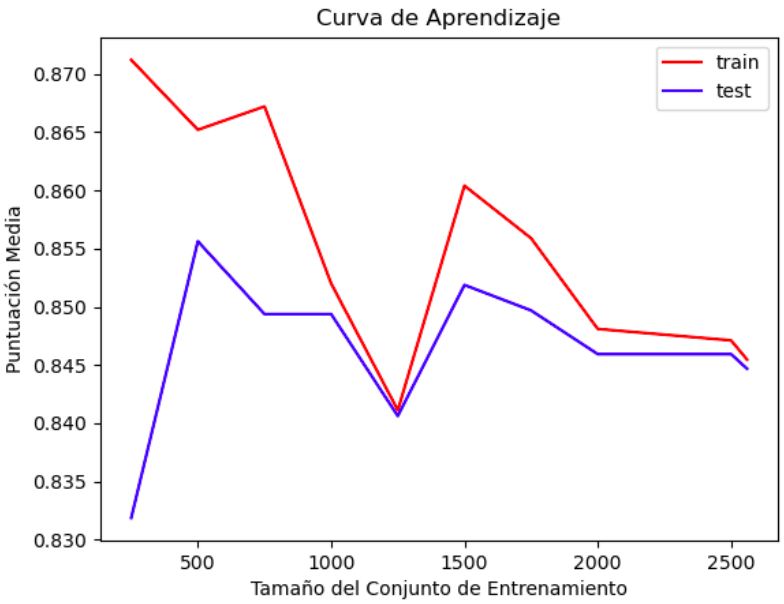


Figura 9. Curva de aprendizaje.

## VI. Técnicas de ajuste de parámetros para la mejora

Como se mencionó anteriormente, ya que se obtuvo una buena precisión en el modelo original de SVM, la cual fue de 82.25%, mas no fue un resultado tan bajo, significa que existen áreas de mejora para el modelo, lo cual puede hacerse a través del uso de técnicas de regularización, el ajustes de parámetros. Así pues, para ello se realizaron tres modificaciones los cuales nos llevaron a cambios de un escenario más favorable.

a. Hiper Parámetro: Peso de clase

   Primeramente, como nos muestra la Figura 10, se ajustó el modelo inicial añadiendo el hiper parámetro de class_weight que incluye la función, la cual permite que nuestro modelo asigne de manera automática pesos inversamente proporcionales. Este cambio se agregó modificando el modelo, con tan solo

añadirle dicho parámetro como balanced, para que se pudieran manejar conjuntos de datos desequilibrados. Con ello nuestro modelo asigna de manera automática pesos inversamente proporcionales a las frecuencias de las clases en los datos de entrenamiento, lo cual ayudará a un mejor desempeño si es que tenemos clases minoritarias.

```python
svm_model_class_weight = SVC(kernel='linear', C=1.0, class_weight='balanced')
```

Figura 10. Código para agregar el parámetro de class_weight al modelo inicial.

Realizando estos cambios se calcularon las métricas, las cuales, presentan un aumento en la precisión con los datos de prueba, sin embargo el recall y el F1-score disminuyeron, por lo que intentaremos usar otra técnica para seguir mejorando.

Tabla IV. Resultados de SVM con class_weight.

| Métrica | Valor |
|---|---|
| Accuracy | 0.8200 |
| Precisión | 0.8080 |
| Recall | 0.8938 |
| F1-score | 0.8487 |

b. Hiper Parámetro: Kernel

El siguiente cambio es que en lugar de utilizar un kernel lineal, usaremos el kernel rbf (radial basis function), el cual se usa en problemas de clasificación y regresión y es capaz de transformar los datos de entrada en un espacio de características de mayor dimensión, para así poder modelar relaciones no lineales.

```python
svm_model_rbf = SVC(kernel='rbf', C=1.0)
```

Figura 11. Código para agregar el parámetro de kernel rbf al modelo inicial.

Con este nuevo cambio de parámetros podemos ver que nuestro modelo con el hiper parámetro kernel como rbf ha aumentado en todas las métricas, lo cual significa que para nuestro modelo es mejor el kernel de Radial Basis Function; no obstante, aún hay oportunidad de tener un modelo más preciso, por lo que seguimos intentando mejorar el modelo SVM.

Tabla V. Resultados de SVM con kernel Radial Basis Function.

| Métrica | Valor |
|---|---|
| Accuracy | 0.9300 |
| Precisión | 0.9231 |
| Recall | 0.9558 |
| F1-score | 0.9391 |

c. GridSearch

La última técnica para ver si se puede hacer una mejoramiento en el modelo, es una técnica de regularización, llamada *GridSearch*. Como se muestra en la Figura 11, importamos *GridSearchCV* para automatizar el proceso de búsqueda de hiper parámetros óptimos para nuestro modelo elegido, lo cual hemos estado haciendo de manera manual desde el inicio e incluso en los mejoramientos anteriores. Esta técnica nos ayudará a hallar los hier parámetros más aptos para la obtención del resultado más óptimo de nuestro modelo de hipertensión.

```python
from sklearn.model_selection import GridSearchCV
```

```
#Tomando en cuanta que nuestro modelo es SVM definimos los hiperparámetros a
buscar, de los cuales queremos identificar cuál es el mejor
hiperparametros = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly'],
'gamma': ['scale', 'auto']}

#Creamos el modelo SVM
svm_model_grid = SVC()

#Se aplica la función Grid Search
grid_search = GridSearchCV(svm_model_grid, hiperparametros, cv=5)

#Se realiza el entrenamiento con los datos correspondientes
grid_search.fit(X_train, y_train)

#Extraemos los mejores hiperparámetros para el modelo SVM
best_params = grid_search.best_params_

#Se entrena el modelo con los mejores hiperparámetros obtenidos con Grid
Search
svm_model_final = SVC(**best_params)

#Se realiza el entrenamiento con los datos correspondientes
svm_model_final.fit(X_train, y_train)

#Se hacen predicciones en los datos de prueba con el mejor modelo
y_pred_test = svm_model_final.predict(X_test)

#Se hacen predicciones en los datos de validación con el mejor modelo
y_pred_val = svm_model_final.predict(X_val)
```

Figura 12. Código para usar Grid Search.

Tabla VI. Resultados de SVM usando Grid Search.

| Métrica | Valor |
|---|---|
| Accuracy | 0.9775 |
| Precisión | 0.9657 |
| Recall | 0.9956 |
| F1-score | 0.9804 |

Como se puede ver en los resultados, utilizando esta técnica es cómo se obtuvieron los mejores resultados del modelo SVM. Gracias a el uso de *GridSearch* se logró identificar que los mejores parámetros para este modelo al utilizar los datos de hipertensión, son C:10, gamma:scale y, kernel:rbf como se había hecho en el paso anterior.

Finalmente, ya que se ha obtenido el mejor resultado de nuestro modelo, volvemos a generar una matriz de confusión, en la que se pueden ver resultados gratificantes para la clasificación de pacientes con y sin hipertensión. Se han clasificado correctamente 166 muestras como clase negativa y 225 muestras como clase positiva, mientras que solamente 8 pacientes se clasificaron incorrectamente como positivos y únicamente 1 paciente con hipertensión se clasifica erróneamente como negativa. Estos resultados con significativamente destinos a los segundos, por lo que nuestro modelo asegura que únicamente 9 pacientes de 400, son los que llegaron a ser clasificados incorrectamente, siendo solamente uno de mayor riesgo, dado que aunque tuviese hipertensión se no está en la clase correcta.

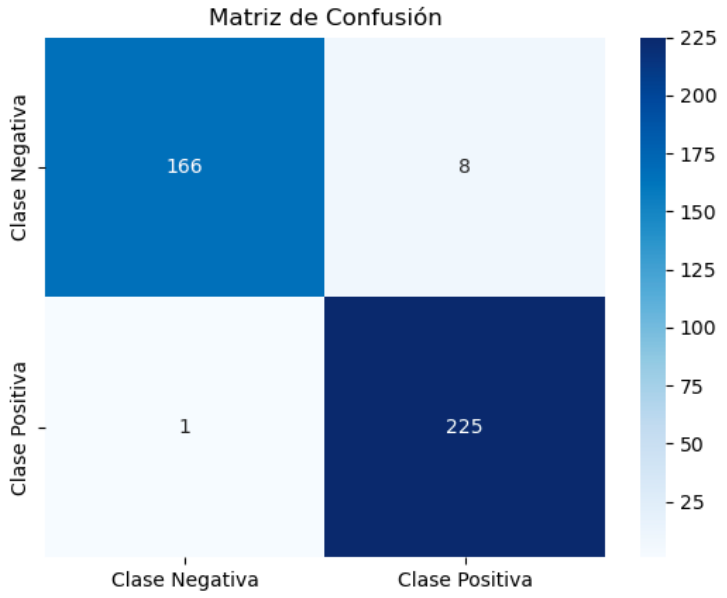


Figura 13. Matriz de confusión del modelo SVM final

## VII. Conclusión

A lo largo de este proyecto, se generó un modelo de Support Vector Machine, con el fin de predecir, con base a las características de los pacientes, si estos tenían o no hipertensión. Así mismo, a lo largo de este se hizo uso de distintas técnicas y parámetros, con el fin de obtener el resultado más óptimo, el cual llegó a tener un 96% de precisión, llegando a ser un modelo extremadamente confiable y eficaz, inclusive desde la primera corrida del modelo, utilizando simplemente los hiper parámetros de default.

Este proyecto demostró que el uso de modelos de Machine Learning, como lo es el SVM, junto con la optimización de hiper parámetros, a la cual se llegó con el uso de Grid Search y, la gestión del desequilibrio de clases, puede mejorar significativamente la capacidad de predicción de un problema de clasificación, en este caso de la hipertensión en pacientes. Así pues, conocer este tipo de aplicaciones es indispensable para los científicos de datos, pues la aplicación en la medicina de la ciencia de datos llega a tener un impacto positivo en la detección temprana de los pacientes con riesgo de hipertensión.