

Implementacion de clasificadores de Scikit-learn en la base de datos Glass.

Echeverry Luis (2420161006)
2420161006@estudiantesunibague.edu.co
Inteligencia Artificial
Programa de Ingeniería Electrónica
Facultad de Ingeniería
Universidad de Ibagué
Ibagué - Tolima
Semestre 2020A
Presentado al ingeniero Jose Armando Fernández Gallego

1. Resumen

Para esta entrega se tiene como objetivo general, continuar con la implementación de los algoritmos de clasificación que se vieron durante el curso en la base de datos seleccionada. Como objetivo específico se tiene la separación de al menos 3 clases presentes en la base de datos, con dos de sus características, haciendo uso de las diferentes herramientas que nos ofrece la librería Scikit-learn de Python. El desempeño de los diferentes algoritmos se evaluará, teniendo en cuenta el porcentaje de clasificación y la gráfica de las regiones de separación, lo que permitirá de esta manera sacar algunas conclusiones importantes acerca de los diferentes algoritmos presentes en este informe. Los clasificadores que se desean implementar son Perceptron, Perceptron + regresión logística, SVM + linear, SVM + RBF, DT, RF. Los cuales están contenidos en la librería Scikit-learn mencionada previamente.

2. Desarrollo

Como se mencionó previamente, es indispensable describir la base de datos seleccionada. Glass Identification Data Set es una recopilación de las características de fragmentos de vidrios obtenidos en escenas de crímenes, realizada por el departamento de ciencias forenses de los Estados Unidos. Está compuesta por un número total de 214 muestras de 6 tipos diferentes de vidrios. Las cuales, a su vez, tienen los valores correspondientes del contenido de óxido de Sodio (Na), Magnesio (Mg), Aluminio (Al), Bario (Ba), Silicón (Si), Potasio (K), Calcio (Ca) y Hierro (Fe). Además del índice de refracción y un número único para cada muestra tomada, parámetros que fueron evaluados al momento de realizar las pruebas necesarias para crear cada una de las muestras.

La razón de la creación de esta base de datos, fue el diseño de un algoritmo que permita clasificar de forma rápida y segura el tipo de vidrio encontrado en una escena del crimen o de un accidente, para que este pueda llegar a servir como evidencia, de ser necesario.

Para este caso en específico se decidió hacer uso del Aluminio (Al) y Sodio (Na) ya que como se puede observar en figura 1, son las dos clases que más se ajustan a una separación lineal entre las tres clases de vidrios a seleccionar.

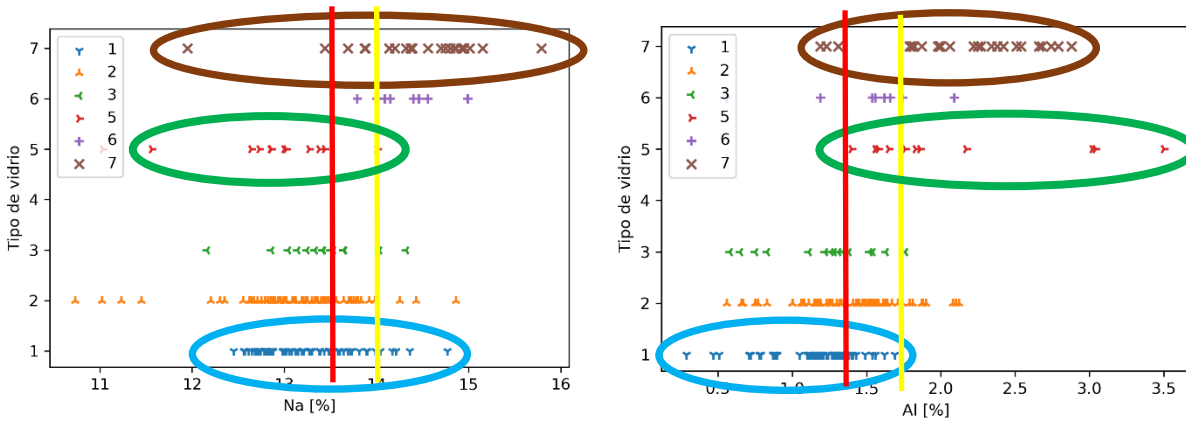


Figura 1. Grafica de Na y Al en cada uno de los 6 tipos de vidrios.

Como segunda, tenemos que los algoritmos que se desean aplicar son el perceptrón, el perceptrón más la regresión logística, máquinas de soporte vectorial (SVM) de forma lineal, máquinas de soporte vectorial (SVM) con kernel de función de base radial, los árboles de decisión y los random forests. Cada uno de estos se encuentra dentro de la librería Scikit-learn de Python, facilitando así su implementación.

3. Diagrama de flujo

A continuación, se presenta el diagrama de flujo que corresponde a la implementación de cada uno de los clasificadores mencionados anteriormente, especificando sus correspondientes salidas y entradas.

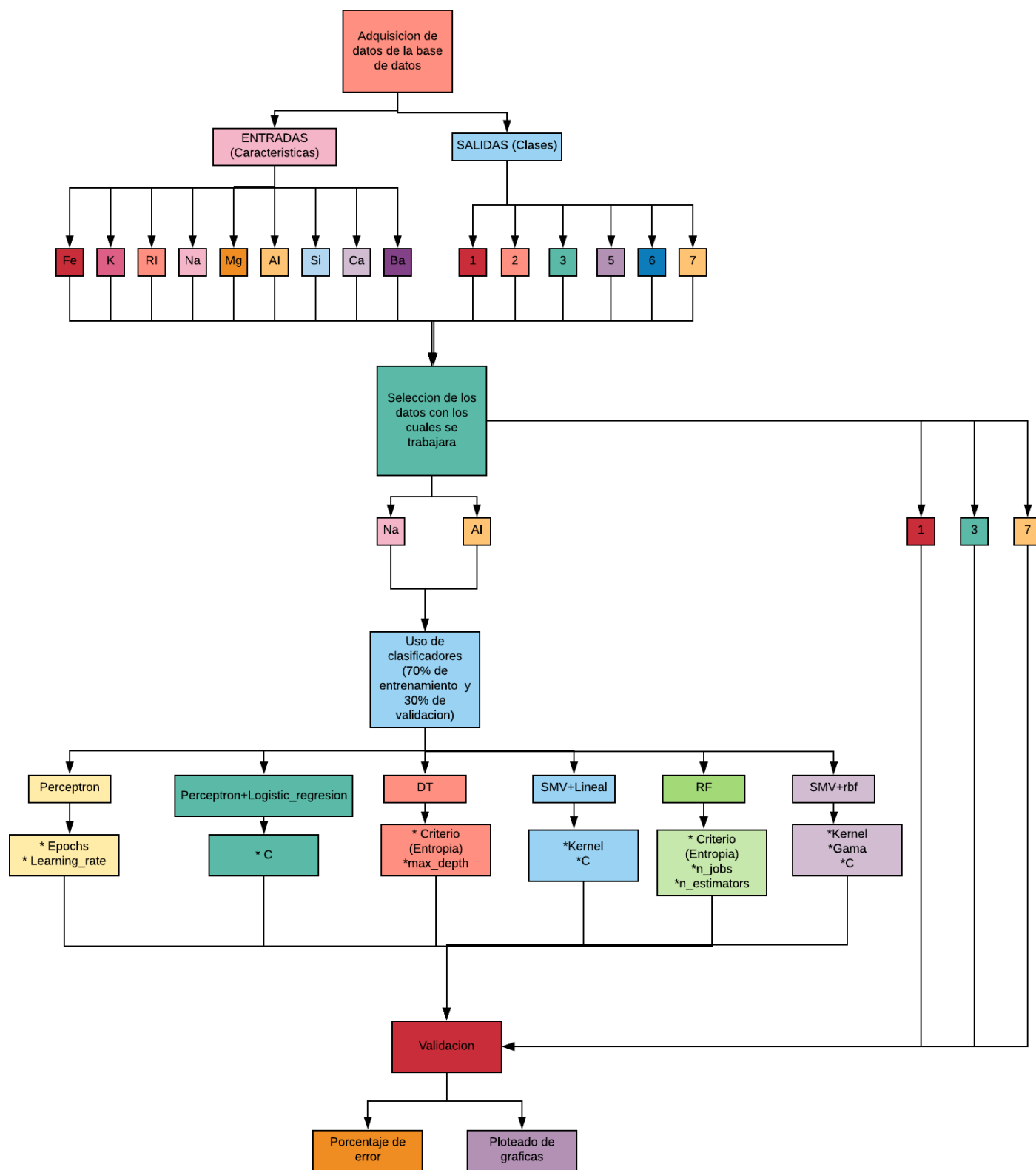


Figura 2. Diagrama de flujo del código a implementar.

4. Resultados

Perceptrón.

Se decidió iniciar por el algoritmo del perceptrón, ya que este fue analizado con mayor profundidad durante el segundo corte de la materia. De forma general, este algoritmo toma los valores de entrada y los multiplica por unos pesos, los cuales se actualizan de forma iterativa en cada época. Al evaluar cada uno de estos valores, se realiza la sumatoria de todos ellos y se hacen pasar a través de una función de activación que, para el caso del perceptrón normal, es una función de escalón unitario. Con el resultado obtenido se evalúa el error y dependiendo de este se actualiza nuevamente cada uno de los pesos.

Para el caso particular del perceptrón en la librería Scikit-learn de Python, hace falta escalar los datos de entrada de la misma manera que se hizo en la entrega anterior. Ya que como se afirma en el libro guía, la estandarización es indispensable para el buen desempeño de algunos algoritmos de clasificación. Para esto, la librería nos presenta las herramientas necesarias para realizar esta tarea, ofreciéndonos la posibilidad de remplazar varias líneas de código por las siguientes:

```
111     # Estandarizacion
112     sc = StandardScaler()
113     sc.fit(X_train)
114     X_train_std = sc.transform(X_train)
115     X_test_std = sc.transform(X_test)
```

Figura 3. Estandarización en Python.

Con esto ya se hace posible la utilización del perceptrón de la librería Scikit-learn de Python como se muestra en las siguientes líneas de código:

```
121     #PERCEPTRON
122     ppn = Perceptron(max_iter=40, eta0=0.1, random_state=1)
123     ppn.fit(X_train_std, y_train)
124
125     #Errores
126     y_pred = ppn.predict(X_test_std)
127     print('Misclassified samples of PPN: %d' % (y_test != y_pred).sum())
128     print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
129     print('Accuracy: %.2f' % ppn.score(X_test_std, y_test))
130
131     #regiones de decisión
132     plot_decision_regions(X=X_combined_std, y=y_combined,
133                          classifier=ppn, test_idx=range(48, 69))
134     plt.xlabel('Porcentaje sodio [%]')
135     plt.ylabel('Porcentaje aluminio [%]')
136     plt.legend(loc='upper left')
137
138     plt.tight_layout()
139     #plt.savefig('images/03_01.png', dpi=300)
140     plt.show()
```

Figura 4. Implementación del perceptrón en Python.

Donde al igual que en la implementación que se realizó en entregas anteriores, los parámetros de entrada son el número de iteraciones y la tasa de aprendizaje. Al evaluar el algoritmo de esta manera obtuvimos los siguientes resultados:

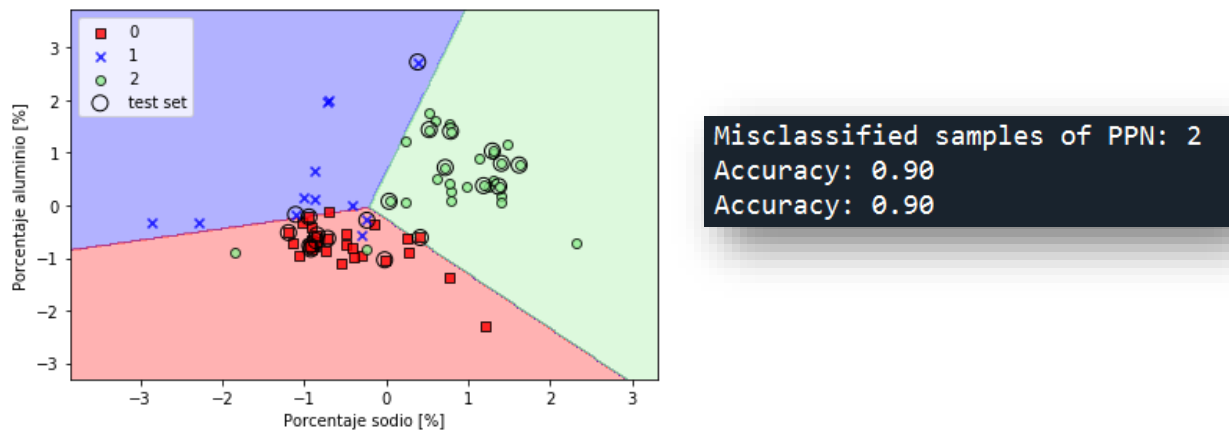


Figura 5. Resultados de implementación del perceptrón en Python.

Con lo anterior podemos analizar que, de forma general, la clasificación de los datos ha sido realizada de una manera correcta. Las regiones fueron separadas de tal manera que el mayor número de datos estuviera dentro de ellas. Pero aquí es donde se pueden observar algunas de las limitantes de este algoritmo, ya que la separación que realiza el perceptrón es de forma lineal, lo que provoca que, si las tres clases no se pueden separar perfectamente con una línea, el error nunca convergerá a cero y los pesos se actualizarán indefinidamente. A pesar de esto, se puede observar que el porcentaje de error fue muy pequeño, ya que de 69 muestras que fueron tomadas, solo 2 fueron clasificadas de forma errónea, dando un acierto de más del 97%.

Perceptrón más la regresión logística.

Para continuar, se decidió seguir con el mismo patrón que lleva el libro, y en este caso se continuó por la modificación del algoritmo del perceptrón, cambiando la función de activación del escalón unitario por la sigmoide. Esto permite que la regresión logística sea capaz de indicar la probabilidad de que un dato pertenezca a una clase específica, a diferencia del perceptrón, el cual solo actualizaba los pesos y basado en esto hacía su predicción.

Ya que este algoritmo basa sus resultados en un modelo probabilístico, podemos inferir de antemano que va a tener un mejor desempeño en la clasificación de los parámetros de nuestra base de datos. Para la implementación, vamos a hacer uso nuevamente de la librería Scikit-learn de Python como se muestra a continuación:

```

142 #PERCEPTRON + LOGISTIC REGRESSION
143 lr = LogisticRegression(C=10.0, random_state=1)
144 lr.fit(X_train_std, y_train)
145 #Errores
146 y_pred = lr.predict(X_test_std)
147 print('Misclassified samples of PPN+LR: %d' % (y_test != y_pred).sum())
148 print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
149 print('Accuracy: %.2f' % lr.score(X_test_std, y_test))
150 #regiones de decisión
151 plot_decision_regions(X_combined_std, y_combined,
152                      classifier=lr, test_idx=range(48, 69))
153 plt.xlabel('Porcentaje sodio [%]')
154 plt.ylabel('Porcentaje aluminio [%]')
155 plt.legend(loc='upper left')
156 plt.tight_layout()
157 #plt.savefig('images/03_06.png', dpi=300)
158 plt.show()

```

Figura 6. Implementación del perceptrón + LR en Python.

Donde de manera intuitiva podemos observar que el único parámetro que se debe definir para la implementación del este algoritmo es el parámetro C. Este es el responsable de evitar un sobre ajuste en de la red, ya que es el inverso del parámetro de regularización λ tal y como se muestra en la siguiente ecuación:

$$C = \frac{1}{\lambda}$$

Ecuación 1. Relación ente C y el parámetro de regularización.

Con esto claro, a continuación se presentan los resultados obtenidos:

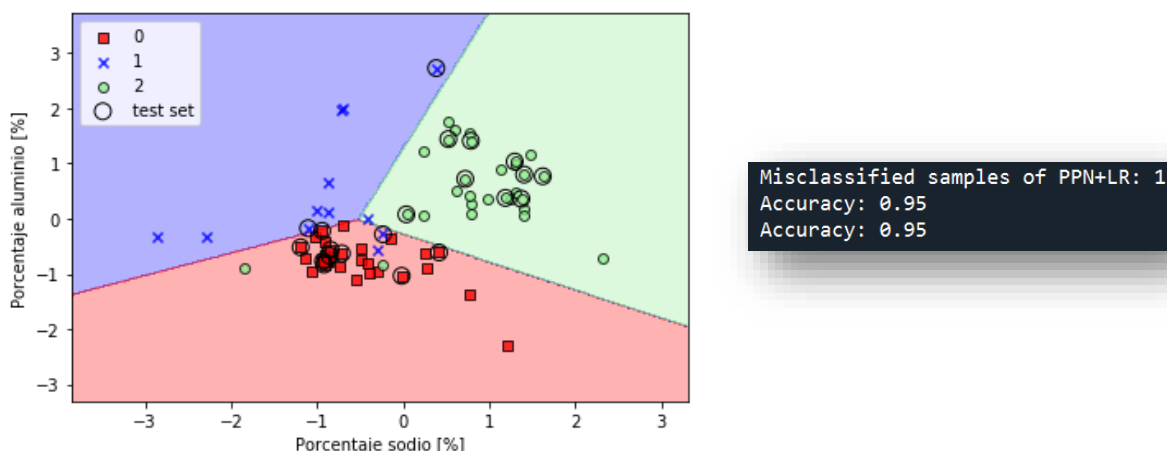


Figura 7. Resultados de implementación del perceptrón + LR en Python.

Observando la gráfica anterior, podemos analizar qué, tal y como se esperaba, el desempeño del algoritmo fue mucho mejor que el del perceptrón, ya que el número de parámetros clasificados erróneamente fue menor, haciendo que el porcentaje de acierto subiera del 97% al 98.5%. a pesar de que se pudo ver una gran mejora en el algoritmo, podemos analizar que aun persiste un error en la

clasificación, esto se debe a que la regresión logística sigue siendo un método de separación de clases lineales.

Máquinas de soporte vectorial.

El siguiente algoritmo que se decidió analizar, es el de las máquinas de soporte vectorial o SVM por sus siglas en inglés. Este tiene como objetivo minimizar el margen que existe entre el hiperplano positivo y negativo. De esta manera se podría decir que las máquinas de soporte vectorial son una modificación adicional que se le realiza al algoritmo del perceptrón, para que este, además de determinar una línea o separación entre los conjuntos de datos, pueda maximizar ese margen hasta que este se encuentre equidistante a los vectores de soporte. Este concepto se ve más claramente en la siguiente gráfica:

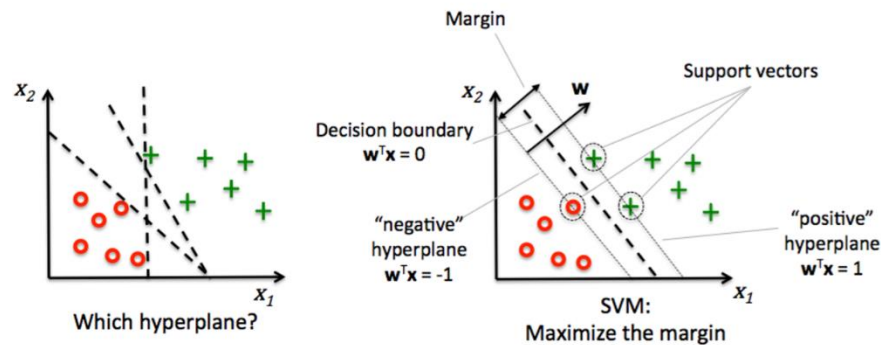


Figura 8. Principio lógico de SVM.

Para la implementación en Python, los parámetros que se deben definir son el kernel, del cual se hablara con mayor profundidad en el siguiente algoritmo, y C, parámetro el cual se mencionó en la sección anterior. se realiza de la siguiente manera:

```

161 #SUPPORT VECTOR MACHINE
162 svm = SVC(kernel='Linear', C=100.0, random_state=1)
163 svm.fit(X_train_std, y_train)
164 #Errores
165 y_pred = svm.predict(X_test_std)
166 print('Misclassified samples of SVM: %d' % (y_test != y_pred).sum())
167 print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
168 print('Accuracy: %.2f' % svm.score(X_test_std, y_test))
169 #regiones de decisión
170 plot_decision_regions(X_combined_std,
171                      y_combined,
172                      classifier=svm,
173                      test_idx=range(48, 69))
174 plt.xlabel('Porcentaje sodio [%]')
175 plt.ylabel('Porcentaje aluminio [%]')
176 plt.legend(loc='upper left')
177 plt.tight_layout()
178 #plt.savefig('images/03_11.png', dpi=300)
179 plt.show()

```

Figura 9. Implementación de SVM en Python.

Con lo anterior podemos inferir que, a pesar de que las máquinas de soporte vectorial son algoritmos un poco más robustos que el perceptrón o la regresión logística, estos siguen siendo lineales, ya que lo que se busca es separar de forma lineal las tres clases presentes en nuestra base de datos. Es por esta razón que el resultado de la implementación del algoritmo es muy similar al de la regresión logística, tal y como se muestra a continuación:

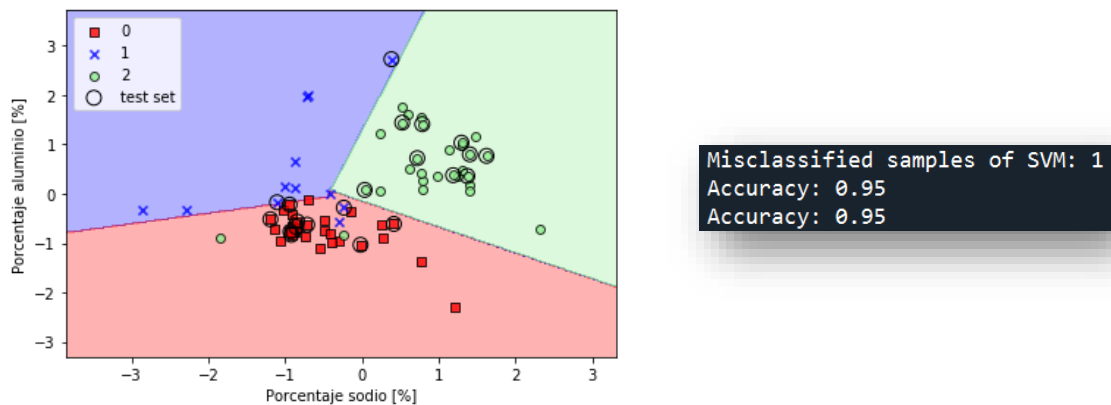


Figura 10. Resultados de implementación de SVM en Python.

Máquinas de soporte vectorial con Kernel de base radial.

Arboles de decisión.

Random forests.