INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY, CAMPUS ESTADO DE MÉXICO.

Escuela de Ingenierías

Ingeniería en Transformación Digital de Negocios

Momento de Retroalimentación: Módulo 2 Análisis y Reporte sobre el desempeño del modelo

Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)

Diana Cañibe Valle A01749422

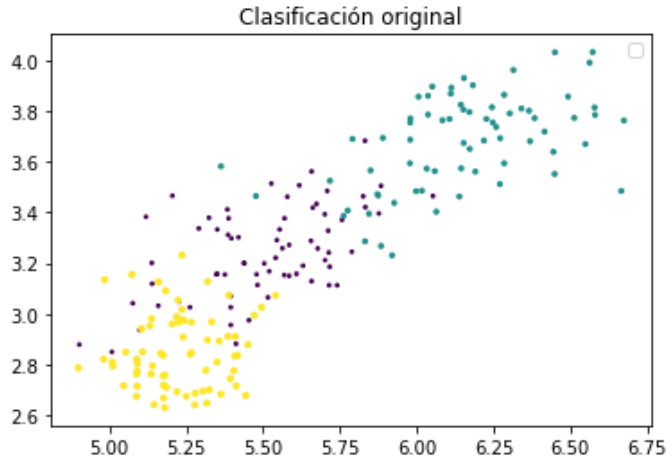
Fecha de entrega: 13 de septiembre del 2022

Análisis y Reporte sobre el desempeño del modelo

El algoritmo utilizado para este reporte es 'K-means', este algoritmo es un modelo de aprendizaje no supervisado, en el cuál los datos proporcionados son agrupados en diferentes grupos, los cuales comparten características similares.

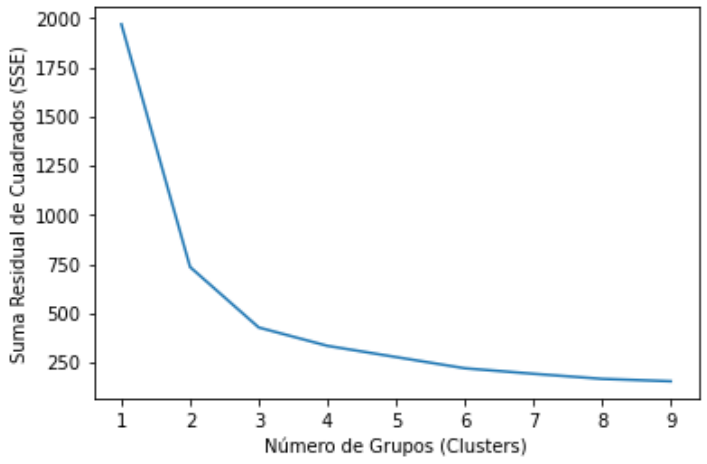
Puntualmente la clasificación realizada para este análisis utiliza un dataset de 3 tipos de semillas de trigo, con 70 observaciones para cada una.
Fuente: http://archive.ics.uci.edu/ml/datasets/seeds

Los datos iniciales son los siguientes:



Nota: El dataset tiene 7 dimensiones para la descripción de cada semilla, por cuestiones prácticas para la gráfica, utilizaré solo 2 (width-eje X, length-eje Y, seed_class-color).

Aunque se conoce el total de grupos que debería haber (3), se realizará también una gráfica de codo para determinar en base a los datos la cantidad de cluster adecuados.

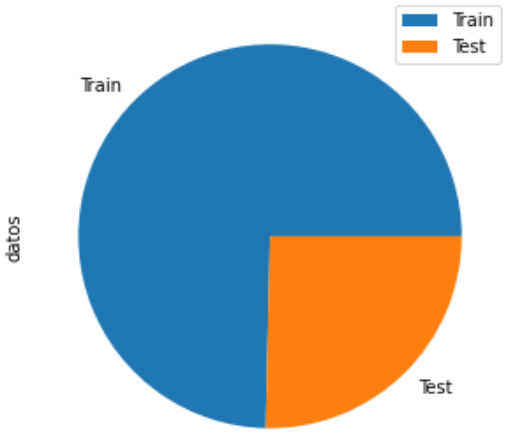


En base a la gráfica podemos observar que 4 o 5 grupos tiene un mejor resultado en la reducción del error (SSE), por lo tanto realizaremos corridas distintas para probar ambos valores y la cantidad original.

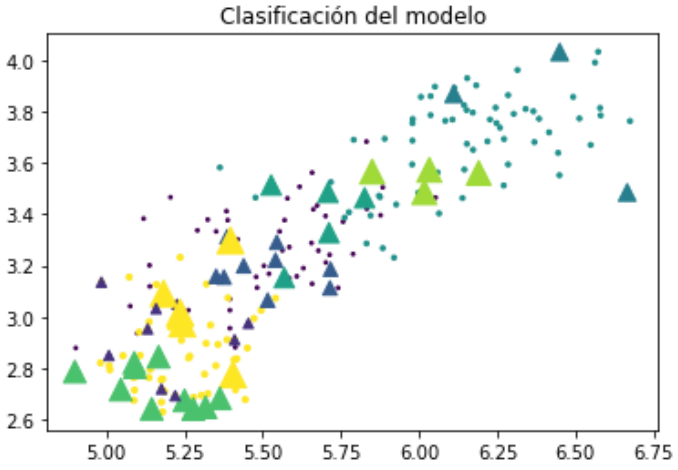
Con el fin de validar la acertividad de la clasificación y poder realizar pruebas, se realiza la partición del total del dataset por medio de la función *train_test_split* de la librería *sklearn.model_selection.* Dicha función separa los datos en una proporción de 75% entrenamiento, 25% prueba.

```
print('Porcentaje entrenamiento')
print((Xtrain.shape[0]/X.shape[0])*100)
print('Porcentaje prueba')
print((Xtest.shape[0]/X.shape[0])*100)
```

```
Porcentaje entrenamiento
74.64114832535886
Porcentaje prueba
25.358851674641148
```



Con estos datos podemos ver la primera versión del modelo y sus resultados



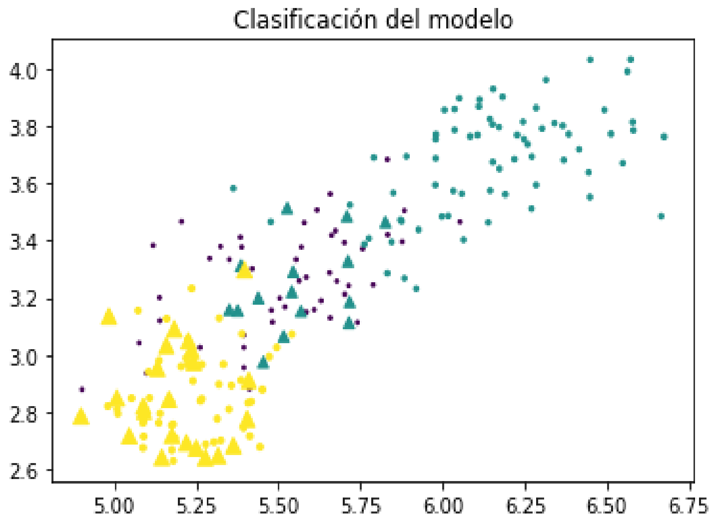

```
#Prediciones con el modelo (Pruebas)
ymodel = model.predict(Xtest)
print('Score:',accuracy_score(ytest, ymodel))

#Gráfica de clásificación según el modelo
plt.scatter(X['length'],X['width'], s=size,c = y)
size=ymodel*30
plt.scatter(Xtest['length'],Xtest['width'], s=size, marker='^',c=ymodel)
plt.title('Clasificación del modelo')
plt.show()
```

Score: 0.07547169811320754

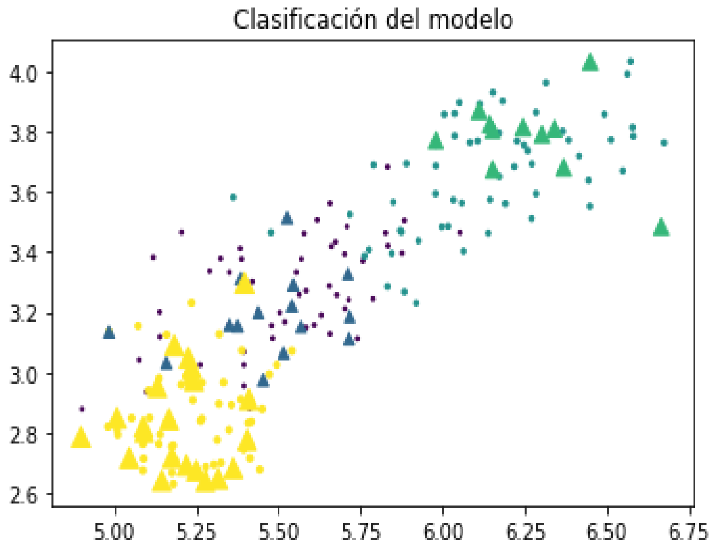
Al correr k-means sin ningún argumento intenta hacer una clasificación de 8 grupos, lo cual resulta en una precisión baja para la predicción de la clasificación de prueba, obteniendo menos del 10% de aciertos, indicando un underfitt con varianza y bias alto, ya que como sabemos por el método del codo, 8 clusters no es un valor adecuado para la clasificación.

Si comparamos los resultados con los valores de 3,4 y 5 clusters obtenemos los siguientes resultados.

3 Clusters	4 Clusters	5 Clusters
Score: 0.2830188679245283	Score: 0.8113207547169812	Score:0.0943396226415094

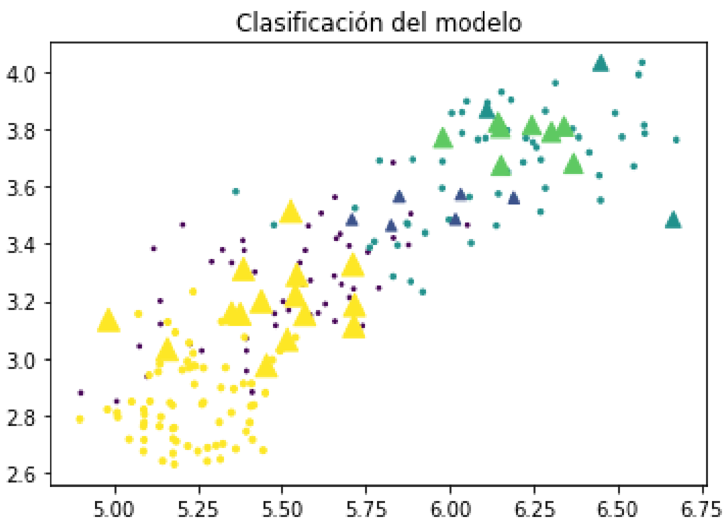


La clasificación con 3 clusters [Gráfica 1] presenta varianza baja, ya que como se ve en la gráfica los valores no están dispersos, sin embargo el bias es alto ya que la clasificación no pertenece al grupo original.



En la gráfica con 4 clusters [Gráfica 2] podemos apreciar que el modelo realiza la predicción acertada de los valores de prueba en comparación a los grupos originales lo que nos indica que la variación y bias son bajos y el ajuste del modelo es adecuado (fit)

Finalmente en el de 5 clusters [Gráfica 3] podemos notar como el score para la predicción de los grupos vuelve a disminuir y es claro que la varianza y bias se vuelven a elevar como en el modelo de prueba con 8 grupos, debido a que se intentan identificar más grupos de los que se pueden evaluar.



Ahora bien ¿porque esto no pasa con 4, y es nuestro mejor resultado? y ¿porque con 3 grupos no es suficiente si es el valor original?, la respuesta es simple, K-means clasifica los grupos con clases partiendo de 0, y nuestros datos inician en la clase 1, y al hacer la validación del score y las predicciones de la variable objetivo los valores no coinciden. Por ello debemos asumir que hay un desplazamiento en los datos de las gráficas y los cluster considerando cada uno como k=n-1, es decir la clasificación de 3 corresponde a 2, la de 4 a 3, la de 5 a 4, y así sucesivamente.

Ajuste de parámetros para mejora del desempeño del modelo

Podemos realizar cambios para obtener mejores resultados para nuestro modelo, podemos decir que ya realizamos uno al principio con el método del codo para determinar la cantidad de clusters adecuados el cual es un parámetro para la función que implementa el algoritmo en la librería.

Otra búsqueda que podemos realizar es por medio de un 'grid search' el cual puede probar versiones diferentes con distintos parámetros para el modelo y validar cual es la combinación que nos proporciona los mejores resultados. Para hacerlo usamos la función *GridSearchCV* de la librería *sklearn.model_selection.*

Como ya hemos solucionado el número de clusters, validaremos el método de inicialización, el cual en la entrega anterior identificamos como relevante para el comportamiento del modelo.

```
#Grid Search para ajuste de hiperparámetros
from sklearn.model_selection import GridSearchCV
parameters={'init':('k-means++','random')}
clf = GridSearchCV(model, parameters)
clf.fit(Xtrain,ytrain)
ymodel = clf.predict(Xtest)
print('Score con grid search:',accuracy_score(ytest, ymodel))
```

Al validar la predicción nuevamente con el modelo con lo que se supone que son los mejores parámetros determinados con la búsqueda, vemos que es el que obtuvimos previamente, por lo tanto para este caso específico los valores por default de la función proporcionan la mejor de las distintas variaciones posibles.

```
Score: 0.8113207547169812
Score con grid search: 0.8113207547169812
```