SVM: Reporte y Análisis de Desempeño

Pablo Ortiz Aurrecoechea A01023791

Septiembre 2023

Introducción

En este reporte, compararemos el desempeño de dos modelos de Máquinas de Soporte Vectorial. En el primer modelo utilizaremos los hiperparámetros predeterminados y en el segundo buscaremos cambiar algunos parámetros para mejorar el funcionamiento del modelo. Para los datos, utilizaremos el *"Breast Cancer Detection Wisconsin Dataset"* para realizar clasificación binaria de tumores cancerígenos.

Selección de Datos

Como se mencionó durante la introducción, para este reporte utilizaremos el *"Breast Cancer Detection Wisconsin Dataset"* para realizar clasificación binaria. Este dataset cuenta con 569 instancias de tumores clasificados como "Benignos" o "Malignos" junto con sus características de dimensión, textura, etc. Este dataset se seleccionó por las razones que aquí se explican:

- **Clasificación Binaria:** Debido a que estamos trabajando con un Clasificador de Soporte Vectorial, es indispensable utilizar un dataset separable binariamente por un hiperplano. Es por eso que, la clasificación entre "Benigno" y "Maligno" hace que el dataset sea perfecto para esta tarea.
- **Facilidad de manejo:** El dataset no cuenta con valores vacíos, cuenta con escalas bastante similares entre las variables (aunque realizamos una normalización) y ya se cuenta pre-cargado en sklearn. Por lo tanto, limpiar, entender y cargar el dataset es muy sencillo sin importar el entorno en el que se esté trabajando.
- **Comparabilidad:** Como buscaremos demostrar que el modelo generaliza, necesitamos un dataset conocido, que sea linealmente separable y que sea fácil de comparar con otros modelos externos.

Análisis Exploratorio y tratamiento de los datos

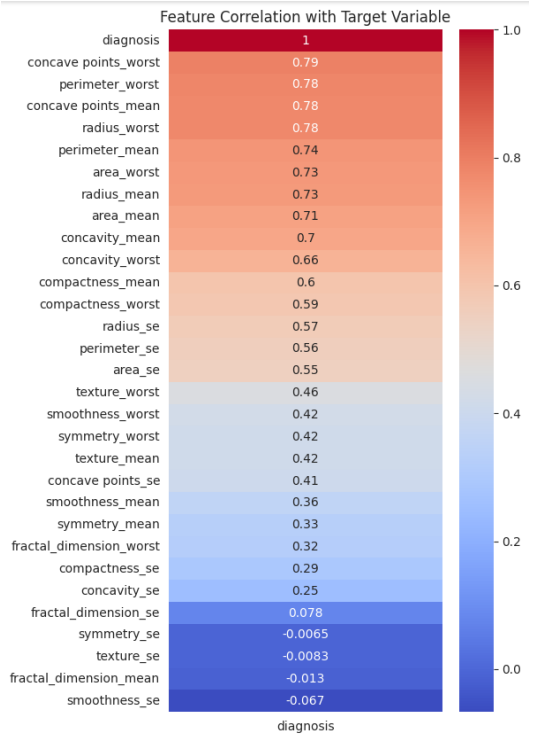
Lo primero que necesitamos conocer, es el tipo de variables con las que estamos trabajando, al igual que si tenemos o no datos vacíos.

```
Data columns (total 32 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      569 non-null    int64
 1   diagnosis               569 non-null    object
 2   radius_mean             569 non-null    float64
 3   texture_mean            569 non-null    float64
 4   perimeter_mean          569 non-null    float64
 5   area_mean               569 non-null    float64
 6   smoothness_mean         569 non-null    float64
 7   compactness_mean        569 non-null    float64
 8   concavity_mean          569 non-null    float64
 9   concave points_mean     569 non-null    float64
 10  symmetry_mean           569 non-null    float64
 11  fractal_dimension_mean  569 non-null    float64
 12  radius_se               569 non-null    float64
 13  texture_se              569 non-null    float64
 14  perimeter_se            569 non-null    float64
 15  area_se                 569 non-null    float64
 16  smoothness_se           569 non-null    float64
 17  compactness_se          569 non-null    float64
 18  concavity_se            569 non-null    float64
 19  concave points_se       569 non-null    float64
 20  symmetry_se             569 non-null    float64
 21  fractal_dimension_se    569 non-null    float64
 22  radius_worst            569 non-null    float64
 23  texture_worst           569 non-null    float64
 24  perimeter_worst         569 non-null    float64
 25  area_worst              569 non-null    float64
 26  smoothness_worst        569 non-null    float64
 27  compactness_worst       569 non-null    float64
 28  concavity_worst         569 non-null    float64
 29  concave points_worst    569 non-null    float64
 30  symmetry_worst          569 non-null    float64
 31  fractal_dimension_worst 569 non-null    float64
```

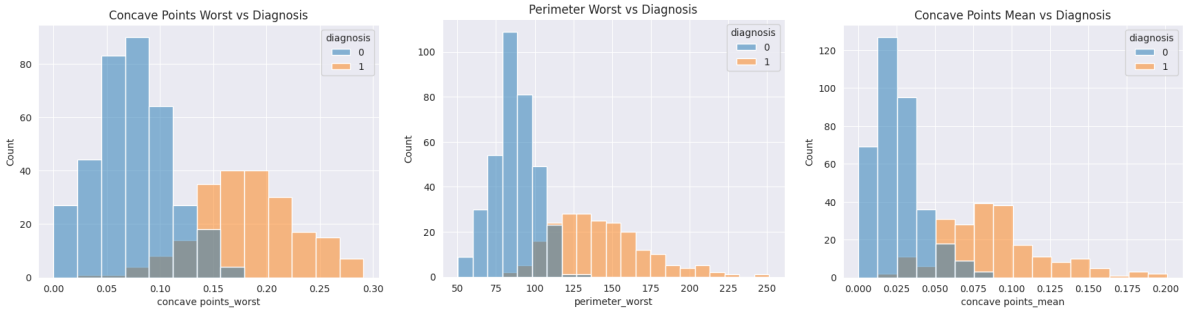
Como se puede apreciar en la figura anterior, el dataset cuenta con una columna categórica (“diagnosis”) y el resto son valores numéricos y no contamos con valores vacíos. Después, es necesario conocer algunas estadísticas descriptivas, sobre todo para identificar si es necesario realizar una normalización de los datos.

	count	mean	std	min	25%	50%	75%	max
diagnosis	569.0	0.372583	0.483918	0.000000	0.000000	0.000000	1.000000	1.00000
radius_mean	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
texture_mean	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
perimeter_mean	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
area_mean	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
smoothness_mean	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
compactness_mean	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
concavity_mean	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
concave points_mean	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120
symmetry_mean	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30400
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09744
radius_se	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87300
texture_se	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88500
perimeter_se	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98000
area_se	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20000
smoothness_se	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03113
compactness_se	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13540
concavity_se	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39600
concave points_se	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05279

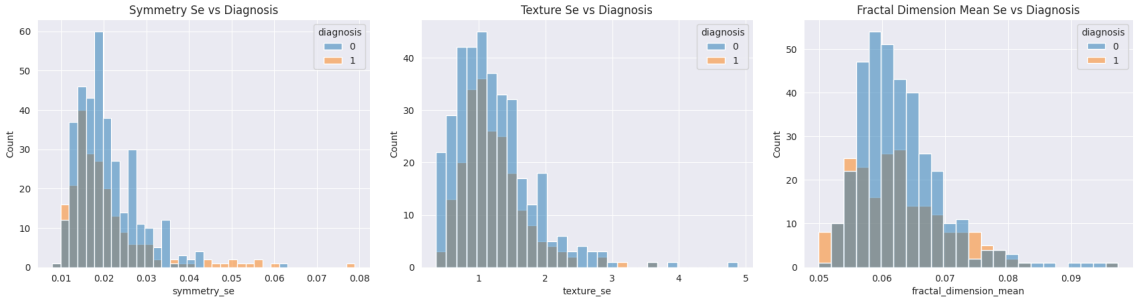
Como podemos ver, algunas variables como *"area_mean"* tienen una escala muy distinta que otras variables como *"concavity_mean"*. A continuación, veremos cómo se comparan las variables más y menos correlacionadas con la clase objetivo.



Gráficas independientes de variables de alta correlación:



Gráficas independientes de variables de baja correlación:



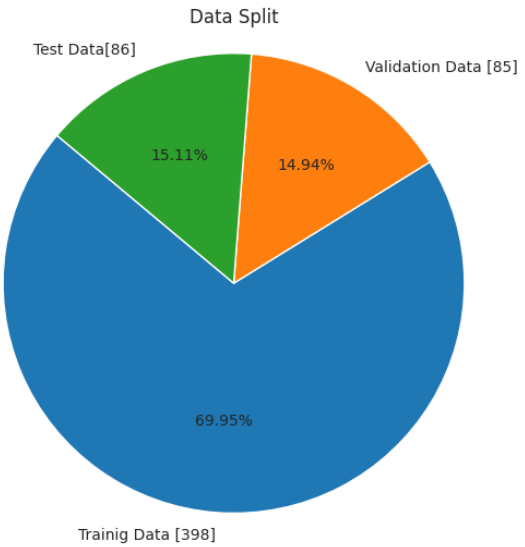
Ya que conocemos un poco sobre el dataset empezaremos con el tratamiento de los datos. Primero que nada, va a ser necesario realizar una normalización para remover el efecto de las diferentes escalas.

```
[17] scaler = MinMaxScaler()

[18] X = scaler.fit_transform(df[['radius_mean', 'texture_mean', 'perimeter_mean',
                                  'area_mean', 'smoothness_mean', 'compactness_mean',
                                  'concavity_mean', 'concave points_mean', 'symmetry_mean',
                                  'radius_se', 'perimeter_se', 'area_se', 'compactness_se',
                                  'concavity_se', 'concave points_se', 'radius_worst',
                                  'texture_worst', 'perimeter_worst','area_worst',
                                  'smoothness_worst', 'compactness_worst', 'concavity_worst',
                                  'concave points_worst','symmetry_worst', 'fractal_dimension_worst']])

[21] y = df["diagnosis"]
```

Una vez normalizada la data, haremos un split entre “test”, “train” y “validation” para realizar las comprobaciones entre bias y varianza.



Primer Modelo con Hiperparámetros Predeterminados

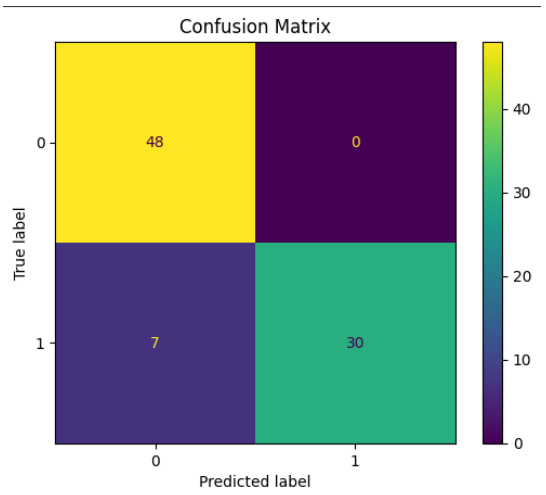
Iniciamos creando el Clasificador de Soporte Vectorial con los valores predeterminados de sklearn. Como podemos ver, de inmediato el modelo presenta buenos resultados. El modelo cuenta con un accuracy general de 92%, con errores para la clase 0 *"Benigno"*.

```
model = SVC()
model.fit(X_train,y_train)
predictions = model.predict(X_validation)
```

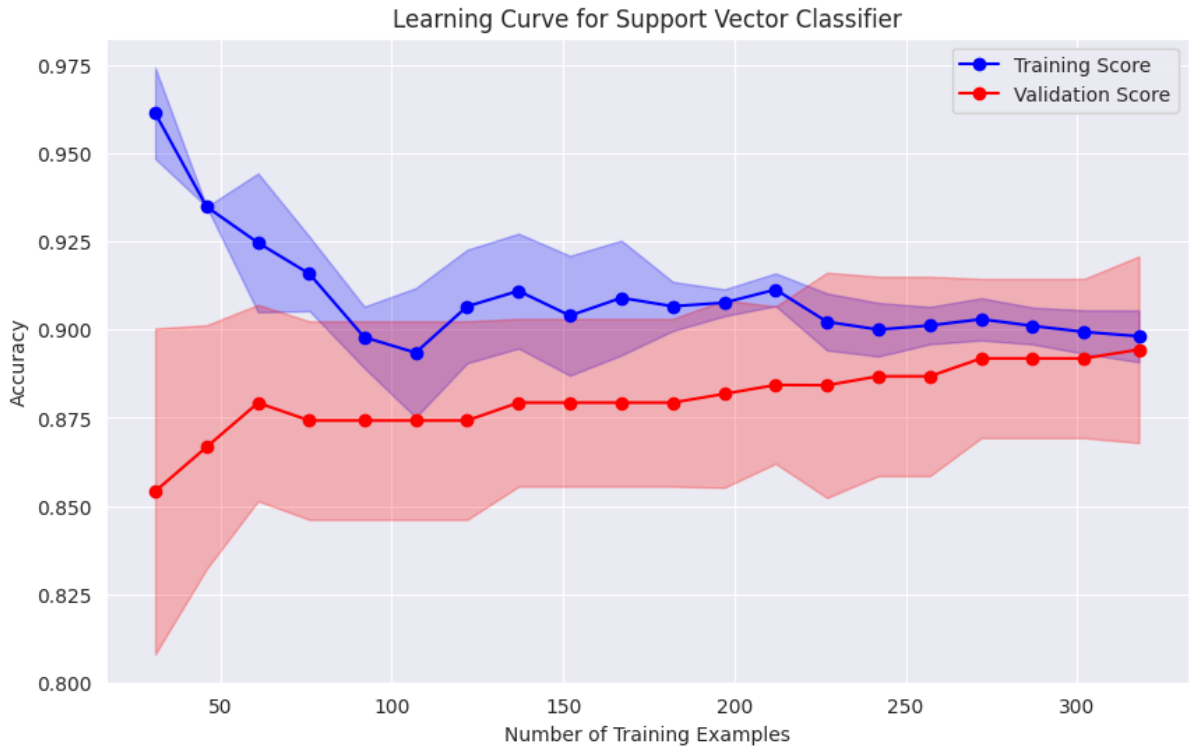
```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93        48
           1       1.00      0.81      0.90        37

    accuracy                           0.92        85
   macro avg       0.94      0.91      0.91        85
weighted avg       0.93      0.92      0.92        85
```



Sin embargo, cuando observamos el aprendizaje del modelo al variar los tamaños del set de entrenamiento y validación podemos ver cómo generaliza y se balancea la varianza y el bias.



Al iniciar el entrenamiento, el modelo cuenta con poca data de entrenamiento y mucha data de validación, lo que genera en muy buen desempeño para la data de entrenamiento pero un bajo accuracy para la data de validación. Esto se genera por un gran overfit de la data de entrenamiento que no generaliza ante la data de validación. Esto se puede interpretar como que el modelo tiene mucha varianza y bajo bias. Mientras el modelo se sigue entrenando con más datos, se puede ver cómo la diferencia entre los resultados de entrenamiento y validación se reduce hasta convertirse en virtualmente 0.

Segundo Modelo con Hiperparámetros Mejorados

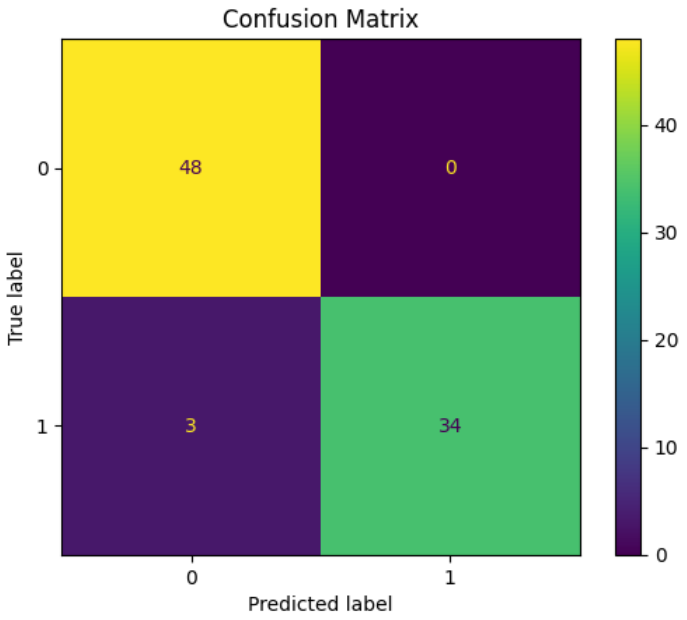
El siguiente paso que realizamos fue variar los valores predeterminados del modelo para buscar generar un mejor desempeño. En este caso, se cambió el valor de “C” de 1 a 0.1 y se cambió el kernel a uno “lineal”. Cómo se puede identificar de inmediato, el accuracy sube de 92% a 96% y los errores en la clase 0 “Benigno” se reducen.

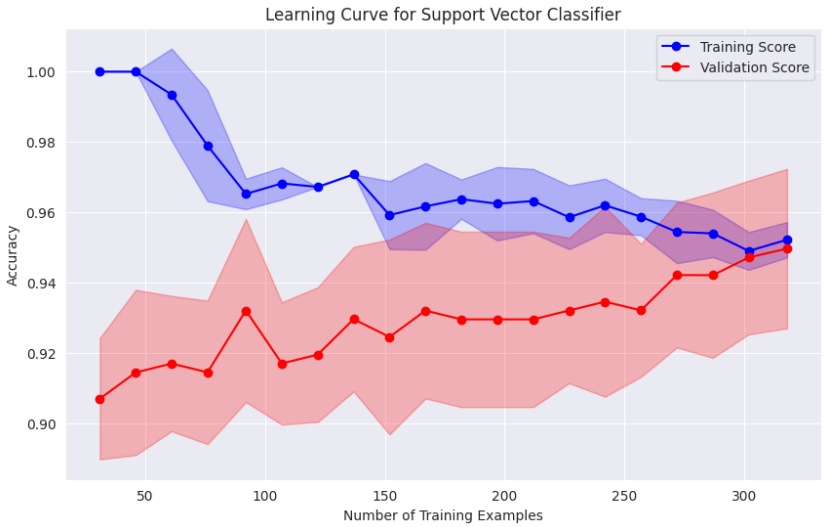
```
model = SVC(C =0.1,kernel = "linear")
model.fit(X_train,y_train)
predictions = model.predict(X_validation)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        48
           1       1.00      0.92      0.96        37

    accuracy                           0.96        85
   macro avg       0.97      0.96      0.96        85
weighted avg       0.97      0.96      0.96        85
```

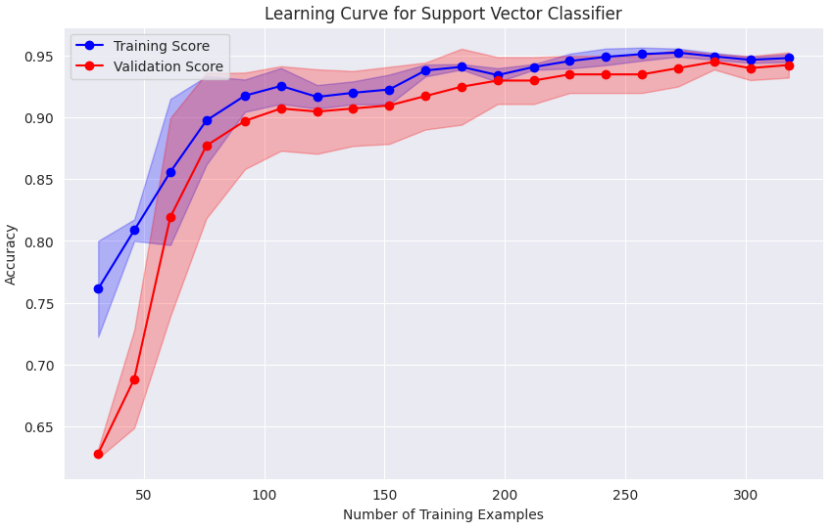




Similar al primer caso, al iniciar el entrenamiento existe una inmensa diferencia entre el desempeño con la data de entrenamiento y la data de validación. El modelo presenta un accuracy perfecto para la data de entrenamiento mientras que la data de validación se encuentra cercano a 91%. De nuevo, este es un ejemplo de alta varianza y overfitting que se va reduciendo mientras el modelo continúa aprendiendo.

Ejemplo de Alto Sesgo y Baja Varianza

En los casos anteriores, vimos cómo el modelo corregía situaciones de overfitting causados por bajo bias y alta varianza. Sin embargo, también se presentan los casos opuestos donde el modelo tiene un alto bias y relativamente baja varianza. A continuación, se muestra un ejemplo del desempeño del segundo modelo pero con la data sin normalizar. Se puede observar claramente cómo, al iniciar el entrenamiento, el desempeño es malo tanto para el set de entrenamiento como el de validación. Al continuar aprendiendo el modelo arregla este problema para alcanzar un buen accuracy de 95%.



Conclusiones

En este reporte, se comparó el desempeño de dos modelos de Máquina de Soporte Vectorial para realizar clasificación binaria de tumores. Como se pudo demostrar, al variar los parámetros de “C” y “kernel”, se pudo mejorar el desempeño del modelo por 4 puntos porcentuales. Adicionalmente, se demostró cómo los modelos generalizan balanceando casos de mucha varianza y bajo sesgo para alcanzar buenos resultados en los datos de entrenamiento y validación. Finalmente, pudimos observar casos opuestos donde la varianza es baja pero el sesgo es alto, factor que también puede causar mal desempeño de los modelos de Machine Learning.