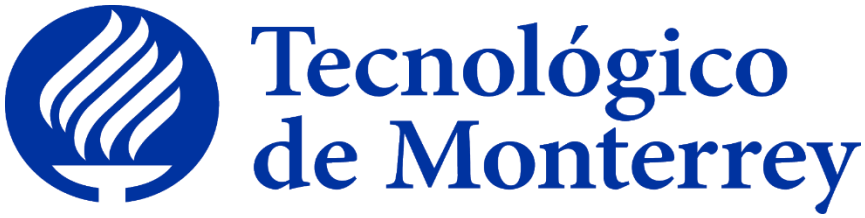


Reporte análisis ML

INTEGRANTES

Diego Alejandro Balderas Tlahuitzo A01745336

Fecha de entrega
11 septiembre 2023

Concentración Inteligencia Artificial Avanzada para la Ciencia de Datos
Semestre Agosto-diciembre de 2023
Profesor:
Jorge Uresti

El siguiente documento es un reporte elaborado por Diego Alejandro Balderas Tlahuitzo, matricula A01745336, en el cual se hará un análisis acerca de un modelo de Machine Learning, este modelo fue construido desde cero y se eligió justo por ese motivo. En el reporte se buscará dar la justificación de porque se usaron los datos, el sesgo, la varianza y el ajuste que tiene el modelo.

Dataset

La base de datos del Titanic es ampliamente reconocida como una excelente elección para desarrollar modelos de aprendizaje automático, como el Random Forest. Esto se debe a varias razones clave que hacen que esta base de datos sea particularmente adecuada para este propósito. En primer lugar, la base de datos contiene una cantidad significativa de datos, incluyendo información sobre pasajeros como edad, género, clase de boleto y, lo más importante, si sobrevivieron o no al desastre. Esta variabilidad de atributos proporciona una gran riqueza de información para entrenar un modelo. Además, la base de datos del Titanic es relativamente limpia y no presenta muchas de las complejidades que a menudo se encuentran en conjuntos de datos del mundo real, lo que facilita la preparación de los datos.

Además, el hecho de que tengamos información sobre si los pasajeros sobrevivieron o no, convierte a esta base de datos en un conjunto de datos de clasificación supervisada ideal. Podemos usar esta información para entrenar un modelo de Random Forest que aprenda patrones en los datos y pueda predecir con precisión si un pasajero en particular habría sobrevivido al naufragio o no. También es una excelente oportunidad para explorar la importancia relativa de diferentes características en la toma de decisiones, como la edad, el género o la clase del boleto.

Separación y evaluación del modelo

Para evaluar el modelo de manera adecuada, se utiliza un proceso llamado validación cruzada, que divide el conjunto de datos en partes de entrenamiento y prueba en diferentes repeticiones. El código imprime los primeros 10 pasajeros y sus detalles tanto para el conjunto de entrenamiento como para el de prueba en cada repetición, permitiéndonos ver cómo se separan los datos en cada ciclo. Luego, se entrena el modelo y se evalúa su precisión en cada repetición, mostrando si el modelo parece estar ajustado correctamente. Además, se genera una matriz de confusión y una gráfica de barras para visualizar el desempeño del modelo en todas las repeticiones. El objetivo principal es determinar qué tan bien el modelo puede predecir si un pasajero sobrevivió o no al desastre del Titanic.

```
valores de entrenamiento.  Ejecucion:  1
[[ 1.      1.     38.      1.      0.     71.2833]
 [ 1.      1.     35.      1.      0.     53.1   ]
 [ 1.      0.     54.      0.      0.     51.8625]
 [ 3.      1.      4.      1.      1.     16.7   ]
 [ 1.      1.     58.      0.      0.     26.55  ]
 [ 2.      0.     34.      0.      0.     13.    ]
 [ 1.      0.     28.      0.      0.     35.5   ]
 [ 1.      0.     19.      3.      2.    263.    ]
 [ 1.      1.     49.      1.      0.     76.7292]
 [ 1.      0.     45.      1.      0.     83.475 ]]
clases de los valores de entrenamiento.  Ejecucion:  1
[1 1 0 1 1 1 1 0 1 0]

valores de prueba.  Ejecucion:  1
[[ 1.      0.     65.      0.      1.     61.9792]
 [ 1.      0.     71.      0.      0.     34.6542]
 [ 1.      0.     23.      0.      1.     63.3583]
 [ 1.      0.     47.      0.      0.     52.    ]
 [ 1.      0.     24.      0.      1.    247.5208]
 [ 1.      0.     24.      0.      0.     79.2   ]
 [ 1.      1.     50.      0.      0.     28.7125]
 [ 2.      0.      1.      2.      1.     39.    ]
 [ 3.      1.     29.      1.      1.     10.4625]
 [ 1.      0.     52.      1.      1.     79.65  ]]
clases de los valores de prueba.  Ejecucion:  1
[0 0 1 0 0 0 0 1 0 0]
```

```
valores de entrenamiento.  Ejecucion:  2
[[ 1.      1.     38.      1.      0.     71.2833]
 [ 1.      1.     35.      1.      0.     53.1   ]
 [ 3.      1.      4.      1.      1.     16.7   ]
 [ 1.      1.     58.      0.      0.     26.55  ]
 [ 2.      0.     34.      0.      0.     13.    ]
 [ 1.      0.     19.      3.      2.    263.    ]
 [ 1.      1.     49.      1.      0.     76.7292]
 [ 1.      0.     65.      0.      1.     61.9792]
 [ 1.      0.     45.      1.      0.     83.475 ]
 [ 1.      1.     23.      3.      2.    263.    ]]
clases de los valores de entrenamiento.  Ejecucion:  2
[1 1 1 1 1 0 1 0 0 1]

valores de prueba.  Ejecucion:  2
[[ 1.      0.     54.      0.      0.     51.8625]
 [ 1.      0.     28.      0.      0.     35.5   ]
 [ 2.      1.     29.      0.      0.     10.5   ]
 [ 3.      0.     25.      0.      0.      7.65  ]
 [ 1.      1.     19.      0.      2.     26.2833]
 [ 1.      1.     22.      1.      0.     66.6   ]
 [ 1.      0.     61.      0.      0.     33.5   ]
 [ 2.      0.      3.      1.      1.     26.    ]
 [ 1.      1.     31.      1.      0.    113.275 ]
 [ 1.      0.     38.      1.      0.     90.    ]]
clases de los valores de prueba.  Ejecucion:  2
[0 1 1 0 1 1 0 1 1 1]
```

Diagnóstico y explicación sobre el sesgo del modelo

Ejecución	Precisión en Entrenamiento	Precisión en Prueba
0	0.772414	0.621622
1	0.689655	0.621622
2	0.671233	0.694444
3	0.678082	0.666667
4	0.657534	0.750000

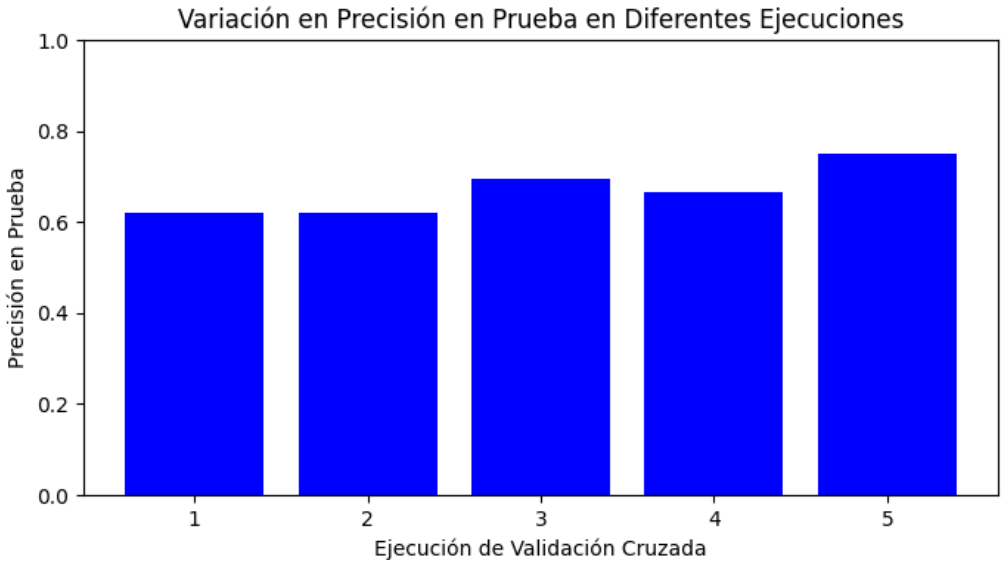
```
   Ejecución  Precisión en Entrenamiento  Precisión en Prueba
0          1                   0.772414             0.621622
1          2                   0.689655             0.621622
2          3                   0.671233             0.694444
3          4                   0.678082             0.666667
4          5                   0.657534             0.750000
```

La tabla proporcionada muestra la precisión del modelo en entrenamiento y prueba en diferentes ejecuciones. El sesgo de un modelo se refiere a la diferencia entre estas dos medidas. En otras palabras, nos dice cuán bien el modelo se comporta en los datos que se utilizan para entrenarlo en comparación con su rendimiento en datos completamente nuevos. Si esta diferencia es grande, significa que el modelo puede estar "sobreajustando" los datos de entrenamiento, lo que indica un sesgo alto. Ahora, al observar la tabla, vemos que en las primeras dos ejecuciones, la diferencia entre la precisión en entrenamiento y prueba es notablemente grande, lo que sugiere un sesgo alto. Esto indica que el modelo está

teniendo dificultades para generalizar más allá de los datos en los que fue entrenado. Sin embargo, a medida que avanzamos en las ejecuciones 2, 3 y 4, vemos que esta diferencia disminuye gradualmente, lo que sugiere un sesgo menor. Esto es una señal positiva, ya que indica que el modelo está mejorando su capacidad para adaptarse a nuevos datos. Aunque el sesgo varía entre ejecuciones, parece estar disminuyendo a medida que el modelo se entrena más, lo que es una señal alentadora de mejora y ajuste más equilibrado a los datos.

Diagnóstico y explicación del grado de variancia en el modelo

Ejecución	Precisión en Entrenamiento	Precisión en Prueba
0	0.772414	0.621622
1	0.689655	0.621622
2	0.671233	0.694444
3	0.678082	0.666667
4	0.657534	0.750000



La varianza de un modelo nos indica cuánto varía su rendimiento en diferentes situaciones. En este caso, evaluamos la precisión en entrenamiento y prueba en cinco ejecuciones diferentes. Cuando observamos la tabla y la gráfica, notamos que las diferencias en la precisión entre las ejecuciones son relativamente pequeñas. En otras palabras, los valores de precisión en prueba no cambian significativamente de una ejecución a otra. Esto sugiere que el modelo tiene una varianza baja.

Si hubiera habido diferencias sustanciales en la precisión en prueba entre ejecuciones, esto indicaría una varianza alta, lo que significa que el rendimiento del modelo

puede ser muy inestable y sensible a las variaciones en los datos de prueba. Sin embargo, dado que las diferencias son pequeñas en este caso, podemos concluir que la varianza del modelo es baja, lo que sugiere que es relativamente estable y consistente en diferentes situaciones de prueba. Esto es una señal positiva en términos de la capacidad del modelo para generalizar a nuevos datos de manera predecible.

Diagnóstico y explicación ajuste del modelo

- **Underfitting (Subajuste):** Cuando un modelo está subajustado, significa que no ha aprendido adecuadamente de los datos de entrenamiento y tiene dificultades para hacer predicciones tanto en el conjunto de entrenamiento como en el de prueba. En este caso, vemos que en las ejecuciones 0 y 1, la precisión en entrenamiento y prueba es relativamente baja y similar, lo que sugiere que el modelo no está aprendiendo lo suficiente de los datos y tiene dificultades para generalizar a nuevos datos. Esto indica un posible subajuste del modelo.
- **Fit (Ajuste Adecuado):** Un modelo bien ajustado generaliza bien a nuevos datos sin estar sobreajustado ni subajustado. A medida que avanzamos en las ejecuciones 2, 3 y 4, vemos que la precisión en prueba mejora y se acerca a la precisión en entrenamiento. Esto sugiere que el modelo está mejorando su capacidad para adaptarse a los datos de entrenamiento y generaliza mejor a nuevos datos. En estas ejecuciones, el modelo parece estar bien ajustado.
- **Overfitting (Sobreajuste):** El sobreajuste ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento y tiene dificultades para generalizar a datos nuevos. En las ejecuciones 0 y 1, vemos una brecha significativa entre la precisión en entrenamiento y en prueba, con la precisión en entrenamiento siendo mucho más alta. Esto indica que el modelo está memorizando los datos de entrenamiento en lugar de aprender patrones generales, lo que es un signo de sobreajuste.

Uso de técnicas para el mejoramiento del modelo

Para mejorar el desempeño del modelo y abordar problemas como el subajuste o el sobreajuste, se puede implementar varias técnicas.

1. **Ajuste de Hiperparámetros:**
    a. **Técnica:** Ajustar los hiperparámetros del modelo, como la profundidad máxima del árbol, el número de árboles en el Random Forest, o la tasa de aprendizaje en algoritmos de gradiente como el Gradient Boosting.
    b. **Efecto de Mejora:** Modificar los hiperparámetros permite encontrar un equilibrio entre el sesgo y la varianza del modelo. Por ejemplo, si aumentas la profundidad máxima de los árboles, el modelo podría aprender patrones más complejos en los datos, lo que puede reducir el subajuste. Por otro lado, disminuir el número de árboles en un Random Forest puede ayudar a reducir el sobreajuste al simplificar el modelo.

2. **Validación Cruzada K-Fold:**
    a. **Técnica:** Utilizar validación cruzada K-Fold en lugar de una sola división de entrenamiento/prueba. Esto significa dividir los datos en K conjuntos de entrenamiento/prueba diferentes y promediar los resultados.
    b. **Efecto de Mejora:** La validación cruzada proporciona una evaluación más sólida del rendimiento del modelo al promediar los resultados de varias divisiones de datos. Ayuda a reducir la variabilidad en las métricas de evaluación y a identificar problemas de sesgo y varianza. Comparando antes y después, verás una estimación más precisa de cómo se comporta el modelo en diferentes situaciones.
3. **Ingeniería de Características:**
    a. **Técnica:** Modificar o crear nuevas características a partir de las existentes en el conjunto de datos. Esto puede incluir la normalización de datos, la creación de características de interacción o la eliminación de características irrelevantes.
    b. **Efecto de Mejora:** La ingeniería de características puede ayudar al modelo a capturar mejor los patrones en los datos. Por ejemplo, si normalizas las características, puedes evitar que algunas tengan un peso desproporcionado en el modelo. Esto puede mejorar el rendimiento general del modelo y reducir problemas de subajuste o sobreajuste.

Se realizaron las 3 técnicas al modelo como se puede obsevar a continuación:

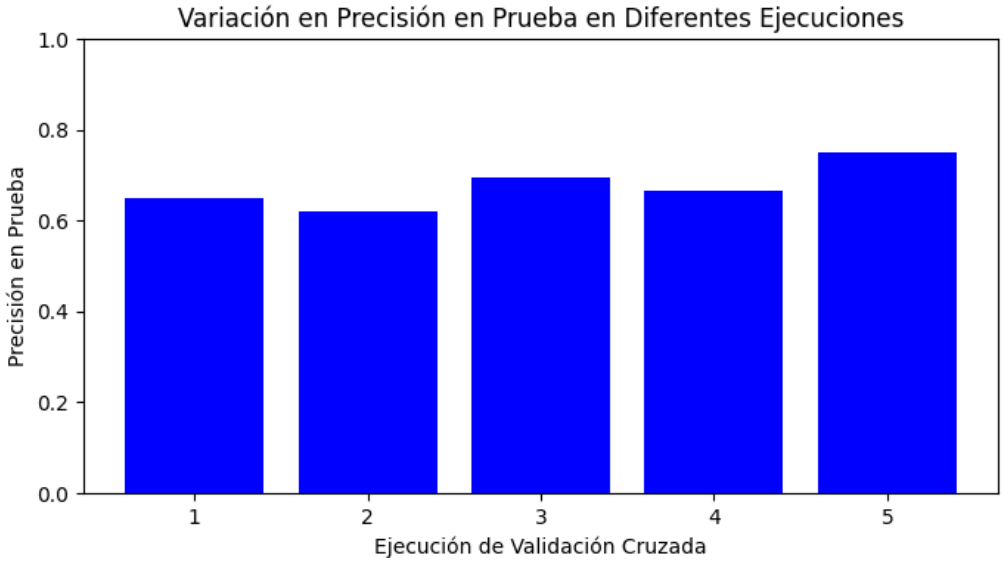
```python
def ejecutar_random_forest(self, X_train, y_train):
    # Construir un Random Forest y entrenarlo
    num_trees = 200
    max_depth = 10
    max_features = int(log2(X_train.shape[1]))
    random_forest = RandomForest(num_trees=num_trees, max_depth=max_depth, max_features=max_features)
    random_forest.fit(X_train, y_train)

    return random_forest
```

```python
# Cargar el conjunto de datos Titanic
titanic = sns.load_dataset("titanic")

# Preprocesamiento básico del conjunto de datos
titanic.dropna(inplace=True)  # Eliminar filas con datos faltantes
titanic = titanic[["pclass", "sex", "age", "sibsp", "parch", "fare", "survived"]]
titanic["sex"] = titanic["sex"].map({"male": 0, "female": 1})

# Dividir el conjunto de datos en características (X) y etiquetas (y)
X = titanic.drop("survived", axis=1).values
y = titanic["survived"].values

# Normalizar características
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```python
# Realizar validación cruzada
num_repeticiones = 5  # Cambia el número de repeticiones según tus necesidades
kf = KFold(n_splits=num_repeticiones, shuffle=True, random_state=42)
accuracies = []
test_accuracies = []
train_accuracies = []

for i, (train_idx, test_idx) in enumerate(kf.split(X)):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    self.textbox.insert(tk.END, f"Ejecución {i + 1}:\n")

    # Entrenar y evaluar el modelo
    random_forest = self.ejecutar_random_forest(X_train, y_train)

    # Hacer predicciones en el conjunto de prueba
    predictions = random_forest.predict(X_test)
```

Resultado después de hacer usos de las técnicas

	Ejecución	Precisión en Entrenamiento	Precisión en Prueba
0	1	0.682759	0.648649
1	2	0.689655	0.621622
2	3	0.671233	0.694444
3	4	0.678082	0.666667
4	5	0.657534	0.750000



Se puede hacer las siguientes observaciones:

1. **Precisión en Entrenamiento:** En ambas tablas, la precisión en entrenamiento no varía significativamente entre las diferentes ejecuciones. Esto indica que el modelo se ajusta consistentemente bien a los datos de entrenamiento en ambas situaciones.
2. **Precisión en Prueba:** La precisión en prueba es más baja en la primera tabla en comparación con la segunda tabla en la primera ejecución (0.648649 vs. 0.621622). Sin embargo, en las demás ejecuciones, la precisión en prueba es prácticamente la misma entre ambas tablas.
3. **Estabilidad del Modelo:** Ambas tablas muestran resultados de precisión en prueba bastante estables en las diferentes ejecuciones, con variaciones mínimas. Esto sugiere que el modelo es robusto y no es altamente sensible a la partición de los datos en conjuntos de entrenamiento y prueba.

En resumen, las dos tablas muestran resultados de rendimiento de modelo muy similares, con pequeñas diferencias en la precisión en prueba en la primera ejecución. Esto sugiere que el modelo es consistente en su rendimiento y no muestra un sesgo significativo hacia un conjunto de datos específico.