# Proyecto Claims Severity Prediction

# ENTREGA FINAL


**INTEGRANTES:**

YOHEL OSVALDO PEREZ GARCIA

TATIANA ELIZABETH SÁNCHEZ SANIN

DANIELA ANDREA PAVAS BEDOYA

**MATERIA:**

Introducción a la Inteligencia Artificial para las Ciencias e Ingenierías

**Profesor:**

RAUL RAMOS POLLAN

UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERÍA

2023

## Introducción:

En la industria de seguros, evaluar la seriedad de un reclamo es crucial para determinar una compensación adecuada y justa para los asegurados.

Para mejorar la precisión de esta evaluación, se inició un proyecto de aprendizaje automático que se centra en el conjunto de datos para predecir la gravedad del daño.

El propósito de este informe es presentar los resultados y conclusiones obtenidos durante el desarrollo de este proyecto. Para hacer esto, utilizamos un conjunto de técnicas y algoritmos de aprendizaje automático para crear un modelo predictivo que puede estimar con precisión la gravedad de un reclamo.
El conjunto de datos utilizado en este proyecto contiene varios atributos relacionados con las reclamaciones, como información del seguro, naturaleza del accidente, características del vehículo (en caso de reclamaciones de automóviles), etc. Este conjunto de datos se ha recopilado durante varios años, lo que proporciona una base sólida para el análisis y la predicción. construcción del modelo.

En las secciones siguientes de este informe, se presentarán detalladamente el proceso de preparación de los datos, la metodología utilizada para el desarrollo del modelo predictivo, así como los resultados y las conclusiones obtenidas.

## Preprocesado de datos

Se realiza una evaluación de la base de datos para determinar si la base de datos de train y test son idénticas. Esto con el fin de evaluar la necesidad de manipulación de los datos.

```
train_features = list(train.columns.values)
train_features.pop() #pop out the loss column
test_features = list(test.columns.values)
print('Son los datasets identicos ? ' + str(train_features == test_features))
```

Se evalúa también si hay clases presentes en test que falten en train y viceversa

## Codificación de datos categóricos y cuantitativos

Luego se hizo codificación de los datos categóricos a.ravés de la función fit_tranform.

Calculamos la asimetría de las columnas continuas utilizando la función skew de scipy.stats. Esto nos ayuda a identificar la distribución de los datos. Aplicamos la transformación Box-Cox a las columnas continuas para corregir la asimetría.

Por último, dividimos los datos en conjuntos de entrenamiento y prueba utilizando train_test_split de scikit-learn.

▾ Encoder

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

```
[ ] categorical_ = [feature for feature in Train_Test.columns if 'cat' in feature]
    continuous_ = [feature for feature in Train_Test.columns if 'cont' in feature]

    #cat_feature = [n for n in joined.columns if n.startswith('cat')]
    #cont_feature = [n for n in joined.columns if n.startswith('cont')]
```

```
[ ] for feature in categorical_: Train_Test[feature] = le.fit_transform(Train_Test[feature])
```

```
[ ] from scipy.stats import skew,boxcox
```

```
[ ] skewed_box = Train_Test.loc[:,"cont1":"cont14"].apply(lambda x: skew(x))
    skewed_box
```

## Modelos

Se tuvieron en consideración varios modelos para el problema planteado y se evaluó su precisión. Entre estos estuvieron XGBoost, Gradient Boosting Regressor, entre otros. Finalmente se llegó a un modelo con xgboost Dmatrix.

### XGBoost:

Está basado en el algoritmo de Gradient Boosting y utiliza árboles de decisión como sus estimadores bases. El XGBRegressor es conocido por su eficiencia y capacidad para manejar conjuntos de datos grandes y complejos.

El modelo utiliza el principio del refuerzo mediante la construcción iterativa de árboles de decisión, donde cada árbol se ajusta a los errores residuales del modelo anterior. A medida que se agregan más árboles, se van corrigiendo los errores y se mejora la capacidad de predicción.

En este modelo se define un diccionario params que contiene los hiperparámetros que se probarán en el modelo XGBoost. Se crea un objeto GridSearchCV que realizará una búsqueda exhaustiva de los mejores hiperparámetros para el modelo. Se utiliza una validación cruzada de 3 folds, una métrica de evaluación de error cuadrático medio negativo (neg_mean_squared_error), se ejecuta en todos los núcleos disponibles (n_jobs=-1) y se muestra información detallada durante el proceso (verbose=True).

Se ajusta el modelo utilizando los datos de entrenamiento (X_train y y_train) con la búsqueda de cuadrícula. Se imprime el mejor conjunto de hiperparámetros encontrados por la búsqueda de cuadrícula.

Se crea un nuevo objeto XGBRegressor con los mejores hiperparámetros encontrados y se ajusta al conjunto de entrenamiento. Se realizan predicciones en el conjunto de prueba y se aplica la función np.expm1() para invertir una transformación logarítmica previa.

Se calcula el error absoluto medio (mean_absolute_error) entre las predicciones y los valores reales del conjunto de prueba.

XGBoost

```
[ ]  1 xg = XGBRegressor()
```

```
1 prams={
2     'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
3      'n_estimators':[100,200,500,1000,2000],
4 }
```

```
[ ]  1 xgb_grid = GridSearchCV(xg,prams,cv = 3,scoring="neg_mean_squared_error",n_jobs = -1,verbose=True)
```

```
[ ]  1 xgb_grid.fit(X_train, y_train)
```

```
[ ]  1 results = pd.DataFrame.from_dict(xgb_grid.cv_results_)
```

```
[ ]  1 results.head(2)
```

```
[ ]  1 print(xgb_grid.best_params_)
```

```
1 final_xg = XGBRegressor(base_score=0.5, booster='gbtree',
2                                     colsample_bylevel=1, colsample_bynode=1,
3                                     colsample_bytree=1, gamma=0,
4                                     importance_type='gain', learning_rate=0.1,
5                                     max_delta_step=0, max_depth=3,
6                                     min_child_weight=1, missing=None,
7                                     n_estimators=2000, n_jobs=1, nthread=None,
8                                     objective='reg:linear', random_state=0,
9                                     reg_alpha=0, reg_lambda=1,
10                                    scale_pos_weight=1, seed=None, silent=None,
11                                    subsample=1, verbosity=1).fit(X_train, y_train)
```

Este código utiliza la búsqueda de cuadrícula para encontrar los mejores hiperparámetros para el modelo XGBoost y luego entrena el modelo con esos hiperparámetros optimizados. Finalmente, realiza predicciones en el conjunto de prueba y calcula el error absoluto

### Gradient Boosting Regressor:

El Gradient Boosting Regressor es un algoritmo de aprendizaje automático utilizado para problemas de regresión, donde el objetivo es predecir valores numéricos continuos. Se basa en el concepto de ensemble learning, que combina varios modelos más débiles para formar un modelo más robusto y preciso. Fue considerado por su tolerancia al sobreajuste.

Se define un diccionario params que contiene los hiperparámetros que se probarán en el modelo Gradient Boosting. Se crea un objeto GridSearchCV que realizará una búsqueda exhaustiva de los mejores hiperparámetros para el modelo Gradient Boosting. Se utiliza una validación cruzada de 3 folds, una métrica de evaluación de error cuadrático medio negativo (neg_mean_squared_error), se ejecuta en todos los núcleos disponibles (n_jobs=-1) y se muestra información detallada durante el proceso (verbose=True):

Se crea un nuevo objeto GradientBoostingRegressor con los mejores hiperparámetros encontrados y se ajusta al conjunto de entrenamiento. Se realizan predicciones en el conjunto de prueba y se aplica la función np.expm1() para invertir una transformación logarítmica previa.

Se calcula el error absoluto medio (mean_absolute_error) entre las predicciones y los valores reales del conjunto de prueba

## Gradient Boosting Regressor

```
[ ]
GBR = GradientBoostingRegressor()
prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[1,2,4],
     'subsample':[.5,.75,1],
     'random_state':[1]
}
```

```
[ ]
gbr_grid = GridSearchCV(GBR,prams,cv = 3,scoring="neg_mean_squared_error",n_jobs = -1,verbose=True)
```

```
[ ]
gbr_grid.fit(X_train, y_train)
```

```
[ ]
print(gbr_grid.best_params_)
```

```
[ ]
GBR_final = GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0,
                                      criterion='friedman_mse',
                                      init=None, learning_rate=0.05,
                                      loss='ls', max_depth=4,
                                      max_features=None,
                                      max_leaf_nodes=None,
                                      min_impurity_decrease=0.0,
                                      min_impurity_split=None,
                                      min_samples_leaf=1,
                                      min_samples_split=2,
                                      min_weight_fraction_leaf=0.0,
                                      n_estimators=2000,
                                      subsample=0.75, tol=0.0001,
                                      validation_fraction=0.1,
```

**Lightbm:**

Es una implementación eficiente y de alto rendimiento del algoritmo de aumento de gradiente (gradient boosting). LightGBM se destaca por su capacidad para manejar conjuntos de datos grandes y complejos, y por su velocidad de entrenamiento y predicción.

Se define un diccionario params que contiene los hiperparámetros que se probarán en el modelo LGBMRegressor. Se crea un objeto RandomizedSearchCV que realizará una búsqueda aleatoria de los mejores hiperparámetros para el modelo LGBMRegressor. Se utiliza una validación cruzada de 3 folds, una métrica de evaluación de error cuadrático medio negativo (neg_mean_squared_error), se ejecuta en todos los núcleos disponibles (n_jobs=-1) y se muestra información detallada durante el proceso (verbose=True). Se imprime el mejor conjunto de hiperparámetros encontrados por la búsqueda aleatoria.

Se carga un conjunto de datos de entrenamiento (Train_Data) y un conjunto de datos de prueba (Test_Data) desde los archivos CSV. Se eliminan las columnas 'id' y 'loss' del conjunto de datos de entrenamiento (Train_Data). Se eliminan la columna 'id' del conjunto de datos de prueba (Test_Data). Se concatenan los conjuntos de datos de entrenamiento y prueba

en un solo DataFrame llamado Train_Test.

Se divide el conjunto de datos transformado nuevamente en los conjuntos de datos de entrenamiento (Train_Data) y prueba (Test_Data). Se dividen los datos de entrenamiento (Train_Data) en conjuntos de entrenamiento y prueba utilizando train_test_split. El 80% de los datos se utiliza para entrenamiento (X_train e y_train) y el 20% se reserva para pruebas (X_test e y_test)

▾ Lightbm

```
[ ]  1 lgb_r = LGBMRegressor(max_depth=-1, random_state=314, n_estimators=5000)
     2 params = {
     3   'boosting_type': ['gbdt','dart','goss','rf'],
     4   'metric': ['l2', 'l1'],
     5   'min_child_weight': [1e-5, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3, 1e4],
     6   'learning_rate': [0.01,0.03,0.05,0.1,0.15,0.2],
     7   'reg_alpha': [1e-2, 1e-1, 1, 1e1, 1e2],
     8   'reg_lambda': [1e-2, 1e-1, 1, 1e1, 1e2]
     9 }
    10 lgbmr_rs = RandomizedSearchCV(lgb_r,params,cv = 3,scoring="neg_mean_squared_error",n_job s = -1,verbose=True)
    11 lgbmr_rs.fit(X_train, y_train)
```

```
[ ]  1 print(lgbmr_rs.best_params_)
```

```
[ ]  1 Train_Data = pd.read_csv('train.csv')
     2 Test_Data = pd.read_csv('test.csv')
     3 r,c = Train_Data.shape   # r -> rows and c --> columns # 188318 and 13
     4 Train_Data.drop(['id','loss'], axis=1, inplace=True True)
     5 Test_Data.drop(['id'], axis=1, inplace=True True)
     6 Train_Test = pd.concat((Train_Data, Test_Data)).reset_index(drop=True True)
     7
     8 categorical_ = [feature for for feature in in Train_Test.columns if if 'cat' in in feature]
     9 continuous_ = [feature for for feature in in Train_Test.columns if if 'cont' in in feature]
    10
    11 for feature in in categorical_: Train_Test[feature] = le.fit_transform(Train_Test[feature])
    12
    13 Train_Test["cont1"], lam = boxcox(Train_Test["cont1"] + 1)
    14 Train_Test["cont2"], lam = boxcox(Train_Test["cont2"] + 1)
    15 Train_Test["cont4"], lam = boxcox(Train_Test["cont4"] + 1)
    16 Train_Test["cont5"], lam = boxcox(Train_Test["cont5"] + 1)
    17 Train_Test["cont6"], lam = boxcox(Train_Test["cont6"] + 1)
    18 Train_Test["cont7"], lam = boxcox(Train_Test["cont7"] + 1)
```

Con esto se logró que el error absoluto medio como 1136, pero en el archivo final se obtuvo una puntuación de 1115,11.

Este puntaje cae exactamente en la posición 261 de 3045 entradas. Entonces, en última instancia, el puntaje que obtenido está en el rango del 10% superior.

## Retos y Consideraciones:

Durante el desarrollo del proyecto de Machine Learning sobre la predicción de la gravedad de reclamos, se han enfrentado diversos retos y consideraciones que han influido en el proceso y los resultados obtenidos. A continuación, se presentan algunos de los principales desafíos y aspectos a tener en cuenta:

1. Calidad y disponibilidad de los datos: Uno de los desafíos iniciales es garantizar la calidad y la disponibilidad de los datos utilizados en el proyecto. Esto implica enfrentar posibles problemas de integridad, consistencia y precisión en el dataset, así como asegurar que se cuente con la cantidad suficiente de registros para un análisis adecuado.

2. Desbalance de clases: Es común que en el dataset de predicción de la gravedad de reclamos exista un desbalance significativo entre las clases de reclamos leves y graves. Este desbalance puede afectar el rendimiento del modelo, ya que este puede tender a favorecer la clase mayoritaria. Es fundamental aplicar técnicas de muestreo estratificado o de ajuste de pesos para abordar este desafío.

3. Selección de características relevantes: El dataset de reclamos puede contener una amplia gama de atributos, pero no todos ellos son necesariamente relevantes para predecir la gravedad. La identificación y selección de las características más informativas es un desafío importante, ya que puede influir en la precisión y el rendimiento del modelo. Se requiere un análisis exhaustivo y experto para determinar qué atributos son más relevantes.

4. Sobreajuste (overfitting) y generalización del modelo: En el proceso de desarrollo del modelo, es fundamental evitar el sobreajuste, es decir, que el modelo se ajuste demasiado a los datos de entrenamiento y no pueda generalizar correctamente en nuevos datos. Es necesario aplicar técnicas de validación cruzada y ajuste adecuado de hiperparámetros para encontrar un equilibrio entre la capacidad de ajuste y la generalización del modelo.

5. Interpretación y explicabilidad del modelo: A medida que se desarrolla un modelo de Machine Learning más complejo, como los algoritmos de ensemble, puede resultar desafiante comprender y explicar las decisiones tomadas por el modelo. La interpretación y explicabilidad del modelo son aspectos importantes, especialmente en el contexto de las industrias aseguradoras, donde se requiere justificar las predicciones y proporcionar transparencia.

6. Actualización y mantenimiento del modelo: Una vez implementado el modelo de predicción de la gravedad de reclamos, es importante considerar su mantenimiento y actualización a medida que se disponga de nuevos datos. Los datos cambian con el tiempo y el modelo puede requerir ajustes periódicos para mantener su rendimiento óptimo.

## Conclusiones:

En conclusión, el proyecto de Machine Learning sobre la predicción de la gravedad de reclamos ha demostrado ser prometedor y ofrece importantes beneficios para el sector asegurador. El desarrollo de un modelo predictivo preciso y eficiente puede mejorar significativamente la evaluación de reclamos, permitiendo una asignación más justa de indemnizaciones y una mejor experiencia tanto para las compañías de seguros como para los asegurados. Sin embargo, es importante tener en cuenta los retos asociados, como el desbalance de clases y la interpretación del modelo, para garantizar su efectividad y aplicabilidad en la práctica. Este proyecto sienta las bases para futuros trabajos en la mejora continua del modelo y la adaptación a las cambiantes necesidades del sector asegurador.