Manejar el Desequilibrio con SMOTE

¿Cuántas veces obtuvo un modelo con una precisión del 99% que no se puede utilizar? Crear modelos de clasificación no es una broma, especialmente cuando hay un desequilibrio de clases en sus datos. Ya sabes, cuando solo hay un fraude en 1000 transacciones.

Verá, identificar 950 de 999 transacciones genuinas es fácil. El truco consiste en identificar correctamente un solo caso de fraude, cada vez.

¿Leer se siente como una pesadilla? Hay una solución sencilla:

Ahí es donde SMOTE (Técnica de sobremuestreo de *minorías sintéticas* ) resulta útil. Puede usarlo para sobremuestrear la clase minoritaria. SMOTE es un tipo de aumento de datos que sintetiza nuevas muestras a partir de las existentes.

Sí, SMOTE en realidad crea nuevas muestras. Está a años luz de la simple duplicación de la clase minoritaria. Ese enfoque crea estúpidamente "nuevos" puntos de datos al duplicar los existentes. Como resultado, no se trae nueva información al conjunto de datos.

**Pero, ¿cómo lo hace SMOTE?**

Selecciona muestras de la clase minoritaria que están cerca y luego traza líneas entre ellas. Los nuevos puntos de muestra se encuentran en estas líneas.

Para ser más precisos, se elige una muestra aleatoria y luego se usa un algoritmo KNN para seleccionar vecinos a los que se dibujan las líneas. Con este procedimiento, puede crear tantas muestras sintéticas como necesite. Esto hace que SMOTE sea perfecto para conjuntos de datos de todos los tamaños.

El único inconveniente real es que los ejemplos sintéticos se crean sin "consultar" a la clase mayoritaria. Esto podría resultar en muestras superpuestas en ambas clases.

Esa es la única teoría que necesitará para comprender este artículo.

Estos son los temas que se tratan a continuación:

* Carga y preparación de conjuntos de datos
* Aprendizaje automático sin SMOTE
* Aprendizaje automático con SMOTE
* Conclusión

Por lo tanto, necesita un conjunto de datos de clasificación que sufra un problema de desequilibrio de clases. Algo como la detección de fraudes con tarjetas de crédito debería funcionar. [Aquí](https://www.kaggle.com/dark06thunder/credit-card-dataset) tienes uno de Kaggle que puedes descargar gratis.

He aquí cómo cargarlo con Python:

|  |  |
| --- | --- |
|  | import numpy as np |
|  | import pandas as pd |
|  | import matplotlib.pyplot as plt |
|  | import seaborn as sns |
|  |  |
|  | df = pd.read\_csv('credit\_dataset.csv') |
|  | df.head() |
|  |  |

Graphical user interface, application

Description automatically generated

Imagen 1 - Jefe de conjunto de datos de fraude con tarjetas de crédito (imagen del autor)

Hay veinte y tantos columnas que preparará en un momento. Primero, exploremos la distribución de la clase objetivo:

|  |  |
| --- | --- |
|  | ax = df['TARGET'].value\_counts().plot(kind='bar', figsize=(10, 6), fontsize=13, color='#087E8B') |
|  | ax.set\_title('Credit card fraud (0 = normal, 1 = fraud)', size=20, pad=30) |
|  | ax.set\_ylabel('Number of transactions', fontsize=14) |
|  |  |
|  | for i in ax.patches: |
|  | ax.text(i.get\_x() + 0.19, i.get\_height() + 700, str(round(i.get\_height(), 2)), fontsize=15) |

Chart, bar chart

Description automatically generated

Ay! Solo alrededor del 1,68% de las transacciones se clasifican como fraude.

Una gran receta para hacer modelos de **baja recuperación de alta precisión** .

Preparación de datos

Lista de cambios iniciales:

* Reasignar gender, cary reality a enteros (0, 1): estas columnas solo tienen dos valores posibles
* Crear variables ficticias para income\_type, education\_type, family\_name, house\_type- de cadenas a binario (0, 1)
* Elimine columnas innecesarias - Unnamed: 0, ID y todas las columnas para las que creó variables ficticias
* Combinar todo en un solo marco de datos

|  |  |
| --- | --- |
|  | **# Remap to integers** |
|  | df['GENDER'] = [0 if x == 'M' else 1 for x in df['GENDER']] |
|  | df['CAR'] = [1 if x == 'Y' else 0 for x in df['CAR']] |
|  | df['REALITY'] = [1 if x == 'Y' else 0 for x in df['REALITY']] |
|  |  |
|  | **# Create dummy variables** |
|  | dummy\_income\_type = pd.get\_dummies(df['INCOME\_TYPE'], prefix='INC\_TYPE', drop\_first=True) |
|  | dummy\_edu\_type = pd.get\_dummies(df['EDUCATION\_TYPE'], prefix='EDU\_TYPE', drop\_first=True) |
|  | dummy\_family\_type = pd.get\_dummies(df['FAMILY\_TYPE'], prefix='FAM\_TYPE', drop\_first=True) |
|  | dummy\_house\_type = pd.get\_dummies(df['HOUSE\_TYPE'], prefix='HOUSE\_TYPE', drop\_first=True) |
|  |  |
|  | **# Drop unnecessary columns** |
|  | to\_drop = ['Unnamed: 0','ID','FLAG\_MOBIL','INCOME\_TYPE','EDUCATION\_TYPE','FAMILY\_TYPE','HOUSE\_TYPE'] |
|  |  |
|  | df.drop(to\_drop, axis=1, inplace=True) |
|  |  |
|  | **# Merge into a single data frame** |
|  | merged = pd.concat([df, dummy\_income\_type, dummy\_edu\_type, dummy\_family\_type, dummy\_house\_type], axis=1) |
|  | merged.head() |

El conjunto de datos ahora se ve así:

Calendar

Description automatically generated with medium confidence

Pero todavía necesita un poco de trabajo. Observe cuán mayores son los valores en income que en no\_of\_child. Eso es lo esperado, pero los algoritmos de aprendizaje automático darán más importancia a las variables a mayor escala.

**Escalado de datos** .

Utilizará MinMaxScalerde scikit-learnpara escalar columnas que tengan valores superiores a 1 en el rango [0, 1]. Aquí 'cómo:

|  |  |
| --- | --- |
|  | from sklearn.preprocessing import MinMaxScaler |
|  |  |
|  | **# Scale only columns that have values greater than 1** |
|  | to\_scale = [col for col in df.columns if df[col].max() > 1] |
|  | mms = MinMaxScaler() |
|  | scaled = mms.fit\_transform(merged[to\_scale]) |
|  | scaled = pd.DataFrame(scaled, columns=to\_scale) |
|  |  |
|  | **# Replace original columns with scaled ones** |
|  | for col in scaled: |
|  | merged[col] = scaled[col] |
|  |  |
|  | merged.head() |

Así es como se ve el conjunto de datos ahora:

Calendar

Description automatically generated with low confidence

Mucho mejor: todo está en el rango [0, 1], todas las columnas son numéricas y no faltan valores.

Esto significa una cosa: el conjunto de datos está listo para el aprendizaje automático.

Aprendizaje automático sin SMOTE

Comencemos con un enfoque ingenuo. Creará un modelo de bosque aleatorio en el conjunto de datos e ignorará por completo el desequilibrio de clases.

Para comenzar, tendrá que dividir el conjunto de datos en partes de entrenamiento y prueba. Solo hay un 1,68% de las transacciones fraudulentas en todo el conjunto de datos. Idealmente, desea que el porcentaje sea aproximadamente el mismo en el tren y en los conjuntos de prueba.

A continuación, le indicamos cómo dividir y verificar el porcentaje de la clase positiva:

|  |  |
| --- | --- |
|  | from sklearn.model\_selection import train\_test\_split |
|  |  |
|  | X = merged.drop('TARGET', axis=1) |
|  | y = merged['TARGET'] |
|  |  |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split( |
|  | X, y, test\_size=0.25, random\_state=42 |
|  | ) |
|  |  |
|  | print(f'''% Positive class in Train = {np.round(y\_train.value\_counts(normalize=True)[1] \* 100, 2)} |
|  | % Positive class in Test = {np.round(y\_test.value\_counts(normalize=True)[1] \* 100, 2)}''') |



Porcentaje de clase positiva en trenes y conjuntos de prueba

Hagámoslo lo más simple posible. Entrenarás un clasificador de **Random Forest** en el set de trenes y lo evaluarás en el set de prueba. La matriz de confusión, la puntuación de precisión y la puntuación de recuerdo le dirán qué tan malo es:

|  |  |
| --- | --- |
|  | from sklearn.ensemble import RandomForestClassifier |
|  | from sklearn.metrics import accuracy\_score, recall\_score, confusion\_matrix |
|  |  |
|  | # Train |
|  | model = RandomForestClassifier(random\_state=42) |
|  | model.fit(X\_train, y\_train) |
|  | preds = model.predict(X\_test) |
|  |  |
|  | # Evaluate |
|  | print(f'Accuracy = {accuracy\_score(y\_test, preds):.2f}\nRecall = {recall\_score(y\_test, preds):.2f}\n') |
|  | cm = confusion\_matrix(y\_test, preds) |
|  | plt.figure(figsize=(8, 6)) |
|  | plt.title('Confusion Matrix (without SMOTE)', size=16) |
|  | sns.heatmap(cm, annot=True, cmap='Blues'); |

Chart

Description automatically generated

Imagen 6: matriz de precisión, recuperación y confusión de un modelo sin usar SMOTE

**El modelo tiene una precisión del 98%, así que ¿dónde está el problema?**

Sí, puede clasificar correctamente casi todas las transacciones genuinas. Pero también clasificó el 91% de las transacciones fraudulentas como genuinas. En pocas palabras, el modelo es inutilizable.

El desequilibrio de clases mató su desempeño. SMOTE puede ayudar.

Aprendizaje automático con SMOTE

Ya sabes qué es SMOTE, y ahora verás cómo instalarlo y usarlo. Ejecute el siguiente comando desde la Terminal:

pip install imbalanced-learn

|  |  |
| --- | --- |
|  | from imblearn.over\_sampling import SMOTE |
|  |  |
|  | sm = SMOTE(random\_state=42) |
|  |  |
|  | X\_sm, y\_sm = sm.fit\_resample(X, y) |
|  |  |
|  | print(f'''Shape of X before SMOTE: {X.shape} |
|  | Shape of X after SMOTE: {X\_sm.shape}''') |
|  |  |
|  | print('\nBalance of positive and negative classes (%):') |
|  | y\_sm.value\_counts(normalize=True) \* 100 |

Text

Description automatically generated

Imagen 7 - Formas y equilibrio de clases después de aplicar SMOTE

Hay 37.000 puntos de datos en lugar de 25.000 y el equilibrio de clases es perfecto: 50:50. A continuación, entrenará el modelo en un nuevo conjunto de datos:

|  |  |
| --- | --- |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split( |
|  | X\_sm, y\_sm, test\_size=0.25, random\_state=42 |
|  | ) |
|  |  |
|  | model = RandomForestClassifier(random\_state=42) |
|  | model.fit(X\_train, y\_train) |
|  | preds = model.predict(X\_test) |
|  |  |
|  | print(f'Accuracy = {accuracy\_score(y\_test, preds):.2f}\nRecall = {recall\_score(y\_test, preds):.2f}\n') |
|  | cm = confusion\_matrix(y\_test, preds) |
|  | plt.figure(figsize=(8, 6)) |
|  | plt.title('Confusion Matrix (with SMOTE)', size=16) |
|  | sns.heatmap(cm, annot=True, cmap='Blues'); |

Chart, treemap chart

Description automatically generated

Imagen 8 - Matriz de precisión, recuperación y confusión de un modelo después de usar SMOTE

El modelo resultante es utilizable, por decir lo menos. SMOTE hizo su trabajo y dio como resultado un modelo que superó significativamente a su versión anterior.

Conclusión

SMOTE en pocas palabras. Puede usarlo siempre que un conjunto de datos sufra un problema de desequilibrio de clases. El método de referencia hoy en día es utilizar tanto el submuestreo como el sobremuestreo, pero ese es un tema para otro momento.

Solo por diversión, puede comparar las clasificaciones erróneas de ambos modelos. Al hacerlo, podría ver si el modelo creado después del sobremuestreo aún clasifica erróneamente los mismos puntos de datos.