**Instagram Profile Data Analysis and Classification**

**Introduction**

In the realm of modern data analysis, classification problems have become increasingly significant, particularly in social media analytics. The project at hand focuses on analyzing and categorizing Instagram profiles based on textual data derived from their biography, captions, and metadata. This effort is structured across three primary notebooks: **models.ipynb**, **processing.ipynb**, and **selected\_model.ipynb**, each contributing to distinct phases of the analysis pipeline.

**Objective**

The core objective of this project is to build an efficient machine learning pipeline capable of categorizing Instagram profiles into predefined categories based on their textual data. The workflow involves:

1. **Data Processing**: Cleaning and preparing raw data to make it suitable for analysis.
2. **Feature Extraction**: Transforming processed text into numerical representations using techniques such as TF-IDF.
3. **Model Training and Selection**: Experimenting with various machine learning models to identify the best-performing approach.
4. **Prediction**: Using the trained model for classifying unseen data.

**Tools and Libraries**

A variety of Python libraries and tools are employed throughout the project to streamline data analysis, machine learning, and model evaluation tasks. Some of the key libraries include:

* **pandas**: For handling and manipulating structured data.
* **NumPy**: To perform numerical operations efficiently.
* **scikit-learn**: Used for feature extraction, model training, evaluation, and hyperparameter tuning.
* **imblearn**: Provides tools like SMOTE (Synthetic Minority Oversampling Technique) to address class imbalance in the dataset.
* **joblib**: For saving and loading trained models and preprocessed objects like vectorizers.
* **TF-IDF Vectorizer**: Converts text data into numerical form based on the Term Frequency-Inverse Document Frequency approach, enabling machine learning algorithms to work with textual data.
* **Jupyter Notebook**: Used as the primary development environment for writing and iterating on Python code interactively.

**Structure of the Notebooks**

1. **processing.ipynb**:
   * Focuses on loading raw data, cleaning textual content, and preparing it for feature extraction.
   * Includes detailed steps for text preprocessing, tokenization, and stopword removal.
2. **models.ipynb**:
   * Covers feature extraction using TF-IDF and a comprehensive comparison of machine learning models like Random Forest, Support Vector Machine, k-NN, Naive Bayes, and Logistic Regression.
   * Explores hyperparameter tuning to optimize the performance of the models.
3. **selected\_model.ipynb**:
   * Delves into the best-performing model, Logistic Regression with a One-vs-Rest (OvR) strategy.
   * Demonstrates predictions using unseen data and discusses how the trained model can be reused in real-world applications.

**1. Data Processing (processing.ipynb)**

The preprocessing phase is crucial for transforming raw textual data into a structured format that machine learning models can interpret. The processing.ipynb notebook handles this by:

1. **Loading Data**:
   * Data is loaded from a .gz file, containing Instagram profile information such as username, biography, captions, and category.
2. **Text Cleaning and Preprocessing**:
   * Text fields are concatenated (e.g., username, biography, captions, etc.).
   * Preprocessing steps include:
     + Tokenization: Splitting text into individual words.
     + Removing non-alphabetic characters.
     + Lowercasing the text.
     + Lemmatization: Reducing words to their base forms using libraries like trnlp.
     + Stopword Removal: Eliminating common words that do not contribute to the meaning (e.g., "ve", "ile").
3. **Saving Cleaned Data**:
   * The cleaned data is saved to a JSON file (processed\_profiles.json), ensuring reusability for subsequent phases.

This preprocessing stage ensures uniformity in text representation and reduces noise, which is critical for effective feature extraction and model training.

**2. Feature Extraction and Model Comparison (models.ipynb)**

Once the data is processed, the next step is to extract meaningful features and experiment with different machine learning models. This phase is handled in models.ipynb.

1. **Feature Extraction with TF-IDF**:
   * Text data is converted into numerical vectors using a Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer. This approach emphasizes words that are significant within a category while discounting commonly occurring terms.
2. **Handling Class Imbalance with SMOTE**:
   * During initial analysis, it was observed that some categories had significantly fewer data points, leading to imbalanced class distributions.
   * To address this, **SMOTE (Synthetic Minority Oversampling Technique)** was employed. SMOTE generates synthetic samples for underrepresented classes by interpolating between existing samples. This ensures a balanced dataset for training, improving the model's ability to generalize across all categories.
3. **Model Training and Comparison**:
   * Multiple classification algorithms were trained on the balanced dataset:
     + **Random Forest**: A robust ensemble method.
     + **Support Vector Machine (SVM)**: Effective for high-dimensional spaces.
     + **k-Nearest Neighbors (k-NN)**: A simple and interpretable approach.
     + **Naive Bayes**: A probabilistic model suitable for text data.
     + **Logistic Regression**: A linear model with strong interpretability.
     + **Logistic Regression with One-vs-Rest (OvR)**: Extends Logistic Regression for multiclass classification.
     + **Decision Tree**: A rule-based algorithm.
   * Each model was evaluated using accuracy and classification reports.
4. **Hyperparameter Tuning**:
   * GridSearchCV was used to fine-tune hyperparameters for models like Random Forest, SVM, and Logistic Regression.
   * This step ensured optimal configurations for each algorithm, further enhancing accuracy.

The results highlighted that **Logistic Regression with OvR**, combined with SMOTE, achieved the highest accuracy and generalizability.

**3. Selected Model and Predictions (selected\_model.ipynb)**

The final notebook, selected\_model.ipynb, focuses on the best-performing model: **Logistic Regression with OvR**.

1. **SMOTE Threshold Optimization**:
   * The initial threshold for SMOTE was set to 200, but further experimentation revealed that a threshold of 600 produced better results, particularly for minority classes.
2. **Model Training**:
   * Logistic Regression with OvR was retrained on the dataset balanced with a SMOTE threshold of 600.
   * The final model was evaluated using a separate test set, yielding excellent performance metrics.
3. **Saving the Model**:
   * The trained model was saved to a .pkl file (logistic\_regression\_ovr\_smote\_600.pkl) for future use.
   * The corresponding TF-IDF vectorizer was also saved, ensuring that new data could be preprocessed consistently.
4. **Making Predictions**:
   * The model was used to classify new Instagram profiles. Data was selectively taken from processed\_profiles.json and passed through the pipeline.
   * Predictions were saved in prediction-classification-round\*.json, enabling easy interpretation and further analysis.

**Why SMOTE Was Essential**

Class imbalance was a critical issue identified early in the process. Without addressing it, the models would have been biased toward the majority classes, ignoring minority categories. By using SMOTE:

* Minority categories were sufficiently represented in the training data.
* The model's performance on underrepresented classes improved significantly.
* Logistic Regression with OvR, in particular, leveraged SMOTE to achieve balanced and accurate predictions across all categories.

**Conclusion**

The systematic progression from data preprocessing to feature extraction, model comparison, and final selection demonstrates a robust approach to solving the classification problem. The combination of:

* Well-preprocessed text data,
* SMOTE for addressing class imbalance,
* And Logistic Regression with OvR for its simplicity and effectiveness,

has resulted in a pipeline that is both efficient and reliable for categorizing Instagram profiles.