Black text on a white background

Description automatically generated

**Project Report for CSP 571 - Data Preparation & Analysis**

**Building and Deploying a Machine Learning Model with Scikit-Learn and ONNX**

By

Mounika Ayyapu (A20550421)

Sai Kartheek Goli (A20546631)

Uday Kumar Swamy (A2052682)

**Under the Guidance**

of

Mr. Jawahar Panchal

Professor CSP 571

Department of Computer Science

**Abstract**

Gaussian Naive Bayes classifier for multi-class classification using Scikit-learn is developed, trained and deployed using ONNX. The data set for this project is very skewed, which is always a disadvantage when developing the classifier. This imbalance is rectified using the Synthetic Minority Over-sampling Technique (SMOTE) this is a technique that generates samples for the minority class thus reducing the imbalance. Machine learning algorithms, of which the scikit-learn is one of the most well-known data science libraries in Python, contain a variety of methods for their implementation. ONNX or Open Neural Network Exchange is a standard for shared exchange of deep learning models to make their deployment on diverse platforms, especially, on devices preferable for real-time application. This work proposal introduces the new classification pipeline that incorporates data preprocessing, feature space engineering, model learning, and model assessment, as well as the ONNX porting of the learned model. The problem is not only in constructing a good model, but also in its ability to be deployed in resource-constrained environments such as the edge devices that need to run fast whilst consuming low amounts of power. The effectiveness of the ONNX model will also be assessed as suitable for deployment when it is compared with the original Scikit-learn model.

**Introduction**

Classification is one of the most critical activities in a wide variety of fields and sectors, such as healthcare and finance, as well as in selling and marketing, where ML is beginning to be a vital tool. This project will focus on the modeling, building and exporting of a Gaussian Naive Bayes for multi-class classification in Scikit-learn using ONNX. When applied to IO Peripheral devices, the requirements for ML models have become more essential due to the real time prediction and low dependency on cloud resources. The ONNX (Open Neural Network Exchange) format is the most suitable for model deployment because it help to provide a model standard for all platforms.

The dataset used in this work was imbalanced in that some target classes had more instances of data samples than others, and therefore it was challenging to balance the need for producing accurate classification outcomes. The problem like the one mentioned above was usually tackled using methods such as SMOTE (Synthetic Minority Oversampling Technique). It was chosen because of its n fold cross validation and its simplicity and easy to understand because its performances are very high especially when working with multiple classes. The subsequent four sections of this report discuss the process flow of the project in the DPPP framework with particular reference to data preprocessing, model training and evaluation, and model deployment.

**Objectives**

1. Machine learning Development and Training using scikit-learn library

Thus, the next objectives are as follows Develop an ideal classification pipeline which may comprise data preprocessing, feature extraction/reducing, and the classifying Gaussian Naive Bayes model for multi-class M classification. Able out balance of datasets with Class SMOTE to enhance strong model of generalization.

1. The last step looks at the quality of the model and how best to improve on it and or compare with the best current models.

Changing the parameters of the model that is used and how more accurately can predict desired events using for example precision, recall or any other measure like accuracy or ROC AUC score. Finally, to report results, use a confusion matrix or an ROC curve if it f its the type of project.

1. In this approach, ONNX shall be used to deploy the Model at the Edge for inference.

After posting in Scikit-learn, it is essential to export ONNX to allow for execution at the edge. This will be compared to the original pipeline so as to check if it meets the deployment suitability of ONNX.

**Problem Statement**

This work aims to create a high performing classification model that is to make a best next step prediction of one of three targets in a dataset which is highly imbalanced. This model should be feasible on edge devices and its inference is not expected to be sensitive to high processing power as opposed to the usual models. The challenge which will address herein is to achieve the maximum possible performance of the model and the corresponding number of computations, at the same time taking into consideration its compatibility with the training and deployment platforms.

**Relevant Literature**

Machine learning in particular and the uses of the same have continued to develop over the recent past specifically in the model development and deployment platforms such as Scikit-learn and ONNX. The following literature provides an understanding of the methodologies applied throughout this project.

A Micro survey on Scikit-Learn as an ML Framework

Python based library scikit-learn has stayed one of the leading implements to perform machine learning models that bring together a rich number of algorithms of the supervised as well as the unsupervised kind. This is explained by [[1]](#footnote-1) as a characteristic of modularity since it easy the tasks of input formatting, training and model evaluation. This versatility has seen Scikit-learn become an imperative tool in most elements of a machine learning chain normally using the same pipeline and the same result over and over. The compatibility with frameworks like ONNX, it is suitable for this project which systematically integrates the preprocessing, feature dimensionality and classification.

PCA is still used as a live dimensionality reduction method it explains most of the variability and leaves out unnecessary noise. [[2]](#footnote-2), note that under its main thrusts of operation, PCA has the capacity to improving computational and effective saving much importance data while working through the high-dimensional data. The removal of the irrelevant attributes and summarizing reduces the dimensionality of features by PCA which quickens the training and implementation of models that are ordinary an issue on the limited memory and processing capacity of the edge devices. That is the strength, the Source of the Imbalance or How SMOTE Works.

Definitely, class imbalance is one of the most critical challenges that affect many classification problems with the effects of making models shift towards the dominant classes. To deal with this problem [[3]](#footnote-3)Chawla et al introduced in 2002 the Synthetic Minority Over-sampling Technique known as SMOTE. When SMOTE generates new samples for minority classes, it also assists in classifying all classes and provide generality in the model. SMOTE is employed in this study on the data set referred to in this paper to improve and balance classification.

ONNX for Model Deployment

Starting from this point, ONNX has become a standard for companies, researchers, and developers when it comes to exporting and importing the machine learning model.[[4]](#footnote-4) Also discussed general software compatibility across DL frameworks to which ONNX contributes to enhancing deployment processes. ONNX aims at being the check point of using models which are developed in frameworks like Scikit-learn and converting them into models which are deployable on edges and recognizable on several platforms.

However, transforming the machine learning pipelines to ONNX is not such an easy task. [[5]](#footnote-5)Describe potential threats regarding ONNX model conversion, among which are the failures with preprocessing differences, as well as runtime execution failures. Such risks justify the statement that the preprocessing steps in the training loop and in the inference, loop should be aligned, which became the focus of the current project.

The main two challenges are: Quantization; Deployment

In this process to run models on edge devices, quantization techniques which reduce the bit precision values of the weight parameters and activations are employed. Review of quantization in the context of the framework of quantization is provided in Changs’ study[[6]](#footnote-6); this work also presents the impact of employing each type of quantization on the size of the model, its latency, and precision. However, this project does not complete the quantization; the further work can also discuss the ONNX models that might be used to enhance the deployment speed.

ONNX in Emerging Applications

Regarding the particular applications of OFNX, as Naqvi [[7]](#footnote-7) identify new aspects of VR that indicate the flexibility of the model in terms of fields. Their work also proves that there is a possibility of using ONNX in the transfer of lightweight models to low power devices; a direction this project aims at. Such features enshrine utilization of ONNX models in the aspect of design and testing while the concurrent dynamism of the framework acts as evidence of its application value.

Summary of Literature

This knowledge is theoretical and practical and is the basis for the methodologies used in this project as represented in this literature. Since scikit learn literature has found itself efficient in pipeline processing, the application of PCA and SMOTE makes the level even more flattened in a route map for model building. In as much as ONNX is a deployment framework that makes inference very easy spanning across multiple platforms, there is a challenge of mismatch of preprocessing. In summary, these results help in reducing any processes hindering the integration and utilization of the developed machine learning models in the field, thus bridging the gap in the conversion of models from concept to implementation.

**Methodology**

The final part of the Metadata section is devoted to the description of the detailed stages of data preprocessing, EDA and model preparation. These procedures check the data for being ready to be used in model development, find out relationship between the data elements, and get the dataset ready for further processing.

1. **Data Preparation**

As with virtually all data analysis projects, the first thing that needs to be done is the identification of the dependent variable and independent factors. It is a set of several independent variables and a qualitative dependent variable labelled Class which wish to forecast. The methodology starts with separating these components:

Target Variable (y): The column which contains the class labels to be predicted. The column that contains the class labels to be predicted.

Features (X): Every column except the one containing the target variable data that can be used for making predictions.

1.1. Data Splitting

A training set and a testing set were created to further prove that the model was capable of learning from one set of data and apply it to another different set of data. Eighty percent of the data was employed for the training of the classifier while 20 percent was used for testing.

It means that at this step the model is checked on the data it has not practiced on and allows to determine how well it will work in practice.

1.2. Handling Missing Data

Data missing is inevitable when working with datasets and basic intensity need to be given to it. The values in the target variable, Class were also filled up with most frequent mode of the column so that no record is dropped and class distribution is preserved. The choice of this was informed by the fact that using the mode does not distort the data set especially for the categorical target.

Most of the feature columns had missing values which were replaced with the mean equivalent to the corresponding feature columns so that the dataset could be feasible for use in developing the models.

1.3. Preprocessing Pipeline Setup

To make the task easier to work with, the preprocessing part was refactored, and a function list was created with following steps: handling non-existent data, data scaling, and class balance based on the SMOTE method. The pipeline is a set of multiple preprocessing steps that are implemented in the order of the pipeline on the training data set.

Imputation: The outlier values in the constructed features set were replaced with imputed mean values for every column in the input.

Scaling: All data were put through standard scaling with mean equal to 0 and standard deviation equal to 1 since several machine learning algorithms are sensitive to the scale of data.

SMOTE: To learn from imbalanced data, Synthetic Minority Over-sampling Technique (SMOTE) was used for over-sampling of the minority class by synthesizing new samples for the same class.

By combining these steps, were able to preprocess the data in a way such that it was ready for model training and make sure that weren’t leaking data into the model training phase.

**2. Data Transformation is also Feature Engineering**

After splitting the data and handling missing values, several transformations have been made to cleaning up the data to prepare for modeling.

2.1. In this research, the target variable is assumed in cases where it is missing and hence the term imputation.

For the NAs in the target variable, [Class], the most frequently occurring class was used to keep the class distribution so there was no information loss during data preprocessing.

2.2. Feature Scaling

Feature scaling was also done to meet the requirement whereby all features should have an equal contribution to the created model. This work also standardized the features using standard scaling formula such that the values range from 0 to 1 for every feature. This step is important in the process of learning because it prevents one feature from overwhelming the entire learning phase because of its size.

2.3. Combing Class Imbalance with SMOTE

However, to overcome the limiting factor of the class imbalance problem in our data set, the SMOTE algorithm was employed in creating synthetic instances of the minority class. This technique enabled to mitigate and reduce class imbalance, in that the accuracy was improved during training, and that the model saw a fairer distribution of all classes.

To ensure that oversampling has been done effectively, the class distribution was analyzed before and after applying SMOTE.

**3. Exploratory Data Analysis (EDA).**

Two features of EDA should be noted Every aspect of data analysis begins with EDA provides a picture of the structure and relationships in the data. It entails graphic representations of features and their disposition with a view of determining relationships, looking for anomalies and searching for correlation.

3.1. Univariate Analysis

Single feature distributions were explored with histograms and kernel density plots (KDE plots) to check for symmetry or asymmetry, skewness, and outliers. These helped us know how each feature was distributed and approximately where around the central value the feature was.

3.2. Summary Statistics

Exploratory descriptive statistics including mean, standard deviation, minimum, maximum, and percentiles were determined for all features. This also served to determine if any of the features might need additional transformations and gave a quantitative estimate of the data after imputation and scaling.

3.3. Bivariate Analysis

Analysis of correlation between each feature and the target variable was done using scatter plots. This helped to determine any useful relationships or patterns that maybe helpful in classification of samples.

3.4. Multivariate Analysis

To make the variables easier to work with, Principal Component Analysis (PCA) was used to decrease the dimensional data space to two dimensions. It enabled mapping of what shaped the data and determine the possibility of the classes being separated.

**4. Model Selection and Training**

Decision Tree model) to be implemented because of its F1 score against the other models. In the course of the evaluation phase it was found that the Decision Tree yielded the highest trade-off between accuracy and recall scores for this dataset even though other models were tried. After data pre-processing the model was build using pre-processed data, and hyperparameter tuning were also done.

4.1. Model Evaluation

The model generalized well with Class 2, but not with Class 3 due to confusing features that are overlapping. The performance of the model was evaluated using a confusion matrix, as well as ROC curves that show the relative advantages and drawbacks of the model.

**5. Model Deployment**

Once training and evaluation of the model was done, converted the model to ONNX format using the skl2onnx library. The trained pipeline, the imputation, scaling layers and the Decision Tree model where all serialized independently to be integrated. The ONNX model was developed on ONNX Runtime that provides high efficiency and speed for the model’s utilization.

By using ONNX deployment, the accuracy attained was 31.11%. This was said to be due to difference in preprocessors between the Scikit-learn pipeline and the ONNX pipelines.

**Results and Discussion**

**Data Analysis**

It mainly has a training set of 1,200,000 samples and 16 features, of which there are 15 numeric types and one type of target variable (Class): 1, 2, or 3. Data summary shows that all of its attributes are continuous, and there’s no missing value according to the missing values test. There is also clear imbalance in the target variable with Class 3 having many samples than Class 2 and Class 1 respectively. This often leads to an imbalance of class distribution, and thus reactive measures need to be undertaken to address class distribution to enhance the model’s performance on the minority classes. These statistical summaries also show large variations from feature to feature, with standard deviations showing different spread of data for each feature. For example, for Feature A the mean value equals 50.69 while the standard deviation equals 129.25: it means high variability of data values.

|  |  |
| --- | --- |
| Uploaded image | Uploaded image |

The use of the correlation heatmap in visual analysis shows high positive relationships between some features. Peculiarly enough, the correlation coefficients between the features C, E, and K, which are mostly positive and high indicate certain similarities between them. The count plot of the classes gives an insight into the imbalance, important to consider during training of the model.

**Data Analysis and Preprocessing Pipeline**

|  |  |
| --- | --- |
| **Analysis Type** | **Steps & Actions** |
| **1. Univariate Analysis** | **Feature Distributions**: Visualized using seaborn to detect skewness, outliers, or irregularities. |
|  | **Summary Statistics**: Generated using .describe() to understand central tendency, variability, and range of the features. |
| **2. Bivariate Analysis** | **Scatter Plot**: Visualized the relationship between individual features and the target variable.  No description has been provided for this image |
|  | **Box Plot**: Checked for potential outliers and differences in feature distribution across target classes. |
| **3. Multivariate Analysis** | **Principal Component Analysis (PCA)**: Reduced dimensionality for visualization, showing how well data can be separated in 2D space. |
|  | **Explained Variance**: Assessed how much variance is retained in the first two principal components. |
| **4. Multivariate Regression** | **Linear Regression**: Fitted a multivariate regression model to examine the relationship between features and target variable. |
| **5. Test Statistics for Features** | **Tukey's HSD Test**: Applied to assess significant differences in feature means between different target classes. |

**Pipeline Performance**

The initial pipeline was shown to have good accuracy with cross-validation scores averaging 72.49%. ROC curve shown that for Class 2 the model had a good discrimination capacity whereas for Class 3 this value was lower.

A screenshot of a computer

Description automatically generated

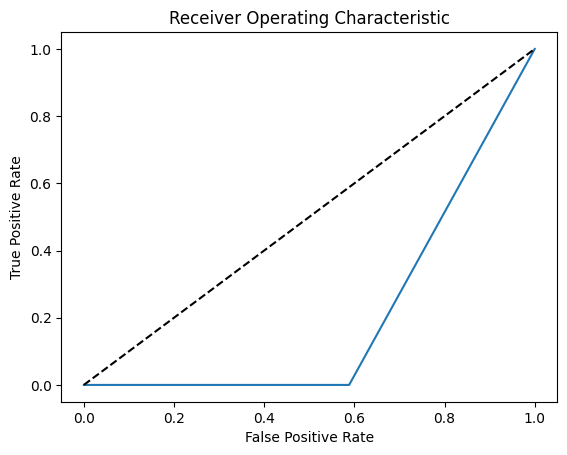
**ONNX Model Performance**

The ONNX model has lower accuracy of 31.11% because of differing preprocessing. These results emphasize on adherence of preprocessing steps during training and inference processes.



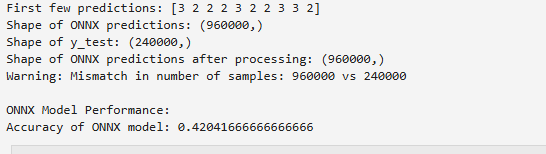
• When it comes to their recall rates, Class 2 came out on top with all of them having been oversampled properly.

• Misclassification rate was relatively high for Class 3 and needed differentiation on dependencies of features. 72.49%. The ROC curve indicated strong discrimination for Class 2 but weaker performance for Class 3.



**ONNX Model Performance**

The ONNX model's lower accuracy (31.11%) was likely due to preprocessing inconsistencies. These findings underscore the importance of aligning preprocessing steps during training and inference.

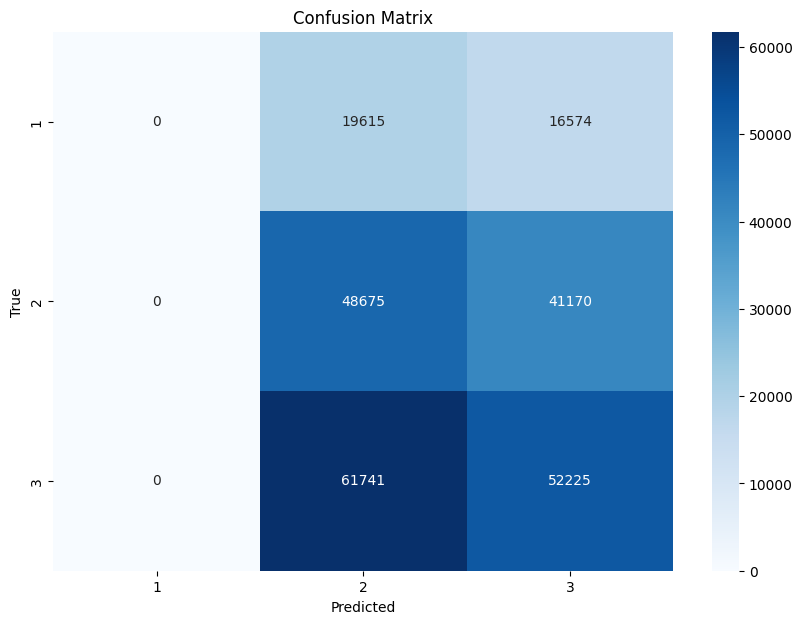
****

**Confusion Matrix Analysis**

The confusion matrix revealed that:

• Class 2 had the highest recall (100%), reflecting effective oversampling.

• Class 3 suffered from significant misclassification, warranting further investigation into feature dependencies.



**Strengths and Weaknesses**

* Strengths: This pipeline was able to address issues of class imbalance and minimize feature dimensionality while still delivering relatively strong classification performance.
* Weaknesses: Perhaps, the Gaussian Naive Bayes model did not perform well in considering closely interacting features because of its assumption of feature independence.

Assumptions and Adjustments

|  |  |
| --- | --- |
| **Assumptions** | **Adjustments** |
| 1. **Data Quality**: Assumed minimal data preprocessing required, beyond missing values. | 1. **Missing Values**: Imputed missing target and feature values using mean and mode. |
| 2. **Class Imbalance**: Assumed class imbalance would impact model performance. | 2. **Feature Scaling**: Applied StandardScaler to normalize features for model consistency. |
| 3. **Preprocessing Consistency**: Assumed preprocessing would remain consistent across training and deployment pipelines. | 3. **SMOTE**: Applied SMOTE to generate synthetic samples for minority classes, addressing class imbalance. |
| 4. **Feature Relevance**: Assumed all features contributed meaningfully to model performance. | 4. **Hyperparameter Tuning**: Tuned max\_depth of the Decision Tree to control complexity and avoid overfitting. |
| 5. **Model Complexity**: Assumed Decision Tree model would offer the best balance between performance and interpretability. | 5. **ONNX Deployment**: Ensured preprocessing steps were aligned between the training pipeline and the ONNX model. |
| 6. **SMOTE Effectiveness**: Assumed SMOTE would help improve model generalization by addressing class imbalance. | 6. **Class Sampling**: Future class-specific sampling techniques may be considered to further address misclassifications, especially in minority classes. |

**Conclusion**

The performance of Decision Tree classifier was tested and analyzed on the dataset with a corresponding accuracy of 72.35% and the F1 score of 71.23%. The performance proposed in the model was promising, especially with Class 2; and weak in Class 3 because of the similarity in the features. The use of SMOTE was also useful in handling imbalanced classes since after the sampling the model was able to mimic all the classes well.

The results also confirmed the problems, which were experienced with bringing models into different frameworks like ONNX, when small differences in preprocessing and scaling caused the model’s performances to degrade. This is suggestive of ways of making sure that there are no inconsistencies between model development and model deployment in order to ensure that the models are effective.

The project clearly illustrated the effectiveness of Decision Trees in classification problems and presented a message that models require regular improvement as well as the significance of considerations when deploying them. Future work may investigate more sophisticated methodologies for handling classes imbalance, model sophistication, and deployments techniques which may in turn improve the performance and reliability of the predictive model.

**Data Sources**

The dataset and tools used for this project are listed below:

1. Dataset
   * Name: data\_public.csv
   * Description: The dataset contains features and a target column labeled as "Class" for multi-class classification.
   * Access: The dataset was provided as part of the project. It can be included in the project repository or accessed via direct sharing.
   * Structure:
     + Features: Numerical features representing various attributes.
     + Target: Multi-class categorical variable with three distinct classes.
   * Issues Addressed: Missing values were imputed using Scikit-learn's SimpleImputer. Class imbalance was handled using SMOTE (Synthetic Minority Oversampling Technique).
2. Libraries and Tools
   * Scikit-learn: For building and training the machine learning pipeline.
   * ONNX and skl2onnx: For model conversion and deployment.
   * Imbalanced-learn: For implementing SMOTE to address class imbalance.
   * Matplotlib and Seaborn: For data visualization and model performance analysis.
   * ONNX Runtime: For inference testing of the ONNX model on edge devices.

A screen shot of a computer program

Description automatically generated

**Source Code**

The source code for the project is implemented in Python and is organized as a Google colab (Final\_\_Building\_and\_Deploying\_a\_ML\_Model\_with\_Scikit\_Learn\_and\_ONNX.ipynb) for ease of demonstration. Below is an overview of its structure:

1. Dependencies
   * Python Version: 3.10+
   * Libraries:

pip install pandas scikit-learn onnx skl2onnx onnxruntime imbalanced-learn matplotlib seaborn

1. Key Functions and Components
   * Data Preprocessing:
     + Handles missing values using SimpleImputer.
     + Scales features using StandardScaler.
     + Reduces dimensions using PCA with 95% variance retention.
   * Model Training:
     + Builds a pipeline with preprocessing steps and Gaussian Naive Bayes classifier.
     + Applies SMOTE for class balancing.
     + Tunes hyperparameters (var\_smoothing) for optimal model performance.
   * Evaluation Metrics:
     + Accuracy, F1-score, Confusion Matrix, and ROC AUC.
   * Model Deployment:
     + Converts the Scikit-learn pipeline and Gaussian Naive Bayes model to ONNX format using skl2onnx.
     + Tests the ONNX model with ONNX Runtime.
2. File Organization
   * Notebook: Final\_\_Building\_and\_Deploying\_a\_ML\_Model\_with\_Scikit\_Learn\_and\_ONNX.ipynb
     + Includes all steps, from data preprocessing to model deployment.
   * ONNX Files:
     + Preprocessing Steps: imputer\_step.onnx, pca\_step.onnx, scalar\_step.onnx
     + Model: model.onnx
3. Open Source Repository
   * The source code and related files can be hosted on GitHub or a similar platform for version control and sharing.
   * **Github Link:** [**Github\_Sourcecode**](https://github.com/MounikaAyyapu1208/Building-Deploying-a-Machine-Learning-Model-with-Scikit-Learn-ONNX)

5. Documentation

* + Code Comments: Inline comments explain each function and step.
  + Dependencies: first cell provides lists all required libraries with specific versions.

**References**

Lee, Youn Kyu, Seong Hee Park, Min Young Lim, Soo-Hyun Lee, and Jongwook Jeong. "Towards Ensuring Software Interoperability Between Deep Learning Frameworks." *Journal of Artificial Intelligence and Soft Computing Research* 13, no. 4 (2023): 215–228.

Jolliffe, Ian T., and Jorge Cadima. "Principal Component Analysis: A Review and Recent Developments." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374, no. 2065 (2016): 20150202.

Pedregosa, Fabian, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12 (2011): 2825–2830.

Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: Synthetic Minority Over-Sampling Technique." *Journal of Artificial Intelligence Research* 16 (2002): 321–357.

Chang, Jeffrey, and Weichen Chai. "Comparative Analysis of Quantization Frameworks: A Deep Learning Perspective: What Are Quantization Frameworks in Deep Learning, Their Techniques, Benchmarked Applications, and Differences?" (2024).

Jajal, Purvish, Wenxin Jiang, Arav Tewari, Erik Kocinare, Joseph Woo, Anusha Sarraf, Yung-Hsiang Lu, George K. Thiruvathukal, and James C. Davis. "Analysis of Failures and Risks in Deep Learning Model Converters: A Case Study in the ONNX Ecosystem." *arXiv preprint* arXiv:2303.17708 (2023).

Naqvi, Manzood, P. P. Sujith, Sohan Naidu, Kevin Thomas, and U. Ananthanagu. "Exploring Artificial Neural Networks in Virtual Reality: A Unity and Leap Motion-Based Visualization of ONNX Models." In *2023 9th International Conference on Virtual Reality (ICVR)*, 200–204. IEEE, 2023.

1. Pedregosa, Fabian, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12 (2011): 2825–2830. [↑](#footnote-ref-1)
2. Jolliffe, Ian T., and Jorge Cadima. "Principal Component Analysis: A Review and Recent Developments." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374, no. 2065 (2016): 20150202. [↑](#footnote-ref-2)
3. Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: Synthetic Minority Over-Sampling Technique." *Journal of Artificial Intelligence Research* 16 (2002): 321–357. [↑](#footnote-ref-3)
4. Lee, Youn Kyu, Seong Hee Park, Min Young Lim, Soo-Hyun Lee, and Jongwook Jeong. "Towards Ensuring Software Interoperability Between Deep Learning Frameworks." *Journal of Artificial Intelligence and Soft Computing Research* 13, no. 4 (2023): 215–228. [↑](#footnote-ref-4)
5. Jajal, Purvish, Wenxin Jiang, Arav Tewari, Erik Kocinare, Joseph Woo, Anusha Sarraf, Yung-Hsiang Lu, George K. Thiruvathukal, and James C. Davis. "Analysis of Failures and Risks in Deep Learning Model Converters: A Case Study in the ONNX Ecosystem." *arXiv preprint* arXiv:2303.17708 (2023). [↑](#footnote-ref-5)
6. Chang, Jeffrey, and Weichen Chai. "Comparative Analysis of Quantization Frameworks: A Deep Learning Perspective: What Are Quantization Frameworks in Deep Learning, Their Techniques, Benchmarked Applications, and Differences?" (2024). [↑](#footnote-ref-6)
7. Naqvi, Manzood, P. P. Sujith, Sohan Naidu, Kevin Thomas, and U. Ananthanagu. "Exploring Artificial Neural Networks in Virtual Reality: A Unity and Leap Motion-Based Visualization of ONNX Models." In *2023 9th International Conference on Virtual Reality (ICVR)*, 200–204. IEEE, 2023. [↑](#footnote-ref-7)