**Project Deliverable Documentation**



**Course**: CE4143/CS4241/IT4230 - Introduction to Artificial Intelligence

**Institution**: ACADEMIC CITY UNIVERSITY

**Name**: NATHANIEL MONNEY

**Index Number**: 10211100403

**PROGRAM:** COMPUTER ENGINEERING

**LECTURER:** MR. GODWIN NTOW DANSO

**Submission Date**: 17th April, 2025

**OVERVIEW**

The AI Model Playground is a Streamlit-based web application designed to showcase practical applications of Artificial Intelligence (AI) and Machine Learning (ML) through an interactive, modular interface. The application enables users to explore and solve diverse AI problems, including:

* **Regression Analysis**: Predicting continuous variables with linear regression, complete with performance metrics and visualizations.
* **K-Means Clustering**: Grouping data points into clusters with interactive visualizations and downloadable results.
* **Neural Network Classification**: Training a feedforward neural network for classification tasks with customizable hyperparameters.
* **LLM-Powered Question Answering**: A Retrieval-Augmented Generation (RAG) system for querying Ghana election data using the Gemini API.

The application integrates data preprocessing, model training, visualization, and evaluation into a unified, user-friendly dashboard, demonstrating proficiency in applying AI concepts.

**Libraries**

* **Python**: Core programming language.
* **Streamlit**: Framework for building the interactive web interface.
* **Pandas & NumPy**: Data manipulation and processing.
* **Scikit-learn**: Machine learning models (LinearRegression, KMeans) and preprocessing.
* **TensorFlow/Keras**: Deep learning for neural network classification.
* **SentenceTransformers & FAISS**: Text embedding and vector storage for the RAG system.
* **LangChain & Google Generative AI (Gemini API)**: LLM backend for question answering.
* **Plotly**: Interactive data visualizations.
* **Matplotlib**: Used in conjunction with Plotly for specific visualizations.

**Web Application**

The AI Model Playground is an interactive web-based platform designed to demonstrate the principles and application of machine learning algorithms using real-world datasets. Among its several modules, two of the most central are the **Regression** and **Clustering** modules. Each is equipped with user-friendly features, visualization tools, and configurable settings to enable both beginners and advanced users to experiment and learn effectively. The following sections provide a detailed walkthrough of these modules and their functionalities.

**Regression Module**

The Regression module of the AI Model Playground is tailored to allow users to perform linear regression analysis using their own datasets. It supports intuitive steps from dataset importation to data preprocessing, model training, and prediction. This guided process is designed to demonstrate the practical applications of linear regression, while also providing valuable insights into the model’s performance.

**1. File Upload and Dataset Preview**

Users initiate the process by uploading a CSV file containing their dataset. This dataset is parsed using the **pandas** library, which enables efficient data handling. Once the file is uploaded, the platform displays the first 10 rows of the dataset using st.write(df.head(10)). This preview step allows users to verify the integrity and structure of their dataset before proceeding.

**2. Data Preprocessing Options**

Data quality is fundamental in machine learning, and this module provides flexible options for preprocessing.

**2.1 Handling Missing Values**

Users can decide how to address missing values through a dropdown menu. The available options include:

* **Keep as-is** (default)
* **Drop rows with missing values**
* **Impute missing numeric values with the median**

These options ensure that users can prepare clean data for training while handling common issues found in real-world datasets.

**2.2 Feature Scaling**

A checkbox is provided to optionally apply **StandardScaler** to the numeric features. This method standardizes the data to a mean of 0 and a standard deviation of 1. Feature scaling is often crucial, especially when using algorithms sensitive to feature magnitude, such as linear regression.

**3. Target and Feature Selection**

The module restricts the target column to continuous variables, specifically price or area, as these are the only suitable columns in the sample dataset. Users then manually select the independent variables (features) from the remaining columns. This helps maintain the integrity of the regression task by ensuring only relevant inputs are used for model training.

**4. Data Encoding**

To handle categorical data, the system automatically encodes any string or object type columns using **LabelEncoder**. This preprocessing step converts non-numeric data into a numerical format suitable for machine learning algorithms without requiring user intervention.

**5. Train-Test Split**

The platform provides a slider to allow users to adjust the test size for model evaluation. The split can be configured between 10% and 50%. Upon selection, the number of samples in the training and test datasets is calculated and displayed, giving users a transparent view of how their data is partitioned.

**6. Model Training**

The core of this module is the training of a **LinearRegression** model from the **scikit-learn** library. The system fits the model using the selected features and target, then performs predictions on both the training and test datasets to evaluate its generalization capability.

**7. Performance Metrics**

To give users insight into how well the model performs, several key performance metrics are computed and displayed:

* **Mean Squared Error (MSE):** Calculates the average of the squares of the prediction errors, giving more weight to larger errors.
* **R² Score:** Indicates the proportion of the variance in the dependent variable explained by the independent variables, ranging from 0 to 1.
* **Mean Absolute Error (MAE):** Represents the average absolute difference between predicted and actual values.
* **Pearson Correlation:** Measures the linear correlation between the actual and predicted values, ranging from -1 to 1.

These metrics allow users to critically assess the effectiveness and accuracy of their regression model.

**8. Visualization**

To enhance interpretability, the module includes visual plots:

* Two regression plots (actual vs predicted) are generated for both the **training** and **test** datasets.
* The visualizations utilize seaborn.regplot, and each plot includes a red regression line to help users visually understand the model's fit.

These plots serve as an intuitive check for linearity and predictive accuracy.

**9. Prediction Section**

The final component of the regression page allows users to test the model in real time. Users can enter new values for the feature variables via form inputs. The system automatically applies necessary encoding and scaling before using the trained model to make a prediction. The result is then displayed instantly, providing a smooth and interactive experience for exploring model outputs.

**Clustering Module**

The Clustering module is designed to introduce users to **unsupervised learning** techniques, specifically **K-Means clustering**. It allows users to uncover hidden patterns in data by grouping similar data points based on feature similarity. The module is equipped to handle messy real-world data, and it provides robust visualization and interpretability tools.

**1. File Upload and Encoding Fallbacks**

Recognizing the variability in dataset formats, this module includes a robust file upload system. It attempts to parse CSV files using various encodings, such as utf-8, latin1, and iso-8859-1. If all standard encoding strategies fail, it falls back to using Python’s native CSV engine. This ensures that the platform gracefully handles unexpected file formats and prevents crashes.

**2. Dataset Preview and Cleaning**

Upon upload, the shape of the dataset and the first few rows are displayed to the user. The cleaning options provided include:

* **Drop rows with missing values**
* **Fill numeric columns with the median and categorical columns with the mode**
* **Fill all missing values with zero**

These options help users quickly sanitize their data for clustering without writing any preprocessing code.

**3. Feature Selection**

Only numeric columns are eligible for clustering. Users must select at least two features, and the platform offers the flexibility of choosing either **2D or 3D** feature combinations for visualization purposes. This ensures meaningful clustering outcomes and clear visual interpretation.

**4. Feature Scaling**

An optional checkbox enables normalization of features using **StandardScaler**. This step is strongly recommended for clustering, as it ensures that all features contribute equally to the distance calculations that underlie the K-Means algorithm.

**5. Elbow Method for Optimal Clusters**

To assist users in choosing an appropriate number of clusters, the system computes the **Within-Cluster Sum of Squares (WCSS)** for values of k ranging from 1 to 10. An elbow plot is generated, allowing users to visually identify the point at which adding more clusters yields diminishing returns—a common heuristic for selecting k.

**6. K-Means Clustering**

After determining the desired number of clusters, the platform performs clustering using the **K-Means** algorithm. Results are:

* Appended as a new **Cluster** column in the dataset.
* Cluster centers (centroids) are computed and displayed to the user.

This helps users gain insight into how the algorithm has grouped the data and where the central tendencies of each cluster lie.

**7. Visualization**

Visualization is a key strength of this module:

* For **2D selections**, a scatter plot is generated.
* For **3D selections**, a 3D scatter plot is rendered using **matplotlib**.
* Cluster **centroids** are plotted in red using an ‘X’ marker for clarity.
* All axes are clearly labeled and formatted for readability.

These plots offer an intuitive understanding of the clustering structure and the distribution of data points.

**8. Scatter Statistics**

The module also generates summary statistics for each cluster, including:

* **Mean values** per cluster for the selected features
* **Count of data points** in each cluster

Additionally, a **bar chart** showing the distribution of data points across clusters is displayed, complete with value labels. This provides a quantitative complement to the visual output.

**9. Download Functionality**

To complete the user workflow, the Clustering module includes a **download button** that enables users to export the updated dataset (including the cluster assignments) as a new CSV file. This allows further analysis or reporting outside the application.

The **Neural Network Classifier Page** is one of the core modules in the AI Model Playground and is designed for advanced classification tasks using deep learning. Built on the robust **TensorFlow/Keras** framework, this module allows users to construct, configure, and train neural networks tailored to specific classification problems. A particularly highlighted use case is **password strength classification**, although the architecture supports any compatible categorical dataset. This page blends automation and customization to provide a flexible interface for both novice users and experienced practitioners.

**1. Dataset Upload and Preprocessing**

The first step in utilizing the Neural Network Classifier module involves the upload of a dataset in CSV format. Once uploaded, the dataset undergoes **automated cleaning**, with rows containing missing values being dropped to ensure data integrity and minimize preprocessing requirements from the user.

A key feature of this module is the intelligent handling of datasets containing a **password column**. If such a column is detected, the system automatically engineers multiple meaningful features to quantify password characteristics. These include:

* **Length** of the password
* Counts of **uppercase letters**, **lowercase letters**, **digits**, and **special characters**
* Number of **unique characters**
* **Entropy**, calculated using **Shannon’s information theory**, to measure the unpredictability or complexity of each password

This comprehensive feature engineering process significantly enhances the model's ability to classify password strength with greater accuracy.

**2. Feature and Target Selection**

The module supports flexible **feature and target selection**. For password classification use cases, the system restricts the target variable to strength (if present) to maintain consistency. If the target column is categorical, it is automatically encoded using **LabelEncoder**, transforming labels into numeric format suitable for model training.

Users are free to select multiple features from the engineered dataset or other uploaded columns to build their custom model. This allows fine-tuning and experimentation with different feature combinations to evaluate their effect on performance.

**3. Data Preprocessing**

Prior to training, the selected features undergo further processing:

* **Categorical features** are encoded to ensure numerical compatibility
* **StandardScaler** is applied to normalize all features, bringing them to a standard scale with a mean of 0 and standard deviation of 1. This is a crucial step, as it ensures stable and efficient learning by the neural network
* The dataset is then split into training and testing subsets using **train\_test\_split**, a standard method for ensuring that model evaluation is performed on unseen data

These steps guarantee that the model is trained on a clean, well-prepared, and unbiased dataset.

**4. Neural Network Configuration**

The neural network architecture is fully customizable, empowering users to experiment with various hyperparameters. Additionally, the module automatically calculates **class weights** to mitigate issues related to class imbalance, which is common in classification tasks like password strength prediction.

Users can specify the following neural network settings:

* **Number of training epochs**, which defines how many times the entire dataset is passed through the model
* **Batch size**, which controls the number of samples processed before the model is updated
* **Learning rate**, a crucial parameter that determines the step size during optimization
* **Number of hidden layers** and **units per layer**, allowing for dynamic construction of the network’s depth and complexity

This level of configurability enables users to fine-tune their models for optimal performance based on their specific datasets.

**5. Model Training**

Once the configuration is complete, the model training begins. A custom **Keras Callback** class is used to provide real-time feedback on the training process. Key features of this component include:

* **Epoch-wise progress updates**, giving users a sense of training duration and current iteration
* **Live plots** displaying the **training and validation loss** and **accuracy curves**, updated in real-time. These plots provide invaluable insight into the model’s learning behavior and help identify issues such as underfitting or overfitting as they emerge

This interactive training experience ensures transparency and keeps users engaged throughout the model development cycle.

**6. Evaluation**

Upon completion of training, the model is evaluated against the test dataset. The module generates a detailed **classification report** containing the following metrics:

* **Precision**: The proportion of true positives among the predicted positives
* **Recall**: The proportion of true positives among the actual positives
* **F1-Score**: The harmonic mean of precision and recall, balancing both metrics
* **Support**: The number of actual occurrences for each class

These metrics are essential for assessing the quality and reliability of the trained model, particularly in multi-class classification scenarios like predicting password strength levels (e.g., weak, medium, strong).

**7. Prediction Options**

The module includes powerful prediction features for both **batch** and **individual** predictions.

**7.1 Batch Prediction**

Users can upload a new CSV file for batch prediction. The uploaded dataset is automatically processed, and the trained model is used to predict the class label for each row. This enables large-scale evaluation of password strength (or other categorical predictions) across a dataset.

**7.2 Real-Time Prediction**

For on-the-fly testing, users can manually enter a single **password** into a form field. The system processes the input in real time and returns:

* **Predicted strength label**
* **Confidence score** (probability of the predicted class)
* A **feature breakdown table**, showing how the input password was interpreted and processed by the model

This feature is particularly useful for demonstrating the classifier’s interpretability and showcasing how neural networks process raw inputs into meaningful outcomes.

# 

# **Large Language Model with RAG**

The **Large Language Model with Retrieval-Augmented Generation (RAG)** module provides an advanced interface for querying documents using natural language. It leverages semantic search and large language models to generate highly relevant and explainable responses. By integrating intelligent chunking, vector-based retrieval, and generative models like Google’s Gemini, this module transforms static documents into dynamic, interactive knowledge sources.

**1. Document Processing Phase**

The document processing pipeline is designed to support a wide range of real-world data formats while ensuring accurate and efficient preparation for semantic search and answer generation.

**1.1 File Ingestion**

The system supports both structured and unstructured document types through two primary formats:

* **CSV Files**: Managed using the pandas library, CSV files are converted into a Markdown table format using the DataFrame.to\_markdown() method. This ensures that tabular data retains its structure and remains digestible for both large language models and text chunking algorithms.

To ensure robustness, all file processing is wrapped in comprehensive error handling mechanisms. Malformed or unsupported files are rejected with descriptive and actionable feedback to maintain system reliability.

**1.2 Text Chunking**

Once documents are ingested, they are segmented into semantically coherent chunks using LangChain’s **RecursiveCharacterTextSplitter**. This method respects natural language structure, prioritizing boundaries such as:

* Double newlines (\n\n)
* Single newlines (\n)
* Common sentence terminators (e.g., ., ,)

The chunking parameters are configured as follows:

* **Chunk Size**: 1500 characters
* **Overlap**: 300 characters

The overlap mechanism ensures contextual continuity between adjacent chunks, which is critical for accurately answering questions that span multiple sections of a document.

**1.3 Vector Embedding**

Each chunk is embedded into a high-dimensional vector space using **SentenceTransformers’ all-MiniLM-L6-v2** model, optimized for performance on CPU environments.

* **Embedding Output**: 384-dimensional vectors
* **Batch Size**: 32 chunks per batch, optimizing runtime and memory usage
* **Vector Store**: FAISS (Facebook AI Similarity Search)
  + **Index Type**: IndexFlatIP (Inner Product similarity)
  + **Preprocessing**: All embeddings are L2-normalized to ensure that inner product similarity reflects cosine similarity

This embedding process forms the semantic backbone of the retrieval system, enabling fast and scalable vector search.

**2. Query Processing Phase**

Once a document has been indexed, users can pose natural language questions that are semantically matched to the most relevant sections of the uploaded document.

**2.1 Question Handling**

User queries are embedded using the same all-MiniLM-L6-v2 model to maintain compatibility and consistency with the document embeddings. Each query is reshaped to fit FAISS’s batch querying format (2D array), preparing it for semantic comparison.

**2.2 Semantic Search with FAISS Indexing**

The embedded query is matched against the FAISS index to retrieve the most semantically relevant chunks.

* **Top-k Retrieval**: The top 5 (k=5) most relevant chunks are retrieved. If the total number of chunks is fewer than 5, the number is dynamically adjusted.
* **Distance Metric**: The system uses inner product similarity, which simulates cosine similarity due to prior normalization.

Robust bounds checking ensures that chunk indices are always valid and runtime errors are prevented during the search process.

**2.3 Confidence Calculation**

To enhance transparency and interpretability, a **confidence score** is computed for each response. This score is derived from a weighted heuristic based on three key components:

* **A. Semantic Similarity (50%)**
  + Based on cosine similarity between the query vector and each retrieved chunk vector.
  + Protected by safety mechanisms to handle shape mismatches and invalid values.
* **B. Distance Score (30%)**
  + The raw FAISS distance values are normalized and inverted into a [0, 1] similarity scale.
  + Ensures semantically closer chunks receive higher weight.
* **C. Coverage Ratio (20%)**
  + Measures the proportion of relevant chunks retrieved (actual / requested).
  + Penalizes incomplete or insufficient retrieval, thus improving accuracy under sparse conditions.

The final confidence score is converted into a percentage and categorized into:

* **High** (75% – 99%)
* **Medium** (50% – 74%)
* **Low** (30% – 49%)

These levels are visually represented with color-coded indicators for user clarity.

**3. Answer Generation with Gemini**

After relevant chunks are retrieved and evaluated, the system constructs a prompt for **Google’s Gemini LLM** (accessed via the google.generativeai API) to generate a natural language answer.

* **Available Models**:
  + gemini-1.5-pro: High-capacity, accuracy-focused model
  + gemini-1.5-flash: Lightweight model optimized for speed

Prompt construction includes the user’s question and all retrieved context. Retry logic is implemented using the tenacity package to gracefully handle temporary API failures or latency issues.

**4. Visualization and Source Attribution**

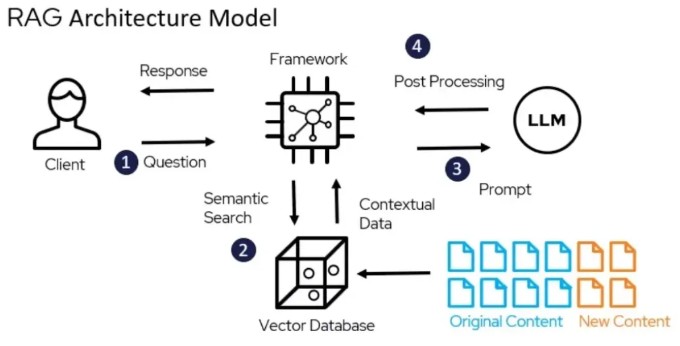
To improve user confidence and model explainability, the interface includes:

* **Confidence Badges**:
  + Green for High confidence
  + Yellow for Medium confidence
  + Red for Low confidence
* **Source Traceability**:
  + A collapsible “View Sources” section reveals the exact document chunks used to construct the response.
  + This promotes transparency and allows users to verify the credibility of the model’s output.

**5. Additional Features**

Several supporting features are built into the module to improve usability and performance:

* **API Key Entry**:
  + Gemini API keys are securely handled and stored temporarily using st.session\_state in Streamlit.
* **Upload & RAG Persistence**:
  + Once a document is processed, its associated chunks and FAISS index are stored in session memory.
  + This avoids redundant processing and speeds up subsequent queries during the same session.



## **LLM RAG Prompt Responses from Chat GPT**

**1. Who won the 2020 election in the Ahafo Region?**  
**Answer:** Nana Akufo Addo of the NPP won the 2020 election in the Ahafo Region with 145,584 votes, which was 55.04% of the total votes.

**2. Which candidate came second in the Ahafo Region during the 2020 election?**  
**Answer:** John Dramani Mahama of the NDC came second with 116,485 votes, which was 44.04%.

**3. What percentage of the total votes did Ivor Kobina Greenstreet receive in the Ahafo Region?**  
**Answer:** Ivor Kobina Greenstreet received 0.07% of the total votes in the Ahafo Region.

**4. How many votes did David Asibi Ayindenaba Apasera receive in the Ahafo Region?**  
**Answer:** David Asibi Ayindenaba Apasera received 83 votes in the Ahafo Region.

**5. Which independent candidate participated in the election in the Ahafo Region and how many votes did he get?**  
**Answer:** Alfred Kwame Asiedu Walker was the independent candidate and he received 103 votes.

## **LLM RAG Prompt Responses from Streamlit Web App**

**Answer:**

Based on the provided data:

1. **Who won the 2020 election in the Ahafo Region?** I don't have enough information to determine the winner. The provided data only shows results for certain "Other" candidates, not the main contenders.
2. **Which candidate came second in the Ahafo Region during the 2020 election?** I don't have enough information to determine who came in second place.
3. **What percentage of the total votes did Ivor Kobina Greenstreet receive in the Ahafo Region?** Ivor Kobina Greenstreet received 0.07% of the votes in the Ahafo Region in 2020.
4. **How many votes did David Asibi Ayindenaba Apasera receive in the Ahafo Region?** David Asibi Ayindenaba Apasera received 83 votes in the Ahafo Region in 2020.
5. **Which independent candidate participated in the election in the Ahafo Region and how many votes did he get?** The provided data does not identify any independent candidates. All listed candidates are affiliated with a party. Therefore, I cannot answer this question.