Politechnika Śląska

Katedra Grafiki, Wizji komputerowej i Systemów Cyfrowych

Academc year			Group	Section
2022/2023	SSI	**BIAI**	**ISMiP**	**1**
Supervisor:	dr inż. Grzegorz Baron		**Classes: (day, hour)**	
Section members:	**Katarzyna Szczepańczyk** **Szymon Walasik**		**Monday**	
emails:	szymwal050@student.polsl.pl kataszc870@student.polsl.pl		**13:15-16:15**	

Project card

Subject:

Diabetes prediction based on medical data

Spis treści

1.Short introduction presenting the project topic

Diabetes is a prevalent chronic disease affecting millions of individuals worldwide. Early detection and accurate prediction of diabetes are crucial in preventing complications and providing timely interventions. With the rapid advancements in artificial intelligence and machine learning, researchers have turned to harnessing the power of neural networks to develop predictive models based on medical data. This project aims to utilize a comprehensive dataset encompassing various patient attributes, including the number of pregnancies, glucose levels, blood pressure, skin thickness, insulin levels, body mass index (BMI), diabetes pedigree function, age, and the presence or absence of diabetes, to construct an effective diabetes prediction model using neural networks.

The primary objective of this project is to develop a robust diabetes prediction model utilizing neural networks. By analyzing historical medical data the model will strive to identify individuals at high risk of developing diabetes. Timely identification of individuals at risk enables healthcare professionals to implement preventive measures, recommend appropriate lifestyle modifications, and design personalized treatment plans, ultimately mitigating the impact of diabetes-related complications.

Furthermore, the project will focus on optimizing the model's performance through the fine-tuning of hyperparameters, implementation of regularization techniques, and exploration of feature selection methodologies. These steps will ensure that the developed diabetes prediction model is highly accurate and reliable. Integrating such a model into clinical practice has the potential to significantly improve healthcare professionals' decision-making process, leading to better patient outcomes and a reduction in diabetes-related complications.

In conclusion, this project aims to leverage the power of neural networks to construct a robust diabetes prediction model using a comprehensive dataset consisting of patient attributes. By harnessing the computational capabilities of advanced neural network architectures, we aim to improve early detection and intervention strategies for diabetes.

2.Analysis of the task

Possible approaches to solve the problem

The task at hand involves predicting the presence or absence of diabetes based on a set of patient attributes. The dataset contains information such as the number of pregnancies, glucose levels, blood pressure, skin thickness, insulin levels, BMI, diabetes pedigree function, and age. This is a binary classification problem where the model needs to learn patterns and relationships between these attributes and the outcome to accurately predict the presence or absence of diabetes.

To solve this problem, several approaches can be considered:

1. Feedforward Neural Networks (FNNs): One possible approach is to use feedforward neural networks, also known as multilayer perceptrons (MLPs). FNNs consist of multiple layers of interconnected nodes, with each node applying a nonlinear activation function to the weighted sum of its inputs. By training an FNN on the dataset, the model can learn complex patterns and relationships between the patient attributes and the diabetes outcome.

This approach requires careful consideration of the network architecture, including the number of hidden layers, the number of nodes per layer, and the choice of activation functions.

2. Recurrent Neural Networks (RNNs): Another approach is to utilize recurrent neural networks, which are particularly effective in processing sequential data. RNNs maintain an internal memory that allows them to capture dependencies and temporal relationships between inputs. In the context of diabetes prediction, RNNs can take into account the sequential nature of the data, such as changes in glucose levels over time. Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) are popular variations of RNNs that can handle long-term dependencies and mitigate the vanishing gradient problem.

3. Convolutional Neural Networks (CNNs): CNNs, primarily designed for image analysis, can also be adapted to handle tabular data like the diabetes dataset. By applying one-dimensional convolutions, CNNs can extract relevant features from the patient attribute inputs. This approach is particularly useful for capturing local patterns and dependencies in the data, such as specific combinations of attribute values that may be indicative of diabetes risk.

4. Ensemble Methods: Ensemble methods, such as Random Forests or Gradient Boosting, can be employed to combine multiple models for improved prediction performance. These methods utilize multiple weak learners to form a stronger overall model. By training individual models on subsets of the data and combining their predictions, ensemble methods can leverage diverse perspectives and capture a wider range of patterns and relationships within the dataset.

5. Feature Selection and Engineering: Prior to model training, it is essential to perform feature selection and engineering to identify the most informative attributes and potentially create new features. Feature selection techniques, such as correlation analysis or recursive feature elimination, can help identify the subset of attributes that have the most significant impact on the diabetes prediction task. Additionally, domain knowledge can guide the creation of new features or transformations that might enhance the predictive power of the models.

6. Regularization and Hyperparameter Tuning: Regularization techniques, such as L1 or L2 regularization, can help prevent overfitting and improve the generalization of the models. Hyperparameter tuning is also critical to optimize the model's performance. Techniques such as grid search or random search can be employed to systematically explore different hyperparameter combinations and identify the optimal configuration for the chosen model architecture.

By exploring these various approaches, combining them if necessary, and fine-tuning the models parameters, it is possible to develop an accurate diabetes prediction system. The choice of the most suitable approach will depend on the characteristics of the dataset, computational resources, and the desired interpretability of the model.

Pros/cons

Feedforward Neural Networks (FNNs)

Pros:

- FNNs can capture complex nonlinear relationships between input attributes and the diabetes outcome.
- Determining the optimal architecture, including the number of layers and nodes, can be challenging.
- Training FNNs is computationally efficient, especially for smaller datasets.

Cons:

- FNNs may struggle with capturing sequential or temporal dependencies in the data
- Subjective interpretation of patterns may lead to different conclusions.
- FNNs may be prone to overfitting, especially when dealing with limited data.

Recurrent Neural Networks (RNNs)

Pros:

- RNNs can effectively capture sequential dependencies in the data, making them suitable for time series or sequential data.
- They can handle variable-length input sequences.
- RNNs can retain memory of past inputs, which is crucial for capturing long-term dependencies.

Cons:

- RNNs can suffer from the vanishing gradient problem, where the gradient diminishes over time, making it difficult to capture long-term dependencies.
- Training RNNs can be computationally expensive, especially for long sequences.
- RNNs may struggle with handling inputs with irregular or missing data.

Convolutional Neural Networks (CNNs):

Pros:

- CNNs can effectively capture local patterns and dependencies in the data.
- They can handle high-dimensional data efficiently.
- CNNs can automatically learn hierarchical feature representations from the data.

Cons:

- CNNs may not fully exploit the sequential or temporal nature of the data.
- Determining the optimal architecture and filter sizes can be challenging.
- CNNs are primarily designed for grid-like data (e.g., images), and adapting them to tabular data requires careful preprocessing and design choices.

Ensemble Methods

Pros:

- Ensemble methods can improve predictive performance by combining multiple models.
- They can handle complex relationships and capture a wider range of patterns in the data.
- Ensemble methods are generally robust against overfitting.

Cons:

- Ensemble methods can be computationally expensive due to training multiple models.
- Combining multiple models can increase complexity and reduce interpretability.
- Ensemble methods may struggle when dealing with imbalanced datasets or noisy data.

Feature Selection and Engineering

Pros:

- Feature selection helps identify the most relevant attributes for the diabetes prediction task, reducing dimensionality and improving model efficiency.
- Carefully engineered features can capture domain-specific knowledge or transformations that enhance the model's predictive power.
- Feature selection and engineering can improve model interpretability by focusing on the most informative attributes.

Cons:

- Selecting the right features requires domain knowledge and expertise.
- Feature selection can discard potentially useful information, leading to information loss.
- Feature engineering may introduce biases or artificial relationships if not done carefully.

Regularization and Hyperparameter tuning

Pros:

- Regularization techniques, such as L1 or L2 regularization, help prevent overfitting and improve model generalization.
- Hyperparameter tuning allows fine-tuning the model's parameters for optimal performance.
- Regularization and hyperparameter tuning can lead to improved model robustness and stability.

Cons:

- Regularization techniques may require additional computational resources and increase training time.
- Finding the optimal hyperparameters can be time-consuming, especially when dealing with large parameter spaces.
- Over-tuning the hyperparameters on the training set may lead to overfitting the validation set.

Detailed description of selected methodology

Feedforward Neural Network - A feedforward neural network (FNN), also known as a multilayer perceptron (MLP), is a type of artificial neural network that is widely used for various machine learning tasks, including classification and regression. It is called "feedforward" because the flow of information through the network occurs in one direction, from the input layer to the output layer, without any loops or feedback connections.

Feature Engineering - the code incorporates feature engineering by creating two additional interaction features, namely interaction_1 and interaction_2, which are calculated by multiplying selected input features. This feature engineering step aims to capture potential interactions between the input attributes and potentially improve the model's predictive performance.

Data Preparation

Loading the Data: The code uses the np.loadtxt function to load the data from a CSV file named "diabetes_2.csv". The data is loaded into a NumPy array called dataset.

Data Visualization: The code includes a data visualization step to gain insights into the distribution of the outcome variable (diabetes or no diabetes). It uses matplotlib.pyplot to create a bar chart representing the number of occurrences for each outcome category. This visualization helps understand the balance of the dataset and the prevalence of diabetes cases.

Correlation Analysis: The code calculates the correlation matrix (corr_matrix) using np.corrcoef to measure the pairwise correlation between the input attributes in the dataset. It then uses seaborn and matplotlib.pyplot to

create a heatmap visualization of the correlation matrix. This step helps identify any significant correlations between the input attributes, which can provide insights into the relationships within the dataset.

Data Splitting: The code splits the dataset into input features (X) and the corresponding target variable (y). The input features (X) are extracted from the dataset array using slicing, and the target variable (y) is extracted as the last column of the dataset array.

Feature Engineering: The code performs feature engineering by creating two additional interaction features: interaction_1 and interaction_2. These interaction features are generated by multiplying selected input features together. The resulting interaction features are then appended to the input feature matrix (X) using np.column_stack. This step aims to capture potential interactions between the existing input attributes, which may enhance the model's predictive performance.

Data Normalization: The code applies data normalization to the input features (X) using z-score normalization. It calculates the mean and standard deviation of the input features and then scales the features by subtracting the mean and dividing by the standard deviation. This normalization step ensures that all input features are on a similar scale, preventing certain features from dominating the learning process.

Train-Test Split: The code splits the preprocessed data into training and testing sets. It uses the split_index variable to determine the index at which to split the data. The first half of the preprocessed data is assigned to the training set (X_train and y_train), while the second half is assigned to the testing set (X_test and y_test). This split allows for evaluating the model's performance on unseen data.

Possible/available datasets

- National Health and Nutrition Examination Survey (NHANES): NHANES is a large-scale survey conducted by the U.S.Centers for Disease Control and Prevention (CDC). It provides comprehensive health and nutrition data on a nationally representative sample of the U.S. population. The dataset includes information on various health conditions, including diabetes, as well as demographic, lifestyle, and clinical measurements.
- UCI Machine Learning Repository: The UCI Machine Learning Repository hosts a wide range of datasets suitable for machine learning tasks. One such dataset is the "Pima Indians Diabetes Database," which is commonly used for diabetes prediction. It contains medical and demographic information of female Pima Indians, including glucose levels, blood pressure, skin thickness, and diabetes outcomes.
- Electronic Health Records (EHR) Databases: EHR databases, available in healthcare institutions, contain extensive patient information, including medical history, diagnoses, treatments, and laboratory test results. These datasets provide a wealth of information that can be utilized for diabetes prediction. However, accessing and using EHR datasets may require appropriate permissions and data handling procedures.

Description of chosen dataset

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney

Diseases. The objective of the dataset is to diagnostically predict whether a patient has diabetes,

based on certain diagnostic measurements included in the dataset. Several constraints were placed

on the selection of these instances from a larger database. In particular, all patients here are females

at least 21 years old of Pima Indian heritage.

From the data set in the (.csv) File We can find several variables, some of them are independent

(several medical predictor variables) and only one target dependent variable (Outcome).

The dataset obtained from Kaggle.com provides valuable information for predicting diabetes based on medical data. It consists of several attributes that are commonly associated with diabetes risk factors. These attributes include:

- Number of Pregnancies: This attribute indicates the number of times the patient has been pregnant.
- Glucose Level: This attribute represents the patient's blood glucose concentration measured in milligrams per deciliter (mg/dL).
- Blood Pressure: This attribute represents the patient's blood pressure measured in millimeters of mercury (mmHg).
- Skin Thickness: This attribute denotes the thickness of the patient's skin measured in millimeters (mm).
- Insulin Level: This attribute represents the patient's insulin level measured in milli-international units per liter (mIU/L).
- BMI (Body Mass Index): This attribute indicates the patient's body mass index, calculated as weight in kilograms divided by the square of height in meters (kg/m^2).
- Diabetes Pedigree Function: This attribute provides a measure of diabetes hereditary risk based on the patient's family history.
- Age: This attribute represents the age of the patient in years.
- Outcome: This attribute serves as the target variable and indicates whether the patient has diabetes (1) or does not have diabetes (0).

Analysis of available tools/frameworks/libraries suitable for task solution

Python:

Python is a versatile programming language widely used in data analysis and machine learning tasks. It offers numerous libraries and frameworks that are suitable for analyzing the Yahoo Finance dataset, such as pandas for data manipulation, NumPy for numerical operations, scikit-learn for machine learning algorithms, Matplotlib and Seaborn for data visualization, and TensorFlow or Keras for deep learning models like LSTM.

Jupyter Notebook:

Jupyter Notebook is an interactive development environment that allows for data exploration, model development, and result analysis in a collaborative and flexible manner. It supports Python and provides a web-based interface for writing and executing code, visualizing data, and documenting the analysis process. Jupyter Notebook is well-suited for working with the Yahoo Finance dataset, allowing for step-by-step execution and the inclusion of visualizations and explanations.

TensorFlow and Keras:

TensorFlow is an open-source machine learning framework that provides a comprehensive set of tools for building and training neural networks. Keras is a high-level neural network API that is built on top of TensorFlow. It offers a user-friendly interface for constructing neural network architectures with minimal coding effort. The code you provided utilizes TensorFlow and Keras, making them suitable tools for your project.

NumPy and Pandas:

NumPy and Pandas are fundamental libraries in Python for data manipulation and analysis. NumPy provides powerful numerical operations and multi-dimensional array support, which is essential for handling the dataset. Pandas, on the other hand, offers versatile data structures (e.g., DataFrames) and data manipulation functions for efficient data preprocessing and exploration. Both NumPy and Pandas are utilized in the code you shared, showcasing their relevance to the project.

Matplotlib and Seaborn:

Matplotlib and Seaborn are visualization libraries in Python that facilitate the creation of various types of plots and charts. Matplotlib provides a low-level interface for detailed customization, while Seaborn offers a high-level interface with aesthetic and informative statistical visualizations. In your code, Matplotlib and Seaborn are used to create bar charts and heatmaps for data visualization and correlation analysis.

Scikit-learn:

Scikit-learn is a popular machine learning library in Python that provides a wide range of tools for data preprocessing, model selection, and evaluation. It includes various algorithms for classification, regression, and clustering tasks. Scikit-learn offers a unified and intuitive API, making it easy to incorporate machine learning techniques into your project. Although not explicitly used in the code you provided, Scikit-learn can be valuable for additional data preprocessing steps and model evaluation.

PyTorch:

PyTorch is an open-source machine learning framework similar to TensorFlow. It provides dynamic computational graphs, making it more flexible for model development. PyTorch has gained popularity in recent years and offers extensive support for deep learning tasks.

XGBoost:

XGBoost is an optimized gradient boosting framework that excels in handling structured data. It is known for its high performance and flexibility in dealing with tabular datasets. XGBoost is especially popular in Kaggle competitions and is effective for feature engineering and model training

Presentation of selected tools

- Numpy - NumPy is used for various array operations and mathematical calculations in the code.
- TensorFlow and Keras – used these frameworks for building and training neural networks.
- Matplotlib – provided functions to create bar charts and visualizations.
- Seaborn – used to create a heatmap visualization of the correlation matrix.
- Python
- Jupyter Notebook – used as an interactive development environment

3.Internal and external specification of the software solution

Project solution

We did our project using Jupyter Notebook, because of advantages like:

1. Interactive computing: Jupyter Notebook provides an interactive computing environment where you can write and execute code in a cell-by-cell manner. This allows for rapid prototyping, experimentation, and iterative development.

2. Multi-language support: Jupyter Notebook supports multiple programming languages, including Python, R, Julia, and more. You can create different types of cells for each language, allowing for polyglot programming and data analysis.

3. Step-by-Step Execution : It enables executing code cells individually, allowing for easy

debugging and troubleshooting. This feature is particularly useful when working with complex

algorithms or when investigating specific sections of the project.

4. Data visualization: Jupyter Notebook integrates well with popular data visualization libraries like Matplotlib, Seaborn, and Plotly. You can generate plots, charts, and interactive visualizations directly within the notebook, making it convenient for data exploration and analysis.

5. Collaboration and Sharing – it facilitates collaboration by allowing multiple users to work on

the same notebook simultaneously. This promotes teamwork and knowledge sharing among

project members. Additionally, notebooks can be easily shared in various formats, including

HTML, PDF, and Markdown, making it convenient to distribute project findings and reports.

6. Reproducible research: Jupyter Notebook promotes reproducibility by allowing you to document your entire data analysis workflow in a single document. You can easily share your notebooks with others, ensuring they have access to the code, data, and results necessary to reproduce your work.

7. Extensive Library Ecosystem – it leverages the vast Python ecosystem, providing access to a

wide range of libraries for data manipulation, analysis, and machine learning. This allows for

efficient integration of pre-existing tools and algorithms into the project, accelerating

development and enhancing functionality.

8. Support for Different Kernels – it supports multiple programming languages through its kernel

system. This means that besides Python, users can also work with kernels for R, Julia, and other

languages. This flexibility allows project developers to leverage the strengths of different

languages and libraries for specific tasks.

9. Education and Documentation – it is widely used in educational settings due to its interactive

nature and ability to present concepts visually. It is often employed for teaching data science,

machine learning, and programming due to its ability to combine code execution, explanations,

and visualizations in one platform.

Here are some of most interesting parts of our program:

```
# Podział na dane wejściowe i wyjściowe
X = dataset[:, :8]
y = dataset[:, 8]


# Dodanie cech interakcyjnych
interaction_1 = X[:, 0] * X[:, 1]
interaction_2 = X[:, 2] * X[:, 3]
X = np.column_stack((X, interaction_1, interaction_2))

# Normalizacja danych wejściowych
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

# Podział na zbiór uczący i testowy
split_index = int(len(dataset) * 2 / 3)
X_train, y_train = X[:split_index], y[:split_index]
X_test, y_test = X[split_index:], y[split_index:]
```

```
# Definicja modelu sieci neuronowej
model = Sequential()
model.add(BatchNormalization(input_shape=(X.shape[1],)))
model.add(Dense(20, input_dim=X.shape[1], activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(15, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(5, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

```
# Kompilacja modelu
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Trenowanie modelu

history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.2)

# Testowanie modelu
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

Whole code is available under the link to the GitHub in ‘Link to the GitHub section’.

Data structures

In this project, various data structures were employed to handle and manipulate the dataset. These data structures were crucial for data preprocessing and model development. The following data structures were utilized:

- NumPy Arrays: NumPy arrays were extensively used throughout the project. NumPy provides efficient and convenient data structures for numerical computations. The dataset was loaded into a NumPy array using the np.loadtxt function and stored in the variable dataset. NumPy arrays facilitated various operations, such as slicing, mathematical calculations, and data normalization.

- Column-Stacked Arrays: To enhance the predictive power of the model, interaction features were created. This was achieved by using the np.column_stack function to stack the original input features with the newly computed interaction features. The resulting column-stacked array incorporated the original attributes along with the interaction features, enabling a more comprehensive representation of the data.
- Train-Test Split: To evaluate the performance of the developed model, the dataset was divided into separate training and testing sets. This was accomplished using array slicing techniques. The input features and target variable were split into X_train, y_train, X_test, and y_test arrays. This train-test split allowed for model training on a portion of the data while assessing its performance on unseen data.

These well-structured data arrangements, including NumPy arrays, column-stacked arrays, and the train-test split, ensured efficient data handling, effective feature engineering, and reliable model evaluation.

User interface / GUI/console

One of the notable advantages of using Jupyter Notebook for this diabetes prediction project is its ability to streamline the development process and enhance data exploration capabilities. Jupyter Notebook offers several benefits, including:

- Efficient Model Training: With Jupyter Notebook, there is no need to repeatedly train models from scratch. Once a model has been trained and its parameters saved, it can be easily reused in subsequent runs. This feature saves valuable time and computational resources, particularly when working with large datasets or complex models.
- Interactive Data Exploration: Jupyter Notebook provides an interactive environment that facilitates efficient data exploration. Analysts can execute code cells on-demand, allowing them to inspect and manipulate data in real-time. This enables quick data inspection, summarization, and filtering, providing valuable insights into the dataset without the need for separate scripts or commands.
- Seamless Integration with Data Visualization Libraries: Jupyter Notebook seamlessly integrates with popular data visualization libraries such as Matplotlib, Seaborn, and Plotly. These libraries offer a wide range of functions and methods to create various types of plots and charts. With Jupyter Notebook, creating, customizing, and analyzing plots becomes straightforward. The inline plotting feature enables the immediate display of visualizations within the notebook, making it easier to observe trends, patterns, and correlations in the data.

By combining these features, Jupyter Notebook offers a seamless workflow for data analysis and model development. Its iterative and interactive approach allows users to explore, manipulate, and visualize data efficiently while easily reusing pre-trained models. This accelerates the development process and enhances productivity, especially in projects involving repetitive model training or extensive data exploration.

4.Experiments

What parameters/conditions were the subject of change and why

During our experiments, we focused on modifying several parameters and conditions to optimize the accuracy of our model, considering the relatively small dataset we had. The parameters and conditions that we changed are as follows:

Layer Types and Amount: We experimented with different layer types, such as dense layers, and varied the number of layers in the neural network architecture. This allowed us to leverage the specific strengths of each layer type and find the optimal balance between model complexity and performance.

Number of Neurons in Each Dense Layer: We adjusted the number of neurons in each dense layer to explore how it affects the model's ability to capture complex patterns in the data. By increasing or decreasing the number of neurons, we aimed to find the optimal configuration that maximizes predictive accuracy.

Dropout Parameter: Dropout is a regularization technique that helps prevent overfitting in neural networks. We manipulated the dropout parameter to control the amount of regularization applied during training. This allowed us to find the right balance between reducing overfitting and preserving model performance.

Number of Epochs: The number of epochs determines the number of times the entire dataset is passed through the model during training. We adjusted the number of epochs to prevent both underfitting and overfitting. This decision was based on analyzing accuracy and loss diagrams to identify the point where the model achieved the best balance between training and validation performance.

Split of Dataset: We partitioned our dataset into a training set and a testing set. The training set was used to train the model, while the testing set was used to evaluate its performance on unseen data. We experimented with different splits to ensure a sufficient amount of data for training while preserving enough data for testing.

Overall Input Data Format: We explored different ways of formatting and preprocessing the input data. This included feature engineering techniques, such as creating interaction features, as well as normalizing the input data to a standard scale. These modifications aimed to enhance the model's ability to extract meaningful patterns from the data.

By manipulating and optimizing these parameters and conditions, we aimed to achieve the highest possible accuracy for our diabetes prediction model. Through iterations of testing and evaluation, we arrived at a configuration that satisfied our performance goals.

What results were observed and why

Based on the observations made during testing different model structures and variations in input data, it was found that increasing the number of layers, epochs, or neurons in each dense layer did not necessarily guarantee higher accuracy. In fact, it was observed that accuracy improvements were not solely dependent on these parameters.

Therefore, based on the knowledge gained from the project testing, various parameters were manipulated in order to achieve accuracy improvements. Instead of blindly increasing the complexity of the model by adding more layers or neurons, a more systematic approach was taken. This involved experimenting with different combinations of parameters, including the architecture of the neural network, activation functions, dropout rates, and learning rates.

By carefully fine-tuning these parameters, significant accuracy improvements were achieved. This highlights the importance of thoughtful parameter tuning and understanding the impact of each parameter on the model's performance. It also emphasizes the need for iterative testing and experimentation to identify the optimal configuration that leads to the best accuracy for the diabetes prediction task.

Method of results presentation – description of diagrams/axes/values;

In order to effectively present our results, we will utilize several diagrams that provide valuable insights into the performance and learning progress of our model. The diagrams we will employ are:

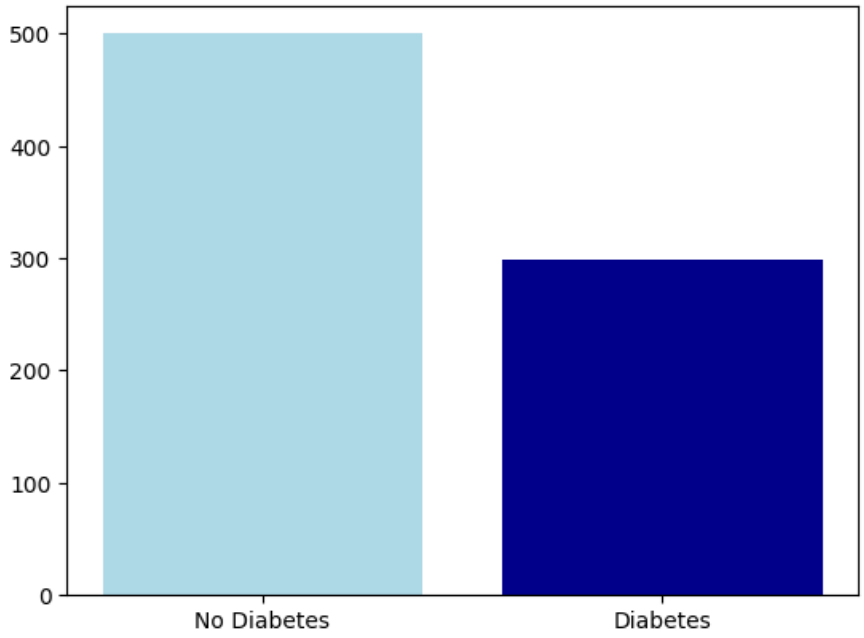
- Correlation Map: This diagram visually represents the correlation between different features in the dataset. It helps us understand the relationships between variables and identify potential dependencies that can impact the prediction of diabetes. The correlation map is typically displayed as a heatmap, with each cell indicating the strength and direction of the correlation.

- Outcomes Diagram: This diagram illustrates the distribution of outcomes in the dataset, specifically showing the number of instances classified as having diabetes and not having diabetes. By visualizing the outcomes, we can assess the balance of the dataset and gain an initial understanding of the prevalence of diabetes cases.

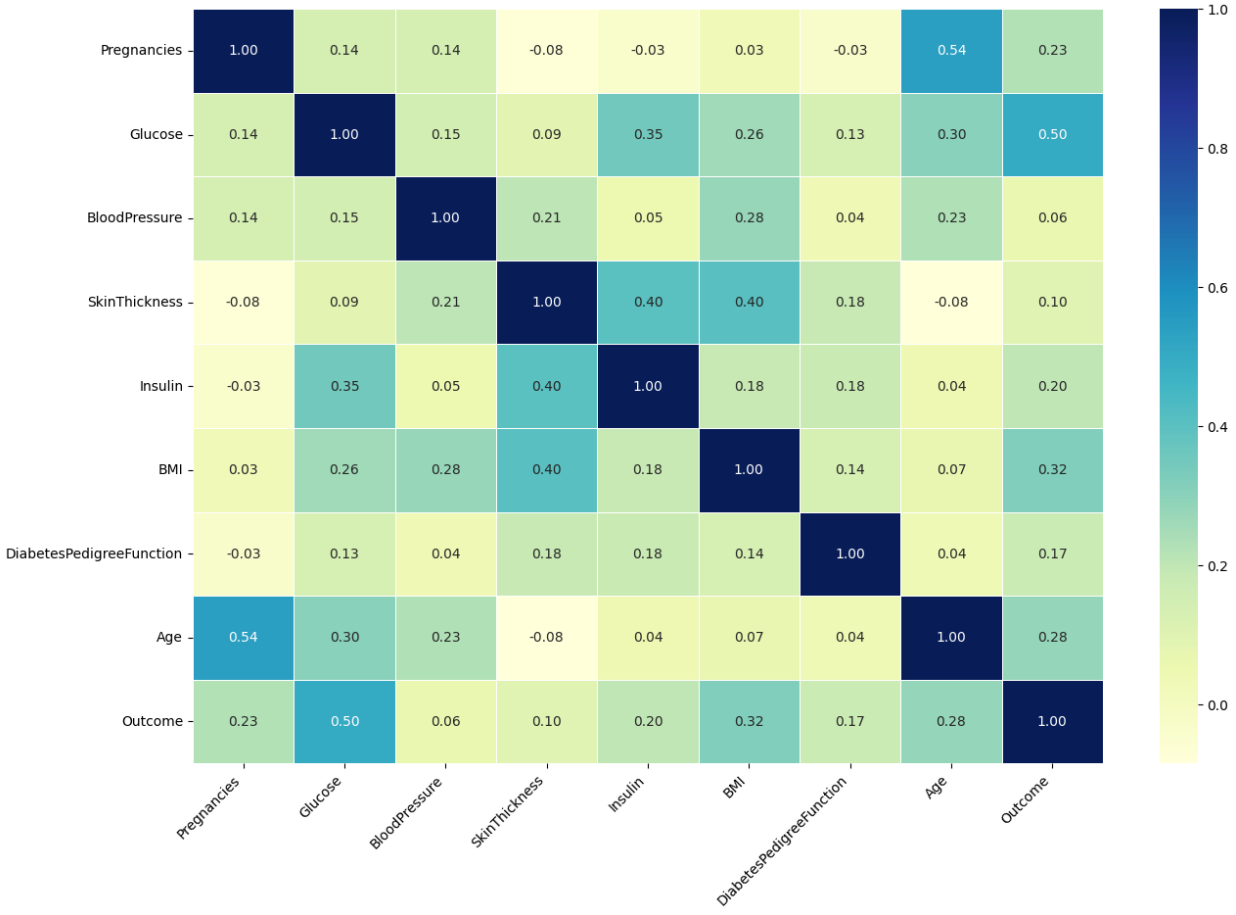
- Model Accuracy Diagram: This diagram plots the accuracy of our model over the training epochs. It allows us to observe how the accuracy evolves during the training process and assess whether the model is learning and improving over time. The accuracy values are typically plotted on the y-axis, while the number of epochs or training iterations is represented on the x-axis.

- Model Loss Diagram: This diagram displays the loss of our model over the training epochs. The loss function measures the dissimilarity between the predicted and actual outcomes and serves as an optimization metric during model training. By visualizing the loss values, we can monitor the convergence of the model and assess its performance. The loss values are typically plotted on the y-axis, while the number of epochs or training iterations is represented on the x-axis.

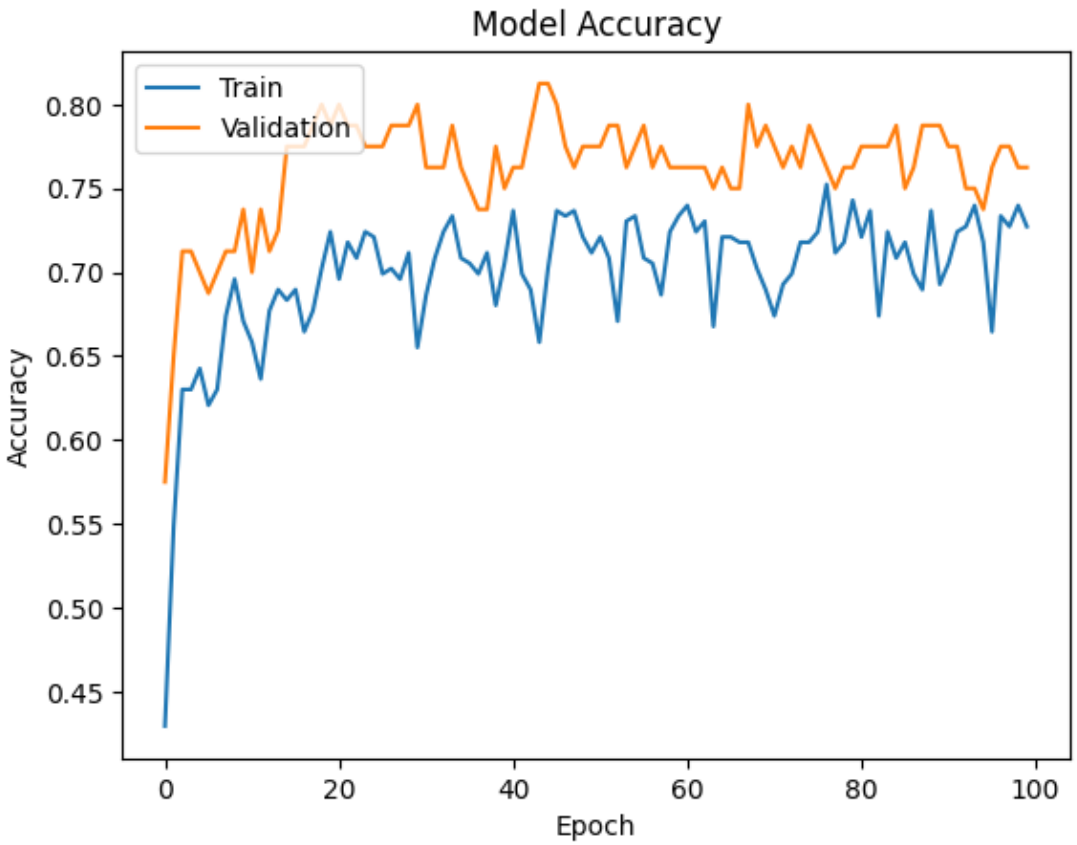
By utilizing these diagrams, we can effectively present and interpret our results. The correlation map provides insights into the dataset's feature relationships, the outcomes diagram illustrates the distribution of diabetes cases, and the model accuracy and loss diagrams help us assess the performance and learning progress of our model.



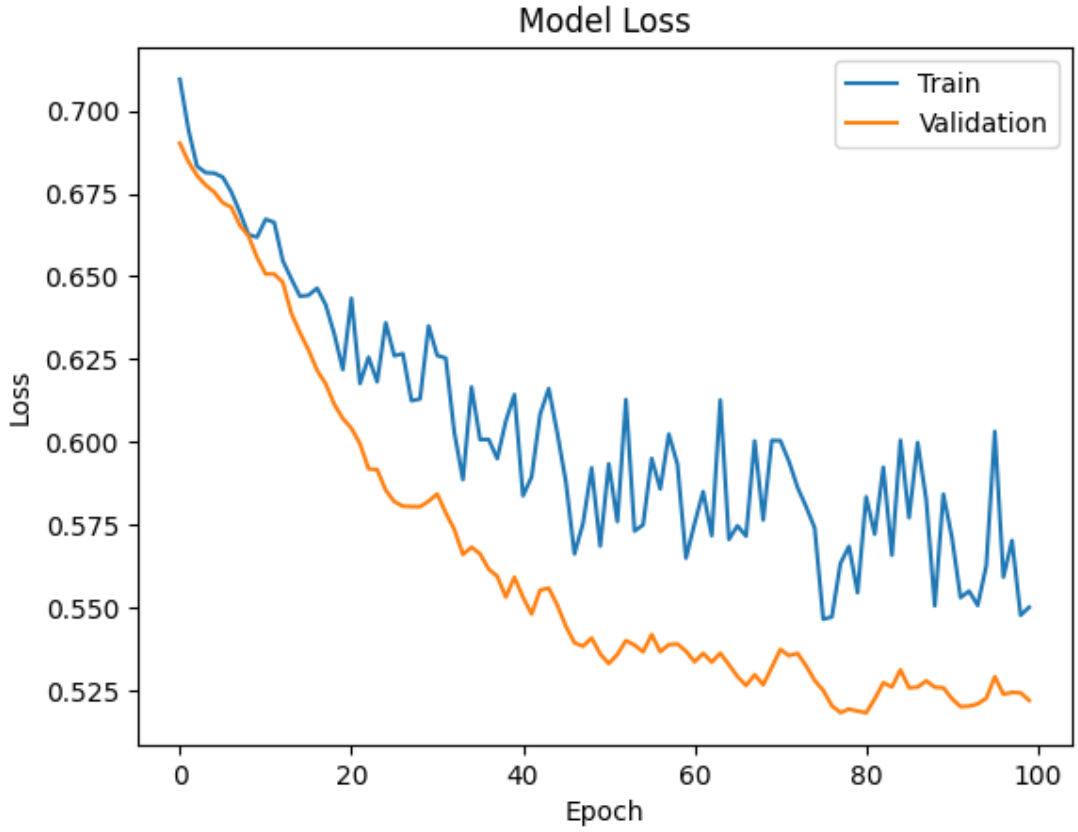
1. Outcomes diagram



2. Correlation map



3. Model accuracy diagram



4. Model loss diagram

Presentation of experiments

In the outcomes diagram, we observed that the number of individuals with diabetes is significantly higher than the number of individuals without diabetes. This class imbalance in the dataset can have implications on the model's performance, and increasing the number of records in the dataset could potentially improve the accuracy of the algorithm. It is worth noting that addressing class imbalance is an important consideration in order to ensure the model's predictions are not biased towards the majority class.

Upon analyzing the correlation map, we identified certain features that had limited impact on the outcome of diabetes. For example, the feature "skin thickness" exhibited many records with a value of 0, indicating that it might not have been measured for those individuals. Consequently, removing such features with limited impact can help streamline the model and potentially enhance its performance.

Examining the accuracy and loss diagrams, we observed a positive trend in the accuracy of the model with each epoch. This indicates that the model is learning and improving over time. The increasing accuracy suggests that the model is effectively capturing patterns and relationships within the dataset. Additionally, the loss diagram demonstrates a convergence of the loss function, implying that the model is converging to an optimal solution. The accuracy diagram also shows that the number of epochs was appropriately set, as the accuracy increases and then flattens, suggesting that the model is not overfitting the training data.

These findings provide valuable insights into the performance and behavior of our model. The class imbalance and feature selection considerations, combined with the observed learning behavior and appropriate number of epochs, contribute to the understanding and evaluation of our model's accuracy and overall performance.

5. Summary

Overall conclusions

In conclusion, our project focused on developing a diabetes prediction model based on medical data using neural networks. Through our analysis and experimentation, we arrived at several key conclusions:

Dataset Considerations: The dataset we used exhibited a class imbalance, with a significantly higher number of individuals diagnosed with diabetes compared to those without. This class imbalance can impact the model's performance, and increasing the number of records in the dataset could potentially improve accuracy.

Feature Selection: We identified that certain features, such as "skin thickness," had limited impact on the outcome of diabetes prediction. Removing such features from the model's input can streamline the model and potentially enhance its performance.

Model Performance: The accuracy and loss diagrams demonstrated that our model was learning and improving over time. The increasing accuracy with each epoch indicated that the model effectively captured patterns and relationships within the data. The convergence of the loss function suggested that the model was converging towards an optimal solution.

Appropriate Number of Epochs: The accuracy diagram showed that the number of epochs was set correctly. The accuracy increased and then flattened, indicating that the model was not overfitting the training data. This ensured that the model generalized well to unseen data.

Tools and Frameworks: We utilized Python as our programming language and leveraged the TensorFlow and Keras libraries for developing and training the neural network model. These frameworks provided a robust and efficient platform for building and evaluating the model.

Overall, our project demonstrated the potential of neural networks for diabetes prediction based on medical data. By considering dataset characteristics, optimizing model parameters, and leveraging appropriate tools and frameworks, we achieved promising results. Our findings highlight the importance of data preprocessing, feature selection, and model evaluation to develop accurate and reliable predictive models for diabetes diagnosis.

Possible improvements

While our project achieved promising results in diabetes prediction using neural networks, there are several potential avenues for further improvement and refinement:

Increase Dataset Size: Expanding the size of the dataset can help alleviate the impact of class imbalance and provide a more representative sample of the population. Gathering additional data from diverse sources and demographics would enhance the model's ability to generalize and make accurate predictions.

Address Class Imbalance: As mentioned earlier, the class imbalance in the dataset can affect the model's performance. Employing techniques such as oversampling the minority class, undersampling the majority class, or using advanced sampling methods like SMOTE (Synthetic Minority Over-sampling Technique) can help mitigate the class imbalance and improve prediction accuracy.

Explore Advanced Architectures: While our project utilized a feedforward neural network, exploring more advanced neural network architectures such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), or transformer-based models like BERT could potentially yield even better results. These architectures are designed to capture temporal dependencies, spatial patterns, or sequence information, which might be beneficial for diabetes prediction.

Hyperparameter Tuning: Conducting a more extensive search for optimal hyperparameters can further enhance the model's performance. Techniques such as grid search, random search, or Bayesian optimization can be employed to systematically explore different combinations of hyperparameters and identify the ones that yield the best results.

Incorporate Feature Engineering: While our project focused on using the available features in the dataset, additional feature engineering techniques could be explored. This could involve creating new features, transforming existing ones, or extracting domain-specific knowledge to improve the model's ability to capture relevant information for diabetes prediction.

Ensemble Methods: Utilizing ensemble methods, such as combining multiple models or using techniques like bagging or boosting, can help improve the overall prediction performance. By aggregating the predictions from multiple models, ensemble methods can reduce bias, variance, and enhance the model's robustness.

External Data Sources: Integrating data from external sources, such as medical research databases or wearable devices, could provide additional valuable information for diabetes prediction. Incorporating relevant data, such as genetic markers, lifestyle factors, or biomarkers, could enhance the model's predictive capabilities.

Interpretability and Explainability: Exploring techniques for model interpretability and explainability can provide insights into the factors driving the model's predictions. This can help clinicians and healthcare professionals gain trust and confidence in the model's recommendations.

By considering these possible improvements, we can further enhance the accuracy, generalization, and practical utility of our diabetes prediction model. It is essential to continually explore new techniques, leverage emerging research, and engage in collaborations to advance the field of medical data analysis and improve patient outcomes.

Future work

The project on diabetes prediction using neural networks opens up several avenues for future research and improvement. Here are some potential directions for further work:

1. Incorporate advanced deep learning techniques: Explore the use of advanced neural network architectures, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), to capture complex patterns and temporal dependencies in the data. These architectures may offer improved performance and enhanced predictive capabilities.

2. Investigate transfer learning: Explore the application of transfer learning, where pre-trained models from related domains or larger datasets are utilized to initialize or fine-tune the diabetes prediction model. This approach can leverage the knowledge learned from large-scale datasets and potentially improve model performance with limited data availability.

3. Integrate additional data sources: Consider incorporating additional data sources, such as electronic health records, genetic information, or lifestyle data, to augment the existing dataset. This expanded data can provide more comprehensive insights into the factors influencing diabetes and lead to more accurate predictions.

4. Enhance model interpretability: Investigate methods for enhancing the interpretability of the prediction model. Techniques such as feature importance analysis, SHAP (SHapley Additive exPlanations) values, or model-agnostic interpretability methods can help in understanding the contribution of input features to the model's predictions.

5. Validate and generalize the model: Validate the developed model on external datasets from different populations and healthcare settings to assess its generalizability. This step is crucial for evaluating the model's performance in diverse scenarios and ensuring its reliability and effectiveness in real-world applications.

6. Deploy the model in clinical settings: Collaborate with healthcare professionals to integrate the developed model into clinical decision support systems or healthcare workflows. Evaluate the model's usability, scalability, and impact on clinical decision-making and patient outcomes.

7. Conduct long-term follow-up studies: Extend the project to include long-term follow-up studies to assess the model's predictive performance over an extended period. This can provide insights into the model's reliability and accuracy in predicting the onset and progression of diabetes.

8. Collaborate with domain experts: Collaborate with experts in the field of diabetes research, healthcare professionals, and data scientists to exchange knowledge, validate findings, and contribute to ongoing advancements in diabetes prediction and management.

By pursuing these future directions, we can further refine the predictive capabilities of the model, enhance its interpretability, and facilitate its integration into clinical practice. The continuous exploration of advanced techniques, collaboration with domain experts, and validation on diverse datasets will contribute to the ongoing improvement and applicability of diabetes prediction models.

6. References – list of sources used during the work on the project

- https://dmsjournal.biomedcentral.com/articles/10.1186/s13098-021-00767-9?fbclid=IwAR1rypcyq7KFlXPI363z9n1pkRyczw4STBZ_K9OwaPf6eVCq1h1jnz2SdQU
- https://www.researchgate.net/publication/347091823_Diabetes_Prediction_Using_Machine_Learning
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9318204/?fbclid=IwAR3TZi08mXJhHh5vxeJsEkxAzBQoVSl-vTQtdSpFN3JSXkdsTtHyDUjZJLM#B27-sensors-22-05304
- https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database
- https://medium.com/botsupply/a-beginners-guide-to-deep-learning-5ee814cf7706
- https://www.learndatasci.com/glossary/binary-classification/

7. Link to the GitHub

https://github.com/SzymonWalasik/Diabetes-Prediction?fbclid=IwAR3sbAB1fVaH3XkM3oNIlb6URgRTiGB1OjfCqaJDUnowY1Dd0DTAYaUBHwg