

CHAPTER 3 – RESEARCH METHODS

3.1 Research Overview

The research overview is shown in *Figure 3.1*. The research begins by gathering and preparing data to train the model. After that, the model will be trained and accuracy check will be done. After accuracy check if performed, fine-tuning will also be performed and comparisons will be made. At the same time, the web-application will be developed in parallel. The model with the best result will be implemented into the web-application at the end.

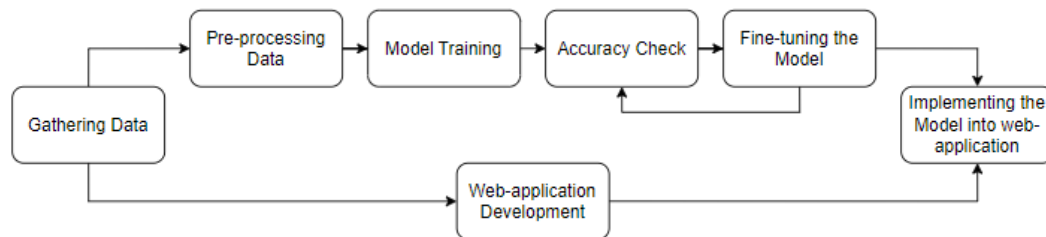


Figure 3.1: Research Overview

3.2 Machine Learning

3.2.1 Experimental Environment

The experiment will be conducted using Python 3.10. The main library used for this research is scikit-learn library since it contains machine learning related libraries. As for the hardware used, it comprises Intel i7-8850H, Nvidia Quadro P600 4GB DDR5, 16 GB RAM, and a cloud notebook in the form of Google Colab with T4 GPU.

3.2.2 Dataset Gathering

The dataset used for this research is taken from past medical records from XYZ Clinic spanning from 2021 to early 2024. These medical records are in the form of textbox which contain information such as symptoms, pre-diagnose, and diagnose, although there are many records that do not have complete information. The diagnoses are

made by experts, in this case senior veterinarians and sometimes junior veterinarians. There are three different datasets obtained from the medical record. The first dataset consists of 68 rows with 3 different labels and the second dataset consists of 176 rows with 5 different labels. There is also a synthetic dataset made with the guidance of a senior veterinarian which consists of 3000 rows with 3 different labels which are not manually assigned by the senior veterinarian. These datasets are used for the multi-class classification training.

There is also a third dataset consisting 155 rows with 5 different labels for training and 34 rows for testing. This dataset is used for both multi-class classification and multi-label classification training. This dataset is chosen to be used for the multi-label classification is because it contains rows with more than 1 labels assigned. For the multi-class classification training done using this dataset, those rows with more than 1 labels are split into 2 different rows instead.

3.2.3 Pre-processing Dataset

The medical records were in the format of paragraphs inside a text box. Therefore, they need to be transformed first into a format that can be processed by the model, in this case CSV. Furthermore, the dataset will be cleaned from empty rows and then transformed into a suitable format for the machine learning.

The attributes for each row are grouped into two; numeric and categorical. In order for the model to comprehend the categorical variables, they need to be transformed into a more readable format. In this case, every single option on each attribute is turned into a binary option. This makes it more readable for the model because it will know whether the existence of an option is true or false.

For the multi-class classification training, the dataset is split into training set and test set with a ratio of 70/30, 75/25, and 80/20. The ratio that gives the best score overall will be used for the final model. The dataset used for the multi-label classification training does not need to be splitted because it already has its own distinct test dataset. The sequential flow of the machine learning program is shown in *Figure 3.2*.

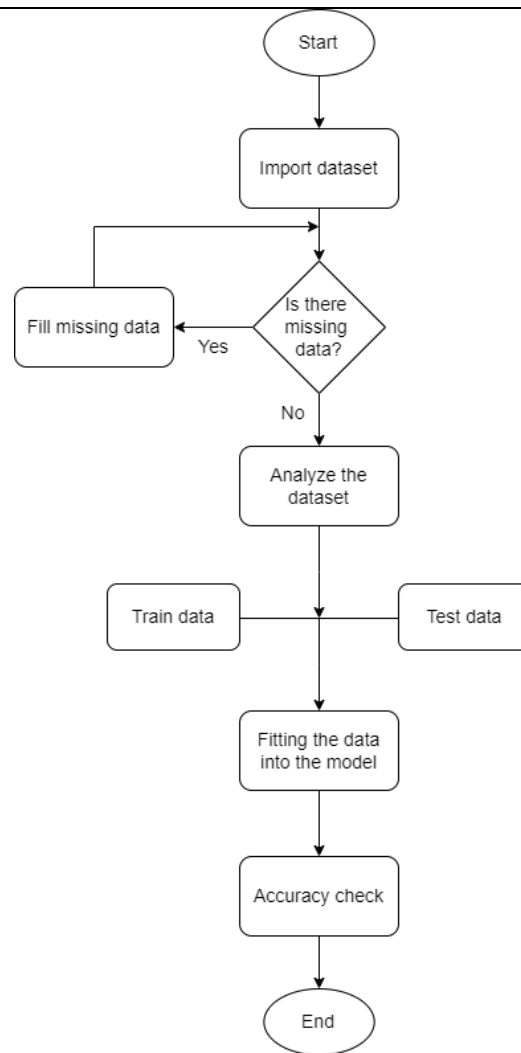


Figure 3.2: Sequential Flow of the Machine Learning Program

3.2.4 Training the Model

The machine learning models used for this research are: Linear Regression, Naïve Bayes, K-Nearest Neighbor, Random Forest Algorithm, and Support Vector Machine.

The parameters of each model will be as shown in *Figure 3.3*.

```
'Logistic Regression': LogisticRegression(random_state=42),  
'Random Forest': RandomForestClassifier(random_state=42),  
'Naive Bayes': GaussianNB(),  
'KNN': KNeighborsClassifier(),  
'SVM' : SVC(probability=True)
```

Figure 3.3: Parameters of Each Machine Learning Model

For Logistic Regression and Random Forest, the parameter *random_state* is set to an integer in order to produce the same results when splitting the test and train datasets across different calls. If the *random_state* is not specified, the test and train datasets will have different values everytime the code is executed.

For Support Vector Machine, the parameter *probability* is set to True to enable probability estimations, which means we can get the probability for each class instead of only getting the class with highest probability.

3.2.5 Accuracy Check

The evaluation metrics for multi-class classification include precision, recall, accuracy, and F1-score. These metrics utilize components like True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). The evaluation of the model's performance includes metrics such as precision, recall, accuracy, and F1-score.

Precision is used to measure how many positive predictions made by the model are correct. The formula to calculate precision is shown in *Equation 3.1*.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (3.1)$$

Recall, is used to measure how many positive cases in the dataset are correctly predicted by the model as shown in *Equation 3.2*.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (3.2)$$

The F1-score calculates the harmonic mean of recall and precision. It symbolizes the harmony between recall and precision. F1 score has the best score at 1 when both precision and recall are 1 and worst score at 0 when either precision or recall is 0. It is also commonly used to measure performance for imbalanced datasets. Generally, an F1-score of 0.5-0.8 is considered good, 0.8-0.9 is considered excellent while above 0.9 is outstanding (Nikolaj Buhl, 2023). F1-score is calculated using *Equation 3.3*.

$$F1\text{-Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.3)$$

Accuracy is used to calculate the percentage of true predictions of all predictions as shown in *Equation 3.4*.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.4)$$

To validate the result of the training, a validation technique called K-Fold Cross Validation is also done. In machine learning, cross validation is used to evaluate a model's performance on unseen data. The dataset is split into folds, with one-fold designated as a validation set and the remaining folds used for training. This process is repeated multiple times with different validation set for each iteration. In order to produce a more accurate evaluation of the model's performance, the results from each fold are summed in the end. The validation method performed in this research uses StratifiedKFold Cross Validation method with 10-fold repetition.

For multi-label classification, the metric used for evaluation is ROC-AUC score. This is a percentage value of the size of Area Under (AUC) the Receiver Operating Characteristic Curve (ROC). The ROC curve plots the True Positive (TP) rate against the False Positive (FP) rate at different classification thresholds. For binary classification, these thresholds represent distinct probabilities used to differentiate between the two classes, showing how effectively the model performs classification.

The model can distinguish between the two classes better if the AUC score is closer to 1, in which case the curve approaches the upper left corner of the graph. Even if two models have comparable accuracies, we can have more faith in the one with a higher AUC score because it is a statistic that takes into account the probability of the class predictions. Generally, an AUC score of 0.7-0.8 is considered good enough, 0.8-0.9 is excellent, and above 0.9 is outstanding (Mandrekar, 2010).

3.2.6 Fine-tuning the Model

In fine-tuning the model, each model has their hyperparameter tuned in order to maximize the performance of the model. To determine the best value for each parameter, RandomizedSearchCV and GridSearchCV are used. They both experiments with a predetermined number of parameter configurations drawn from the designated distributions and will return the best score achieved by a certain set of parameters. The difference is, RandomizedSearchCV will only try out several combination of hyperparameters meanwhile GridSearchCV will try out all possible combinations. Fine-tuning is performed only for the multi-label classification because it will be the one implemented for the diagnose system instead of the multi-class classification.

```
lr_params = {  
    'solver': ['liblinear'],  
    'penalty': ['l1', 'l2'],  
    'C': np.logspace(-4, 4, 20),  
}
```

Figure 3.4:Parameters for Logistic Regression

The parameters for Logistic Regression model are shown in *Figure 3.4*. Solver is the algorithm to be used in the optimization problem. Algorithm 'liblinear' is chosen because it has a good result for small datasets. Penalty refers to the norm of penalty used. C is the regularization parameter which is inverse of regularization strength.

```
rf_params = {  
    'n_estimators': np.arange(100, 1050, 50),  
    'max_depth': [5, 7, 9, 11],  
    'min_samples_leaf': [3, 5, 7, 9],  
    'min_samples_split': [2, 4, 6, 8],  
}
```

Figure 3.5:Parameters for Random Forest

The parameters for Random Forest model are shown in *Figure 3.5*. Parameter `n_estimators` is used to state the total number of trees in the forest. `Max_depth` is used to specify the maximum depth of the tree. `Min_samples_leaf` specifies the minimum number of samples required to be at a leaf node while `min_samples_split` is the minimum number of samples required to split an internal node.

```
nb_params = {  
    'var_smoothing': np.logspace(0, -9, num=200)  
}
```

Figure 3.6:Parameter for Naïve Bayes

The only parameter available for Naïve Bayes model is shown in *Figure 3.6*. Var_smoothing determines the amount of the largest variation among all features that is added to variances for computation stability.

```
knn_params = {  
    'n_neighbors': [3, 5, 7, 9, 11, 13, 15, 17],  
    'weights': ['uniform', 'distance'],  
    'metric': ['minkowski', 'euclidean', 'manhattan']  
}
```

Figure 3.7:Parameters for K-Nearest Neighbor

The parameters for K-Nearest Neighbor model are shown in *Figure 3.7*. Parameter n_neighbors is used to determine the number of neighbors by default. Weights is used to specify the weight function used in prediction; uniform means every point in every neighborhood has the same weight. while distance means a query point's closest neighbors will have more of an impact than its farther neighbors. Metric specifies the metric to be used for distance computation.

```
svm_params = {  
    'C': [0.1, 1, 10, 100, 1000],  
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],  
}
```

Figure 3.8:Parameters for Support Vector Machine

The parameters for Support Vector Machine model are shown in *Figure 3.8*. Kernel specifies the kernel type to be used in the algorithm. Same as in Logistic Regression, C is the regularization parameter which is inverse of regularization strength. Gamma is the kernel coefficient used.

3.3 Web-application Development

Before the making of this research, XYZ Clinic already has a fully functional web-application built using ReactJS. The existing web-application has CRUD feature for:

- Appointment
- Customer
- Medical Record
- Overview
- Patient

The new features added into the web-application that is related with this research are:

- Diagnose
- Disease
- Symptom

The architecture diagram of the system is shown in *Figure 3.9* with the green box highlighting the newly added feature. The activity diagram of the newly added feature is shown in *Figure 3.10* and the use-case diagram is shown in *Figure 3.11*.

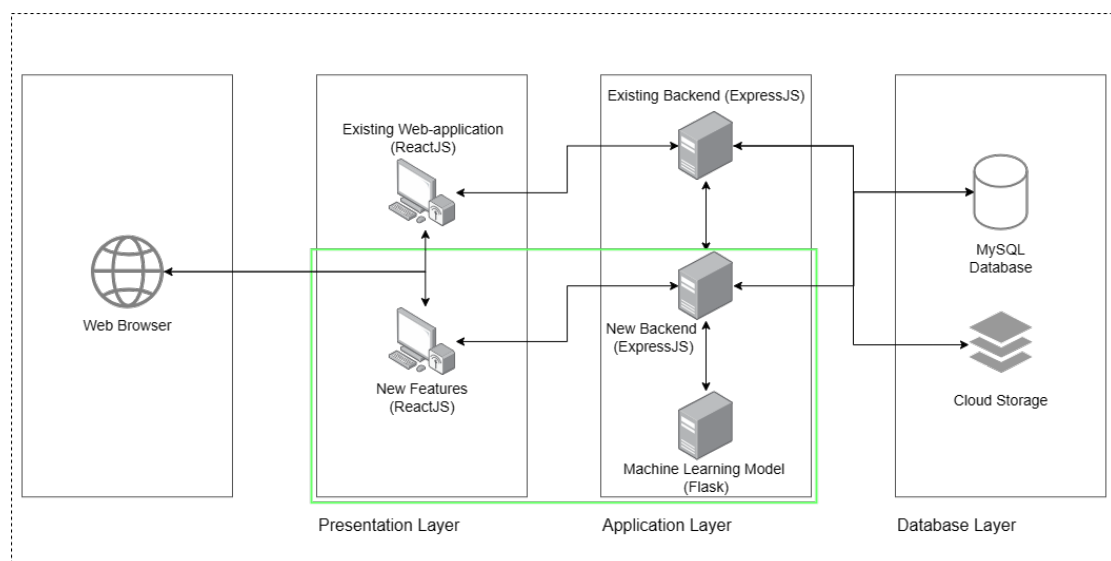


Figure 3.9:Architecture Diagram of the System

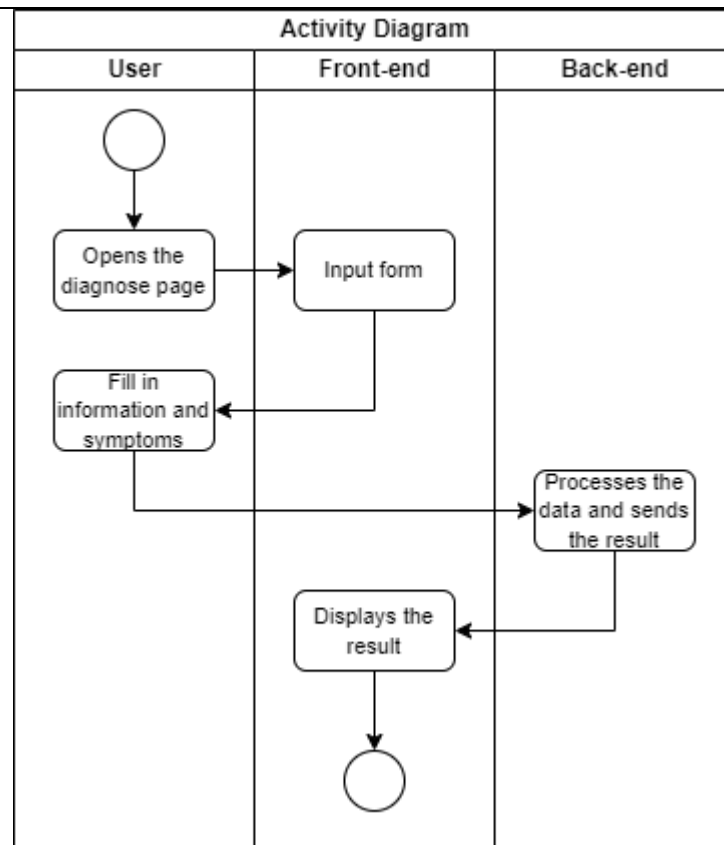


Figure 3.10:Activity Diagram of the Web-application

The user, in this case junior veterinarians will have to open the diagnose page from the existing web-application in order to see the input form. The input form will then be filled out with information regarding the patient and available symptoms.

This form will then be sent to the ExpressJS backend and the form data will be processed and sent to the Flask server to get a prediction result from the machine learning. The prediction result will be passed back to the ExpressJS backend and sent to the frontend to be displayed.

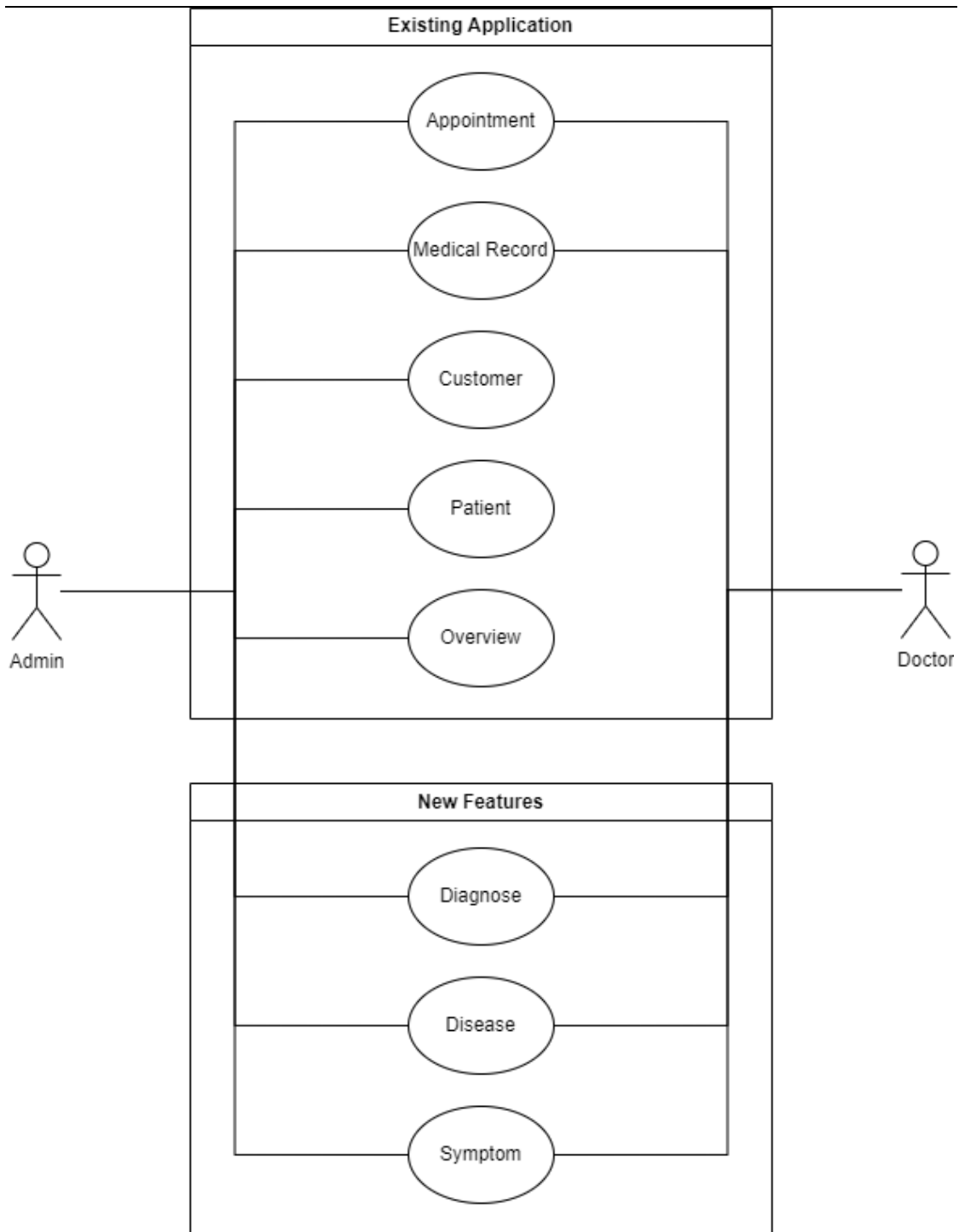


Figure 3.11:Use Case Diagram for the Web-application

3.4 Integrating the Model into the Web-application

The model will be deployed using Flask on a cloud server. An API will be made so the web-application will be able to connect to the Flask server. This API will only be called when the diagnose function is called.