PHYSICIAN FRIENDLY MACHINE LEARNING

by
Meghana Padmanabhan

A thesis submitted to the Department of Electrical and Computer Engineering,
Cullen College of Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in Computer and Systems Engineering

Chair of Committee: Dr. Hien Van Nguyen
Committee Member: Dr. Zhu Han
Committee Member: Dr. Saurabh Prasad

University of Houston
May 2020

Dedication

I dedicate my work to my family and friends. A special feeling of gratitude to my loving parents whose encouragement and support instilled courage in me. I also dedicate this thesis to my many friends who have supported me throughout the process. I will always appreciate all they have done.

Acknowledgements

I wish to sincerely thank my advisor Dr. Hien Van Nguyen for his continued support, advice, encouragement and most of all patience throughout the entire process. His expertise and generosity with feedback made the completion of this research an enjoyable experience. Thank you Dr. Zhu Han and Dr. Saurabh Prasad for agreeing to serve on my committee. Special thanks goes to the staff of the Department of Electrical and Computer Engineering for their continued support.

Abstract

Artificial Intelligence (AI) and Machine Learning (ML) today has infiltrated almost all fields, helping catch patterns and make interesting conclusions from data. The surge in AI over the years can be attributed to two main facts: Increase in computation power of newer systems and the availability of data, both of which serve as seeds for building good prediction models. Medicine has slowly but steadily adopted AI over the years. Yet, traditional heuristic approaches and experience of physicians and doctors is heavily relied upon to this date. This thesis proposes two machine learning tools that can help doctors, physicians and medical researchers with their diagnosis and treatment procedures.

Proposal 1 discusses Automatic Machine Learning (AutoML), which is a tool that helps automate the process of ML model building and fine-tuning, taking away the onus of fine-tuning model parameters from the programmer. The resistance towards adoption of ML in the medical community stems from the idea that the tools and knowledge are only accessible to highly trained ML experts. This proposal is an attempt at breaking this age-old perception by proposing Auto-ML as a tool to build good ML models. The experiment done to substantiate this claim is to have a graduate student with sufficient experience in ML, manually build and fine-tune ML models on two publicly available cardiovascular disease prediction data-sets over a month and compare the performance with that of Auto-ML. The results prove that Auto-ML is capable of building models of similar accuracies in a time span of 30 minutes per data-set, with just a few lines of code. This should provide enough empirical evidence and encourage doctors to adopt ML as part of their research.

Proposal 2 discusses the power of visualization of Convolutional Neural Networks (CNN) in performing classification tasks and how they help develop trust in doctors and

medical researchers about model predictions. Gradient-weighted Class Activation Mapping (Grad-CAM) is used as a tool to generate localization maps indicating regions in the image that contributed to a certain prediction from the CNN, thereby instilling trust in medical professionals.

Table of Contents

Dedication ii

Acknowledgements iii

List of Tables ix

List of Figures x

Abbreviations xii

1 PROPOSAL 1 - AUTOMATIC MACHINE LEARNING 1

1.1 Introduction 1

1.2 Taxonomy 3

1.3 Datasets 3

1.3.1 Heart UCI dataset 3

1.3.2 Cardiovascular disease dataset 4

1.4 Auto-Sklearn 4

1.5 Literature survey 7

1.6 Human strategy 10

1.6.1 Human strategy outline 10

1.6.2 Human attempt procedure 12

1.6.3 Model descriptions 15

1.6.3.1 Logistic Regression classifier 15

1.6.3.2 Support Vector Machines 18

1.6.3.3 Decision Trees . . . 20
1.6.3.4 Random Forests . . . 22
1.6.3.5 K Nearest Neighbors . . . 23
1.6.3.6 Bagging Classifiers . . . 25
1.6.3.7 Adaboost classifier . . . 25
1.6.3.8 Multi-layer perceptron (MLP) classifier . . . 26
1.6.3.9 Gradient boosted trees . . . 28
1.6.3.10 Voting classifiers . . . 30
1.6.4 Results and discussion . . . 30
1.6.4.1 AutoML Benefits Complex Datasets More . . . 30
1.6.4.2 Comparison of AutoML's and Graduate Student's Test-Set Performances . . . 31
1.7 Conclusion . . . 33

2 PROPOSAL 2 - NEURAL NETWORK VISUALIZATION 35

2.1 Introduction . . . 35
2.2 Visualization tools for CNNs . . . 37
2.3 Grad-CAM visualization . . . 40
2.3.1 Guided Grad-CAM . . . 43
2.4 ABMR . . . 44
2.4.1 ABMR classification by pathologists . . . 44
2.5 Machine classification . . . 45
2.6 Results and discussion . . . 47
2.6.1 Dataset . . . 47
2.6.2 Training . . . 48
2.6.2.1 Training curves . . . 48
2.6.2.2 Visualizations . . . 48
2.7 Conclusion and future work . . . 51

REFERENCES 54

APPENDICES **65**

A Daywise performance on Heart UCI and Cardiovascular Disease Dataset . . 65

B Descriptions and parameter settings of algorithms used by the student . . . 71

List of Tables

1.1 13 attributes of UCI Heart dataset. 4

1.2 12 attributes of Cardiovascular Diseases dataset. 5

1.3 Accuracies reported by previous studies on Heart UCI Dataset compared to accuracies of the graduate student and AutoML 33

1.4 Comparison of AutoML and graduate student's classification performances and total time on UCI test set 33

1.5 Comparison of AutoML and graduate student's classification performances and total time on Cardiovascular test set 33

2.1 Performance (Accuracy) of ResNet models on test set 48

List of Figures

1.1 The Auto-Sklearn pipeline [25] contains three main building blocks: 1) Data preprocessor, 2) Feature preprocessor, and 3) Estimator or machine learning algorithms. 5
1.2 Python code for using Auto-Sklearn to train a classifier for any dataset. . . . 7
1.3 Process adopted by human to build good models 11
1.4 Sample distributions of the two datasets 12
1.5 Sample cross validation curve [71] 15
1.6 Optimal hyperplane for SVM [19] 18
1.7 Sample Decision tree [66] 21
1.8 Sample K nearest neighbor classification [71] 23
1.9 MLP with one hidden layer [71] 27
1.10 Validation accuracy over 18 days by the graduate student on the Heart UCI dataset 31
1.11 Validation accuracy over 15 days by the graduate student on the Cardiovascular Disease dataset 32

2.1 Sample visualization map using Activation maximization [88] 38
2.2 Original image and image with occlusion map [88] 38
2.3 Original image and image with saliancy map [88] 39
2.4 Original image and image with gradient map [88] 40
2.5 Layerwise visualization of classification task from VGG16 [88] 40
2.6 Glomerulus images with varying degrees of ABMR [82] 45

2.7 Building bock of residual networks [36] 46
2.8 Building block in ResNet34 and "bottleneck" building block in ResNet-50/101/152. [36] 47
2.9 ResNet-34 [36] 47
2.10 Preliminary CNN solution for a classification task in transplant nephropathology [7] 48
2.11 Validation accuracy curves for the three ResNet networks 49
2.12 Training and validation accuracy curves ResNet50 49
2.13 True positive predictions with Grad-CAM visualization 50
2.14 True negative predictions with Grad-CAM visualization 50
2.15 False predictions with Grad-CAM visualization 51
2.16 Heatmaps from network and regions nephropathologists see [7] 51
2.17 Wrong predictions made from cropped images show that the whole image (several features from Glomerulus) is required to make correct prediction . 52

Abbreviations

AI	Artificial Intelligence
AutoML	Automatic Machine Learning
ML	Machine Learning
RFE	Recurrent Feature Elimination
SVM	Support Vector Machine
RBF	Radial Basis Function
ANN	Artificial Neural Networks
MLP	Multi-layer Perceptron
ReLU	Rectified Linear Unit
DL	Deep Learning
CNN	Convolutional Neural Network
CAM	Class Activation Mapping
Grad-CAM	Gradient-weighted Class Activation Mapping
ABMR	Antibody mediated rejection
VQA	Visual Question Answering
LSTM	Long Short Term Memory
WSI	Whole Slice Image
LM	Light Microscope
EM	Electron Microscope

Chapter 1

PROPOSAL 1 - AUTOMATIC MACHINE LEARNING

1.1 Introduction

The process of building good machine learning (ML) models is usually long and expensive involving several steps like data cleaning, relevant feature selection, appropriate model selection, hyper-parameter tuning and deployment [47]. Not only is the process time-expensive, but also involves a lot of calculated guessing and trial-and-error. As stated by the 'No free lunch' theorem, there is no algorithm that can achieve good performance on all problems [109]. This process thus requires expertise in machine learning as well as computer programming, thereby discouraging others that could benefit from the power of ML from using it. Also, given the amount of skill and competence required, the price to build a good ML model is expensive [111]. Automatic Machine Learning (AutoML) is an attempt at bringing the power of Machine Learning closer to those that do not possess the skills to manually train and fine-tune models. The idea of AutoML is to allow users to simply input data and get predictions on unseen data at the click of a button. AutoML aims to automate as many of the above steps mentioned. AutoML can help medical researchers and doctors use ML more extensively in their work.

AutoML can be explained as the end-to-end process of searching and finding the best

AI model configuration for an arbitrarily given dataset. Each configuration is the result of looking for the best pre-processing step, algorithm, optimization method, and hyper-parameter. Given the enormity of the search space, the processing of looking for the best configurations is computationally expensive. But with the advent of advanced technologies like the Graphics Processing Unit (GPU) and Tensor Processsing Unit (TPU) and better search algorithms, AutoML methods have been able to scale up dramatically.

The are several AutoML libraries commonly in use today. Libraries like AutoSklearn [25], MLBox [68], TPOT [69], H20 AutoML [34] and Auto-Keras [44] are popular in use today. Many major technological companies, aware of the potential of AutoML have their own Auto Machine Learning platforms like Google Cloud AutoML [30] and Microsoft AutoML [64]. This experiment uses AutoSklearn to perform Automatic feature preprocessing, model selection and hyper-parameter tuning on data. AutoSklearn uses the built-in software package Sklearn's functionalities to automate the pipeline by choosing a good algorithm, good hyper-parameters and feature preprocessing steps. It tackles the problem using Bayesian optimization [98]. A choice is made at each stage of the pipeline from 15 Classifiers, 14 pre-processing methods and 4 data pre-processing algorithms. An ensemble is created from the executed models during optimization to serve as the final model. AutoSklearn has been the winner at the ChaLearn AutoML challenge. It has been tested over 100 diverse datasets and substantially outperforms all of its competitors [25].

It is therefore evident that AutoML has great potential across industries. Despite this fact, AutoML has not been well-studied in biomedical applications. This proposal aims to make the following contributions:

- Investigate the performance of AutoML on cardiovascular disease prediction datasets.
- Compare the performance of AutoML with that of manually built solutions by a graduate student with significant experience in ML and computer programming.
- Detailed analysis, discussion and comparison of the performance of AutoML versus the graduate student.

1.2 Taxonomy

- **Feature Engineering:** The process of constructing a good feature set from data so that the learning tool can obtain a good performance.
- **Search Space:** All candidate classifiers and their corresponding hyper-parameters.
- **Optimizer:** Agent that goes through search space trying to find a good model/ algorithm.
- **Feedback:** The metric value that is used to judge a model/ algorithm's performance.
- **Evaluator:** Agent that evaluates a model/algorithm for goodness-of-fit and generates feedback for optimizer to use.

1.3 Datasets

The two datasets used for this set of experiments are the **Heart UCI (University of California, Irvine)** dataset [43] and the **Cardiovascular Disease datasets** [105], both publicly available datasets. The two datasets are described in more detail in the following sections.

1.3.1 Heart UCI dataset

The Heart UCI dataset contains data of patient records with the target field referring to the presence or absence of heart disease. The database has 76 attributes, but only 13 are used for our experiments to make our results comparable to previous machine learning papers. Table 1.1 shows the selected attributes and their properties. This dataset has in total 303 records, which is relatively small given that a typical machine learning dataset contains several thousands to hundreds of thousands of data points. There have been multiple works investigating the performances of different machine learning algorithms on this dataset. The popularity of this dataset makes it easy to know how competitive the results of the graduate student are as well as how the performance of the AutoML method compares to human–experts' systems. The target variable in this dataset is 'Target' in Table

1.1. Of the 303 records, 138 records are that of patients with Target 0 and 165 records with Target 1.

Table 1.1: 13 attributes of UCI Heart dataset.

Attribute	Type	Description
Age	Continuous	Age in years
Sex	Discrete	1 = male, 0 = female
Cp	Discrete	Chest pain type (4 values)
Trestbps	Continuous	Resting blood pressure (in mm Hg on admission to the hospital)
Chol	Continuous	Serum cholestoral in mg/dl
Fbs	Discrete	Fasting blood sugar > 120 mg/dl 1 = true; 0 = false
Restecg	Discrete	Resting electrocardiographic results (values 0,1,2)
Thalach	Continuous	Maximum heart rate achieved
Exang	Discrete	Exercise induced angina (1 = yes; 0 = no)
Oldpeak	Continuous	ST depression induced by exercise relative to rest
Slope	Discrete	The slope of the peak Exercise ST segment (values 0,1,2)
Ca	Discrete	Number of major vessels (0-4) colored by flourosopy
Thal	Discrete	Nature of defect, values (0-3)
Target	Discrete	Presence or absence of heart disease, values (1,0)

1.3.2 Cardiovascular disease dataset

The cardiovascular disease dataset consists of 70,000 records of patients' data with the target (Cardio) describing the presence or absence of heart disease using 11 features as described in Table 1.2 [105]. The input features are of three types: objective (containing factual information), examination (containing the results of a medical examination) and subjective (containing information given by the patient). The target variable in this dataset is 'Cardio' in Table 1.2. Of the 70,000 records, 35,021 records are that of patients with Cardio 0 and 34,979 records are that of patients with Cardio 1.

1.4 Auto-Sklearn

Auto-Sklearn was proposed in [25]. The name was motivated by Scikit-Learn [71], a popular generic machine learning toolbox. Auto-Sklearn automates the process of building AI model by utilizing a large number of machine learning classifiers (14 in total) and pre-processing steps (14 feature processing methods, and 4 data preprocessing methods)

Table 1.2: 12 attributes of Cardiovascular Diseases dataset.

Attribute	Type	Description
Age	Continuous	Age of the patient in days
Gender	Discrete	1 - women, 2 - men
Height (cm)	Continuous	Height of the patient in cm
Weight (kg)	Continuous	Weight of the patient in kg
Ap_hi	Continuous	Systolic blood pressure
Ap_lo	Continuous	Diastolic blood pressure
Cholesterol	Discrete	1: normal, 2: above normal, 3: well above normal
Gluc	Discrete	1: normal, 2: above normal, 3: well above normal
Smoke	Discrete	whether patient smokes or not
Alco	Discrete	Alcohol intake-Binary feature
Active	Discrete	Physical activity-Binary feature
Cardio	Discrete	Presence or absence of cardiovascular disease

in the Scikit-Learn toolbox. That includes logistic regressions, support vector machines, random forests, bagging, boosting, and neural networks. Figure 1.1 shows the graphical illustration of the pipeline. Given the training data, Auto-Sklearn first selects an appropriate set of data preprocessing steps such as rescaling or imputation of missing values. It then passes the processed data to the feature processing block, which further normalizes the data or reduces their dimensions using standard techniques such as principal component analysis [46] and independent component analysis [42]. Finally, data are passed to the estimator block, which selects and trains machine learning algorithms to predict desirable outputs from input data samples.

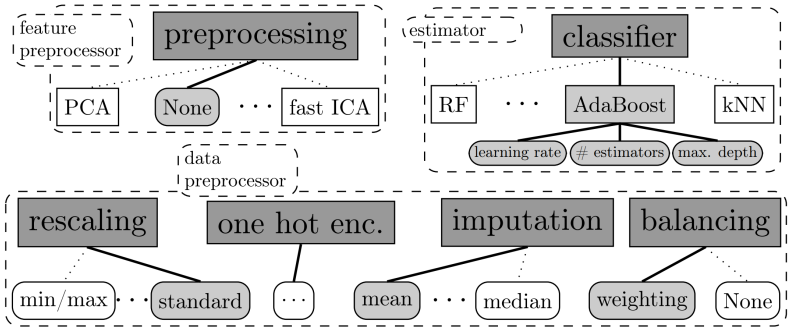

Figure 1.1: The Auto-Sklearn pipeline [25] contains three main building blocks: 1) Data preprocessor, 2) Feature preprocessor, and 3) Estimator or machine learning algorithms.

Auto-Sklearn defines AutoML as the process of automatically producing test-set pre-

dictions (without any human intervention) given a fixed computational budget. Here, computational budget means computer run time or computer memory usage. Auto-Sklearn combines traditional machine learning techniques with a Bayesian optimization framework to search for the best combination of AI models and parameters. It also introduces several notable improvements compared to previous approaches [35]. *First*, it uses prior experience on other datasets to create a good model initialization for a new dataset. The central intuition is that domain experts derive knowledge from previous tasks. Motivated by this observation, Auto-Sklearn employs a similar strategy. It collects a set of 38 *meta-features*, or vector descriptions of dataset properties that would help to determine appropriate algorithms that would likely perform well on a particular dataset. Examples of meta-features include statistics about number of data samples, data dimensions, classes, and skewness. Based on these features, Auto-Sklearn make a rough suggestion for what algorithms, pre-processing, and other hyper-parameters will work well on a particular dataset. Bayesian optimization further refines and improves the model. *Second*, instead of outputting one model, it use a weighted combination of multiple best-performing models. This is similar to the ensemble method in random forests [11] that combines multiple random trees to reduce the prediction variance. Empirical studies found that this modification significantly improve the robustness of the final model [25].

A non-technical person will find Auto-Sklearn intuitive and easy to learn. Figure 1.2 shows the code for training a classifier for an arbitrary dataset. It essentially contains only four lines of code. The first line loads the Auto-Sklearn library, assuming that this library is already pre-installed in the computer. The second line of code creates an instance of the classifier. One can think of this as a placeholder for the final classifier. The third line of code calls the function **.fit** to train (aka fitting) the final classifier given the training data **X_train** and the corresponding labels **y_train**. The last line calls the function **.predict** to make the predictions on the test data **X_test**.

```
import autosklearn.classification as automl
classifier = automl.AutoSklearnClassifier()
classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
```

Figure 1.2: Python code for using Auto-Sklearn to train a classifier for any dataset.

1.5 Literature survey

Review of the AutoML problem and techniques to solve it: Q. Yao et al [111] provide an in-depth discussion of the AutoML problem setup and the techniques employed to solve it, along with providing a detailed analysis of these techniques and the reasons for their successful operations. The authors describe the AutoML problem as a combination of automation and ML. The steps involved in building a good ML model include problem definition, feature engineering, model selection, optimization, evaluation and deployment. AutoML can help automate feature engineering, model selection, optimization (algorithm selection) and evaluation, thereby allowing experts to focus on problems with more applications and business value. The AutoML problem is defined as a tool that maximizes performance limited by minimum (or no) human assistance and computational budget. The 3 main goals of AutoML include providing a good generalization of performance across different input data, minimal human assistance and good computational efficiency (good performance under limited budget). The idea is to allow experts to focus on defining the problem, collecting the data and deployment and allowing ML to become more accessible to everyone, not just to human experts. The authors summarize all AutoML steps into two broad components, the *Evaluator*, which measures the performance/ algorithm and sends feedback to the "optimizer" and the *optimizer*, that goes through the *search space* and updates the configurations of the learning tool with the help of the feedback provided by the evaluator. An optimizer for the AutoML problem is based on the nature of search space, feedback and how much of the search space needs to be covered before a good, suitable model can be found. Several search approaches are discussed for traversing the search space; Simple approaches and Derivate-free approaches. *Simple approaches* include search techniques like grid search (which involves enumeration of every possible config-

uration in search space (requires discretization in continuous search space)) and random search, where the search space is randomly sampled. *Derivative free optimizers* select new configurations based on evaluation results from previous samples. Common derivative free optimizers include *Heuristic Search*, an algorithm that works by reaching an optimal configuration by selecting a suitable population. Heuristic search approaches differ in the way the population is selected. Some popular heuristic search optimizers include Particle Swarm Optimizers and Evolutionary algorithms. *Model-Based Derivative-Free Optimizers* build models on new samples and utilizes the model performance generated by the evaluator to generate new models. Some common examples include Bayesian optimization, Classification based optimization and Simultaneous Optimistic Optimization.

AutoSklearn: AutoSklearn [25] is an award winning AutoML tool [32] based on the Scikit-learn library that automates the process of feature preprocessing (using 14 feature preprocessing methods), data preprocessing (using 4 data preprocessing methods), classifier selection (using 15 classifiers) and hyper-parameter optimization. AutoSklearn won the first phase of the ChaLearn challenge [48]. The AutoML problem is formulated as a CASH (Combined algorithm selection and optimization) problem, where the problems of model selection and hyper-parameter optimization are tackled jointly. The system is inspired by AutoWEKA [103], which combines the WEKA learning framework with Bayesian optimization. Additional strength of AutoSklearn comes from Meta-Learning, a technique used to initialize the Bayesian optimizer using results from previously used similar datasets and ensamble construction from configurations previously used by the optimizer. AutoSklearn uses Meta-Learning in the following way. It uses 140 datasets from the OpenML repository [106], computes meta features for all these datasets (examples of meta features include number of data points, features, classes etc.) and stores ML framework instantiations that performs well on these datasets. Now, given a new dataset, its meta-features are computed and the L1 distance of these meta-features to meta-features from the offline datasets are computed and the ML framework instantiations of the closest 25 datasets from the meta-feature space are used to warm start the Bayesian optimizer. Automated ensamble construction is used as follows. The ensamble construction process is a greedy selection process that iteratively adds models that the evaluator has travelled

through in the search space and weighs these models based on the performance on a hold-out cross validation dataset. These ensambles are shown to outperform the individual models [33][51] and prevents wasteful loss of models that are evaluated in the search space.

Performance evaluation of Auto-Sklearn: Vanilla Auto-Sklearn (without meta-learning and ensamble construction) performed better than AutoWEKA on 85% of the presented datasets and statistically tied with hyperopt-sklearn on 56% of the datasets. Performance of AutoSklearn (with meta-learning and ensamble construction) is evaluated on a broad range of binary and multiclass classification datasets from the OpenML repository, using Balanced Classification Error Rate (BER) (average of the proportion of wrong classifications) as a metric to circumvent the problem of imbalanced classes. The performance is studied under rigid time constraints, limiting the total CPU time to 1 hour and the run time for a single model to 6 minutes. The results indicate that meta-learning substantially improves performance right from the first model to the final model. Use of ensamble construction improved the performance significantly early-on when meta-learning is used, which can be attributed to the fact that meta-learning generates good models right from the start. Nonetheless, ensamble construction is shown to improve the performance of Vanilla AutoSklearn when run longer.

Cardiovascular disease risk prediction models: Given that cardiovascular diseases are the leading cause of death today [63] and the availability of tremendous amount of cardiovascular data, there have been numerous studies in the past to get machine learning models to deduce patterns in the data to allow for early detection of heart diseases. Multiple standalone machine learning models and hybrid models have been proposed [15]. Vembandasamy et al. [107] propose the use of Naive Bayes classifier for prediction of heart disease on a dataset from a leading diabetic research institute in Chennai, India containing 500 records and 10 attributes. The Naive Bayes classifier attained accuracy of 86.4%. Shouman et al. [94] propose the Decision Tree classifier on the benchmark Heart UCI (University of California, Irvine, CA, USA) dataset by applying several tuning techniques to Decision Trees like different combinations of discretization, tree types, voting, etc. to identify a reliable, robust and accurate method of classification. The final reported accuracy is 84.1%. Srinivas et al. [100] propose more complicated data mining algorithms.

The technique involves the extraction of significant patterns from the dataset, choosing patterns with values greater than a prescribed threshold and using five different mining goals. The reported accuracy is 83.7%. Tomar et al. [104] use Least Squares Twin Support Vector Machines [28] for diagnosis of heart diseases using the grid-search approach for hyper-parameter selection and F-scores as the evaluation metric on the heart UCI dataset. Reported accuracy is 85.59%. Several ensemble classifiers, which are a weighted combination of simple classifiers have also been seen to work well with heart disease prediction. Pouriyeh et al. [75] use the Decision Tree classifier, Naïve Bayes classifier, Multilayer Perceptron, K-Nearest Neighbor classifier, Single Conjunctive Rule Learner and Radial Basis Function with Support Vector Machines both individually and in combination on the Heart UCI dataset. In addition, bagging, boosting and stacking techniques have been applied on each of the above-mentioned classifiers. The best performing classifier was reported to be a combination of the Support Vector Machine and the Multilayer Perceptron and the reported accuracy is 84.81%. Bashir et al. [5] propose the use of an ensemble classifier that uses an enhanced bagging approach with the multi-objective weighted voting scheme. Five different base classifiers including Naïve Bayes, linear regression, quadratic discriminant analysis, instance-based learner and support vector machines are used. Five different heart disease datasets are used. The experimental evaluation shows that the proposed framework achieves diagnosis accuracy of 84.16%.

1.6 Human strategy

1.6.1 Human strategy outline

This section describes an outline of the strategy adopted by the human to come up with a model that performs well on both cardiovascular disease risk prediction problems.

1. **Define the problem**: In this case the problem is defined as finding the best ML model that can classify a human as having or not having cardiovascular disease. The datasets including the independent variables and the target variable are described in tables 1.1 and 1.2.

2. **Preprocess the data**: This includes data preprocessing and feature preprocessing. The dataset is first checked for missing values (since both datasets do not contain missing values, this step is ignored). As part of feature preprocessing, the nature of features in the datasets are identified. This include identifying continuous and categorical features. If the feature is identified as categorical, a decision is made on whether it is cardinal (feature that describes quantity), ordinal (feature that describes position) or nominal (feature that is used as a name, label or identifier) in nature by studying more about the feature. Nominal features are label-encoded [101]. Continuous features on the other hand are scaled so that the values are normalized into a particular range allowing for faster convergence with certain algorithms [92]. The dataset is then split into train and test sets [4].

3. **Obtain good learning performance:** This is a two step process. First is to use intuition/personal experience given the data to *select the model and tweak model parameters*. Second is to *obtain feedback* about the performance of the model and adjust the configurations based on performance. These two steps are repeated until we run out of computational time (15 days per dataset in our case) [111].

The procedure is described in figure 1.3

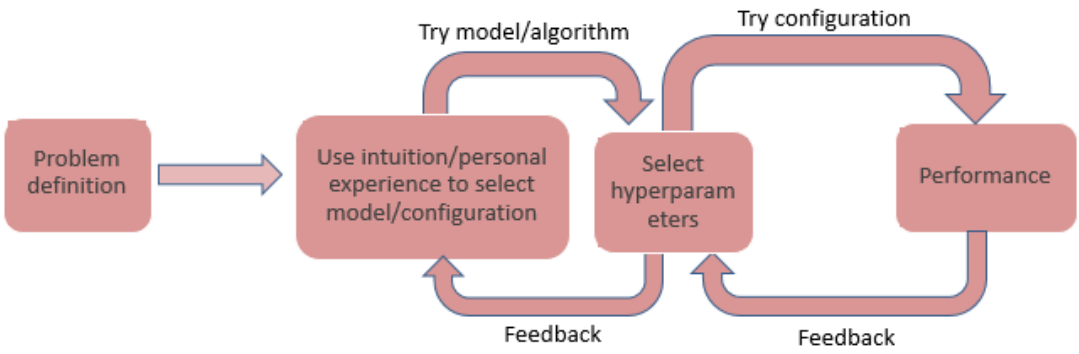

Figure 1.3: Process adopted by human to build good models

1.6.2 Human attempt procedure

1. **Understanding the datasets:** The total number of data samples is in the Cardiovascular disease dataset is 70000 and in the Heart UCI dataset in 303. Figure 1.4 denotes the distributions or positive and negative samples in the datasets.

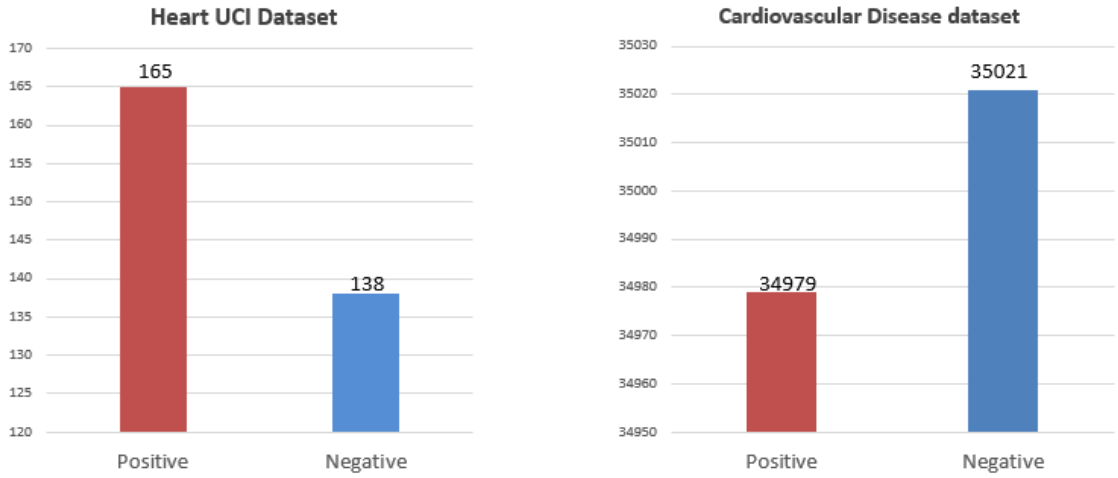

Figure 1.4: Sample distributions of the two datasets

2. **Dataset split:** The dataset is split into a train dataset, with 80% of the samples and test dataset, with 20% of the samples. Data split is performed in a stratified fashion [3]. Stratified sampling allows to select a sub-population from the entire dataset that best represents all the data points. k-fold cross validation [29] is performed with models fitted on the train dataset. Training dataset is divided into k subsamples, training is performed on exactly k-1 subsamples and the model is validated on 1 subsample. This process is repeated k times so that all subsamples serve as the validation subsample once. It is extremely useful with smaller datasets like the Heart UCI dataset where there aren't enough samples to use for hold-out cross validation. With some models, k-means cross validation is shown to be resource/time draining especially when used with the Cardiovascular disease dataset. In such cases, k-fold cross validation is replaced with hold-out cross validation.

3. **Model and hyper-parameter fitting:** The general strategy followed is as follows.

   (a) Classifiers are used with their default hyper-parameters to obtain the baseline

performance of the model on the data. The following models are fitted on the data.

- Logistic Regression [39]
- Support Vector Machines (with linear and Radial Basis Function (RBF) kernels) [90]
- k-nearest neighbors [73]
- Decision trees [2]
- Random Forests [2]
- Gradient boosted trees [27]
- Adaboost classifiers [61]
- Bagging previously used classifiers [102]
- Voting classifiers [85]

(b) For Linear models, including Logistic regression and Support Vector Machines (SVM) with linear kernels (non tree based models), feature selection techniques including F-test, Mutual Information test and Recursive feature elimination (RFE) are performed prior to model fitting. F-test is a statistical test and a popularly used feature selection technique that captures linear dependencies between a feature and the target variable. In simple terms, the F-statistic measures the ratio of the variances between each feature and the target to provide an indication of the correlation between them. Mutual Information [72] at a high level can be thought of as reduction in the uncertainty of the target variable given the information about any variable. A high value of mutual information is indicative of high reduction in uncertainty, a low value indicates a lower reduction in uncertainty and zero mutual information means that the two variables are independent of each other. Recursive feature elimination [110] is a greedy approach to feature selection that recursively selects a smaller and smaller subset of features from the whole dataset by assigning importance scores to features by fitting the said estimators on them and pruning the less important features from the dataset. Tree based models do not require explicit use of feature selection

prior to training because they implicitly perform feature selection using Gini, entropy or Chi-Square methods to ensure that the most important features lie closer to the tree root [93]. These algorithms will be discussed in depth in the following section.

4. **Hyper-parameter selection:** A model hyper-parameter is a value that is set before training and whose value doesn't change during training. Hyper-parameters selection can be a tedious task and the choice of the hyper-parameter can affect the model performance greatly. The challenges to hyper-parameter search include complex high-dimensional search spaces, randomness in the search space and highly complex objective functions [17]. Hands-off optimization algorithms like grid-search and random search (used for manual tuning as part of this experiment) are resource and time expensive since time is wasted in searching through unpromising areas of the search space. A more informed search strategy like Bayesian optimization, which minimizes a certain objective function is more popularly in use today since it has been shown to outperform the hands-off optimization approaches on the test set in fewer iterations [74]. This approach is used by AutoSklearn.

   In this experiment, once the baseline performance is obtained using models with their default configurations, the cross validation accuracy of the model is obtained over range of values for a given hyper-parameter using Cross validation curves. Cross validation curves are used to obtain optimal values of single hyper-parameters to validate a model using a scoring metric like the accuracy. The generalization error of a model is composed of Bias, Variance and noise. Bias in an indication of the performance of the estimator on various datasets, variance is the sensitivity of the estimator on the training dataset and noise is the property of the data [71]. High bias implies the estimator has underfit the training data and high variance implies the model has overfit the training data. Cross validation curves (CV curves), like in figure 1.5 help spot that value of the hyper-parameter at which the performance of the evaluator on the validation set begins to drop and that of the training set begins to rise, and at this point the estimator is devoid of bias and variance. From the CV

curve in fig 1.5, it can be seen that the correct choice of the hyper-parameter gamma for SVM would be around 10^{-3}. At this point the training score spikes while the cross validation score begins to fall [71].

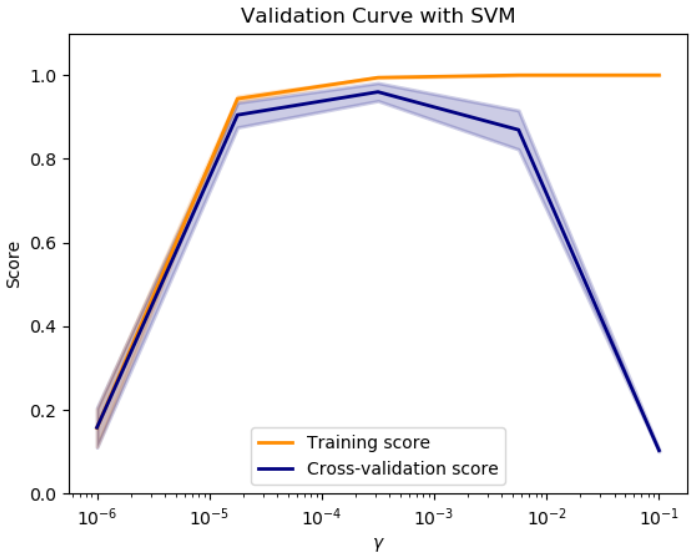

Figure 1.5: Sample cross validation curve [71]

1.6.3 Model descriptions

A description of all the classifiers fitted to the data are presented in this section.

1.6.3.1 Logistic Regression classifier

Logistic Regression is a basic statistical model that is used to classify categorical targets. Most popular applications are the categorization of tumors as malignant or not and categorization of e-mail as spam or not. Logistic regression is of three types; Binary logistic regression (used to predict between two target classes), Multinomial logistic regression (used to predict between multiple classes) and Ordinal Logistic regression (used to predict between three or more categories, example movie ratings) [86]. Since both the datasets in this experiment are to predict the presence/absence of cardiovascular disease in patients, we use binary logistic regression. The equations associated with Binary logistic regression

are

$$
\begin{aligned}
&\textit{Output = 0 or 1} \\
&\textit{Hypothesis } Z = WX + b, \\
&\textit{where X is the input, W is the learnable feature weights and b is the bias} \\
&h_\theta(x) = sigmoid(Z), \\
&\textit{where sigmoid(t)} = \frac{1}{1+e^{-t}}.
\end{aligned} \tag{1.1}
$$

The Binary cross entropy loss function used with logistic regression is presented in equation

$$Cost\ (h_\theta(x)\ ,\ y)\ =\ -y\ log(h_\theta(x))\ -(1-y)\ log(h_\theta(x)). \tag{1.2}$$

Cost function J(θ) is minimized (using Gradient Descent [83]) until the optimal set of parameters θ is obtained.

The accuracies computed using default scikit-learn hyper-parameter values serve as a baseline performance measure.

Feature Selection: Feature selection (selecting the right set of features) can help with improving the efficiency and accuracy of the model. Three feature selection techniques are used with Logistic Regression; F-test, Mutual Information test and Recursive feature elimination technique. These feature selection techniques serve as a means to the reduce the dimensionality of the data to boost model performance on high-dimensional datasets.

F-test: The F-test provides the ANOVA F-value between the the individual features and the target and measures the variance between the target and feature population. The following equation

$$F\ =\ \frac{Between\ group\ variance}{Within\ Group\ Variance} \tag{1.3}$$

describes the computation strategy for ANOVA F-test. F-test captures the linear relationships between the features and the target. The k-best features selected based on the F-test is selected using scikit-learn's k best features module and the choice of k is made by testing

the model performance against these k features.

Mutual information test: Mutual information between two random variables is a measure of the correlation between two random variables. The formal definition of the mutual information between two random processes X and Y is described in equation

$$I(X,Y) \;=\; \sum_{x\in X}\sum_{y\in Y} P(x,y)\; log\; \frac{P(x,y)}{P(x)\;P(y)}. \tag{1.4}$$

In this equation [52], P(x,y) is the joint probability distribution of X and Y and P(x) and P(y) are the marginal distributions of X and Y. Mutual information test captures the non-linear relationships between the features and the target. The k-best features based on the f-test and mutual information test using scikit-learn and the choice of k is made by testing the model performance against these k features.

Recurrent Feature Elimination: As the name suggests, recursive feature elimination recursively removes features until the best performing model is achieved. Scikit-learn's 'Selectbestk' is used to select the k best features in the dataset.

Hyper-parameter selection: Two important hyper-parameters are tuned using cross validation curves; *maximum iterations* (maximum number of iterations for the solvers to converge) and *C (inverse of the regularization term)*. The maximum number of iterations dictates the number of steps the solver takes until convergence. Regularization is used to improve the generalization performance of the model and reduce variance (overfitting) so that the model does not memorize the patterns in the training data and is capable of generalizing well to an unseen dataset. The regularization term can be thought of as the penalty term added to the cost function to keep the model from learning too much from the training data. Scikit-learn supports three regularization types; L1 regularization, L2 regularization and Elastic net, which is a linear combination of the L1 and L2 regularization. L1 and L2 regularizations can be described as

$$L1\; reg \;=\; \frac{\lambda}{2m} \sum_{j=1}^{num\_features} |\theta_j|. \tag{1.5}$$

$$L2\ reg\ = \frac{\lambda}{2m} \sum_{j=1}^{num\_features} \theta_j^2, \tag{1.6}$$

where θ is the set of parameters and m is the number of data samples. The hyper-parameter C is the inverse of the regularization term λ. A larger value of C would therefore indicate a smaller regularization. The optimal value for the two hyper-parameters is obtained using cross validation curves.

1.6.3.2 Support Vector Machines

A support vector machine is a non-probabilistic binary classifier that can effectively map the data points in space so that separate categories are divided by the widest gap. Figure 1.6 depicts the SVM hydroplane with the largest margin. Points that are closest to the hyperplane are selected from both classes and are called Support Vectors. The distance between the lines and the support vector is called margin. The goal of the lagorithm is to maximise this margin. For this reason, the SVM is also called the large margin classifier.

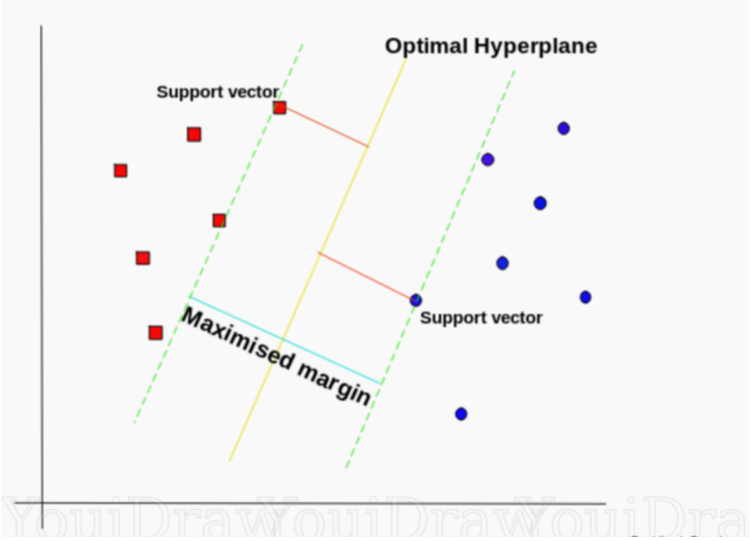

Figure 1.6: Optimal hyperplane for SVM [19]

SVMs can perform non-linear classifications using the kernel trick, by mapping the data points into high dimensional feature space. There are multiple Support Vector Machine Kernels including Linear kernel, Gaussian Kernel, Gaussian Radial Basis function (RBF), Laplace RBF kernel, Hyperbolic tangent kernel, Sigmoid kernel, Bessel function

of the first kind Kernel, ANOVA radial basis kernel and Linear splines kernel in one-dimension [18]. Scikit-learn supports SVM with Linear kernel, Radial basis function kernel and Polynomial kernel. The cost function for Linear SVM is

$$Hypothesis\ h_\theta(x) = \begin{cases} 1\ if\ \theta^T x\ >=\ 0 \\ 0\ otherwise \end{cases}$$

SVM Cost function

$$Cost(h_\theta(x,y) = \begin{cases} max(0, 1-\theta^T x)\ if\ y = 1 \\ max(0, 1+\theta^T x)\ if\ y = 0 \end{cases} . \tag{1.7}$$

Regularized SVM Cost function

$$J(\theta)\ =\ C\ [\sum_{i=1}^{m}\ y^{(i)}\ Cost_1(\theta^T(x^{(i)}))\ +(1-y^{(i)})\ Cost_0\ (\theta^T(x^{(i)}))]\ +\frac{1}{2}\ \sum_{j=1}^{n}\ \theta_j^2,$$

where θ is the learned parameters, x is the set of input features, y is the ground truth labels, C is the regularization parameter, n is the number of input features and m is the number of data points.

As for non-linear kernels, the cost function simply becomes equation

$$J(\theta)\ =\ C\ [\sum_{i=1}^{m}\ y^{(i)}\ Cost_1(\theta^T(f^{(i)}))\ +(1-y^{(i)})\ Cost_0\ (\theta^T(f^{(i)}))]\ +\frac{1}{2}\ \sum_{j=1}^{n}\ \theta_j^2, \tag{1.8}$$

where x is replaced by f, a function of x.

This proposal makes use of the RBF kernel in addition to the linear kernel. The function f in this kernel is as described in the following equation

$$f\ =\ Similarity(x,l)\ =\ exp\frac{||x-l||^2}{2\sigma^2}, \tag{1.9}$$

where l is the landmark data points (which are manually picked randomly from the data points), f is the set of new features and σ^2 is the variance of the Gaussian function. This goes to say that the new features f are decided based on the proximity to the landmarks l.

Feature Selection Feature selection techniques used with Linear SVM include the *F-*

test, *Mutual information test and RFE*. Since, scikit-learn's RFE module does not support SVM with RBF kernel as the base estimator, only F-test and mutual information test are performed. With each test, the optimal number of features that maximises the model accuracy is selected and SVM is fitted to the data.

Hyper-parameter selection: Validation curves are plotted to select the three important hyper-parameters, gamma, C and maximum iterations. *C* serves as a regularization parameter. A large C implies low bias and high variance. The penalty used is L2 regularization. *Gamma* is a Gaussian kernel parameter and is used with SVM with RBF kernel. gamma can be defined as $\frac{1}{2\sigma^2}$. Intuitively, gamma can be thought of a factor that decides the reach of influence for a single training example. A small value of gamma means a large variance, meaning two points that are far away from each other obtain a large similarity score. On the other hand, a large value of gamma would mean the radius of influence would only include the data point causing the model to overfit, irrespective of the value of C. Therefore, selection of optimal value of gamma is essential to strike a balance between bias and variance. *Maximium iterations* decides the number of iterations that the algorithm takes towards convergence.

1.6.3.3 Decision Trees

A decision tree is a simple yet powerful classification tool that organizes the data points in a tree-like structure, where each node denotes a test in an attribute, each branch represents the test result and the leaves hold the class labels. The idea is to predict the target variable by simple rules learned from the data features [71]. There ideas are depicted on a sample decision tree in figure 1.7. The leaves of the tree predict a Diseased and a Healthy case that serve as the target variable. The node at the top of the tree is called the root node and is the best predictor. The best predictors are at the top nodes of the tree. The strongest distinguishing feature in the decision tree represented in figure 1.7 is therefore the root node 'Locus A'. Decision trees can work with numeric and categorical features.

Given that the best predictors serve as the top nodes of the tree, decision trees therefore do not require explicit feature selection prior to model fitting.

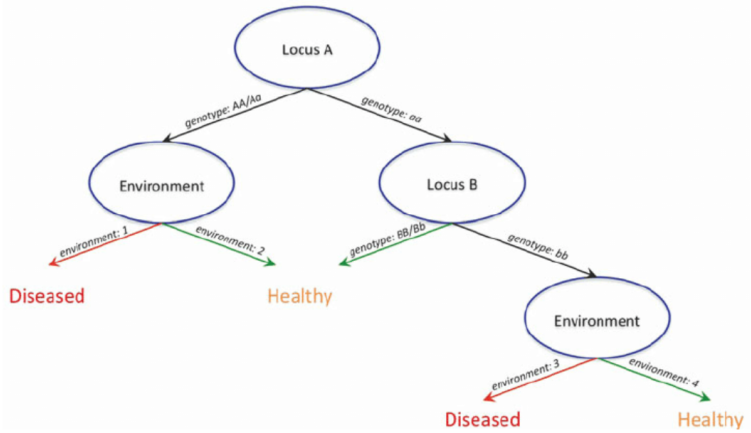

Figure 1.7: Sample Decision tree [66]

Hyper-parameter selection: There are several important hyper-parameters that decide the quality of decision trees. The following hyperpearmeters are carefully selected using cross validation curves.

Maximum depth of the tree is the longest path from the root node to the leaf. The depth of the tree decides the complexity of the tree and hence how well the tree fits the training data. A tree too deep may overfit the data. Decision tree pruning is a technique that reduces the depth/size of the decision tree by removing portions of the tree that have lower predictive power for classification, thereby helping combat overfitting [62].

Split criterion (popularly known as the goodness-of-split criterion) is the function that decides the quality of split at every node. Scikit-learn offers gini and entropy (Information Gain) as split criteria. The Gini impurity function is defined as

$$\phi(p) \;=\; \sum_{j=1}^{C} p_j \,(1 - p_j), \tag{1.10}$$

where $\phi(p)$ is the impurity function and C is the number of classes/partitions and the Entropy impurity function is defined in equation

$$\phi(p) \;=\; -\sum_{j=1}^{C} p_j \, log(p_j), \tag{1.11}$$

where $\phi(p)$ is the impurity function and C is the number of classes/partitions. In other

words, Gini is measuring how often a random chosen data point would be labelled incorrectly, while Entropy (a more computationally intensive technique) measures the disorder in the group by target variable [12].

1.6.3.4 Random Forests

A random forest, as implied by its name, is an ensemble of several decision trees. All decision trees in the forest produce classification results of their own and the class with most votes is the predicted class. The principle is that several uncorrelated decision trees will outperform any of the individual trees. For this idea to work, the individual decision trees must generate diverse results and the generated results must be uncorrelated with each other. It does so using Bagging (or Bootstrap aggregation methods) and Feature randomness. *Bagging* methods take advantage of the fact that decision trees are extremely sensitive to the data they are training on. Different trees are made to work with data points randomly sampled from the data with replacement [54]. *Feature randomness* is a technique in which the individual trees select from a random subset of the features. These two techniques ensure a diverse set of trees that produce uncorrelated results on the samples.

Since random Forests are inherently feature selectors, the application of **feature selection** prior to model fitting is not required.

Hyper-parameter selection: Apart from the individual decision tree hyper-parameter selection from the previous section like the the depth of the tree, an important hyper-parameter is the the number of estimators (or trees in the forest). In general, an increase in the number of trees should improve the performance and generalize better to unseen data, however beyond a certain number of trees, the benefit from having increased number of trees will be lower that the additional computation cost. The rate of increase in performance will begin to drop beyond a certain number of trees. Usually once the number of trees reach about 100, the desired accuracy is met [57]. The best number of trees is determined using cross validation curves.

1.6.3.5 K Nearest Neighbors

K nearest neighbor is an instance based learning algorithm that works on the principle that a query point most likely represents the class of it's nearest neighbors. As part of model training, the data points are simply represented as vectors in a multidimensional feature space and their labels are stored. The output of a data point during test phase is the class membership decided by majority vote of it's K nearest neighbors. The algorithm is therefore sensitive to the choice of K. It is also possible to decide on how much weight is given to the vote of the neighbors. All K neighbors could be given equal weights or data points can be weighted based on their proximity to the data point (known as 'uniform' weights and 'distance' weights in Scikit-learn). Figure 1.8 represents a 3 class (Orange, teal and blue) classification of data points using the K nearest neighbor algorithm leaving all data points within a region or nearest neighbors (represented by their corresponding color) to belong to the same class.

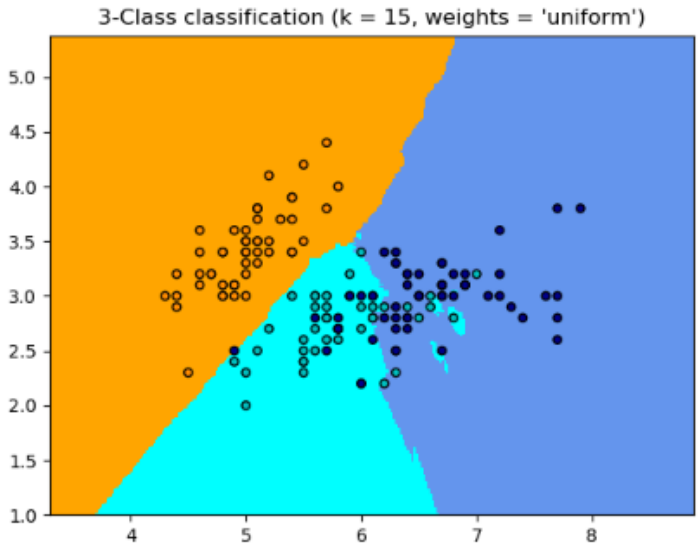

Figure 1.8: Sample K nearest neighbor classification [71]

Hyper-parameter selection: As discussed above, this algorithm is highly sensitive to the choice of several hyper-parameters. *Number of neighbors, weights assigned to the neighbors' vote and distance metric (denoted by p in Scikit-learn)* are all tuned using Cross validation curves.

Number of neighbors (K) is a difficult hyper-parameter to tune because of the vastness of the search space. A small value of K will cause the neighboring points to have a very high influence on the class of a data point and can cause noisy results. Although a high value of K will result in smoother decision boundaries, this will lead to high bias and low variance and increased computational time [21]. The choice of K becomes harder with larger data sizes. Thus, with larger datasets like the Cardiovascular disease dataset, the rule of thumb is used to select K, which states that the optimal value of K is usually odd and approximately the square root of the number of samples in the dataset [45]. For smaller datasets like the Heart UCI Dataset, cross validation curve is used to tune the value of K.

The choice of *weights* can potentially impact the smoothness of the decision rule. Weighted KNN can potentially help with cases where the distance of a data point from the nearest neighbors is very high (can occur in cases where the size of the dataset is small) and the closest neighbors (not all K neighbors) only indicate the true label in the class. Scikit-learn provides two weight parameters 'uniform' and 'weighted', where 'uniform' means the votes from all K neighbors are given equal weight, while 'weighted' means the the weight given to neighbors' votes vary based on their distance from the data point.

The *Power parameter for the Minkowski metric (p)* is the distance metric that would be used in the computation of distances between the data points. If p=1, the distance metric is Manhattan distance (L1 norm). Manhattan distance between two points is the distance between points measured along the right axes. If two point in N-dimensional space are depicted as P1: (x1,x2,x3,.....,xN) and P2:(y1,y2,y3,.....,yN), then the Manhattan distance between the two points is depicted as

$$Manhattan\ distance(P1, P2) \ = \ \sum_{i=1}^{N} \ |x_i - y_i|. \tag{1.12}$$

If p=2, the distance metric is Euclidian distance (L2 norm). It is the length of the straight line connecting two points. If two point in N-dimensional space are depicted as P1: (x1,x2,x3,.....,xN) and P2:(y1,y2,y3,.....,yN), the Euclidian distance between the two points is depicted as

$$Euclidian\ distance(P1, P2) \ = \ \sum_{i=1}^{N} \ |x_i - y_i|^2. \tag{1.13}$$

For arbitrary p, the distance metric is Minkowski distance . It is a generalization of the Euclidian distance and the Manhattan distance. If two point in N-dimensional space are depicted as P1: (x1,x2,x3,.....,xN) and P2:(y1,y2,y3,.....,yN), the Minkowski distance between the two points is depicted as

$$Minkowski\ distance(P1, P2) \ = \ (\sum_{i=1}^{N} \ |x_i - y_i|^p)^{\frac{1}{p}}. \tag{1.14}$$

1.6.3.6 Bagging Classifiers

Bagging is a ensemble technique aimed at stabilizing and improving the accuracy of machine learning models using model averaging techniques. Bagging classifiers are the generic case of the random forest algorithm discussed previously. While random forests are bagged decision trees, it is possible to bag several other classifiers using Scikit-learn's *Bagging classifier*. Bagging is the process of drawing random samples from a dataset with replacement [9]. The idea is to aggregate the results of different estimators that are trained with random subsets of data and features from the original dataset. The idea is to reduce overfitting or high variance. Bagging is a combination of contributions from Pasting (random subsets drawn from random samples) [10], Bagging (random samples drawn with replacement) [9], Random Subspaces (random subsets of the features) [40], Random Patches, a combination of random samples and features [58]. In this proposal, we use bagging classifier with decision trees and K nearest neighbors as the base estimators.

Hyper-parameter selection: The base estimators, decision trees and K nearest neighbors are hyper-parameter tuned using the steps mentioned previously. The hyper-parameters specific to the bagging classifier that are tuned using cross validation curves include *Maximum features* and *Maximum samples*. *Maximum samples* is the maximum number of samples that is used to train each estimator, and the *maximum number of features* is the maximum number of features used to train each estimator.

1.6.3.7 Adaboost classifier

Adaboost classifier [26] sequentially fits weak learners on versions of data modified based on the performance of previous learners. The predictions from all the weak learners

(classifiers with low variance) are combined using majority vote. A certain weight is applied to each data sample. Initially they are assigned uniform weights equal to $\frac{1}{N}$, where N is the number of data samples. As training proceeds, at each iteration, the weights are modified based on the performance of the learners on the data samples. Incorrectly predicted data samples are assigned higher weights, and those predicted correctly are weighted less. This forces the learners to focus on incorrectly predicted data samples. This technique is used to increase variance. Adaboost classifiers are typically used with weak estimators as base classifiers. In this proposal, Adaboost classifiers are used with Decision tree classifiers and Support vector classifiers as the base estimators.

Hyper-parameter selection: *Learning Rate* is the only hyper-parameter tuned using cross validation curves for two base estimators, Decision tree estimator and SVM with RBF kernel.

Learning rate for an Adaboost algorithm is a shrinkage parameter that is responsible for keeping variance under check by slowing down learning from the individual estimators.

1.6.3.8 Multi-layer perceptron (MLP) classifier

An MLP is a class of Artificial Neural Networks (ANN) that contain multiple layers of perceptrons. A perceptron is a unit that performs a weighted linear combination of several inputs and applies suitable non-linear activation on it. An MLP classifier is trained using backpropagation (using supervised learning), in which parameters/weights are updated based on the partial derivative of the cost function with respect to the parameter. The classifier learns a mapping function $f(.) : R^m -> R^o$ using the training data, where m is the input dimension and o is the output dimension. The typical structure for MLP is described in figure 1.9. As seen in figure 1.9, an MLP has an input layer in which each neuron represents an input feature, one or more hidden layers that transform the input as as

$$a_k = G(\sum_{i=1}^{n} (w_i * x_i)), \tag{1.15}$$

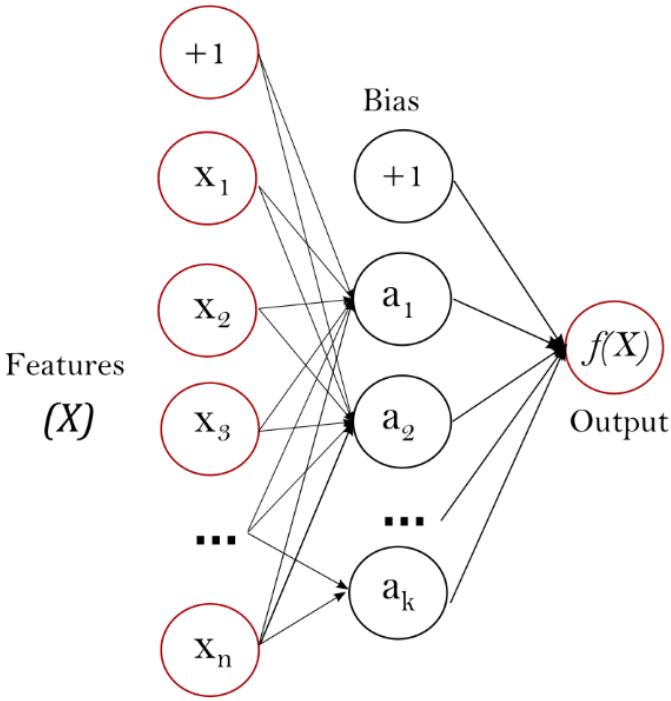

Figure 1.9: MLP with one hidden layer [71]

where G(.) is an activation function. Two most common activation functions are Sigmoid represented as

$$sigmoid(x) = \frac{1}{1+e^{-x}}. \tag{1.16}$$

ReLU (Rectified Linear Unit) represented as

$$relu(x) = max(0, x). \tag{1.17}$$

and tanh (which is tanh(x)). These activation functions allow the network to learn non-linear mappings between the input and output.

Hyper-parameter selection: MLP classifiers have several hyper-parameters that can be optimized. Number of hidden layers, the size of the hidden layers, the size or number of neurons in the hidden layers, the solver used for backpropagation, regularization parameters and learning rate among others. Given that the hyper-parameter space is vast,

rule-of-thumb is resorted to when selecting the number of hidden layers and number of units in the hidden layer. The *number of hidden layers* decides the nature of mapping that the network learns from the input to output. Rule-of-thumb [37] states that with most problems, there is no reason or need for more than one hidden layer. One hidden layer makes the network capable of learning a continuous mapping from the finite input space to the finite output space. So, one hidden layer is used. *Number of neurons in the hidden layer* is another important hyper-parameter. While too few neurons causes underfitting wherein the network doesn't learn much from the data, too many neurons causes overfitting wherein the network learns too much from the training data and doesn't generalize to a new set of data points. Besides, too many neurons can cause increase in training time so much so that the network may not be trained enough in the budgeted time. Thus, the right balance needs to be found and this is a challenging task. Heaton [37] state that the optimal choice is between the input and output layer size, two-thirds the size of the input layer plus output layer and lesser than twice the size of the input layer. *Maximum iterations* decides when the algorithm stops learning. For stochastic solvers like Stochastic Gradient Descent [81] and Adam optimizers [50], maximum iterations is also the number of epochs (number of times each data point goes through the network), while for other optimizers, it is the number of gradient steps that are taken by the optimizer. The algorithm is halted when either the maximum number of iterations is hit or when the tolerance ϵ (when the change in loss between two consecutive iterations is less than a value ϵ) is reached.

1.6.3.9 Gradient boosted trees

Gradient boosting can be seen as the boosting models discussed in section 1.6.3.7 but with generalization that comes from optimizing an arbitrary loss function [8]. Mason et al. [60] introduced boosting algorithms as an iteratively solved functional gradient descent algorithm, which in other words is an algorithm that minimizes a cost function by iteratively choosing weak learners that point in the direction of negative gradient. Gradient boosting is most commonly used with decision trees as the base classifier. The following

equations describe gradient boosted trees.

$$
\begin{aligned}
&\textit{If } h_i(x) \textit{ is the predictions from the } i^{th} \textit{ learner and } F(x) \textit{ is the prediction from all previous learners,}\\
&F(x) \;=\; \sum_{i=1}^{M} \gamma_i h_i(x)\\
&\textit{where } \sum_{i=1}^{M} \gamma_i h_i(x) \textit{ is the weighted sum of all classifiers}\\
&\textit{New classifier } F_m(x) \;=\; F_{m-1}(x) \;+\; \gamma_m h_m(x)\\
&\textit{where } \gamma_m \;=\; \arg\max_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))\\
&\textit{where loss function } L \;=\; yP \;\; log(1 + e^P)\\
&\textit{where P is the model prediction.}
\end{aligned}
\tag{1.18}
$$

The loss function is the deviance loss for binary classification. The idea behind gradient descent is to start with a weak learner h_0 and make a prediction $F_0(x)$ with the weak learner, find the residual or loss $\epsilon_1 = y - F_0(x)$. Then a new learner h_1 fits the residual ϵ_1 such that $F_1(x) = F_0(x) + h_1(\epsilon_1)$. The new residual is now $\epsilon_2 = y - F_1(x)$ and new predictions now become $F_2(x) = F_1(x) + h_2(\epsilon_2)$. So, in essence every new learners learns enough to cover for residuals or what the previous learners have not learnt. And each new learner is weighted by minimizing the loss functions over weight space.

Hyper-parameter selection: Hyper-parameters for gradient boosted trees include the hyper-parameters for the base classifiers (decision trees) like the tree depth, split criterion etc. and the hyper-parameters from gradient descent include the number of estimators and learning rate. Decision trees are used with the best tree parameters selected during hyper-parameter tuning of simple decision tree models. The most important parameters that are tuned using cross validation curves are *learning rate* and *number of estimators*. The learning rate, also called the shrinking parameter shrinks the weight γ as seen in equation 1.18 of each new learner by the learning rate. The number of estimators is the number of trees there is in the model or the number of boosting stages. So there is a subtle trade-off between these two parameters. Both these parameters are fine-tuned using cross validation

curves.

1.6.3.10 Voting classifiers

The voting classifier is intended to serve as a combination of equally well performing classifiers that can balance out their individual weaknesses to produce predictions better than the individual models themselves. The combination can use majority vote of individual classifiers (hard vote) or averaged probabilities of individual predictions (soft vote) to make predictions.

Hyper-parameters optimized: Voting classifiers offers the choice of tuning two important hyper-parameters; the *estimators* or the individual classifiers and the *voting technique*. The individual estimators list is tuned using intuition by combining two well performing classifiers that make relatively uncorrelated errors. The voting mechanism is left at default hard voting, which takes maximum vote to make predictions.

1.6.4 Results and discussion

1.6.4.1 AutoML Benefits Complex Datasets More

In this section, we compare the classification accuracies obtained by the student with that of AutoML. Since the student selects the final model based on the best validation accuracy, this experiment will show how quickly the student can find a good model for a particular dataset. Figures 3 and 4 show the classification over 15–18 day periods for UCI Heart and Cardiovascular Disease datasets, respectively. For both datasets, AutoML achieves competitive validation accuracies compared to that of the student. On the UCI-Heart dataset, the student was able to find a good model from the first day. Since this dataset only has a small number of samples (303 in total), simple classifiers such as linear support vector machine and logistic regression tend to work well. Moreover, the small dataset size makes it faster to run an algorithm and thus reduce the overall development time. On the Cardiovascular Disease dataset, it took the student significantly longer time (seven days) to find a good model. This could be because the second dataset is more complex, demonstrated by the lower validation accuracy of linear classifiers compared

to the previous dataset. Moreover, the number of data points is also significantly larger (70,000 of Cardiovascular Disease vs. 303 of UCI-Heart). Our experiment suggests that the time-saving factor is larger for more complex datasets when using AutoML instead of manual model search.

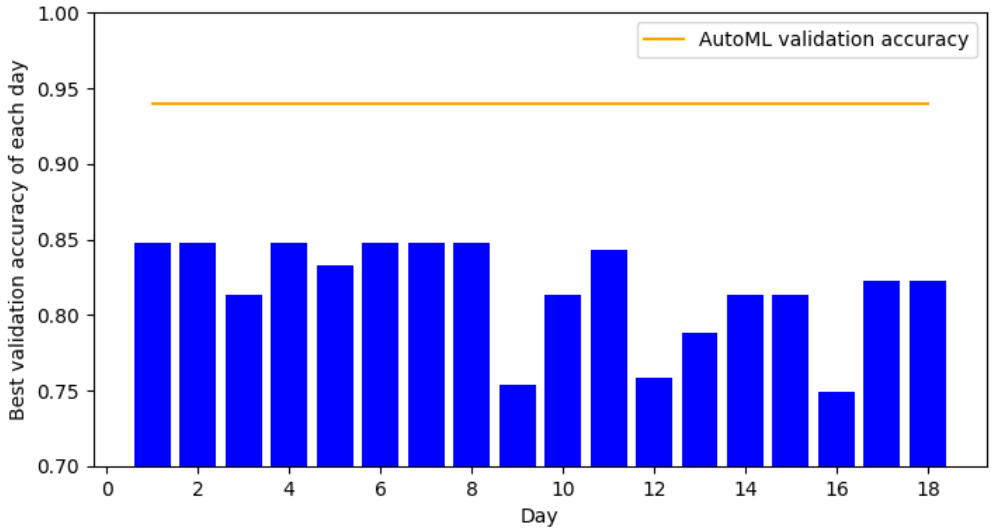

Figure 1.10: Validation accuracy over 18 days by the graduate student on the Heart UCI dataset

1.6.4.2 Comparison of AutoML's and Graduate Student's Test-Set Performances

Once the final models are selected for the two datasets based on the best validation accuracies, the student performed inference on the test sets to obtain the final performance measures. Note that AutoML models were evaluated on exactly the same test sets to make the results comparable. AutoML achieves slightly better mean accuracy for the UCI-Heart dataset, and similar accuracy for the Cardiovascular Disease dataset compared to the student. In addition, AutoML achieves significantly better areas under curves on both datasets. This suggests that AutoML classifiers generalize much better than their manual counterparts. Most importantly, AutoML only takes 30 min to build a competitive classifier for each dataset, compared to long periods of time (432 h for UCI and 360 h for Cardiovascular datasets) taken by the graduate student to develop similar classifiers.

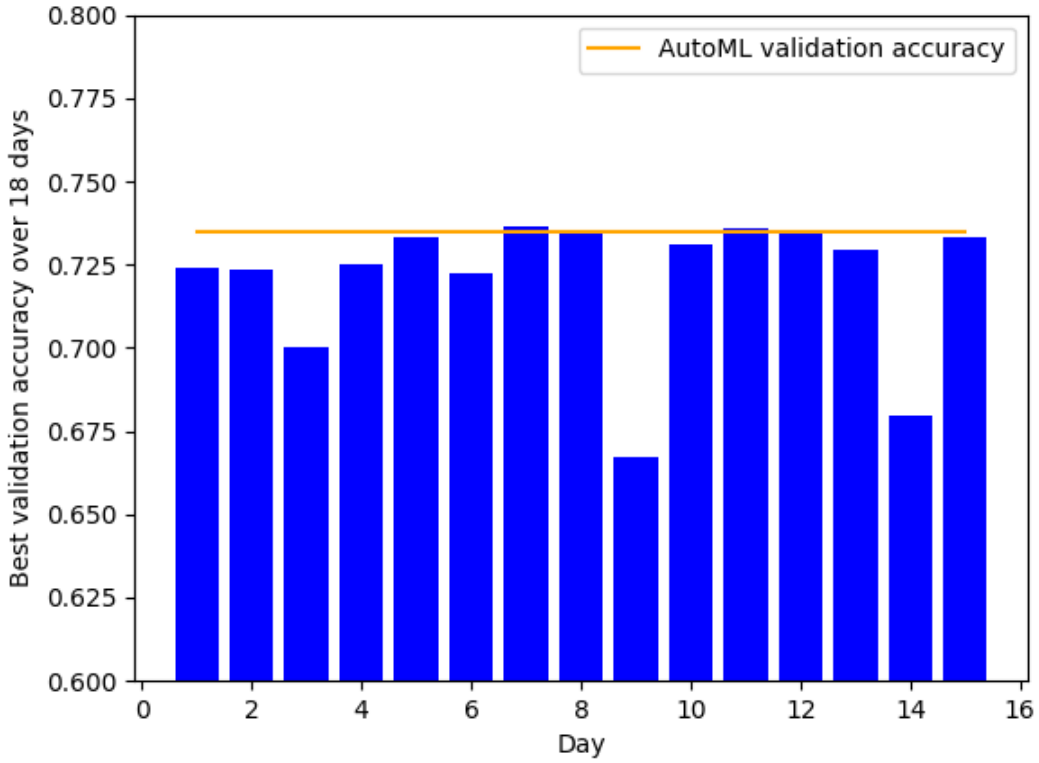

Figure 1.11: Validation accuracy over 15 days by the graduate student on the Cardiovascular Disease dataset

Other state-of-the-art studies 1.4 on the Heart UCI dataset have shown results comparable with those of the graduate student and AutoML. Table **??** presents performance accuracies from training different machine learning models on the Heart UCI dataset and table 1.5 presents performance accuracies from training different machine learning models on the Cardiovascular disease dataset. The accuracies obtained by the student and AutoML are on par with those reported in the recent literature. This further supports the claim that the AutoML method is able to quickly find competitive classifiers with minimal human effort.

The Heart UCI dataset contains 76 features, but only 13 most-important features are included since most studies and published papers utilize them to build machine learning models on. This makes it possible to compare our results to these published papers, in order to serve as the baseline to check if the results obtained by the Graduate student and Auto-sklearn are competent enough. However, the potential downside to reduced feature

Table 1.3: Accuracies reported by previous studies on Heart UCI Dataset compared to accuracies of the graduate student and AutoML

Author	Reported accuracy
Shouman et al. [94]	0.841
Duch et al. [22]	0.856
Wang et al. [108]	0.8337
Srinivas et al.[100]	0.837
Tomar and Agarwal [104]	0.8559
Graduate student (this paper)	0.84
AutoML (this paper)	0.85

Table 1.4: Comparison of AutoML and graduate student's classification performances and total time on UCI test set

	Accuracy	AUC-ROC	AUC-PR	Total time (hours)
Graduate student	0.84	0.82	0.80	432
AutoML	0.85	0.93	0.94	0.5

Table 1.5: Comparison of AutoML and graduate student's classification performances and total time on Cardiovascular test set

	Accuracy	AUC-ROC	AUC-PR	Total time (hours)
Graduate student	0.74	0.73	0.68	360
AutoML	0.74	0.8	0.79	0.5

space is loss of information. The other features (not included within these 13 attributes) include information on the subject's response to exercise Electrocardiogram and cigarette smoking habits among others [43].

1.7 Conclusion

This study intends to propose the use of AutoML for adoption in the clinical domain by breaking the perception that machine learning is accessible to trained experts only. For the first time, we evaluate the performance of an AutoML library (Auto-Sklearn) on two cardiovascular disease datasets and compare the results to that obtained by a graduate student after a month of effort in training multiple classifiers on the datasets. These two cardiovascular datasets contain clinical data from trial subjects and whether or not they

have cardiovascular disease, so that given a new subject's data, the model (learned patterns from given data) can predict the presence or absence of cardiovascular disease with a reasonably good accuracy. The results indicate that the graduate student and AutoML report similar accuracies on the two datasets, on par with other state-of-the-art studies. The area under curves for AutoML is significantly higher indicating that the model built by AutoML generalizes better than that of the graduate student. Besides, the time taken by AutoML to produce these results is just around 30 minutes per dataset, which is significantly less compared to about 400 hours taken by the graduate student. The number of lines of code for AutoML is also significantly lesser compared to the several hundred code lines used by the graduate student, hence justifying the ease of use. Thus, our experimental results strongly suggest that AutoML is a promising approach that enables non-technical users to quickly build competitive machine learning models that work as well as those designed by humans with experience in machine learning. This finding is expected to change the way biomedical researchers and physicians view machine learning. The development of AutoML technology is likely to make machine learning tools more accessible to and speed up the research discovery process in the clinical community. Although this study focuses on cardiovascular disease datasets, we conjecture that the key findings related to the efficiency and efficacy of AutoML will hold for other biomedical datasets. In the future, we will investigate the effects of AutoML on other clinically relevant tasks such as tumor detection and segmentation from medical images. Another important advantage of AutoML techniques is that they can incorporate additional constraints when searching for AI models. For example, physicians might want to maximize the classification accuracy while ensuring the classifier's sensitivity is higher than a certain threshold. Such constraints are hard to optimize in the traditional AI framework. Our future work will evaluate this complex scenario. We expect that the advantage of AutoML will be more prominent when the complexity of the task increases.

Chapter 2

PROPOSAL 2 - NEURAL NETWORK VISUALIZATION

2.1 Introduction

The size of medical data and scan images has grown leaps and bounds due to advancements in acquisition devices. Medical image interpretations are now susceptible to subjectivity and experience of the interpreter. With growing quantum of this data, the scan reports seen by medical experts are also subject to human error. State-of-the-art deep learning architectures are now being used to interpret scan images for medical image classification and segmentation tasks [79]. The combination of high-performance computing and state-of-the-art ML techniques provides efficient diagnosis free from human errors. Deep learning today can perform more than just diagnosis. Some common tasks include image fusion, segmentation, high accuracy surgery guidance, scan image segmentation, identification and labelling [53]. The idea is to extract as much meaning from the images as possible and use them efficiently. Because of the differences in scan images from patient to patient and the associated complexities, having traditional ML algorithms extract meaning from the images was not a viable option anymore. This has given rise to more complex deep learning networks that are able to process this convoluted medical data. Deep neural architecture is loosely inspired by the functioning of human brain. The basic unit in a deep

learning architecture is a neuron, which receives several input values, linearly combines them and passes the result through a non-linear layer to generate the output. A layered architecture of these units form a deep learning architecture. Addition of more and more layers helps unveil complex patterns in data. In some tasks like identifying tumors in MRI scans and cancer in blood, deep learning algorithms are performing better than humans [1]. With this kind of proficiency, deep learning has a major impact in computer vision and medical imaging. Some networks that are having major impact on medical diagnosis today include Convolutional Neural Networks (CNN), Deep Neural Networks (DNN), Recurrent Neural Network (RNN) [65], Deep Conventional Extreme Learning Machine (DC-ELM) [70], Deep AutoEncoder (dA) [41], Deep Boltzmann Machine [87] etc. As the size of input medical data increases and the nature of this data becomes more and more complex, different network architectures are coming to the forefront. Deep learning has mostly impacted ophthalmology, pathology, oncology and radiology among others [79]. Google's DeepMind [76] and IBM Watson [38] are making strides in understanding and interpreting medical images.

Despite all the strides that Deep Learning has made in medicine, it is faced with multiple challenges. The performance of a machine learning model depends on the quantity of data. In cases where the algorithm is performing supervised learning, this large quantities of data is required to be labelled and/or annotated by human experts (physicians), which requires a lot of time and is prone to errors. Second issue is that of unbalanced data. This can be caused by rare diseases or disorders, which is commonplace in medicine. Third and the most significant challenge is that of unexplainability of deep learning techniques. Despite its breakthrough performances over the past years and its ability to open new possibilities, Deep Learning suffers from one big issue, the 'Black box' problem, which is the inability to understand how the network arrived at a prediction. With deeper models, the results became more and more unexplainable, leaving medical experts unable to follow the strategies adopted by the algorithm. In other words, the data fed into the network and the output is inscrutable, leaving the network in between a black box, thereby raising trust issues in medical experts. A group of physicians from BMJ Clinical Research called the ML tools as "rendering verdicts with no accompanying justifications" [24].

Visualizing the deep network is one way to understanding neural networks and break their black box perception. It enables us to 'see' what features the network uses to classify an image into a particular class. There are several techniques by which CNNs can be visualized. Activation maximization, Occlusion maps, Saliancy maps, Class Activation maps (Gradient weighted) and Layerwise output visualization are some of them. Each of these techniques will be discussed on further detail in the following sections.

This proposal focuses on classification of Antibody mediated rejection (ABMR) in kidney(/renal) transplantation patients caused by the rejection of transplanted organ by antibodies from the recipient acting against donor-specific HLA molecules, blood group antigen (ABO)-isoagglutinins, or endothelial cell antigens [97]. This is an impediment to the long-term success of kidney transplantation, but is common and occurs in upto 40% of patients after a year of transplant. A CNN is trained to classify a kidney slice image as having/not having ABMR and the regions in the slice contributing to the said decision are extracted (using Gradient-specific class activation maps) and verified by pathologists.

Section 2.2 discusses the general techniques adopted for visualizing deep networks briefly, following which section 2.3 discusses Grad-CAM visualization technique adopted in this project in detail. Section 2.4 introduces the ABMR classification problem and discusses the general guidelines that are to be followed by pathologists worldwide to arrive at a classification decision and the associated complexities. Section 2.5 discusses the CNN used for making this classification in detail. Section 2.6 provides a report of the results obtained from machine classification using CNN and the corresponding heatmaps showing regions "seen " by the network, along with a comparison of what a pathologist "sees" versus what the network is "seeing" to make a decision.

2.2 Visualization tools for CNNs

There are several strategies by which visualization of CNNs can be achieved [88].

- Activation Maximization [67]: This technique leverages the idea that every layer in a CNN looks for a certain pattern in the output from the previous layer, and the activation from that layer is maximized when the input pattern is found. This is

done by computing the gradient of the activation loss with respect to the input and updating the input as seen in the following equation.

$$gradient = \frac{\partial ActivationMaximizationLoss}{\partial input}. \tag{2.1}$$

An activation maximization map to classify an image as having an elephant or not is dhown in figure 2.1. This helps us perform a sanity check to identify if the network is indeed using features of an elephant (like trunk and tusk) to make this classification.

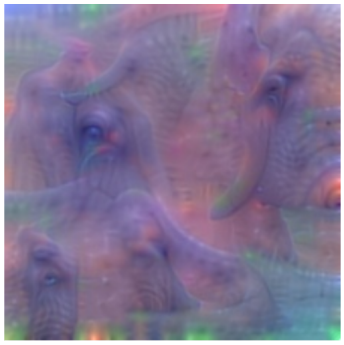

Figure 2.1: Sample visualization map using Activation maximization [88]

- Occlusion Maps: Occlusion maps work by identifying important regions (/patches) in the image. As the name suggests, occlusion maps work by occluding (or masking) some portion of the image and use the CNN to compute the probability that the image still belongs to the original detected class. A decrease in probability would mean the occluded portion is pivotal to the detection of that class, else, it is not. Figure 2.2 shows the regions which the classification network sees using occlusion maps.

Figure 2.2: Original image and image with occlusion map [88]

- Saliancy Maps [96]: Saliancy maps are yet another technique that works on the principle of gradients. It computes the impact of each input pixel value on the output. The gradient of the output is computed with respect to each input image pixel. This is an indication of the effect of each pixel on the output. Positive gradient values are an indication that small increments to the pixel values increase the output. The gradient is computed as in the following equation.

$$gradient = \frac{\partial output}{\partial input}. \tag{2.2}$$

Figure 2.3 indicate the original image and the saliancy map for classifying the image as having/not having a dog.

Figure 2.3: Original image and image with saliancy map [88]

- Class Activation Maps (Gradient Weighted): The idea of gradient weighted class activation maps is to weight the activation maps based on the gradients or how much they affect the output. Gradient weighting takes the feature map from the penultimate layer of the CNN (before the classification layer), computes the gradient of the output with respect to this feature map and average pool all the gradients and multiply the feature map with the pooled gradient. Fig 2.4 shows an image and the corresponding gradient map.

- Layerwise output visualization: As the name suggests, this helps us visualize what features each layer is learning. This helps us compare the performance of each layer and consequently add or remove layers to improve the performance of CNNs. Figure 2.5 shows the outputs from each layer of VGG16 [96]. As the image indicates, lower

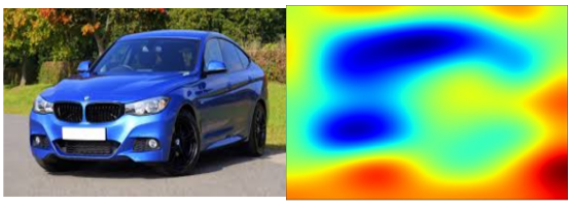

Figure 2.4: Original image and image with gradient map [88]

layers learn simple features like lines, but consequent layers learn more complex objects like roof, door etc.

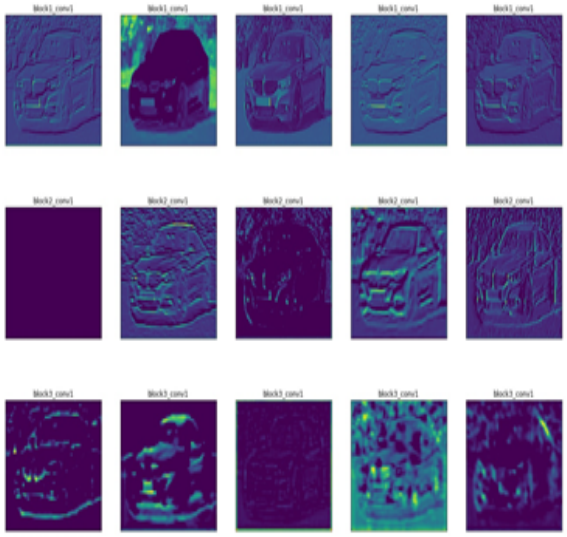

Figure 2.5: Layerwise visualization of classification task from VGG16 [88]

2.3 Grad-CAM visualization

The idea behind Grad-CAM (introduced in [91]) is to use the gradients for any target class up until the final convolutional layer to produce a map that marks regions that are pivotal to the prediction of that class. The speciality of Grad-CAM is that it works with a wide range of CNN network types, including simple classifiers with fully-connected layers like ResNet, DenseNet, VGG etc, structured output models like image captioning networks, CNNs with multi-modal inputs or reinforcement learning, without the need to alter the structure of the network. In addition to visualizing the performance of Neural

Networks, Grad-CAM provides insights into wrongly predicted input examples, helping experts delve into the problem further and obtain suitable solutions. This further helps domain experts discern a "strong" model from a "weak" one and establish trust in the neural network.

The authors of this paper strongly believe in having more "transparent" models that are capable of explaining the reasons for their prediction. They believe that in all stages of AI, model explainability comes in handy. For example, at the beginning stages when performance of AI was poorer than its human counterpart, model explanability helped human experts work on the models to better it, when the performance of AI is in par with that of humans, AI is required to build trust among users and when AI performs better than humans, model explainability can teach humans how to make better decisions.

With earlier visualizations, the complexity of a model and its explainability are inversely related. Several visualization techniques like Class Activation Mapping (CAM) [113] identify regions in the image that the network sees but can be applied to CNNs that do not contain the fully connected layers. In other words, one had to forgo model complexity to perform visualization. But, Grad-CAM is capable of working with highly complex models as discussed previously without the need to perform any architectural changes to the model. A good visualization is one that is both highly discriminative to individual classes and of high resolution. Grad-CAM visualizations combine the benefits of Pixel-space gradient visualizations like Guided-back propagation [99], deconvolution [112] and CAM [113] that generate highly class discriminative mappings.

This paper introducing Grad-CAM makes several other contributions:

- For captioning and visual question answering (VQA) tasks, Grad-CAM applied on the network combination of CNN+LSTM models are good at pinpointing image regions that contribute to a caption prediction.

- Helps uncover bias in some models that CNNs learn, helping ensure more fair and bias-free outcomes.

- Generate textual explanations for model classification using "Neuron importance" [6] or identifying the weight of a neuron towards making a certain prediction.

Several visualization strategies have been used in the past. Techniques like **Pixel based visualizations** [95] that highlight important pixels using the partial derivative of class scores versus pixel intensities and **Guided Backpropagation** [99] **Deconvolution** [112], where the idea is to make changes to the gradient to improve quality are not class discriminative, although they produce fine-grained visualizations. Techniques that **Maximally activate network units** [23] and **Invert latent representations** [59] are not specific to the image but generate a common map for the model.

Marco Tulio Ribeiro et al. [80] provide a standard for evaluating the model visualizations. The idea behind **Weakly supervised localization** [16] is to use class labels and localize corresponding objects in the image. Such techniques (including CAM), replace fully connected layers in CNNs with convolutional layers and perform global average pooling [55] in order to obtain feature maps for individual classes. Since the feature maps prior to the softmax layer are tampered with, this may cause inferior performance. Grad-CAM offers a new way of combining feature maps with gradients that doesn't need any architectural changes to the model. There are several other methods that perform step-by-step perturbation (or masking) of the input image, like in the case of occlusion maps discussed earlier.

The final fully connected layer right before the softmax classification layer typically looks for class-specific information from the whole image (regions in the image that is associated with the label). Grad-CAM taps into the gradients coming into the layer to identify the importance of each neuron for this particular classification. This can be used to obtain the neuron importance from any layer, not just the last fully connected layer.

The idea is to obtain class discriminative localization map as follows.

$$
\begin{aligned}
&\textit{Localization Map } L^c_{GradCAM} \in \mathbb{R}^{u*v} \\
&\textit{where c -> class} \\
&u, v -> \textit{ width, height of image for class c} \\
&y^c -> \textit{ output at the fully connected layer before softmax} \\
&\textit{Compute } \frac{\partial y^c}{\partial A^k} \textit{ where } A^k -> \textit{ feature map activations of convolution layer.}
\end{aligned}
\tag{2.3}
$$

The gradients flowing backwards are global average pooled over all pixels to get neuron importance weights, α^c_k as in the following equation.

$$
\alpha^c_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A^k_{ij}}. \tag{2.4}
$$

The final map is obtained by

$$
\alpha^c_k = ReLU\ (\sum_k \alpha^c_k A_k), \tag{2.5}
$$

where ReLU is the activation function.

As can be seen from the above equation, the result is a course heatmap of the same shape as the input image. ReLU is applied to the linear combination of all the maps so that only those pixels whose increase in intensity leads to an increase in y^c are extracted.

2.3.1 Guided Grad-CAM

Guided Grad-CAM is the fusion of Grad-CAM's ability to be class discriminative, identify appropriate regions in the image according to class and Guided Backpropagation's ability to provide fine-grained visualizations. Guided backpropagation captures those pixels in the image that activate neurons and not those that suppress them. To apply guided backpropagation to Grad-CAM, the Grad-CAM map, $L^c_{Grad-CAM}$ is upsampled by bilinear interpolation to obtain an input-image sized mask and element-wise multiplication of the two is performed. This yield the benefit of both class-discriminative and fine-grained pixel space visualization.

2.4 ABMR

ABMR is the rejection of allograft caused by antibodies in the organ recipient [97]. It usually occurs in patients over a year after transplantation but but could occur (in <40% of the cases) in patients even within a year of desensitization [13]. Desensitization is the process of removing antibodies that could attack foreign tissue from the blood stream of the recipient prior to transplantation [49]. ABMR is thought to be the leading cause of graft loss. Treatment options are limited and guided by randomized clinical trials in small groups [89]. The level of evidence for treatment options are limited today, but there are numerous options for classification of ABMR [82].

2.4.1 ABMR classification by pathologists

Diagnostic criteria for classification of kidney-slice as having/not-having ABMR follows the Banff classification [82]. Banff lesion score is an international consensus among pathologists to classify a kidney whole slice image (WSI) as having/not-having ABMR. It is intended as a guide to physicians for assessing and obtaining Banff lesion scores [82].

This proposal works on Glomerulus (a bunch of capillaries at the end of the kidney tubule where the filtering of waste from water happens) whole slide images. In general, deposits in the glomerular capillaries is indication of poor renal allograft survival and ABMR [31]. Banff cg score (double contours in the glomerulus basement membrane) is an indication of both the presence and extent of double contours or lamination in the affected glomerulus. In general, the extent is scored from 0 to 3 based on the percentage of capillary loops with double contours. Figure 2.6 depicts four glomerulus images affected by ABMR to different degrees. These stains can be taken using Light microscopy (LM) or Electron Microscopy (EM). Image A in fig 2.6 is a slice with double contours (short black arrow indicates original basement membrane and short red arrows point to new formation) visible using EM. Long black arrow is an indication of cell swelling. The swelling accompanied by a minimum of 3 glomerular capillaries noticed using EM gives it a Banff lesion score of cg1a. Image B has a Banff lesion score of cg1b, double contours (identified by red arrows) affect atleast 25% of all capillary loops. Image C has a Benff lesion score of cg2, where

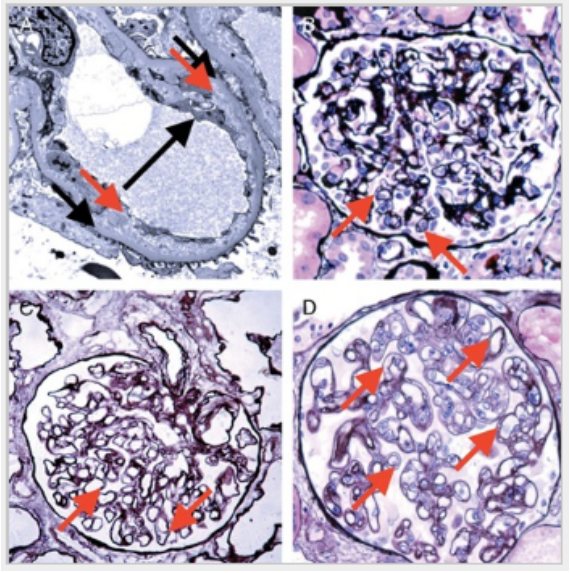

Figure 2.6: Glomerulus images with varying degrees of ABMR [82]

double cappilaries affect 26-50% of all capillaries, and image D with a Benff score of cg3 affects atleast 50% of all glomerulus capillaries [82].

Thus, the classification of glomerulus slice images into ABMR/No ABMR is clearly a problem that requires the extraction of unique features from the glomerulus image. Given that CNNs today are capable of performing feature extractions at superhuman level accuracies [56], the multiple layers in the network should be able to extract meaning from the glomerulus images and make appropriate classifications.

This proposal performs classification of whole slide glomerulus images into ABMR/No ABMR using Residual Networks [36], identify regions in the slice image that the network looks at to make the classification decision and obtains pathologist review to confirm if the regions being looked at by the CNN and the pathologist are the same.

2.5 Machine classification

Classification in this problem is achieved using Residual Networks (ResNet). The ResNet model introduced by Kaiming He et al. [36], was conceived to solve the 'degradation problem'.

Degradation problem: As more and more of layers are stacked in a network to unearth

more and more complex features, the network suffers from accuracy saturation beyond which the performance begins to fall. This degradation is not a result of overfitting.

ResNets address this problem by making each stacked layer fit a residual mapping instead of the desired underlying mapping, i.e, if a layer's mapping is $\mathscr{H}(x)$, the stacked non-linear layers are made to fit $\mathscr{F}(x) := \mathscr{H}(x) - x$. Now, the original mapping becomes $\mathscr{H}(x) := \mathscr{F}(x) + x$. $\mathscr{F}(x) + x$ can be realized using a combination of the 'Feed-forward connection' $\mathscr{F}(x)$ and the 'shortcut connection' x, which learn identity mapping. In short, each block (may consist of several layers) and its output can be represented as in figure 2.7. The advantage of 'shortcut connections' is that no extra parameters need to be learned,

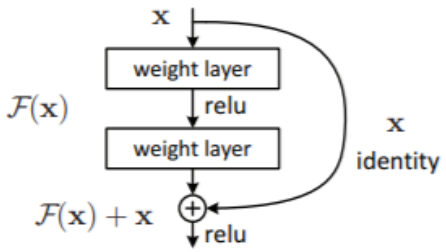

Figure 2.7: Building bock of residual networks [36]

thereby not adding to the computational complexity and can be solved using the same solvers like stochastic gradient descent. O Russakovsky et al. [84] show that ResNets enjoy increased accuracy even at high depths unlike other stacked networks that suffer 'degradation' as depths increase. Each building block in this network is defined as in the equation

$$
\begin{aligned}
& y = \mathscr{F}(x, W_i) + x \\
& \textit{where x is the input vector to the block} \\
& \textit{y is the output vector from the block} \\
& W_i \textit{ are the learned weight.}
\end{aligned}
\tag{2.6}
$$

The function $\mathscr{F}$ is flexible and can cover two or three layers as in figure 2.8. ResNet-x translates to a residual network with x layers. ResNet-34 and it's blocks are depicted in figure 2.9.

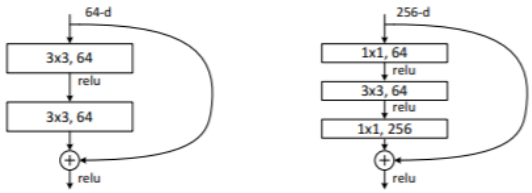

Figure 2.8: Building block in ResNet34 and "bottleneck" building block in ResNet-50/101/152. [36]

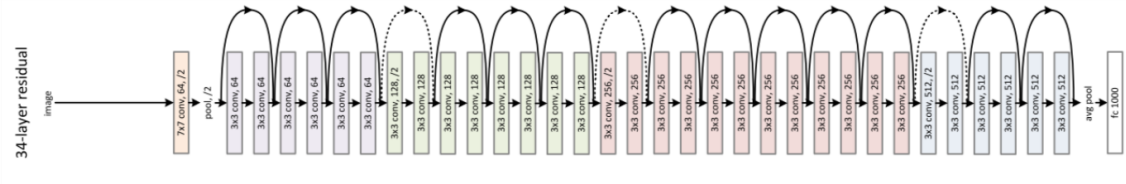

Figure 2.9: ResNet-34 [36]

Performance: ResNet-152 (ResNet with 152 layers) performs with 6.6% lesser top-1 error rate and 3.6% lesser top-5 error rate than VGG-16 [96] on the ImageNet [20] validation dataset, despite being less complex (with 8 times more depth) than VGG-16, showing that deeper networks can be used to extract more complex features, circumventing the 'degradation' problem. This performance of ResNet on the ImageNet task won the first place on the ILSVRC 2015 classification task. An ensemble of ResNets achieves an error of 3.57% on the ImageNet test set.

2.6 Results and discussion

2.6.1 Dataset

The dataset consists of 274 total Glomerulus whole slide images, 113 images with ABMR and 161 with no ABMR. For training purposes, the dataset is split into 196 training images (82 images with ABMR and 114 without ABMR) and 78 testing images (31 with ABMR and 47 without ABMR).

2.6.2 Training

ResNet50, ResNet101 and ResNet152 are trained on the images over 200 epochs, with a constant learning rate of 10^{-3}. Two sets of experiments are performed; one with using just random rotation for data augmentation [77] and another with using random rotation along with random crop as augmentation. The training process is illustrated in figure 2.10.

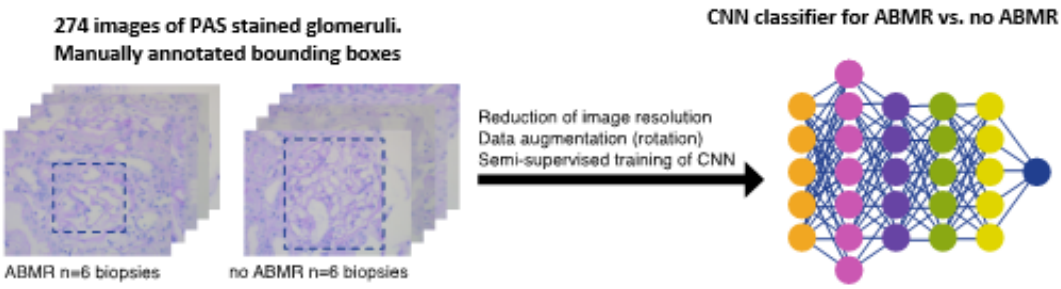

Figure 2.10: Preliminary CNN solution for a classification task in transplant nephropathology [7]

Maximum accuracy obtained on the test results are listed in Table 2.1.

Table 2.1: Performance (Accuracy) of ResNet models on test set

Network	With random rotation (no crop)	With random rotation and crop
ResNet50	98.72%	91.02%
ResNet101	96.15%	93.59%
ResNet152	96.15%	91.03%

2.6.2.1 Training curves

Figure 2.11 shows the smoothed validation curves for the three ResNet training networks. Resnet50 displays slightly higher validation accuracy over the other two.

Figure 2.12 is the smoothed training and validation accuracy curve for ResNet50.

2.6.2.2 Visualizations

Grad-CAM is used to perform visualizations on the test images to indicate the regions in the slide that the network sees in order to make the classification decision. As discussed

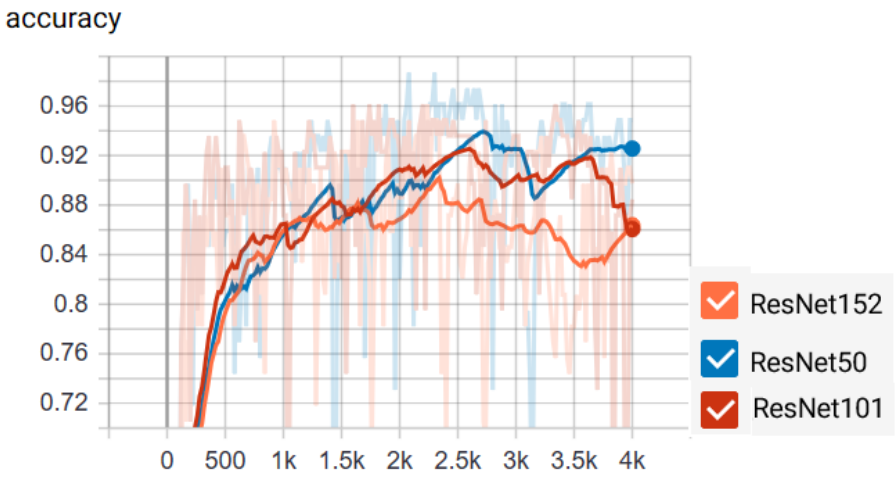

Figure 2.11: Validation accuracy curves for the three ResNet networks

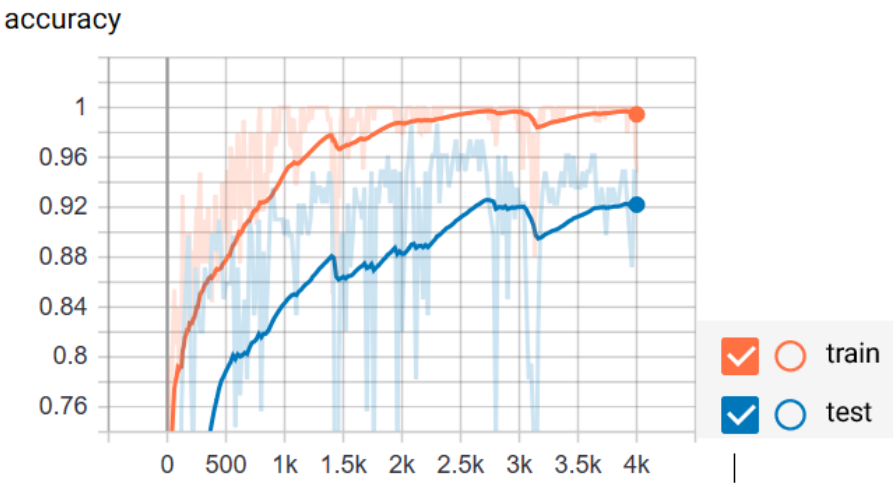

Figure 2.12: Training and validation accuracy curves ResNet50

in section 2.3, Grad-CAM can be used to generate class discriminative visualizations. The following images show some visualizations generated by Gad-CAM.

Fig 2.13 shows a true positive class predicted by the model and the corresponding Grad-CAM map.

Fig 2.14 shows a true negative class predicted by the model and the corresponding Grad-CAM map. Fig 2.16 shows wrongly predicted by the model and the corresponding Grad-CAM map.

As discussed in section 2.4.1, a trained human nephropathologist would look for fea-

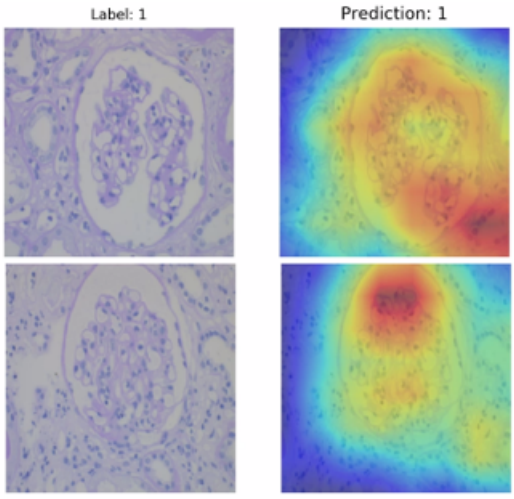

Figure 2.13: True positive predictions with Grad-CAM visualization

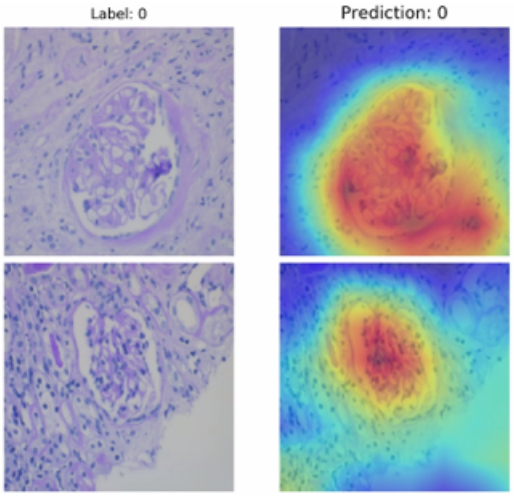

Figure 2.14: True negative predictions with Grad-CAM visualization

tures of glomerulitis before making the ABMR versus not ABMR decision. Some of these features include glomerular basement membrane splitting, secondary focal and segmental glomerulosclerosis [7].

It can be seen from the wrong predictions in figure 2.16 that the regions that the network focuses on are largely outside of the glomerular tuft, alerting the pathologist to a possible misclassification. In contrast, the heatmap in the example with the correct classification highlights important regions in the glomerulus that is indicative of ABMR.

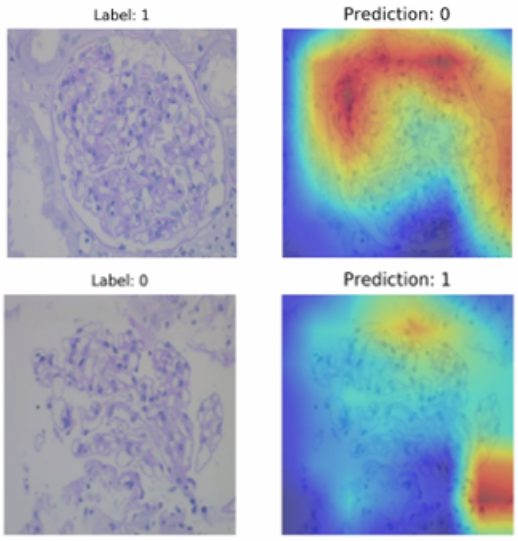

Figure 2.15: False predictions with Grad-CAM visualization

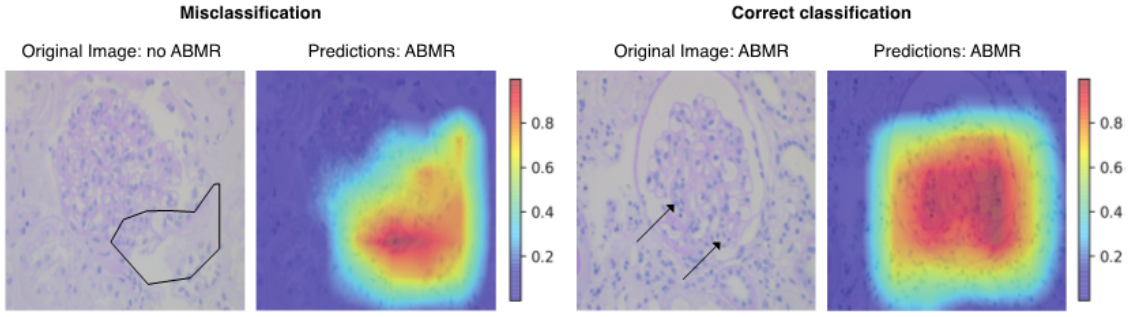

Figure 2.16: Heatmaps from network and regions nephropathologists see [7]

It is interesting to note that cropping the image as an augmentation step lowers the classification accuracy as seen from table 2.1, further reinforcing the fact that several features from the glomerulus as a whole is required to make the ABMR versus no ABMR decision as discussed in section 2.4.1. Wrong predictions made post image crop is shown in figure 2.17.

2.7 Conclusion and future work

This proposal works on using visualization techniques to understand how CNNs are operating on the input image ans consequently making decisions. This helps build trust in

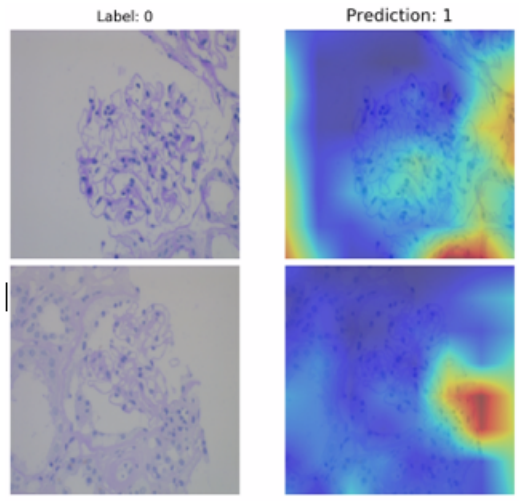

Figure 2.17: Wrong predictions made from cropped images show that the whole image (several features from Glomerulus) is required to make correct prediction

medical professionals who are skeptical about AI and hence break it's 'black box' perception. AI, specifically in nephropathology (dealing with diagnosis of kidney diseases) is still at it's infancy. Nephropathology is complicated because of the need to integrate multiple factors before coming to a diagnosis. AI can aid nephrologists in several ways by helping them with repetitive tasks that require high attention to detail [7].

In this proposal, Grad-CAM is used to generate visualizations of Glomerulus whole slide images post their classification using ResNets. ResNets with several different number of layers are trained to classify the images. The generated visualizations are studied to understand what the network 'sees' to make correct predictions and where the network 'sees' when failing to make the right predictions. These results are compared with what and where a specialist is likely to see when making a similar decision. If wrong classifications are made, visualizations are used to check if wrong portions of the slide are focused on while making that decision. This would help a specialist know exactly why the network is failing.

Visualization tools are greatly helpful to enable medical experts to get a thorough insight into the network. Today new and improved visualization tools have been proposed like Grad-CAM ++, which build on Grad-CAM and provides human-interpretable visual explanations for highlighted regions in the image [14]. Ablation based class activation

mapping [78] is shown to perform better than Grad-CAM using ablation analysis techniques instead of the gradient approach to determine the weights of the feature maps with respect to each class. Thus, the avenues for understanding a network and innards of its functioning are plenty. Thus, future work would be to focus on integrating neural networks with such techniques that can help medical specialists fully understand neural architectures and their functioning better.

References

[1] A. S. Ahuja. "The impact of artificial intelligence in medicine on the future role of the physician." In: *PeerJ* 7 (2019), e7702.

[2] J. Ali, R. Khan, N. Ahmad, and I. Maqsood. "Random forests and decision trees." In: *International Journal of Computer Science Issues (IJCSI)* 9.5 (2012), p. 272.

[3] A. Alonso Abad. Wiley statsref: Statistics reference online. Wiley, 2015.

[4] Amitabha Dey. *Data Preprocessing for Machine Learning*. `https://medium.com/datadriveninvestor/data-preprocessing-for-machine-learning-188e9eef1d2c`. Online; accessed 20 April 2020. 2018.

[5] S. Bashir, U. Qamar, and F. H. Khan. "BagMOOV: A novel ensemble for heart disease prediction bootstrap aggregation with multi-objective optimized voting." In: *Australasian physical & engineering sciences in medicine* 38.2 (2015), pp. 305–323.

[6] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. "Network dissection: Quantifying interpretability of deep visual representations." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 6541–6549.

[7] J. U. Becker, D. Mayerich, M. Padmanabhan, J. Barratt, A. Ernst, P. Boor, P. A. Cicalese, C. Mohan, H. V. Nguyen, and B. Roysam. "Artificial intelligence and machine learning in nephropathology." In: *Kidney International* (2020).

[8] G. Biau, B. Cadre, and L. Rouvière. "Accelerated gradient boosting." In: *Machine Learning* 108.6 (2019), pp. 971–992.

[9] L. Breiman. "Bagging predictors." In: *Machine learning* 24.2 (1996), pp. 123–140.

[10] L. Breiman. “Pasting small votes for classification in large databases and on-line.” In: *Machine learning* 36.1-2 (1999), pp. 85–103.

[11] L. Breiman. “Random forests.” In: *Machine learning* 45.1 (2001), pp. 5–32.

[12] L. Breiman. “Some properties of splitting criteria.” In: *Machine Learning* 24.1 (1996), pp. 41–47.

[13] K. Budde and M. Dürr. *Any progress in the treatment of antibody-mediated rejection?* 2018.

[14] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian. “Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks.” In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 839–847.

[15] R Chitra and V Seenivasagam. “Review of heart disease prediction system using data mining and hybrid intelligent techniques.” In: *ICTACT journal on soft computing* 3.04 (2013), pp. 605–09.

[16] R. G. Cinbis, J. Verbeek, and C. Schmid. “Weakly supervised object localization with multi-fold multiple instance learning.” In: *IEEE transactions on pattern analysis and machine intelligence* 39.1 (2016), pp. 189–203.

[17] M. Claesen and B. De Moor. “Hyperparameter search in machine learning.” In: *arXiv preprint arXiv:1502.02127* (2015).

[18] Dataflair team. *Kernel Functions-Introduction to SVM Kernel Examples.* `https://data-flair.training/blogs/svm-kernel-functions/`. Online; accessed 29 March 2020. 2018.

[19] Dataflair team. *Support Vector Machines(SVM) — An Overview.* `https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989`. Online; accessed 29 March 2020. 2018.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “Imagenet: A large-scale hierarchical image database.” In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[21] Dhilip Subramanian. *A Simple Introduction to K-Nearest Neighbors Algorithm.* https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e. Online; accessed 29 March 2020. 2019.

[22] W. Duch, R. Adamczak, and K. Grabczewski. "A new methodology of extraction, optimization and application of crisp and fuzzy logical rules." In: *IEEE Transactions on Neural Networks* 12.2 (2001), pp. 277–306.

[23] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. "Visualizing higher-layer features of a deep network." In: *University of Montreal* 1341.3 (2009), p. 1.

[24] Eric Bender. *Unpacking the Black Box in Artificial Intelligence for Medicine.* https://undark.org/2019/12/04/black-box-artificial-intelligence/. Online; accessed 1 May 2020. 2019.

[25] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. "Efficient and robust automated machine learning." In: *Advances in neural information processing systems*. 2015, pp. 2962–2970.

[26] Y. Freund and R. E. Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting." In: *European conference on computational learning theory*. Springer. 1995, pp. 23–37.

[27] J. H. Friedman. "Stochastic gradient boosting." In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378.

[28] G. Fung and O. L. Mangasarian. "Incremental support vector machine classification." In: *Proceedings of the 2002 SIAM International Conference on Data Mining*. SIAM. 2002, pp. 247–260.

[29] T. Fushiki. "Estimation of prediction error by using K-fold cross-validation." In: *Statistics and Computing* 21.2 (2011), pp. 137–146.

[30] Google Cloud. *Cloud AutoML.* https://cloud.google.com/automl. Online; accessed 19 March 2020. 2017.

[31] V. Goutaudier, H. Perrochia, S. Mucha, M. Bonnet, S. Delmas, F. Garo, V. Garrigue, S. Lepreux, V. Pernin, and J.-E. Serre. "C5b9 deposition in glomerular capillaries is associated with poor allograft survival in antibody-mediated rejection of kidney allograft." In: *Frontiers in immunology* 10 (2019), p. 235.

[32] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macià, B. Ray, M. Saeed, and A. Statnikov. "Design of the 2015 chalearn automl challenge." In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2015, pp. 1–8.

[33] I. Guyon, A. Saffari, G. Dror, and G. Cawley. "Model selection: Beyond the bayesian/frequentist divide." In: *Journal of Machine Learning Research* 11.Jan (2010), pp. 61–87.

[34] H2O. *AutoML: Automatic Machine Learning*. `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`. Online; accessed 19 March 2020. 2017.

[35] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. "The WEKA data mining software: an update." In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[36] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[37] J. Heaton. Introduction to neural networks with Java. Heaton Research, Inc., 2008.

[38] R. High. "The era of cognitive systems: An inside look at IBM Watson and how it works." In: *IBM Corporation, Redbooks* (2012), pp. 1–16.

[39] J. M. Hilbe. Logistic regression models. CRC press, 2009.

[40] T. K. Ho. "The random subspace method for constructing decision forests." In: *IEEE transactions on pattern analysis and machine intelligence* 20.8 (1998), pp. 832–844.

[41] C. Hong, J. Yu, J. Wan, D. Tao, and M. Wang. "Multimodal deep autoencoder for human pose recovery." In: *IEEE Transactions on Image Processing* 24.12 (2015), pp. 5659–5670.

[42] A. Hyvärinen, J. Karhunen, and E. Oja. Independent component analysis. Vol. 46. John Wiley & Sons, 2004.

[43] Janosi, A. and Steinbrunn, W. and Pfisterer, M. and Detrano. *Heart Disease Data Set.* //archive.ics.uci.edu/ml/datasets/Heart+Disease(accessedon10July2019). Online; accessed 30 April 2020. 2018.

[44] H. Jin, Q. Song, and X. Hu. "Auto-keras: An efficient neural architecture search system." In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 1946–1956.

[45] M. Jirina, M. Jirina, and K Funatsu. "Classifiers based on inverted distances." In: *New fundamental technologies in data mining*. Vol. 1. InTech, 2011, pp. 369–387.

[46] I. Jolliffe. Principal component analysis. Springer, 2011.

[47] KDnuggets. *Frameworks for Approaching the Machine Learning Process.* https://www.kdnuggets.com/2018/05/general-approaches-machine-learning-process.html. Online; accessed 19 March 2020. 2018.

[48] KDNuggets. *Contest Winner: Winning the AutoML Challenge with Auto-sklearn.* https://www.kdnuggets.com/2016/08/winning-automl-challenge-auto-sklearn.html. Online; accessed 20 April 2020. 2016.

[49] KDnuggets. *Kidney Desensitization Program Frequently Asked Questions*. https://www.uwhealth.org/transplant/kidney-desensitization-program-frequently-asked-questions/10618. Online; accessed 3 May 2020. 2015.

[50] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[51] A. Lacoste, M. Marchand, F. Laviolette, and H. Larochelle. "Agnostic Bayesian learning of ensembles." In: *International Conference on Machine Learning*. 2014, pp. 611–619.

[52] E. G. Learned-Miller. "Entropy and mutual information." In: *Department of Computer Science, University of Massachusetts, Amherst* (2013).

[53] J.-G. Lee, S. Jun, Y.-W. Cho, H. Lee, G. B. Kim, J. B. Seo, and N. Kim. "Deep learning in medical imaging: general overview." In: *Korean journal of radiology* 18.4 (2017), pp. 570–584.

[54] A. Liaw and M. Wiener. "Classification and regression by randomForest." In: *R news* 2.3 (2002), pp. 18–22.

[55] M. Lin, Q. Chen, and S. Yan. "Network in network." In: *arXiv preprint arXiv:1312.4400* (2013).

[56] P. Liskowski, W. Jaśkowski, and K. Krawiec. "Learning to play othello with deep neural networks." In: *IEEE Transactions on Games* 10.4 (2018), pp. 354–364.

[57] M. Liu, R. Lang, and Y. Cao. "Number of trees in random forest." In: *Computer Engineering and Applications* 51.5 (2015), pp. 126–131.

[58] G. Louppe and P. Geurts. "Ensembles on random patches." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2012, pp. 346–361.

[59] A. Mahendran and A. Vedaldi. "Visualizing deep convolutional neural networks using natural pre-images." In: *International Journal of Computer Vision* 120.3 (2016), pp. 233–255.

[60] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. "Boosting algorithms as gradient descent." In: *Advances in neural information processing systems*. 2000, pp. 512–518.

[61] S. Mathanker, P. Weckler, T. Bowser, N Wang, and N. Maness. "AdaBoost classifiers for pecan defect classification." In: *Computers and electronics in agriculture* 77.1 (2011), pp. 60–68.

[62] M. Mehta, J. Rissanen, and R. Agrawal. "MDL-Based Decision Tree Pruning." In: *KDD*. Vol. 21. 2. 1995, pp. 216–221.

[63] W. G. MEMBERS, V. L. Roger, A. S. Go, D. M. Lloyd-Jones, E. J. Benjamin, J. D. Berry, W. B. Borden, D. M. Bravata, S. Dai, and E. S. Ford. "Heart disease and stroke statistics—2012 update: a report from the American Heart Association." In: *Circulation* 125.1 (2012), e2.

[64] Microsoft. *Microsoft AutoML.* https://www.microsoft.com/en-us/research/project/automl/. Online; accessed 19 March 2020. 2017.

[65] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. "Recurrent neural network based language model." In: *Eleventh annual conference of the international speech communication association*. 2010.

[66] R. M. Nelson, M. Kierczak, and Ö. Carlborg. "Higher order interactions: detection of epistasis using machine learning and evolutionary computation." In: *Genome-Wide Association Studies and Genomic Prediction*. Springer, 2013, pp. 499–518.

[67] A. Nguyen, J. Yosinski, and J. Clune. "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks." In: *arXiv preprint arXiv:1602.03616* (2016).

[68] Nicolas CHEREL,Mohamed MASKANI,Henri GERARD. *Home - Welcome to ML-Box's official documentation*. https://mlbox.readthedocs.io/en/latest/. Online; accessed 19 March 2020. 2017.

[69] R. S. Olson and J. H. Moore. "TPOT: A Tree-Based Pipeline Optimization Tool for Automating." In: *Automated Machine Learning: Methods, Systems, Challenges* (2019), p. 151.

[70] S. Pang and X. Yang. "Deep convolutional extreme learning machine and its application in handwritten digit classification." In: *Computational intelligence and neuroscience* 2016 (2016).

[71] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[72] H. Peng, F. Long, and C. Ding. "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy." In: *IEEE Transactions on pattern analysis and machine intelligence* 27.8 (2005), pp. 1226–1238.

[73] L. E. Peterson. "K-nearest neighbor." In: *Scholarpedia* 4.2 (2009), p. 1883.

[74] F. Pfisterer, J. N. van Rijn, P. Probst, A. Müller, and B. Bischl. "Learning Multiple Defaults for Machine Learning Algorithms." In: *arXiv preprint arXiv:1811.09409* (2018).

[75] S. Pouriyeh, S. Vahid, G. Sannino, G. De Pietro, H. Arabnia, and J. Gutierrez. "A comprehensive investigation and comparison of Machine Learning Techniques in the domain of heart disease." In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2017, pp. 204–207.

[76] J. Powles and H. Hodson. "Google DeepMind and healthcare in an age of algorithms." In: *Health and technology* 7.4 (2017), pp. 351–367.

[77] PyTorch. *Tochvision.Transforms*. `https://pytorch.org/docs/stable/torchvision/transforms.html`. Online; accessed 2 May 2020. 2016.

[78] H. G. Ramaswamy. "Ablation-CAM: Visual Explanations for Deep Convolutional Network via Gradient-free Localization." In: *The IEEE Winter Conference on Applications of Computer Vision*. 2020, pp. 983–991.

[79] M. I. Razzak, S. Naz, and A. Zaib. "Deep learning for medical image processing: Overview, challenges and the future." In: *Classification in BioApps*. Springer, 2018, pp. 323–350.

[80] M. T. Ribeiro, S. Singh, and C. Guestrin. ""' Why should i trust you?" Explaining the predictions of any classifier." In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[81] H. Robbins and S. Monro. "A stochastic approximation method." In: *The annals of mathematical statistics* (1951), pp. 400–407.

[82] C. Roufosse, N. Simmonds, M. Clahsen-van Groningen, M. Haas, K. J. Henriksen, C. Horsfield, A. Loupy, M. Mengel, A. Perkowska-Ptasinska, and M. Rabant. "A 2018 reference guide to the Banff classification of renal allograft pathology." In: *Transplantation* 102.11 (2018), pp. 1795–1814.

[83] S. Ruder. "An overview of gradient descent optimization algorithms." In: *arXiv preprint arXiv:1609.04747* (2016).

[84] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein. "Imagenet large scale visual recognition challenge." In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[85] D. Ruta and B. Gabrys. "Classifier selection for majority voting." In: *Information fusion* 6.1 (2005), pp. 63–81.

[86] Saishruthi Swaminathan. *Logistic Regression — Detailed Overview.* `https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc`. Online; accessed 27 March 2020. 2018.

[87] R. Salakhutdinov and G. Hinton. "Deep boltzmann machines." In: *Artificial intelligence and statistics*. 2009, pp. 448–455.

[88] Saurabh Pal. *A Guide to Understanding Convolutional Neural Networks (CNNs) using Visualization.* `https://www.analyticsvidhya.com/blog/2019/05/understanding-visualizing-neural-networks/`. Online; accessed 2 May 2020. 2019.

[89] B. Sautenet, G. Blancho, M. Büchler, E. Morelon, O. Toupance, B. Barrou, D. Ducloux, V. Chatelet, B. Moulin, and C. Freguin. "One-year results of the effects of rituximab on acute antibody-mediated rejection in renal transplantation: RITUX ERAH, a multicenter double-blind randomized placebo-controlled trial." In: *Transplantation* 100.2 (2016), pp. 391–399.

[90] B. Schölkopf, A. J. Smola, and F. Bach. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[91] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. "Grad-cam: Visual explanations from deep networks via gradient-based localization." In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[92] Shaurya Uppal. *Label Encoder vs. One Hot Encoder in Machine Learning.* `https://www.geeksforgeeks.org/python-how-and-where-to-apply-feature-scaling/`. Online; accessed 24 April 2020. 2018.

[93] Y.-S. Shih. "A note on split selection bias in classification trees." In: *Computational statistics & data analysis* 45.3 (2004), pp. 457–466.

[94] M. Shouman, T. Turner, and R. Stocker. "Using decision tree for diagnosing heart disease patients." In: *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*. 2011, pp. 23–30.

[95] K. Simonyan, A. Vedaldi, and A. Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." In: *arXiv preprint arXiv:1312.6034* (2013).

[96] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[97] N. Singh, J. Pirsch, and M. Samaniego. "Antibody-mediated rejection: treatment alternatives and outcomes." In: *Transplantation Reviews* 23.1 (2009), pp. 34–46.

[98] J. Snoek, H. Larochelle, and R. P. Adams. "Practical bayesian optimization of machine learning algorithms." In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[99] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. "Striving for simplicity: The all convolutional net." In: *arXiv preprint arXiv:1412.6806* (2014).

[100] K Srinivas, B. K. Rani, and A Govrdhan. "Applications of data mining techniques in healthcare and prediction of heart attacks." In: *International Journal on Computer Science and Engineering (IJCSE)* 2.02 (2010), pp. 250–255.

[101] Sunny Srinidhi. *Label Encoder vs. One Hot Encoder in Machine Learning*. `https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc273365621`. Online; accessed 24 April 2020. 2018.

[102] C. D. Sutton. "Classification and regression trees, bagging, and boosting." In: *Handbook of statistics* 24 (2005), pp. 303–329.

[103] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms." In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 847–855.

[104] D. Tomar and S. Agarwal. "Feature selection based least square twin support vector machine for diagnosis of heart disease." In: *International Journal of Bio-Science and Bio-Technology* 6.2 (2014), pp. 69–82.

[105] Ulianova, S. *Cardiovascular Disease Dataset*. `https://www.kaggle.com/sulianova/cardiovascular-disease-dataset`. Online; accessed 5 May 2020. 2018.

[106] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo. "OpenML: networked science in machine learning." In: *ACM SIGKDD Explorations Newsletter* 15.2 (2014), pp. 49–60.

[107] K Vembandasamy, R Sasipriya, and E Deepa. "Heart diseases detection using Naive Bayes algorithm." In: *International Journal of Innovative Science, Engineering & Technology* 2.9 (2015), pp. 441–444.

[108] S.-j. Wang, A. Mathew, Y. Chen, L.-f. Xi, L. Ma, and J. Lee. "Empirical analysis of support vector machine ensemble classifiers." In: *Expert Systems with applications* 36.3 (2009), pp. 6466–6476.

[109] D. H. Wolpert and W. G. Macready. "No free lunch theorems for optimization." In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[110] K. Yan and D. Zhang. "Feature selection and analysis on correlated gas sensor data with recursive feature elimination." In: *Sensors and Actuators B: Chemical* 212 (2015), pp. 353–363.

[111] Q. Yao, M. Wang, Y. Chen, W. Dai, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang. "Taking human out of learning applications: A survey on automated machine learning." In: *arXiv preprint arXiv:1810.13306* (2018).

[112] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks." In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[113] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. "Learning deep features for discriminative localization." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.

Appendices

A Daywise performance on Heart UCI and Cardiovascular Disease Dataset

Validation accuracy and areas under curves on UCI Heart dataset over different days				
	Algorithm	**Validation accuracy**	**Area under PR**	**Area under ROC**
1	Logistic regression with default parameters	0.78878	0.9011128	0.89478
	Linear SVM with default parameters	0.84756	0.896318	0.895693
	SVM with RBF kernel with default parameters	0.808	0.891344	0.88138
2	Logistic Regression (with default parameters) run on best feature subset selected using F-test	0.8128	0.8880569	0.88159
	Logistic Regression (with default parameters) run on best feature subset selected using mutual information test	0.808048	0.8975	0.89674
	Linear SVM (with default parameters) run on best feature subset selected using F-test	0.84756	0.896318	0.895693
	Linear SVM (with default parameters) run on best feature subset selected using mutual information test	0.84756	0.896318	0.895693
	SVM with RBF kernel (with default parameters) run on best feature subset selected using F-test	0.8329	0.904811	0.89211665
	SVM with RBF kernel (with default parameters) run on best feature subset selected using mutual information test	0.823	0.903723	0.8940077
3	Hyperparameter selected SVM (with RBF kernel)	0.813	0.886121	0.88038
	Decision Tree classifier with default parameters	0.699878	0.66709	0.697596
	Random forest with default parameters	0.783536	0.8882388	0.8793
	Logistic Regression (with default parameters) run on best feature subset selected using RFE	0.8129	0.890859	0.894463

4	Linar SVM (with default parameters) run on best feature subset selected using RFE	0.84756	0.896318	0.895693
	Hyperparameter selected SVM (with RBF kernel) run on best feature subset selected using F-test	0.8329	0.904811	0.89211
	Hyperparameter selected SVM (with RBF kernel) run on best feature subset selected using mutual information test	0.82292	0.904803	0.89203
	Hyperparameter selected decision tree	0.74402	0.745473	0.7643768
	Extra Trees classifier with deafult parameters	0.79841	0.8530859	0.86375
5	Hyperparameter selected Logistic Regression	0.80817	0.89997	0.89817
	Hyperparameter selected Logistic Regression run over best feature subset selected using RFE	0.82256	0.8974	0.89742
	Bagged decision tree with default bagging parameters	0.74939	0.8477148	0.837058
	Hyperparameter selected random forest	0.813048	0.894048	0.88423
	Hyperparameter selected extra trees classifier	0.8328048	0.894655	0.888
6	Hyperparameter selected linear SVM run on best feature subset selected using F-test	0.8278	0.896389	0.896012
	Hyperparameter selected linear SVM run on best feature subset selected using mutual-information-test	0.80792	0.8896	0.88867
	Hyperparameter selected linear SVM run on best feature subset selected using RFE	0.84756	0.89753	0.899134
	K Nearest neighbors run with deafult parameters	0.7639	0.833959	0.8481658
	Hyperparameter selected KNN	0.79829	0.826397	0.8450444
	MLP with default parameters	0.84292	0.894446	0.8930052
7	Hyperparameter selected Linear SVM	0.84756	0.89753	0.89913

8	Hyperparameter selected Linear SVM run on best feature subset selected using the F-test	0.84756	0.89753	0.89913
	Hyperparameter selected Linear SVM run on best feature subset selected using the Mutual information test	0.8228	0.89753	0.89913
	Bagged decision tree with hyperparameter selected bagging parameters	0.74402	0.74547	0.7643768
9	GBT with default parameters	0.754268	0.863732	0.8455228
10	Bagged KNN with default bagging parameters	0.79317	0.8785502	0.8726475
	Bagged decision tree with hyperparameter selected bagging parameters	0.813048	0.85738	0.864297
11	MLP using hold out cross validation dataset to adjust the hidden layer sizes	0.84292	0.894446	0.8930052
12	GBT with early stopping	0.74926	0.8598156	0.84612668
	Adaboost classifier with default parameters	0.758658	0.8153026	0.8051948
13	Hyperparameter selected GBT	0.78829	0.848089	0.84422419
14	Adaboost with SVM (RBF kernel) as base estimator with default boosting parameters	0.813414	0.891518	0.8836409
15	Adaboost with SVM (RBF kernel) as base estimator with hyperparameter selected boosting parameters	0.813414	0.891518	0.8836409
16	Adaboost (with Decision tree as the base estimator) with default boosting parameters	0.749268	0.859368	0.832627
	Adaboost (with Decision tree as the base estimator) with hyperparameter selection of boosting parameters	0.749268	0.859368	0.832627
17	Voting classifier with Logistic Regression and SVM (with RBF kernel) as the base estimators	0.8228048	0.8999173	0.8955798

Validation accuracy and areas under curves on Cardiovascular Disease dataset over different days				
Day	**Algorithm**	**Validation accuracy**	**Area under PR**	**Area under AUC**
1	Logistic Regression with default parameters	0.723714	0.76851	0.7865167
	Linear SVM with default parameters	0.723785	0.7719484	0.7885721
2	Logistic regression (with default parameters) run on best feature subset selected using mutual_information test	0.723714	0.768510596	0.78651675
	Logistic regression (with default parameters) run on best feature subset selected using F-test	0.723714	0.768510596	0.78651675
3	Decision Trees with default parameters	0.633875	0.584521224	0.6338543
	Random forest with default parameters	0.700071799	0.7220255	0.75192896
4	Logistic regression (with default parameters) run on best feature subset selected using RFE	0.723714	0.768510596	0.78651675
	Hyperparameter selected decision trees	0.72516	0.74735396	0.77891727
	Extra Trees classifier with default parameters	0.6923212	0.714689946	0.747498768
5	Hyperparameter selected logistic regression	0.72216	0.766638	0.785086
	Hyperparameter selected Logistic regression run on best feature subset selected using RFE	0.72216	0.766638	0.785086
	Bagged decision tree with default tree and default bagging parameters	0.69457	0.71740559	0.746457
	Hyperparameter selected random forest	0.733285	0.782507669	0.799203889
	Hyperparameter selected extra trees classifier	0.724499714	0.76900928	0.786646167
6	Hyperparameter selected Logistic regression run on best feature subset selected using F-test	0.72216	0.766638	0.785086
	Hyperparameter selected Logistic regression run on best feature subset selected using mutual-information test	0.721767	0.7667635	0.78516308
	K nearest neighbors (KNN) with default parameters	0.6490004	0.6518057	0.6937506
	Hyperparameter selected KNN	0.6633037	0.695850793	0.725343673
	**Bagged KNN with default bagging parameters and hyperparameter selected KNN (using hold out cross validation)	0.672857143	0.617624076	0.672843752
7	Bagged decision tree with default bagging parameters and hyperparameter selected trees	0.72442867	0.7543878	0.78348809
	Bagged decision tree with Hyperparameter selected bagging parameters and hyperparameter selected trees	0.7365357	0.784403633	0.80085053

8	MLP with default parameters	0.7344998	0.7839049	0.80077518
9	KNN with n_neighbors selected by common practice	0.66307	0.7055441	0.721821787
	KNN with weights parameter 'uniform'	0.663303765	0.69585079	0.72534367
	KNN with weights parameter 'distance'	0.66719669	0.70235477	0.72515836
10	MLP trained with hidden layer sizes selected using common practice	0.730107	0.776824243	0.794522289
	Gradient Boosted Trees (GBT) with default parameters	0.73574998	0.786339793	0.802404022
11	**Using hold out cross validation dataset to adjust the hidden layer sizes in MLP	0.728571429	0.672313798	0.7285437
	GBT with hyperparameter search	0.73551789	0.785422854	0.801896237
12	Adaboost classifier with default parameters	0.7294466	0.7746921	0.79510308
13	Adaboost with decision tree as the base classifier	0.6798037	0.72085929	0.73191027
14	Adaboost with decision tree as the base classifier and hyperparameter selected of boosting parameters	0.733339338	0.779142077	0.797869636
	Voting classifier with Logistic Regression and Random Forest classifiers as base estimators	0.732178	0.779432729	0.7968886
15	**SVM with RBF Kernel (with default parameters) with PCA applied input data	0.735267857	0.6771019	0.735248314
	**SVM with Linear Kernel (with default parameters) with PCA applied input data	0.725803571	0.674904287	0.725755039
	Voting classifier with gradient Boosted trees and Adaboost classifier as the base estimators	0.73560714	0.785655199	0.802340698

B Descriptions and parameter settings of algorithms used by the student

Dataset-- Heart UCI

Training data size= 203 samples (108 1s and 95 0s)

Test data size = 100 samples (57 1s AND 43 0s)

k-fold cross validation on training data with k=5

Algorithm	Description	Package	Parameters
Logistic Regression	Logistic Regression with default parameters	sklearn.linear_model. LogisticRegression	LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	F-classif feature selection technique applied to select best feature subset (or k) by plotting k versus validation accuracies. Logistic regression with best feature subset and default parameters is applied to data	sklearn.feature_selection.SelectKBest sklearn.linear_model. LogisticRegression	SelectKBest(self, score_func=f_classif, k=10) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	mutual_info_classif feature selection technique applied to select best feature subset (or k) by plotting k versus validation accuracies. Logistic regression with best feature subset and	sklearn.feature_selection.SelectKBest sklearn.linear_model. LogisticRegression	SelectKBest(self, score_func=mutual_info_classif, k=12) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1,

	default parameters is applied to data		class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	Recursive Feature Elimination (RFE) feature selection technique applied to select best feature subset by plotting subset size versus validation accuracies. Logistic regression with best feature subset and default parameters is applied to data	sklearn.feature_selection.RFE sklearn.linear_model.LogisticRegression	RFE(self, estimator, n_features_to_select=12, step=1, verbose=0) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	Hyperparameter tuned logistic regression	sklearn.linear_model.LogisticRegression sklearn.model_selection.validation_curve	LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=3.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=15, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	Best hyperparameters applied to logistic regression with f-classif.	sklearn.feature_selection.SelectKBest sklearn.linear_model.LogisticRegression	SelectKBest(self, score_func=f_classif, k=13) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=3.0, fit_intercept=True, intercept_scaling=1,

			class_weight=None, random_state=None, solver='warn', max_iter=15, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	Best hyperparameters applied to logistic regression with mutual_info_classif.	sklearn.feature_selection.SelectKBest sklearn.linear_model.LogisticRegression	SelectKBest(self, score_func=mutual_info classif, k=12) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=3.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=15, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	Best hyperparameters applied to logistic regression with RFE.	sklearn.feature_selection.RFE sklearn.linear_model.LogisticRegression	RFE(self, estimator, n_features_to_select=11, step=1, verbose=0) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=3.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=15, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
Linear Support Vector Classifier	SVC with Linear kernel with default parameters	sklearn.svm.SVC	SVC(self, C=1.0, kernel='linear', degree=3, gamma='auto_deprecated', coef0=0.0,

			shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
	F-classif feature selection technique applied to select best feature subset (or k) by plotting k versus validation accuracies. Linear SVC with best feature subset and default parameters is applied to data	sklearn.feature_selection.SelectKBest sklearn.svm.SVC	SelectKBest(self, score_func=f_classif, k=19) SVC(self, C=1.0, kernel='linear', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
	mutual_info_classif feature selection technique applied to select best feature subset (or k) by plotting k versus validation accuracies. Linear SVC with best feature subset and default parameters is applied to data	sklearn.feature_selection.SelectKBest sklearn.linear_model.LogisticRegression	SelectKBest(self, score_func=f_classif, k=17) SVC(self, C=1.0, kernel='linear', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape

			='ovr', random_state=None)
	Recursive Feature Elimination (RFE) feature selection technique applied to select best feature subset by plotting subset size versus validation accuracies. Linear SVC with best feature subset and default parameters is applied to data	sklearn.feature_selection.RFE sklearn.svm.SVC	RFE(self, estimator, n_features_to_select=16, step=1, verbose=0) SVC(self, C=1.0, kernel='linear', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
	Hyperparameter tuned Linear SVC	sklearn.svm.SVC sklearn.model_selection.validation_curve	SVC(C=1.2, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='linear', max_iter=600, probability=1, random_state=None, shrinking=True, tol=0.001, verbose=False)
	Best hyperparameters applied to linear SVC with f-classif.	sklearn.feature_selection.SelectKBest sklearn.svm.SVC	SelectKBest(self, score_func=f_classif, k=19) SVC(C=1.2, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='linear',

			max_iter=600, probability=1, random_state=None, shrinking=True, tol=0.001, verbose=False)
	Best hyperparameters applied to linear SVC with mutual_info_classif.	sklearn.feature_selection.SelectKBest sklearn.svm.SVC	SelectKBest(self, score_func=mutual_info_classif, k=19) SVC(C=1.2, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='linear', max_iter=600, probability=1, random_state=None, shrinking=True, tol=0.001, verbose=False)
	Best hyperparameters applied to linear SVC with RFE.	sklearn.feature_selection.RFE sklearn.svm.SVC	RFE(self, estimator, n_features_to_select=18, step=1, verbose=0) SVC(self, C=1.2, kernel='linear', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=600, decision_function_shape='ovr', random_state=None)
SVC with RBF Kernel	SVC with RBF kernel with default parameters	sklearn.svm.SVC	SVC(self, C=1.0, kernel='rbf', degree=3, gamma='auto_deprecate

			d', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape ='ovr', random_state=None)
	F-classif feature selection technique applied to select best feature subset (or k) by plotting k versus validation accuracies. SVC (with RBF kernel) with best feature subset and default parameters is applied to data	sklearn.feature_selectio n.SelectKBest sklearn.svm.SVC	SelectKBest(self, score_func=f_classif, k=10) SVC(self, C=1.0, kernel='rbf', degree=3, gamma='auto_deprecate d', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape ='ovr', random_state=None)
	mutual_info_classif feature selection technique applied to select best feature subset (or k) by plotting k versus validation accuracies. SVC(with RBF kernel) with best feature subset and default parameters is applied to data	sklearn.feature_selectio n.SelectKBest sklearn.svm. SVC	SelectKBest(self, score_func=mutual_info _classif, k=10) SVC(self, C=1.0, kernel='rbf', degree=3, gamma='auto_deprecate d', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape

			='ovr', random_state=None)
	Hyperparameter tuned SVC with RBF kernel	sklearn.svm.SVC sklearn.model_selection.validation_curve	SVC(self, C=1.0, kernel='rbf', degree=3, gamma= 0.1, coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=300, decision_function_shape ='ovr', random_state=None)
	Best hyperparameters applied to SVC (RBF kernel) with f-classif.	sklearn.feature_selection.SelectKBest sklearn.svm.SVC	SelectKBest(self, score_func=f_classif, k=13) SVC(self, C=1.0, kernel='rbf', degree=3, gamma= 0.1, coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=300, decision_function_shape ='ovr', random_state=None)
	Best hyperparameters applied to SVC (RBF kernel) with mutual_info_classif.	sklearn.feature_selection.SelectKBest sklearn.svm.SVC	SelectKBest(self, score_func=mutual_info_classif, k=10) SVC(self, C=1.0, kernel='rbf', degree=3, gamma= 0.1, coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=300, decision_function_shape

			='ovr', random_state=None)
Decision Tree classifier	Decision Tree classifier with default parameters	sklearn.tree.DecisionTreeClassifier	DecisionTreeClassifier(self, criterion="gini", splitter="best", max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0., max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0., min_impurity_split=None, class_weight=None, presort=False)
	Hyperparameter selected decision tree	sklearn.model_selection.validation_curve sklearn.tree.DecisionTreeClassifier	DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0,min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')
	Bagged decision tree with default tree and default bagging parameters	sklearn.ensemble.BaggingClassifier sklearn.tree.DecisionTreeClassifier	BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2,

			min_weight_fraction_leaf=0.0, presort=False, random_state=None,splitter='best'),bootstrap=True,bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10,n_jobs=None,oob_score=False,random_state=None, verbose=0,warm_start=False)
	Bagged decision tree with default bagging parameters and hyperparameter selected decision tree	sklearn.ensemble.BaggingClassifier sklearn.tree.DecisionTreeClassifier	BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0,min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None,splitter='best'),bootstrap=True,bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
	Bagged decision tree with hyperparameter selected bagging parameters and hyperparameter selected Decision Tree	sklearn.ensemble.BaggingClassifier sklearn.tree.DecisionTreeClassifier	BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease

			=0.0,min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'), bootstrap=True, bootstrap_features=False, max_features=7, max_samples=0.8, n_estimators=10, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
Random Forest Classifier	Random forest classifier with default parameters	sklearn.ensemble.RandomForestClassifier	RandomForestClassifier (bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
	Hyperparameter selected random forest classifier	sklearn.model_selection.validation_curve sklearn.ensemble.RandomForestClassifier	RandomForestClassifier (bootstrap=True, class_weight=None, criterion='gini', max_depth=3, max_features='auto', max_leaf_nodes=None,

			min_impurity_decrease=0.0, min_impurity_split=None,min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0,n_estimators=55, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
Extra Trees classifier	Extra Trees classifier with default parameters	sklearn.ensemble import ExtraTreesClassifier	ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
	Extra trees classifier with hyperparameter selection	sklearn.model_selection.validation_curve sklearn.ensemble import ExtraTreesClassifier	ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini', max_depth=5, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2,

			min_weight_fraction_leaf=0.0, n_estimators=55, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
K nearest neighbors classifier	KNN classifier with default parameters	sklearn.neighbors. KNeighborsClassifier	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=5, p=2, weights='uniform')
	Hyperparameter selected KNN	sklearn.model_selection .validation_curve sklearn.neighbors. KNeighborsClassifier	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=3, p=1, weights='uniform')
	Bagged hyperparameter selected KNN	sklearn.ensemble.BaggingClassifier sklearn.neighbors. KNeighborsClassifier	BaggingClassifier(base_estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=3, p=1, weights='uniform'), bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
MLP Classifier	MLP with default parameters	sklearn.neural_network. MLPClassifier	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,

			beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)
	Using hold out cross validation dataset to adjust the hidden layer sizes and max_iterations in MLP	sklearn.neural_network.MLPClassifier	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)
Gradient Boosted Trees (GBT)	GBT with default parameters	sklearn.ensemble.GradientBoostingClassifier	GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.1,

			loss='deviance', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)
	GBT with early stopping (to counter overfitting)	sklearn.ensemble.GradientBoostingClassifier	GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.1, loss='deviance', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, n_iter_no_change=5, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.2, verbose=0, warm_start=False)

	GBT with hyperparameter search	sklearn.model_selection.validation_curve sklearn.ensemble.GradientBoostingClassifier	GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.1, loss='deviance', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=37, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)
Adaboost classifier	Adaboost classifier with default parameters	sklearn.ensemble.AdaBoostClassifier	AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=None)
	Boosting RBF SVM with Adaboost	sklearn.ensemble.AdaBoostClassifier	AdaBoostClassifier(algorithm='SAMME.R', base_estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf', max_iter=300, probability=1, random_state=None, shrinking=True, tol=0.001, verbose=False),

			learning_rate=1.0, n_estimators=50, random_state=None)
	Boosting decision trees with AdaBoost	sklearn.ensemble.AdaBoostClassifier	AdaBoostClassifier(algorithm='SAMME.R', base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'), learning_rate=1.0, n_estimators=50, random_state=None)
Voting classifier	Voting classifier with the Logistic Regression and Support vector classifier as estimators	sklearn.ensemble.VotingClassifier sklearn.linear_model.LogisticRegression sklearn.svm.SVC	VotingClassifier(estimators=[('logistic', LogisticRegression(C=3, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=15, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False)), ('svm', SVC(...bf', max_iter=100, probability=1, random_state=None, shrinking=True, tol=0.001, verbose=False))],

			flatten_transform=None, n_jobs=None, voting='soft', weights=None)

Cardiovascular disease dataset

56,000 training samples (27983 1s and 28017 0s)
14,000 testing samples (6996 1s and 7004 0s)

k-fold cross validation (with k=10) used and when computational complexity is high (too much time for evaluation), hold-out cross validation is used.

Algorithm	**Description**	**Package**	**Parameters**
Logistic Regression	Logistic Regression with default parameters	sklearn.linear_model. LogisticRegression	LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	F-classif feature selection technique applied to select best feature subset (or k) by plotting k versus validation accuracies. Logistic regression with best feature subset and default parameters is applied to data	sklearn.feature_selection.SelectKBest sklearn.linear_model. LogisticRegression	SelectKBest(self, score_func=f_classif, k=11) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)

	mutual_info_classif feature selection technique applied to select best feature subset (or k) by plotting k versus validation accuracies. Logistic regression with best feature subset and default parameters is applied to data	sklearn.feature_selection.SelectKBest sklearn.linear_model.LogisticRegression	SelectKBest(self, score_func=mutual_info_classif, k=11) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	Recursive Feature Elimination (RFE) feature selection technique applied to select best feature subset by plotting subset size versus validation accuracies. Logistic regression with best feature subset and default parameters is applied to data	sklearn.feature_selection.RFE sklearn.linear_model.LogisticRegression	RFE(self, estimator, n_features_to_select=11, step=1, verbose=0) LogisticRegression(self, penalty='l2', dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
	Hyperparameter tuned logistic regression	sklearn.linear_model.LogisticRegression sklearn.model_selection.validation_curve	LogisticRegression(C=0.2, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=7, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001,

			verbose=0, warm_start=False)
	Best hyperparameters applied to logistic regression with f-classif.	sklearn.feature_selection.SelectKBest sklearn.linear_model.LogisticRegression	SelectKBest(self, score_func=f_classif, k=11) LogisticRegression(C=0.2, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=7, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False)
	Best hyperparameters applied to logistic regression with mutual_info_classif.	sklearn.feature_selection.SelectKBest sklearn.linear_model.LogisticRegression	SelectKBest(self, score_func=mutual_info classif, k=10) LogisticRegression(C=0.2, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=7, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False)
	Best hyperparameters applied to logistic regression with RFE.	sklearn.feature_selection.RFE sklearn.linear_model.LogisticRegression	RFE(self, estimator, n_features_to_select=9, step=1, verbose=0) LogisticRegression(C=0.2, class_weight=None, dual=False, fit_intercept=True,

			intercept_scaling=1, max_iter=7, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False)
Support Vector Machine	Support Vector Classifier with Linear kernel and default parameters	sklearn.svm.SVC	SVC(self, C=1.0, kernel='linear', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
	Support Vector Classifier with Linear kernel and Principal component analysis (PCA) applied on input	sklearn.decomposition.PCA sklearn.svm.SVC	PCA(copy=True, iterated_power='auto', n_components='mle', random_state=None, svd_solver='full', tol=0.0, whiten=False) SVC(self, C=1.0, kernel='linear', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shap

			e='ovr', random_state=None)
	Support Vector Classifier with RBF kernel and Principal component analysis (PCA) applied on input	sklearn.decomposition.PCA sklearn.svm.SVC	PCA(copy=True, iterated_power='auto', n_components='mle', random_state=None, svd_solver='full', tol=0.0, whiten=False) SVC(self, C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
Decision Tree classifier	Decision Tree classifier with default parameters	sklearn.tree.DecisionTreeClassifier	DecisionTreeClassifier(self, criterion="gini", splitter="best", max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0., max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0., min_impurity_split=None, class_weight=None, presort=False)
	Hyperparameter selected decision tree	sklearn.model_selection.validation_curve sklearn.tree.DecisionTreeClassifier	DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease

			=0.0, min_impurity_split=1e-06, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=0, splitter='best')
	Bagged decision tree with default tree and default bagging parameters	sklearn.ensemble.BaggingClassifier sklearn.tree.DecisionTreeClassifier	BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0,min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'), bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
	Bagged decision tree with default bagging parameters and hyperparameter selected decision tree	sklearn.ensemble.BaggingClassifier sklearn.tree.DecisionTreeClassifier	BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=1e-06,

			min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=0, splitter='best'), bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
	Bagged decision tree with hyperparameter selected bagging parameters and hyperparameter selected Decision Tree	sklearn.ensemble.BaggingClassifier sklearn.tree.DecisionTreeClassifier	BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=9, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'), bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=30, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
Random Forest Classifier	Random forest classifier with default parameters	sklearn.ensemble.RandomForestClassifier	RandomForestClassifier(bootstrap=True,

			class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
	Hyperparameter selected random forest classifier	sklearn.model_selection.validation_curve sklearn.ensemble.RandomForestClassifier	RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=7, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=69, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
Extra Trees classifier	Extra Trees classifier with default parameters	sklearn.ensemble import ExtraTreesClassifier	ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None,

			min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
	Extra trees classifier with hyperparameter selection	sklearn.model_selection.validation_curve sklearn.ensemble import ExtraTreesClassifier	ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini', max_depth=25, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
K nearest neighbors classifier	KNN classifier with default parameters	sklearn.neighbors.KNeighborsClassifier	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=5, p=2, weights='uniform')
	Hyperparameter selected KNN	sklearn.neighbors.KNeighborsClassifier	KNeighborsClassifier(algorithm='auto', leaf_size=30,

			metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=20, p=2, weights='uniform')
	Hyperparameter selected KNN with weights=”uniform”	sklearn.neighbors. KNeighborsClassifier	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=20, p=2, weights='uniform')
	Hyperparameter selected KNN with weights=”distance”	sklearn.neighbors. KNeighborsClassifier	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=20, p=2, weights='distance')
	Bagged hyperparameter selected KNN	sklearn.ensemble.BaggingClassifier sklearn.neighbors. KNeighborsClassifier	BaggingClassifier(base_estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=20, p=2, weights='uniform'), bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0, n_estimators=10, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)
	KNN with n_neighbors selected by common practice	sklearn.neighbors. KNeighborsClassifier	KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=237, p=2,

			weights='uniform')
MLP Classifier	MLP with default parameters	sklearn.neural_network.MLPClassifier	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9,n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)
	MLP trained with hidden layer sizes selected using common practice	sklearn.neural_network.MLPClassifier	MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(6), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)
	Using hold out cross validation dataset to	sklearn.neural_network.MLPClassifier	MLPClassifier(activation='relu', alpha=0.0001,

	adjust the hidden layer sizes in MLP		batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(35, 35, 35), learning_rate='constant', learning_rate_init=0.001, max_iter=1000, momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)
Gradient Boosted Trees (GBT)	GBT with default parameters	sklearn.ensemble.GradientBoostingClassifier	GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.1, loss='deviance', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=100, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)

	Hyperparameter selected GBT	sklearn.ensemble.GradientBoostingClassifier	GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.1, loss='deviance', max_depth=5, max_features=6, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=50, n_iter_no_change=None, presort='auto', random_state=0, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)
Adaboost classifier	Adaboost classifier with default parameters	sklearn.ensemble.AdaBoostClassifier	AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=None)
	Adaboost with decision tree as the base classifier	sklearn.ensemble.AdaBoostClassifier	AdaBoostClassifier(algorithm='SAMME.R', base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2,

			min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'), learning_rate=1.0, n_estimators=50, random_state=None)
	Adaboost with decision tree as the base classifier and hyperparameter selected boosting parameters	sklearn.ensemble.AdaBoostClassifier	AdaBoostClassifier(algorithm='SAMME.R', base_estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=1e-06, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=0, splitter='best'),learning_rate=1.0, n_estimators=50, random_state=None)
Voting classifier	Voting classifiers with Logistic regression and RandomForest as estimators	sklearn.ensemble.VotingClassifier sklearn.linear_model.LogisticRegression sklearn.ensemble.RandomForestClassifier	VotingClassifier(estimators=[('lr', LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False)), ('rf', RandomForestClassifier

			(bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)], flatten_transform=None, n_jobs=None, voting='soft', weights=None)
	Voting classifiers with Gradient Boosting Classifier and Adaboost classifier as estimators	sklearn.ensemble.VotingClassifier sklearn.ensemble.GradientBoostingClassifier sklearn.ensemble.AdaBoostClassifier	VotingClassifier(estimators=[('gbt', GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.1, loss='deviance', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1,...m='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=None))], flatten_transform=None, n_jobs=None, voting='soft', weights=None)