CS3099 Junior Honours Software Project

Final Report

Machine Learning Team 7

Kira Kim
Tom Oliver
Yizhe Sun
Victor Vasilyev
James Moran
Tianyu (Tom) He

University of
St Andrews

April 23, 2018

Abstract

The aim of this year's Junior Honours project was to create an online system for the management, distribution and analysis of pathology data for use by the University Medical department. The year had been split into 3 main groups with each one tackling a different aspect of the problem (backend, frontend and a machine learning server). A combination of the three components from any teams from the three main groups could, in theory, connect together to form a fully working system. To that end, emphasis has been put on collaboration and class meetings to discuss and define a common protocol for inter-group communication. As one of machine learning groups, we have made progress via development of refined design and implementation over time and enhanced the quality of the implementation according to the user stories for effective analysis application and satisfactory user experience.

Declaration

We declare that the material submitted for assessment is our own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 10,117 words long, including project specification and plan. In submitting this project report to the University of St Andrews, we give permission for it to be made available for use in accordance with the regulations of the University Library. We also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis. We retain the copyright in this work, and ownership of any resulting intellectual property.

Contents

1 Introduction

In the era of big data, data analysis has started to take an integral part in research for various industry sectors. Medical data science applies big data analysis techniques for finding relationships in data gathered from patients. Advancements made in data analytic and machine learning techniques has accelerated research in this field.

In the light of such context, the JH project is centered around building a system which can support the analysis of data. There are two problems we had to solve: providing the functionality of the machine learning algorithms required in the specification, and integrating the machine learning components within a web server following a protocol agreed upon by the year group.

With the above objectives, we have implemented basic functionalities, along with the implementation of advanced specification devised by the consultation with the product owner. Our active inter- and intra- group collaborations helped with continuous design development and with connecting with other groups to produce a final application.

2 Key Concepts

Before starting our project we initially conducted in-depth research of machine learning methodology and algorithms. Thorough research we have undertaken during the first semester has allowed us to devise an intuitive design and efficient implementation over time. The following summarises our understanding of the key concepts which we have researched and implemented.

2.1 Supervised Learning

Supervised learning is to find a function that maps an input to an output based on example input-output pairs. It infers a function from labelled training data consisting of a set of training examples. A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. Supervised learning problems can be further grouped into regression and classification problems.

2.1.1 Classification

Classification predictive modelling is the task of approximating a mapping function f from input variables X to discrete output variables Y. The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation. For this project, the classification algorithms we have used include: Naive Bayes Classifier (three variants), Support Vector Machines Classifiers, and Random Forest Classifiers.

Random Forests

Random forest is an example of ensemble learning, where multiple decision trees constitute a forest. This technique is applicable for both classification and regression, for discrete and continuous target variables respectively.

Every decision tree is built via random selection of samples and random selection of features. These are then formulated as a series of questions; each narrows down the number of samples applicable, for the ultimate discrimination of samples according to their belonged subsets. Every question is chosen with a variable, to be optimal on the level to divide the subsets to the target values. Ultimately, leaf nodes of top-down decision trees will yield predicted target values.

Random Forest Classifiers

For classification, each tree will classify each observation into its predicted target class; an ultimate decision of its classified target class will be upon the one with the majority across decision trees; for the maximum probability of achieving a correct classification.

The process of tree construction is undertaken along with continuous measurement of the quality of the split. For classification, the metric used is the Gini index. It computes the probability of an element having an incorrect labelling amongst randomly labelled elements of a set, according to the underlying distribution of such allocation.

Support Vector Classifiers

To linearly separate binary data, support vector classifiers work by finding the hyperplane (a $n-1$ subspace, for 2d it would be a line, 3d a plane) that separates the points of both classes with the widest margin. To find the hyperplane the SVC finds the closest points to the hyperplane (support vectors), draws a connecting line between the midpoints of the support vectors of both classes, and uses the line that bisects and is perpendicular to the connecting line as the separator. Depending on the type of data used, separation of points can be done linearly or using techniques such as kernel trick: mapping to a higher dimension.

It may be desirable to classify points into more than 2 classes. There are two methods to do this, either classifying each class against the rest of the points (one vs. rest), or classifying each class against every other class (one vs. one) and then combining that information. One vs. rest needs fewer classifications and is more sensitive to class imbalances in comparison to one vs. one.

Naive Bayes Classifiers

Naive Bayes Classifiers exploit Bayes' theorem:

$$P(H \mid E) = \frac{P(E \mid H) \times P(H)}{P(E)}$$

Bayes' theorem allows to calculate the probability a hypothesis H is true given some event E. This is also known as the "posterior" probability. $P(H)$ is known as the "prior", the probability of the hypothesis being true before the event occurring. Depending on the input data, we may want to use different versions of the classifier. Gaussian Naive Bayes assumes a Gaussian distribution of the data enabling use on datasets containing continuous attributes (e.g core body temperature). Multinomial Naive Bayes is suited to datasets containing discrete features (e.g occurrence counts of some medical observation). Bernoulli Naive Bayes is best suited to classifying data sets with boolean features (e.g has necrosis happened?).

2.1.2 Regression

Regression predictive modelling is the task of approximating a mapping function (f) from input variables (X) to a continuous output variable (y). A continuous output variable is a real-value, such as an integer or floating point value. These are often quantities, such as amounts and sizes.

Random Forest Regressors

For regression, an overall output of random forest of decision trees will be the mean prediction value amongst the trees. The value calculated for every split is the Mean Squared Error (MSE) that denotes the expected value of error or loss resulting in the split. Given total n number of samples, if $\hat{y_i}$ and y_i denote the predicted value of the ith sample and its true value respectively, the following formula is used.

$$MSE(y, \hat{y_i}) = \sum_{i=1}^{n-1} (y_i - \hat{y_i})^2$$

This is analogous to Gini index but under continuous variable setting and the decrease in the error value is notable down the tree path.

The random forest algorithms can produce graphical output such as feature importance bar plots and visualisations of decision trees constructed at training. The feature importance plot is useful for having a general blueprint of individual contributions of features and to what extent they affect the overall decision.

Random forests generate decision trees (10 by default, can be modified as part of the advancedTraining endpoint) with features such as questions asked for every split. class or value predicted at the leaf nodes. Users can notice which random features are selected and the quality of split for each question. The current implementation of tree construction has no limit on the maximum depth; this is negligible for sufficiently enough number of samples for classification while, for regression, however, it results in relatively large trees.

Support Vector Machine Regressors

Support Vector Regressors work in a similar way to other regressors - they minimise a cost function in order to find the line of best fit. The difference with a support vector regressor is that it is able to use the kernel trick and fit the points to a curve rather than a more simple linear solution which allows for potentially more accurate predictions.

2.2 Survival Analysis

Kaplan-Meier Estimates

Often in the field of medicine, a researcher may like to know the probability of an event occurring at or before a certain time, such as time until death or time until drug relapse. To accommodate this we can employ different techniques, such as the Kaplan-Meier estimate. The Kaplan-Meier algorithm allows us to estimate the survival function $S(t)$ that gives the probability that an event has not occurred yet at time t. It takes the form of

$$S(t) = exp(-\int_0^t \lambda(z)dz)$$

where $\lambda(z)$ is the hazard function:

$$\lambda(t) = \lim_{\delta t \to 0} \frac{Pr(t \leq T \leq t + \delta t | T > t)}{\delta t}$$

that defines how likely it is that the event would occur at time t given that it has not occurred before.

The survival function and, more specifically, the resulting graph can give the researcher information about the nature of the hazard and comparing the Kaplan-Meier curves of multiple data-sets (e.g. a patient that has been given a drug and a patient that has been given a placebo) can give insight into whether the variable has any effect on the survival of the subjects.

Cox Regression

The Kaplan-Meier estimate, although a good technique, does not account for the differences between subjects and so treats all subjects equally and assigns every one of them the same hazard function. We may want to research the effect of a particular variable on the survival probability curve of a subject without having seen one like it before - e.g. if we're looking to predict how long it takes for a subject to relapse after recovering, but we have never observed a patient with that combination of sex, age and blood type before. The Cox model allows us to do just that by assigning every covariate x_i an index b_i based on the observed subjects such that

$$\lambda(t|X) = b_0(t)exp(\sum_{i=1}^{|X|} b_i x_i)$$

The above formula implies that a hazard function is obtained by multiplying a base hazard function b_0 that is independent of any covariates by a constant factor associated with every possible combination of covariate values, calculated using the factors assigned to covariates by the trained model.

2.3 Performance Metrics

Including to the desired performance metrics requested by the product owner, some additional metrics are researched.

Receiver Operating Characteristic Curves

The Receiver Operating Characteristic Curve (ROC Curve) is a graphical representation of the trade-off between sensitivity and specificity. True positive rates ($Sensitivity$) are plotted against false positive rates ($1 - Specificity$). Specificity and Sensitivity are given by:

$$1 - Specificity = \frac{F_p}{F_p + T_n}$$

$$Sensitivity = \frac{T_p}{T_p + F_n}$$

Using T_P for true positives, F_p for false positives, T_n for true negatives, and F_n for false negatives.

ROC Curves are useful in assessing the ability to discriminate targets. The curve displays values in each threshold which constitute of a point that yields an optimal value of sensitivity with the lowest possible corresponding false positive rate. A numerical measure of the quality of an ROC curve is the area under the curve defined within the range 0 to 1, where higher the value the more accurate the model is.

Since two graphs for tests with the same area under the curve value may differ in shape, the user may then decide which analysis technique to use according to their objective such as attaining high sensitivity, high specificity etc.

Precision and Recall Curve

Precision is the proportion of values that have been correctly classified as being in the positive class and is given as:

$$Precision = \frac{T_p}{T_p + F_p}$$

Where T_P represents true positives, F_p represents false positives and F_n represents false negatives.

Recall is the proportion of true positive values that have been correctly classified and is given as:

$$Recall = \frac{T_p}{T_p + F_n}$$

The Precision-Recall Curve shows the trade-off between the two factors at different thresholds. High precision indicates a low rate of false positives while high recall indicates a low false negative rate. Ideally, a large area under the curve is desirable - with high recall and high precision the classifier will identify most, if not all, of the positive results and the results will be accurate.

In the context of the medical data used for this project, if a user is trying to make a cancer diagnosis, very high recall is desirable as a false negative could be deadly for the patient. False positives, while may result in wasted resources and distress for the patient, are less likely to result in a deadly scenario and so are less important.

For multi-class data it is possible to use micro-averaged or macro-averaged performance as a way to assess how the classifier performed over all classes. Macro-averaged performance simply averages the precision and recall calculated for each class. Micro-averaged performance sums the individual true positives, false positives and false negatives and calculates the precision and recall based on the combined values. As we don't know the distribution of the data we receive we have to assume the classes could be very biased. For this reason, the micro-averaged curve is preferable as it is not affected as much by a class with more extreme precision and recall values.

A common plot shows that as recall increases, precision generally decreases. The AP (average precision) score gives you an indication of how well the classifier performs and can be used to compare algorithms.

With our limited data and imbalanced classes, the precision indicated by the plot we produced is sub-optimal at any level of recall and the trade-off would have to be considered when deciding on a recall threshold.

After implementing the curves as part of specification requirements, we went beyond the scope by implementing tools to illustrate how accurately models make predictions on previously unseen test sets.

Confusion Matrix

The confusion matrix is used to display the output generated by classification models. The desired output is an identity matrix, values to be 1 on diagonal entries and 0 elsewhere; this indicates that all samples are correctly predicted. In general, the values for every entry are normalized to be part of the range $[0, 1]$, so that performance can be compared relative to different classes. Models have shown varying competency in successful predictions. While we can see that models have generally performed better on balanced sets, they have struggled with those that are imbalanced. Introducing this tool helped with identifying areas of improvement for each algorithm which could be implemented in the next iteration.

Regression Plots

To test our regression algorithms we plotted the predicted values against the true values. This provides a visual representation of the output which can be used with the r^2 value to assess the performance of the algorithm. The best possible performance would be to make zero errors which would exhibit a line of best fit indistinguishable from $y = x$. These plots are analogous to confusion matrices.

2.4 Preprocessing

Standard Scaler

Standard scaler standardizes features by removing the mean and scaling to unit variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature does not more or less look like standard normally distributed data (e.g., Gaussian with 0 mean and unit variance). In this project, we bind the standard scaler and support vector machine estimators for in the testing.

Robust Scaler

Robust Scaler scales features using statistics that are robust to outliers. It removes the median and scales the data according to the quantile range (defaults to interquartile range, from 25th quantile to 75th quantile) The centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Median and interquartile range are then stored to be used for later data using the transform method.

Standardization of a dataset is a common requirement for many machine learning estimators. Typically this can be achieved by using standard scaler to remove the mean and scaling to unit variance. However, often the outliers can influence the sample mean/variance negatively. In these scenarios the median and the interquartile range give better results.

Min Max Scaler

Min-Max scaler transforms features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, i.e., between zero and one. The motivation to use this scaling includes robustness to very small standard deviations of features and preserving zero entries in sparse data.

Max Abs Scaler

Max-Abs scaler was specifically designed for scaling sparse data. It can also achieve the same functionality as Min-Max scaler for scaling features so that the maximum absolute value of each feature is scaled to unit size.

Variance Threshold

Variance threshold is a feature selection algorithm for removing all low-variance features, and therefore improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.

Select K Best

Select k best is a univariate feature selection; it works by selecting the best features based on univariate statistical tests. Select k best, as its name suggests, selects features according to the k highest scores. There are wide ranges of choices of the score for the selection, which include ANOVA F-value between label/feature, chi-squared stats of non-negative features for classification tasks, etc.,

3 Design and Implementation

In this section we will discuss our design and implementation decisions for both the basic specification and our own extended design

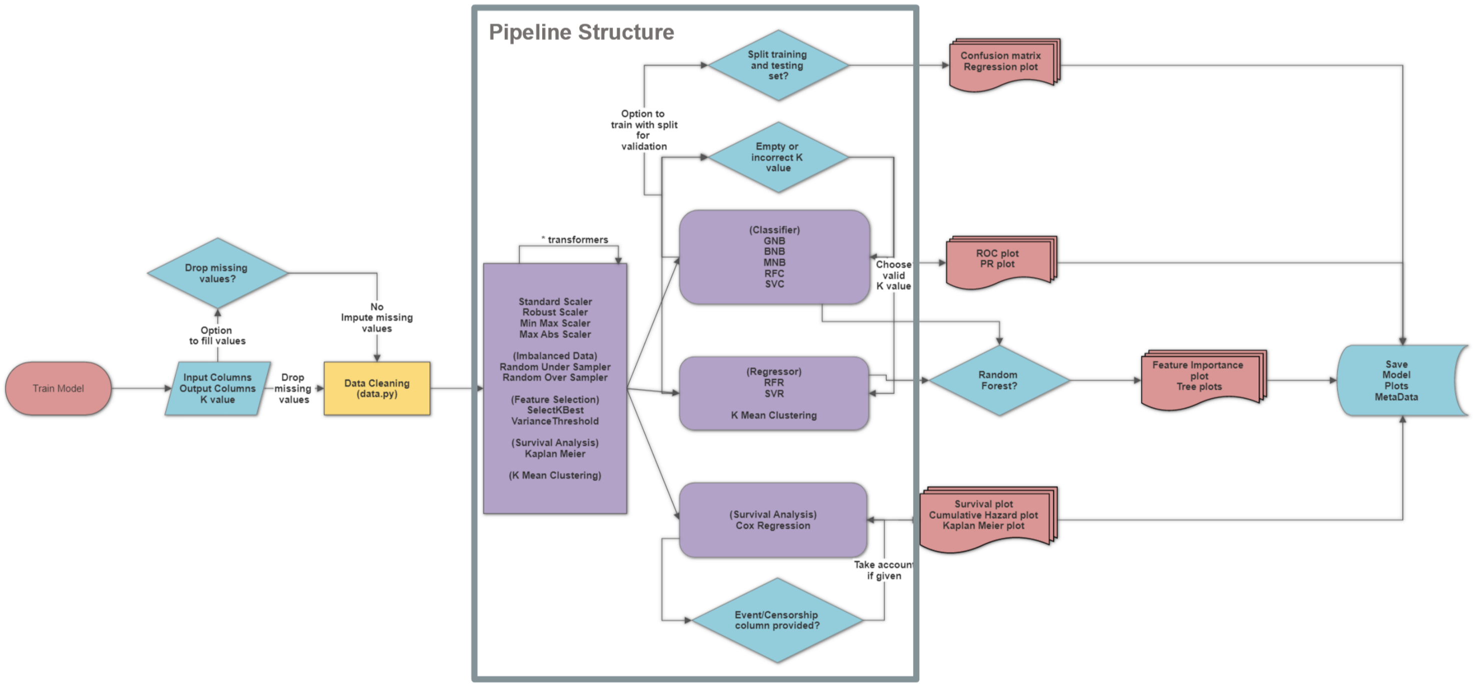


Figure 1: Diagram showing Pipeline Design Structure for Training

3.1 Design

Figure 1 above shows the data flows in a model training request as an example. When designing our project we made it as extendable as possible in order to give the user a large amount of control as we have made progress towards extensions. At every stage the user is able to specify how to treat the data if they wish to but we also still providing sensible defaults. Ultimately, the design we have derived is not limited to fulfil the basic specification, but applicable to advanced computations according to the user options.

3.1.1 Basic Specification Design

In order to adhere to the basic specification provided in appendix B, we have narrowed down the scope of user decisions and expanded the application of automated processes. Every request containing some data for training, suggestion, and prediction will begin with data cleaning. Data cleaning will entail formatting data for training: dealing with missing values by dropping the samples as appropriate, handling nominal, ordinal, and numerical data accordingly. It is ensured that data formatting is done equally for data used for training and input for prediction. The user will define an algorithm to build a model and a k value for cross validation.

Training will include fitting under k-fold cross validation followed by the fit of data as a whole. A model will be built according to the hyperparameter values set as defaults. Any standard preprocessing steps required before training will be applied to the data internally. During training, the accuracy score is computed and any necessary graphs will be plotted. The plots will be exported and displayed to the user to assess output. Any statistics will be displayed to the user as metadata.

3.1.2 Advanced Specification Design

After the implementation of standard functionalities, we have received further requests from the product owner. We have decided to create an extended version of our application fulfilling the standard specification to enable while equipped with some additional functionalities to perform more complex data analysis. For a full list of these, see appendix C.

By following the diagram shown in 1, the decision questions displayed denote the user options when it comes to creating a model for training. Any number of transformations can be applied to the data followed by the final estimator. For example, imputer $\rightarrow$ Kaplan Meier estimate $\rightarrow$ Cox regression. In this case the Imputer transformer step allows missing values in a column to be estimated rather than dropping the sample altogether.

The output from the imputer step now becomes the input for the Kaplan-Meier estimation. Finally the Cox regression algorithm is trained on output data from the last intermediate step. Each transformer and estimator step can be combined with hyper-parameters enabling the user to experiment and fine tune the system. This functionality is aimed at the experienced data scientist who would otherwise feel constrained by the offers presented by the default jobs.

The user is also able to specify the test/train split ratio performed on the data. This means a portion of the data can be set aside for testing which can be used to assess performance of the model. This is visualised by confusion matrix for classifier algorithms and regression plots for regression algorithms.

3.2 Implementation

We decided to use Python for this project due to its abundance of libraries and widespread adoption within the machine learning sphere. Libraries available include scikit-learn and scikit-survival which both have a strong community and numerous tutorials. Libraries such as flask make creating restful APIs very simple.

Our initial implementation followed an object-oriented design centered around algorithms, as shown in appendix D. Each algorithm is dealt with as a class with a hierarchical structure. A common set of attributes and methods are inherited. We have considered minimal code redundancy and good inheritance structure as priorities, along with a good representation of semantics of machine learning algorithms.

Over time, the implementation has changed according to design shown in 1, which was a decision made after reading through much of the scikit-learn documentation. With a motivation to improve performance of a model created, implementation of intermediate steps before building the final estimator was sought. This can be satisfied by a pipeline structure. Every training request will integrate all the steps chosen, creating an object that can be used as a single estimator. This will prevent any information leakage and guarantee that all preprocessing steps are applied correctly. By the inter-linkage, a model will be uniformly defined as a pipeline, regardless of the choice of estimator. This abstraction of algorithms to a pipeline meant flexibility in threading transformers and efficiency in training performance.

3.3 Advanced Specification Implementation

3.3.1 Data Preprocessing for Imbalanced Data

As suggested by the Product Owner, most of the medical data we will be receiving is imbalanced. Therefore it is crucial to have preprocessing steps for making the dataset more balanced to ensure the performance of our classification algorithms. In this project, we introduced three approaches to resolve this problem: over-sampling, under-sampling, and a combination of those two.

For over-sampling, we have implemented a naive strategy using random over sampler which generates new samples by randomly sampling with replacement of the currently available samples. Moreover, we have also introduced two popular methods to over-sample minority classes: (i) Synthetic Minority Oversampling Technique (SMOTe) and (ii) Adaptive Synthetic (AdaSyn) sampling method. These algorithms can be used in the same manner. SMOTe might connect inliers and outliers while AdaSyn might focus solely on outliers which, in both cases, might lead to a sub-optimal decision function. In this regard, SMOTe offers three additional options to generate samples. Those methods focus on samples near of the border of the optimal decision function and will generate samples in the opposite direction of the nearest neighbours class.

For under-sampling, there are two different types of algorithms: prototype generation and prototype selection. For the prototype generation algorithms, given an original data set S, it will generate a new set S' where $|S'| < |S|$ and $S' \notin S$. In other words, the prototype generation technique will reduce the number of samples in the targeted classes, while the remaining samples are generated and not selected from the original set. The prototype generation algorithm we have implemented in this project is *cluster centroids*. It performs under-sampling by generating centroids based on clustering methods, offering an efficient way to represent the data cluster with a reduced number of samples. On the contrary to prototype generation algorithms, prototype selection algorithms will select samples from the original set S. Therefore, S' is defined such as $|S'| < |S|$ and $S' \in S$. The prototype generation algorithm we have implemented in this project is the *random under sampler*, which is a fast and easy way to balance the data by randomly selecting a subset of data for the targeted classes.

For the combination of over-sampling and under-sampling, when users use SMOTe for generating samples (since SMOTe does not have any knowledge regarding the underlying distribution), some noisy samples can be generated. Hence, it can be beneficial to apply an under-sampling algorithm to clean the noisy samples. Two methods are usually used in the literature: (i) Tomek's link and (ii) edited nearest neighbours cleaning methods. Imbalanced-learn provides two ready-to-use samplers SMOTETomek and SMOTEENN. In general, SMOTEENN cleans more noisy data than SMOTETomek. In this project, we have implemented these two samplers.

3.3.2 Neural Networks for Image Analysis

The basic neural networks are made up of layers of interconnected nodes containing an activation function that fire depending on different values of the previous layer. Patterns are encoded into the input layer which connects to a series of hidden layers before connecting to the output layer where the result is output. Each connection has a weight which determines how much a previous node affects the next node. Additionally, each node has a bias that determines what level of input will result in the node activation. These are updated during the training phase trying to model the function that maps the inputs to the correct outputs.

In our project we used neural networks for image analysis. Traditionally neural networks require a large amount of data in order to be effective. Unfortunately, often when dealing with medical data, only a limited number of examples are available. To deal with this we used a technique called transfer learning. Although transfer learning as a concept has been known since the mid 90's it has only received widespread attention relatively recently. Transfer learning uses a network that has been pre-trained with a large amount of data from a similar domain. It adds a few layers to the network and trains them for a few epochs to recognise the new categories. Depending on the problem, it then trains the whole neural network to recognise the new categories.

We were unable to find a network that had been pre-trained on medical data. Despite this we pressed on and tried the technique anyway. We used a pre-trained network called Resnet34. Resnet was trained by Microsoft in 2015 and was the first deep neural network to pass human performance on image recognition in the competition ILSVRC. This was trained on images you would traditionally take pictures of, for example animals and household objects, and so was not ideally suited to medical images which are mainly microscope images.

The package we used added 2 layers to the architecture which we trained for 3 epochs. Then, knowing that the model was large and not trained on the domain that it was being applied to, we trained the whole network for 31 epochs. Throughout we used cyclic cosine annealing, an interesting, relatively new approach to cycle the learning rate throughout training in order to find a stable minimum. We also used different learning rates for different layers, such as a lower learning rate for lower layer as those generally are finding basic shapes and patterns that should be consistent across both domains.

We found that this technique worked very well, improving the final score of the validation test from about 50% to just over 90% when training on the same architecture and with the same training method but instead using the pre-trained model. The training loss was still higher than the validation loss indicating that there had not been over-fitting.

We were not provided with any labelled medical data and so we found some images of different type of labelled blood cells at
https://www.kaggle.com/paultimothymooney/blood-cells
It contains images of 410 blood cells, although each image has been augmented to create 12500 images (around 3000 for each class). We only realised too late that this did mean that some the validation data *may* have been augmented images of data used in the training set which could make the results too optimistic.

Nevertheless, the low number of images used and the high accuracy obtained in their classification compared to model that had not been pre-trained shows that transfer learning lends itself well to medical images and would hopefully be useful for anybody using our took with a limited number of patent images. With future development this could provide a powerful tool to help doctors make diagnosis. Additionally when we tested the implementation on images closer to the original domain, for example identifying cats and dogs, we were getting accuracies of 97%+ indicating this approach has a lot of potential.

3.3.3 Symbolic Regressor Using the Genetic Algorithm

Genetic algorithms try to find a mathematical expression that produces accurate output values from the input values. They start by taking a set of random expressions. A subset of these expressions is then put into a tournament where each is scored on how well it predicts. The best expression of each tournament is moved onto the next generation. They do not move on unaltered; there are some techniques you can use to combine them. For example, two trees may be combined in some way, possibly by extending the subtrees appropriately. This whole process runs for some generations before the most successful one is finally chosen to be used as the regressor in the last generation.

In our project, we have attempted to implement a genetic symbolic regressor. However, due to time constraints and it not being part of the specification and therefore supported by the frontend, the genetic algorithm is not exposed over API.

3.3.4 K-Means Clustering

The k-means clustering algorithm attempts to split a given anonymous dataset with no class identity to a fixed number (k) of clusters.

Initially, k number of so-called centroids are chosen randomly. A centroid is a data point at the center of a cluster. These centroids are used to train a kNN classifier. The resulting classifier is used to classify (using k = 1) the data and thereby produce an initial randomized set of clusters. After that, each centroid is set to the arithmetic mean of the cluster it defines. The process of classifying the data points and adjusting the centroid is repeated until the values of the centroids stabilize. The final centroids will be used to produce the final classification/clustering of the input data, effectively turning the set of initially unlabelled/unclassified data points into a set of data points, each with a class identity.

3.3.5 Feature Selection

In order to simplify the models so that it is easier to interpret by researchers/users, training times are reduced, and generalization by reducing over-fitting is enhanced, it is possible to add feature selection before actually passing the raw data inputs to the classifiers.

3.3.6 Estimator Suggestion

For the suggestion of the best estimator for a certain data target, the main design approach is first to determine what type of the problem is: classification, regression, or survival analysis. The parameter provides information about the target type, which includes discrete, continuous, and lifetime. For discrete data target, it is going to be a classification problem; for continuous data target, it is going to be a regression problem; for lifetime data target, it is going to be a survival analysis. If problem type has been determined as regression or classification, an exhaustive search will be performed on every kind of estimator of that type. Each estimator will be run using GridSearchCV with its default parameter set, and the optimum result will be stored for comparison. For SVC and SVR, since it is a very common practice to scale the data, the standard scaler is applied before these estimators. If the determined problem type is survival analysis, it will return the Cox Regression as this is the only survival analysis estimator currently incorporated in the system.

3.3.7 Cap False Negative Rate

As the Product Owner suggests, it may be useful for a user to specify a limit on the false negative rate, and get recommended combinations of models and corresponding parameters that fulfill the requirement. Since it is not possible to get FNR directly from the sklearn API we use the following formula: $FNR = 1 - Recall(TPR)$ We uses *precision_recall_fscore_support* for getting the recall, which is the second float of the returned list. Since the FNR is specific to the classification problem, only the estimator and parameter sets for classification will be run. Each estimator will be run using GridSearchCV with its default parameters set, and the false negative rate that meets the requirements will be stored in a tuple along with the details of the estimator, its parameters and preprocessing if there is any. The output will be a list of tuples, with the FNR in ascending order.

3.3.8 Kaplan-Meier as an Intermediate Step

The Kaplan-Meier estimate does not take into account covariates - it simply estimates the survival function of the given dataset. It can therefore not be used to predict the survival function of a set of covariates. Because of this, we have decided that it cannot be set as a final estimator, but can rather be included in a pipeline as an intermediate step that plots an estimate of the survival function.

3.4 Server

3.4.1 Year Standard API Explanation

The following diagram (figure 2) shows the information flow of a single start training request. Each arrow represents an HTTP request. As you can see from the diagram, HCI first sends an HTTP request to the back end that contains the job id, input columns, project name, etc. This request is forwarded to the ML server where the following checks are made:

1. The HCI client is properly authenticated to request the training to start.

2. The project the user is accessing is valid

At this point, the ML server requests the data file from the BE and selects the correct columns. The job can now be started. On completion, the trained model is sent back to the BE for storage.

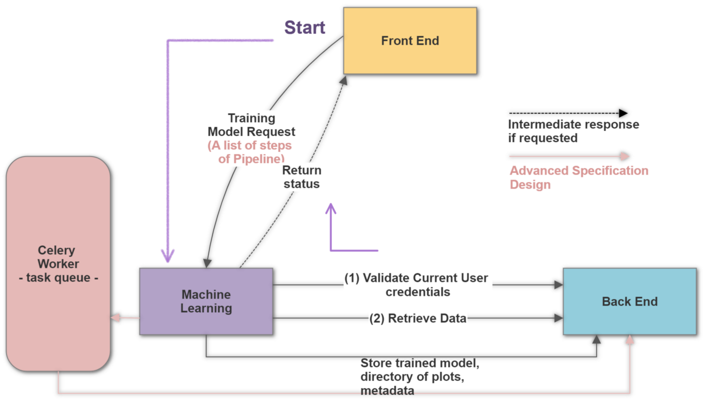

Start
Front End
Training Model Request
(A list of steps of Pipeline)
Return status
Intermediate response if requested
Advanced Specification Design
Celery Worker - task queue -
Machine Learning
(1) Validate Current User credentials
(2) Retrieve Data
Back End
Store trained model, directory of plots, metadata

Figure 2: Year Standard API Diagram with Team Extended Design

3.4.2 Release 2 Proposed API (Not Final)

The intermediate release included our proposed API which has been reviewed after the penultimate release. The motivation was to simplify the ML design which later has been merged with the year standard API. First, the HCI client sends the start training request as before. The backend server then makes sure the client is authenticated and that the request is valid. The backend then selects the specified columns from the specified file and includes this in the request it makes to the ML server. At this point, the ML server can begin training model. Upon completion of the model training process, the ML server, stores the model in the backend.

3.4.3 Final API and Protocol

The API used to implement our ML server for the final release can be found at:
https://app.swaggerhub.com/apis/syzroy/ML7-API/1.1/
It is a modification of the year standard API.

The year standard API can be found at:
https://app.swaggerhub.com/apis/JH-Project/machine-learning-api/1.1/, with credits to Ryan Wilson and Daphne Bogosian.

The basic protocol used to implement our ML server can be found at:
https://github.com/CS3099JH2017/cs3099jh/blob/master/protocols/ML01.md
There has been a series of discussion between the whole year including some

meetings with the product owner. The design decisions made on the protocols and the standard API are a result of these discussions.
Our API fully supports the standard API, with extra support for our advanced features.

3.4.4 Modifications made comparing to the standard API

1. Added error checking for the required fields in the swagger. For example, when sending a basic training request, "job_id" is a required field. In our last release, we needed to do validation manually, for example to check whether the "job_id" field is present or not. In this release, we do not need to do any manual validation as the swagger generated code deals with this.

2. Since error checking is added to the swagger, some error messages are not the same as specified by the standard API. This is because when swagger does error checking, it gives back slightly different error messages. Also, when some parameters are missing in a request, instead of getting 404 not found errors, we will now get 400 Bad Request error, which will make more sense.

3. Add in advanced features, such as train advanced, get transformers and get estimators. Details can be found at the link above to our API.

3.4.5 Server structure

Note: To implement the API, we used the swagger generated code as a foundation from which to build upon.

1. The '__main__.py' file is the main file for running the server

2. All server side code files are in 'venv/src/server' directory

3. All '.py' files in the 'server/models' directory are generated by swagger, they contain the oop structure for all objects needed to parse requests and responses

4. The 'server/swagger' directory contains the API description in the form of a '.yaml' file, which will be used by swagger for resource routing and error checking

5. The 'encoder.py' file contains the JSON encoder used to parse JSON requests to python dictionaries

6. The 'util.py' file contains functions to convert dictionaries to actual models used by the server

7. The 'server/controllers' directory contains the actual implementations of our service endpoints

8. The 'server/worker' directory contains the celery worker configurations and model training tasks. Detailed explanations of the 'celery worker' can be found in section 4.4.5.

3.4.6 Intermediate Server Design (Release 1)

During the first release there was not a consensus on the year group protocol, for this reason we initially wrote our own API so the lack of a standard protocol did not hinder our development. This API was very basic, and details of this API can be found in our Release 1 report. By release 2 we had adopted the newly decided year group standard protocol.

Only one of the design decisions from the first release persisted into subsequent releases, the "celery worker". Without an asynchronous structure, model training processes using machine learning algorithms may take a long time to finish, and during that time the server would not be able to receive requests. To address this we decided that model training should be a background task. For this purpose, we used a distributed task queue called Celery, and we chose Redis as the message broker for the celery worker to receive and send information regarding training tasks. A full list of dependencies can be found in the appendix.

3.4.7 Intermediate Server Design (Release 2)

We adapted to the year standard API since the second release.
For the second release, the server functionalities we implemented are:

1. Getting all available jobs (a job is to train a specific machine learning model using a particular algorithm, with specified arguments - this is explained in the protocol).

2. Start training a machine learning model.

3. Get the training status of a specific model.

4. Stop the training of a specified model, and delete the model from backend if training has finished.

5. Giving predictions using specific models and input data.

To test our server's functionalities, we have built a simple backend server. The design decisions for the mock backend will be found in the extension section.

For the second release, we implemented authentication so that every request, except for getting available jobs, will need to be authenticated with the backend. As stated in the extension section, for testing purposes, our test backend only accepts "Bearer 12345" as the authorisation token.

For the celery worker, we used to have 7 different training tasks in the first release. In the second release, we generalised them into a single general training task to avoid creating unnecessary dictionaries. After a task has finished training, the model will be sent to the backend in binary form to be saved.

For the 'get jobs' functionality, we created a dictionary to store all the job information when the server is running. We can use temporary storage because this dictionary will only be accessed when the server is running, and the contents of this dictionary will not be changed during run-time. Therefore, there will not be any data loss.

For model training, a lot of error checking when parsing JSON requests is implemented inside the 'create_model' function. Since the API description (yaml file) does not handle any error checking, the code ended up being quite verbose. This is resolved in the final release by modification of the "yaml" file as explained before.

To delete a training model, the code currently only deletes the model locally. The ability to delete the model from the backend is not implemented yet because we have not implemented the protocol to delete a file in the mock backend server. This is resolved in the final release by modifying the test backend to support file deletion.

For prediction, the process is that our server requests the data and model file from backend and then deserialises the binary data received back to machine learning models. After that, the predict() method is called on the model.

To get the status, in this release, we did not implement the ability to get real-time training progress. Instead, it will just be 1 or 0 when the task succeeded or failed respectively. The progress tracking is added in to the server in the final release.

To get the list of models, we simply just call 'get status' on every model within the given project.

3.4.8 Final server design

For the final release, we implemented all the requirements proposed by both the "CS3099_spec_ML_1.pdf" given at the beginning of this school year, and the nearly all of the "ML01: Machine learning basic specification", which is the year standard protocol.

There is one functionality in the protocols which we did not implement: Stop a partially trained model, and save the already trained parts. After discussions, we decided not to provide partial training support because we could not find a library that supported partially trained models. It is not in the basic specification, and the product owner has not shown significant interest in this area.

Due to time constraints we were not able to add all functionalities to the server:

1. The endpoint to cap false negative rates for all classifier algorithms. We have implemented the functionality to cap false negative rates, as explained in 4.3.2, and we have tests to prove it is working. Since this advanced feature will not be supported by backend groups and HCI groups, we decided that it is optional to implement an endpoint for this feature.

2. For similar reasons as the last scenario, we decided that it is optional to implement an endpoint for this feature.

Implementation changes comparing to the second release:

Authentication
In addition to the authentication implemented from last release we now also try to verify authentication by accessing the ”current user” information from the backend. By doing so, we can avoid unneeded error checking and we should have a faster response time. We used the ”test backend” to test for this functionality. Note that if a request does not have an authorisation token, it will fail immediately without accessing the backend.

Starting a machine learning job Modifications since the second release are the following.

1. For any algorithm other than cox's regression, requests with more than one target column more will receive a ”Bad Format” error. In the case of cox's regression each of the two target columns will be time stamps. For each row in the training data we can find the duration by subtracting one from the other, this preprocessing step is applicable to cox's regression only.

2. Integrating the server to fit our pipeline design for model training. The explanation about our pipeline design can be found at Section 4.

For a basic training request, we will train the model using optimised default hyper-parameters for the pipeline. By doing so, the end-user does not need an in-depth understanding of the underlying structures.

3. For Cox regression, we added one optional Boolean parameter called "event_column". After discussion, it was decided by the whole year that the first column of the input columns will be the event column for Cox regression. Therefore, when this Boolean value is set to true, we will regard the first column of the input columns as the event column. If it is set to false or this parameter is not given, we will create an event column during data preprocessing.

4. The label encoder we get after data preprocessing will be saved into the pipeline object to be stored in the backend. The label encoder is needed by prediction in order to translate the codes back to the original labels for display to the user.

5. If the target is discrete, the model can only be trained on classifier algorithms, such as Gaussian Naïve Bayes. If the target is continuous, the model can only be trained on regression algorithms, such as Random Forest Regressor.

6. After the model is trained, as advised in the standard API, we create a bundled "report" folder on the backend under the given "output_file_path", and then upload all the plots generated into that folder, along with a single JSON "report" file for HCI to use. A detailed explanation of what a "report" file is can be found on the HCI protocol here (under "Requirement: Report files"): `https://github.com/CS3099JH2017/cs3099jh/blob/master/protocols/UI01.md`

Delete a model After authorisation whether the project exists or not is checked. If it does not exist, a "project not found" error is returned. If this model is not currently training, the backend is checked for the model and is removed if present. If it is not currently in training and not in the backend, this model does not exist. In this case the "model not found" error is sent.

Getting predictions from a model After getting the model in binary form, we decode the binary data back to the model object in python. We then call the ".predict()" method using the supplied input and target columns. After getting the prediction results, we check the first element of the result list to determine the type of the results. This is required as part of the http response as specified by the group API.

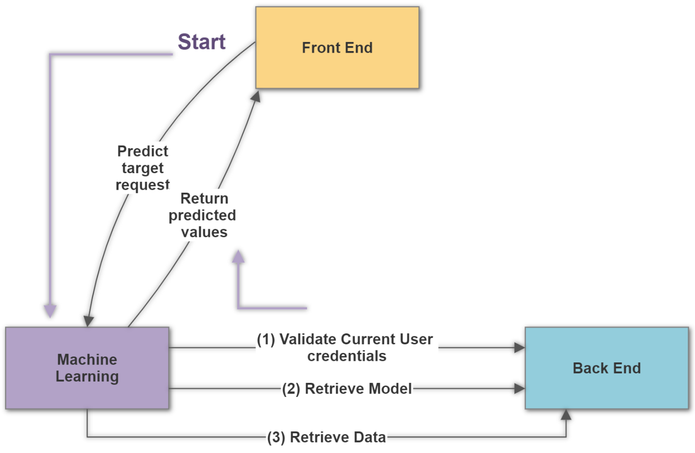


Figure 3: API Diagram for Predict Request

Algorithm suggestion Upon receiving a request, the type of the target column is checked. If it is discrete then all classifier algorithms are run to get the accuracy scores for each of them. If the target type is continuous, regression algorithms are run instead of classifiers. The results will be returned as a list of ”jobs” associated with an accuracy score. Lifeline predictions are not available for algorithm suggestion.

In this release the server has been modified to accommodate the pipeline control flow added in the second release. The server now includes 2 new endpoints providing descriptions of transformers and estimators supported. These and other functionalities were developed in collaboration with HCI1 to provide an intuitive way to specify steps.

File Storage Decisions (not stateless) For file storage on the server side, we created a ”Model” object for each data training task, to store necessary information such as time of creation, and job id. Also, we created two dictionaries, one for mapping between model ids with Model objects, and the other one for mapping between model ids with file ids (we will only get a file id when we store models on the backend). Therefore, our server is not stateless. We fully understand that by doing so, we may suffer from data loss when our server crashed. However, due to the nature of this project, we can not get a backend group to test file storages with until very late, and there is limited time for us to implement a fully working backend by ourselves. Therefore, to ensure the submitted server is fully tested, we did not change the server to be stateless.

Advanced server features

/transformer This endpoint returns a dictionary mapping transformer names to their descriptions and supported hyper parameters.

/estimator This endpoint returns a dictionary mapping estimator names to their descriptions and supported hyper parameters.

/trainAdvanced This endpoint is used to start a training job of a number of steps consisting of one ore more transformers followed by an estimator. Within each step any number of hyper parameters can be included as well as the following global parameters k, split_ratio and event_column.

4 Software Development Tools

4.1 SCRUM Process

All scrum documentation up to date can be found in the "Release_2" directory of the project, along with the project README, which contains instructions on how to run the server and the unit test suite. The unit test suite, tests.py executes all the tests using predefined input datasets demonstrating the main functionalities we maintained.

SCRUM Process was applied extensively for our intra-group collaboration. We have rotated the role of scrum master for every sprint, whoever was given the role was responsible for updating the scrum backlog and leading stand-up meetings during a sprint. Outcomes of scrum meetings are all stored for the record. The Product owner has changed regularly throughout the sprints since each of us has researched different aspects of the project and came up with suggestions by expanding on the initial requirements in the specification.

Team Structure and Role Assignment

As was the procedure previously, the team has assigned a scrum master for every sprint in the rotation. The scrum master was responsible for overseeing the progress of each team member and the team. The role included managing a *milestone* (a management tool for a sprint) and recording meeting outcomes in a backlog file on gitlab. Every milestone typically contained tasks (issues) assigned to team members. The task assignment was always done at the sprint planning meeting upon agreement of all members.

Meetings

Regular stand-up meetings typically consist of sharing information about the project organisation, updating the members on the progress of each other by presenting their work and commenting on the future tasks to be done. At least two meetings have been arranged for every sprint. The minutes of the meetings are always held in gitlab for future reference and to inform any members in absence.

Scrum Reflections

Efficient use of the scrum process and features of version control such as issues and branches has enabled our team to work in a coordinated manor. Working on separate branches means members were able to work without interfering with each other. Use of gitlab issues meant that every member of the team had a clear task in front of them.

Despite producing tangible results, there were areas that could be improved. There was a tendency to underestimate the workload while assigning tasks to each team members. Of course there are always unforeseen circumstances that disrupt the workflow, e.g. illness that are not always communicated with the Scrum Master promptly. Were we to use the scrum process again, allowing for more leeway around the deadlines for completing issues means that disruption can be minimised and issues are less likely to spill over to subsequent sprints.

4.2 Communication and Management

We have used various communication methods according to our availability and circumstances during holidays. We used Skype calls for weekly sprint planning and retrospective meetings and Facebook group messenger for stand-up meetings and design discussions, while year-wide discussions were mostly done through Slack.

4.3 Team Review of Challenges faced

We believe that we worked very well as a team, adhering to the SCRUM process and spreading tasks evenly among members to produce a working piece of software aimed at satisfying the user's needs. However, we did face some challenges:

- The entirety of our team worked remotely during the winter vacation when our three sprints took place. There were teammates working in China, Russia, and the UK at the same time. The different time zones and conflicting working hours have undermined some effectiveness. Since China's Great Firewall blocked Facebook Messenger, which is one of the major tools for team communication, we had to switch our group discussion to Skype. However, this challenge quickly vanished after everyone came back to university.

- Scrum masters often found it challenging to constantly keep up with the updates from each developer for the duration of each sprint. There was often an information delay which caused some confusion and trouble for their management. Retrospective meetings have helped in receiving feedback for this role and for the future release. It has been agreed that issue boards can be used more effectively as a point of updates for each developer, writing comments on their progress.

5 Inter-group Collaborations

From the very start, we made extensive efforts to collaborate with other groups, particularly backend and HCI groups that would be connecting to our system.

It was initially hard to identify what stage other groups were at and how ready they were for inter-group communications but after the small demo to the project owner on the 27th, we identified an HCI group, HCI-1, which we felt were doing well and made plans with them to collaborate.

We attempted to work with some backend groups but found none were at a stage to start to collaborate or even start connecting although we informed them of our ideas and they were supportive.

In the meeting with HCI-1, we proposed the idea of orientating the models around the idea of a pipeline and presented the features we currently had and planned to implement. They agreed the idea was useful and intuitive and from that together we developed the idea from both a front-end and machine learning perspective.

Towards the deadline we were assigned another group with whom we were to do the demo. We immediately created a group chat, explained how to use our system and provided all the endpoints necessary to connect. Progress is on-going as neither the HCI nor the Backend group assigned to us had a fully working implementation, but we provided all the help we can and are getting closer to full integration.

6 Presentation of Case Study

6.1 Basic Specification

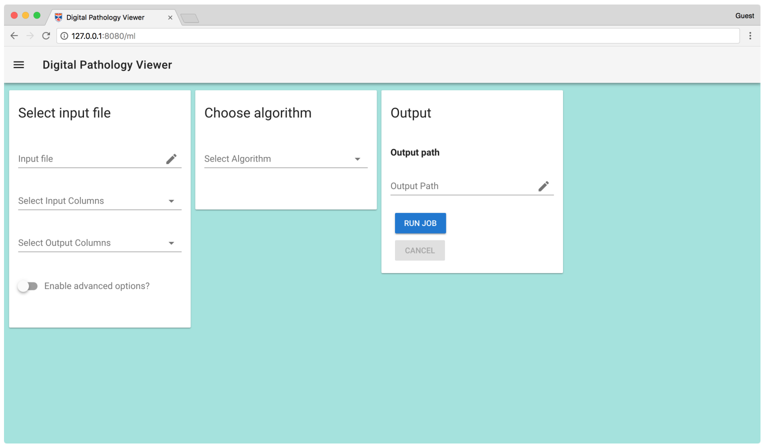


Figure 4: The basic training as seen through HCI1's user interface

As shown in figure 4 the user can have key components to be added, which will be passed to the ML.

7 Evaluation

7.1 Comparison to the Basic Specification

We have a complete working program that implements all features of the basic requirements.

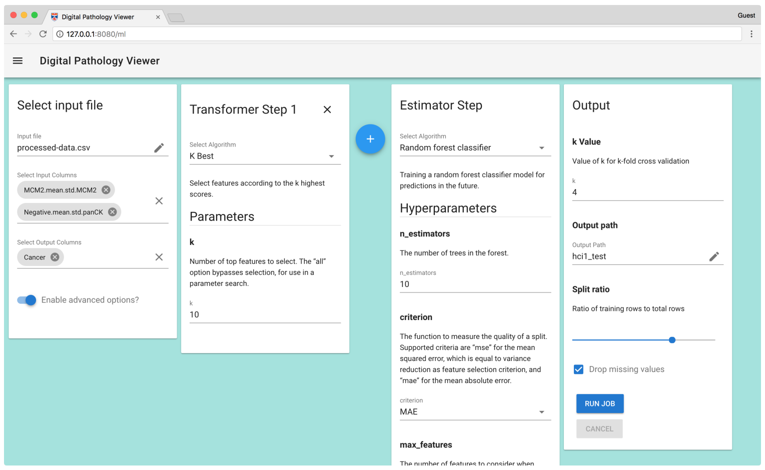
We have created a good framework for data analysis, providing a large number of machine learning algorithms in an adaptable fashion and giving the user a lot of control over the process.

One thing to note is that we didn't make it medicine-specific despite medical analysis being the original premise. It would have been very hard to orientate the design of our program around medical data analysis as that would require much domain specific expertise along with guaranteed consistent data insights and it has been stressed by the module coordinator right from the start that that is not required.

That said, it is still more than sufficient for powerful data analysis on medical data.

7.2 Comparison to Other Groups

Throughout the project, we maintained consistent contact with the project owner. Through this communication, we developed the advanced specifica-



Select input file

Input file: processed-data.csv

Select Input Columns: MCM2.mean.std.MCM2, Negative.mean.std.panCK

Select Output Columns: Cancer

Enable advanced options?

Transformer Step 1

Select Algorithm: K Best

Select features according to the k highest scores.

Parameters

k

Number of top features to select. The "all" option bypasses selection, for use in a parameter search.

k: 10

Estimator Step

Select Algorithm: Random forest classifier

Training a random forest classifier model for predictions in the future.

Hyperparameters

n_estimators

The number of trees in the forest.

n_estimators: 10

criterion

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "mae" for the mean absolute error.

criterion: MAE

max_features

The number of features to consider when

Output

k Value

Value of k for k-fold cross validation

k: 4

Output path

Output Path: hci1_test

Split ratio

Ratio of training rows to total rows

✓ Drop missing values

Figure 5: The advanced implementation of pipeline as seen through HCI1's user interface

tion taking into account desirable features from the user's perspective. This allowed us to move beyond just using the basic specification and create a product that would be more useful for users compared to other groups.

7.3 Comparison to Previous Works

There are many similar professional software services available with providing similar functionality. They all have a strong focus on data parsing and presentation of various metrics and mainly allow the user to analyse R functions. Our project is then a good development on this; it is very extendable, and as python is currently becoming increasingly popular among data scientists, new packages would be easy to integrate with our program.

8 Testing

8.1 Effectiveness of the Added Preprocessing

To make sure that the preprocessing algorithms we introduced into the system do actually improve the performance of the classification estimators and therefore are valuable to have as extensions, we made some comparison tests in tests.py and advanced_tests.py for some classifiers with or without preprocessing steps.
One example is the Bernoulli Naive Bayes Classification with Binary Data with or without Random Under Sampler and Standard Scaler. The performance for the classifier is much better with those preprocessing steps, which is evident in the following supporting screenshots and plots.

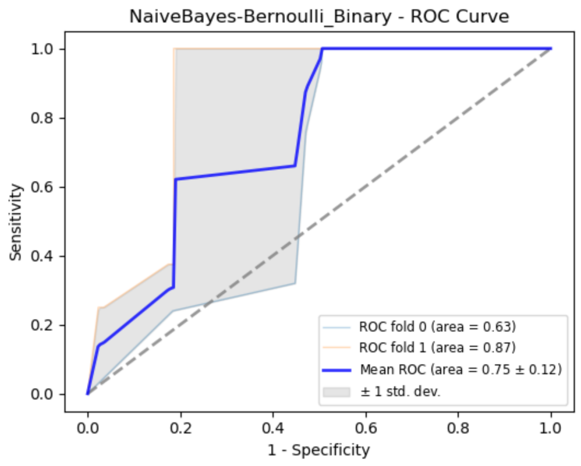


Figure 6: ROC curve plot of Bernoulli Naive Bayes model alone

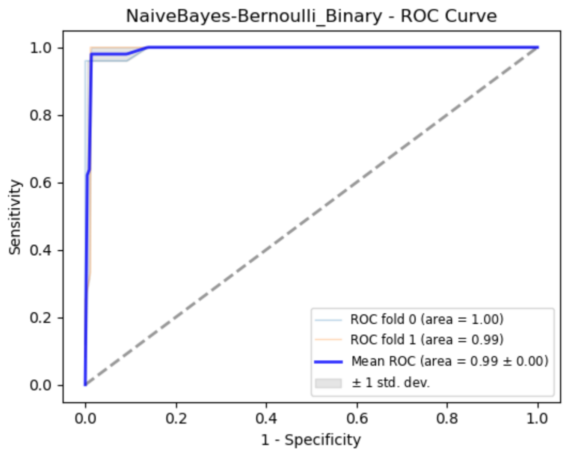


Figure 7: ROC curve plot of Bernoulli Naive Bayes with pre-processing steps

For the ROC Curve, the Bernoulli Naive Bayes Model with pre-processing steps has far larger area under the curve than the one without pre-processing steps.

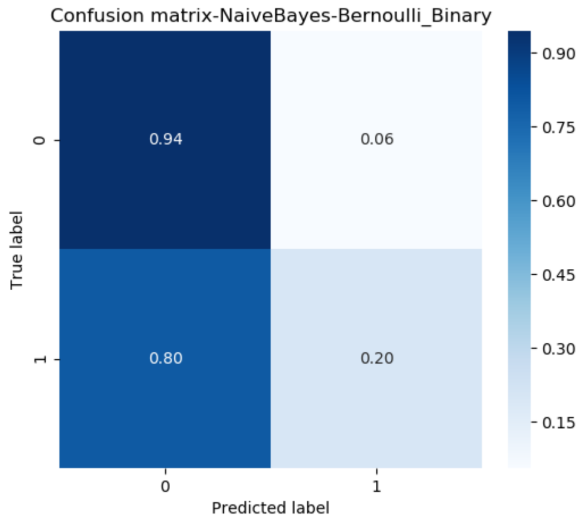


Figure 8: Confusion matrix showing prediction performance of Bernoulli Naive Bayes model alone

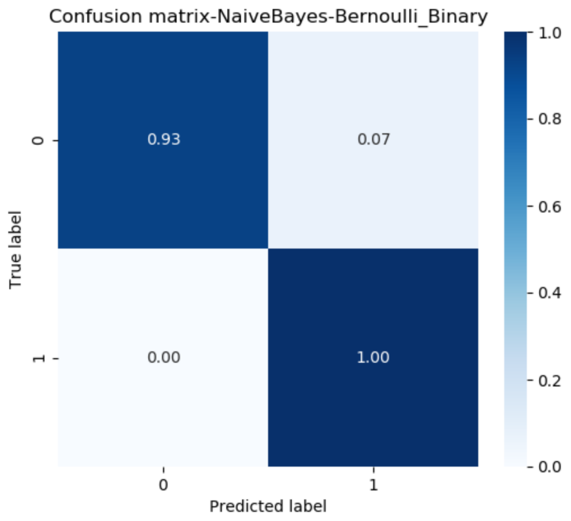


Figure 9: Confusion matrix showing prediction performance of Bernoulli Naive Bayes with pre-processing steps

For the Confusion Matrices, for the Bernoulli Naive Bayes Model with pre-processing steps, much higher percentage of elements are correctly predicted into the right class, which can be seen from the two very dark areas on the top-left and bottom-right, and two very light areas on the top-right and bottom-left.

8.2 Python Unit Test Suite

We have an extensive unit testing suites including testsṗy and advanced _testsṗy, which include testing of key algorithms implemented. These are separated in order to distinguish the standard and advanced specification implementation. The latter will test the functionality of combining various transformers and estimators. More specific test cases are written in testing scripts such as image_analysistestṗy and in dataṗy. We can therefore compare and assess the performance of the models; this can be easily done by looking at the graphical outputs of many formats such as plots of confusion matrices and curves: the graphical output allows us to not only check for the existence of errors but also perform a sanity check of the machine learning implementations.

8.3 API Testing

To check that our server is coherent with the API, we have initially attempted to create a continuous integration suite using the CI runner provided with the gitlab. The suite tested functionality such as starting the training process for a model on selected data with a chosen algorithm and checking the status of the training process. However, due the semi-random nature of the output produced by our server (the random model UUID to be precise) it is difficult to check the output against some expected text. Because of this, and because of writing requests with the *curl* command being very error-prone, we have decided to give up on the idea of a full CI suite and test our server using a Node.js script instead.

Node.js is a language made for networking tasks, so it is particularly well suited for testing an HTTP server. The test script uses the *Mocha* and *Chai* unit testing frameworks and is therefore also structured in the form of a unit test suite. To test the server, the script sends HTTP requests to the server and checks that the response is of the form described in the API. It also sends malformed and unauthorised requests to ensure that the server responds to those properly.

In order to obtain data, our server relies on a backend server. In order to be able to test our server we have implemented a simple mock backend to serve data files. The backend protocol we followed can be found here:

https://github.com/CS3099JH2017/cs3099jh/blob/master/protocols/BE01.md
To simulate the file storage system on the backend we used python dictionaries in order to easily retrieve data and do error checking. To simulate authentication, we only accept one token 'Bearer 12345' through authentication. If the authentication token is not given or that token is not the correct one, it will just return '401' error. For the final release we have also added support for user credentials for the default user "Bearer 12345". We also added a delete endpoint so that models can also be deleted using a mapping from model id to file path.

Note that only certain functionalities are implemented for the mock backend as it only exists in order to test our server.

The test suites have been invaluable for ensuring our program works as expected throughout the duration of the project and has helped to uncover multiple bugs. The specific instructions for running the unit test suite and the API test suite are included in the README.

9 Conclusion

We have created the machine learning component of the medical data analysis tool as described in the specification. Our machine learning server meets the basic requirements and also includes advanced features requested by the product owner. We have adhered to the agile development philosophy and the scrum process as has been encouraged throughout this module.

For the duration of this project and especially for this final release, our main objective has been to create a component that ultimately does not only show our skill but is also useful to the Product Owner. In order to achieve that, we communicated extensively with the Product Owner to tailor the system as much as possible to their needs and with other groups, so that in the end we may be able to showcase a working application.

Appendices

A Libraries used

All libraries used can be found in the dependencies text file hosted on gitlab.
https://gitlab.cs.st-andrews.ac.uk/cs3099group-ml-7/project-code/blob/master/venv/dependencies.txt

"Project objectives, specification, plan, and interim report as submitted during the year, followed by a list of changes that should be made in the light of experience, e.g. additional references, new software modules, changed specifications."

B Basic Product Backlog

The copy of basic specification is here.
https://studres.cs.st-andrews.ac.uk/CS3099/0-General/CS3099_spec_ML_1.pdf
The copy of modified basic specification agreed by the year group is here.
https://github.com/CS3099JH2017/cs3099jh/blob/master/protocols/ML01.md
The copy of our first product backlog is here written in code format and as request responses.
https://gitlab.cs.st-andrews.ac.uk/cs3099group-ml-7/project-code/blob/future-development/Release_1/ProductBacklog2711.md

C Extended Product Backlog

Product Backlog For Extensions
This following new user stories have been included for the extended specification:

- The user can choose to pre-process data using different scalers before model training.
- The user can split a dataset into a training set and a test set. The test set will be used to make predictions on the model. The user can define the proportions of the split or use default values otherwise.
- The user can choose feature selection in the model construction.

- The user can define hyper-parameter values on which to build a model. The model will be built with default optimal values otherwise.
- The user can choose to build a model under fixed values of False Negative Rate or True Positive Rate as preferred.
- The user can specify the number of runs to obtain a trained model.

D Initial Design

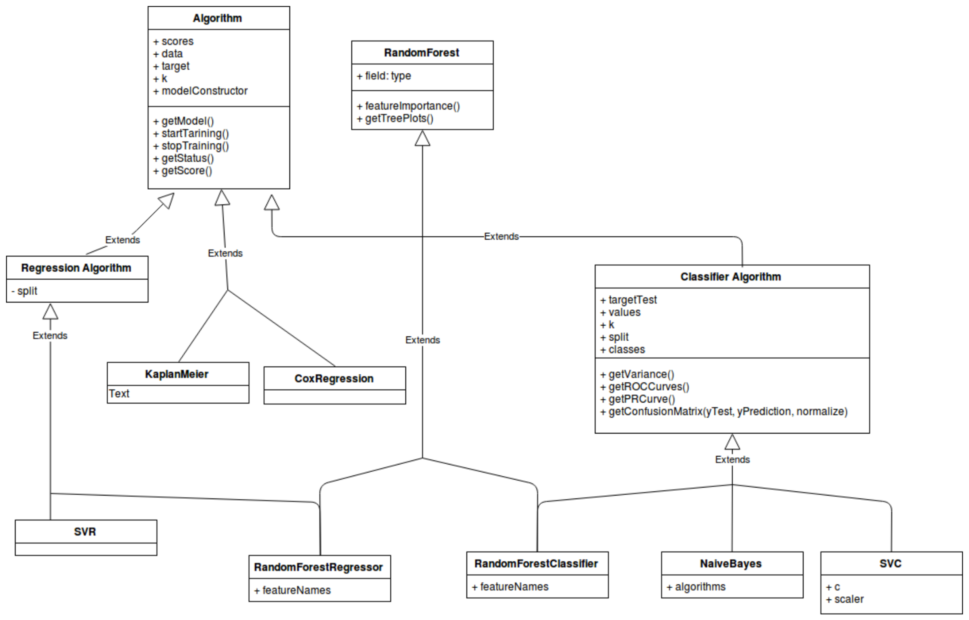


Figure 10: UML Class Diagram