Breast Cancer Tumour Classification With FIS

Jon-Paul Boyd
School of Computer Science and Informatics
De Montfort University
United Kingdom

***Abstract*— Of all cancers occurring in women, breast cancer is the most common, with 2 million new cases and over 0.5 million deaths globally each year. Many doctors are overwhelmed, having to deal with up to 70 patients a day. Mistakes in diagnosis can have deadly consequences. This report presents a fuzzy logic knowledge-based inference system for the binary classification of breast cancer tumours, taking an extensive data-driven, supervised learning approach in defining linguistic terms, implication rule sets and membership functions, with the Wisconsin Diagnostic Breast Cancer dataset used as the empirical, impartial problem domain expert. A large set of configuration variants that includes 5 defuzzification methods are tested, prioritising the classification accuracy of malignant tumours. The best FIS model classified all test set malignant tumors correctly with an overall accuracy of 93.6%, outperforming the logistical regression, decision tree and random forest machine learning methods.**

Index – Fuzzy Logic Decision Support, Breast Cancer, Classification, Wisconsin Diagnostic Breast Cancer, Skfuzzy

I. Introduction

Breast cancer is the most common of all cancers diagnosed in woman, with 1 in 8 at risk of developing it during their lifetime [1]. Over 0.5 million died from breast cancer in 2011 [2], while more recently in 2018, 2 million new cases were reported globally [3]. In the UK alone, between 2015 and 2017, there were 11,400 deaths each year, or 31 per day [4]. However it is not a disease solely of the western world, as 50% of cases and 58% of deaths occur in less developed countries where the survival rate falls to below 40%, compared with the U.S. at ~80% and Japan at ~60% [2].

Such low survival rates are attributed to discovery only at late-stage, primarily due to absence of early detection programmes, poor diagnosis and limited medical resources [2]. The latter issue is exemplified by the authors of [5] in researching fuzzy inference systems classifying prostate cancer, where in China over 5000 patients are allocated to one physician seeing over 70 patients a day. Understanding the increased potential for fatal misdiagnosis in such overwhelmed healthcare systems, these statistics are especially tragic given that "w*hen diagnosed at its earliest stage, all (100%) people with breast cancer will survive their disease for one year or more*" [4].

The aim therefore is the design and implementation of a decision system supporting medical practitioners in correctly classifying breast tumours as benign or malignant using key features identified in dataset analysis. Given the mortality rates, highest possible prediction performance of malignant tumours is prioritised but demands acceptable benign prediction performance. The Wisconsin Diagnostic Breast Cancer Dataset (WDBCD) [6] containing sampled tumour data labelled as benign or malignant is used to train and validate a fuzzy inference system (FIS) modelling the problem using fuzzy logic (FL), a concept first introduced by Zadeh in 1965 [7]. This will facilitate the assembly of a transparent knowledge base of *IF...THEN* rules, where with fuzzy logic sample input features such as tumour radius and texture can be graded as "*how benign*" or "*how malignant*" with a value between 0 and 1, rather than 0 or 1 as per Boolean logic. These grades are then combined to determine a fuzzy output, which is then transformed into a binary tumour type classification.

With such an intuitive knowledge base, both the system rules and conclusion can be understood and interpreted by the medical practitioner, providing a far more "*explainable A.I.*" solution than machine learning (ML) alternatives such as neural networks. Leveraging FIS in this classification problem seems appropriate as "*diagnosis implies complex data involving several levels of uncertainty and imprecision*" [8] that can be modelled by fuzzy grading of features. This is further supported with the authors of [9] explaining that "a *single disease may manifest itself quite differently, depending on the patient, and with different intensities*" and "*a single symptom may correspond to different diseases. On the other hand, several diseases present in a patient may interact and interfere with the usual description of any of the diseases*".

In similar WDBCD classification work, Nauck et al used a neuro-fuzzy approach to achieve an accuracy of 95.06%, while decision trees were used by both Quinlan and Lavanya et al, obtaining accuracies of 94.74% and 92.97% respectively [10]. These results will be used in addition to scoring accuracy with logistical regression (LR), decision tree classifier (DTC) and random forest classifier (RFC) models to benchmark classification performance of this study's FIS models. Given the domain criticality, sensitivity as the proportion of malignant tumours predicted correctly will be scored for each test scenario.

This paper is organized as follows. Section II provides an analysis of the WDBCD to understand it's features, characteristics and distribution. Section III presents the FIS design and implementation, while Section IV evaluates testing results. Section V concludes. Note all work done on experimentation setup given in *Appendix A*.

II. Dataset Analysis

Analysis of the WDBCD helps understand its information content and help guide FIS configuration. There are 569 sample observations of breast cancer tumour. Each comprises 30 independent real-number features that are "*computed from a digitized image of a fine needle aspirate (FNA) of a breast mass*" and describe "*characteristics of the cell nuclei present*" in images [6]. A sample *ID* and dependant binary classification label D*iagnosis* is also included. Class labels *B* (benign) and *M* (malignant) are mapped to 0 and 1 respectively to support mining and machine learning methods requiring numerically-typed targets.

A full feature list with distribution statistics including mean, max and standard deviation (SD) is given in *Appendix B*. 357 samples (62.74%) belong to class *B* and while minority class *M* has 212 samples (37.26%). There are no duplicate samples nor input features with constant value, null value or exhibiting sparsity (> 99% observations with zero value).

A. Feature Importance

Given the full WDBCD consist of 30 features, it is desirable to use only a subset for 2 main reasons. First, to reduce the complexity of the FIS *IF...THEN* rule implementation, and secondly, reduced dimensionality will allow the modelled expert system to better generalise over previously unseen data. Other secondary advantages include easier human interpretation of rules and classification outcomes, shorter model training time and simpler solution maintenance. Both a univariate feature selector using a chi-squared scoring function and an RFC were used to rank the top 15 features on their importance to target label *Diagnosis*, as listed in *Appendix B (UniChi Rank, RFC Rank)*. With prior success applying RFC feature subsets in machine-learning classification [11], the selection by this method (Fig. 1) is used, with all other features removed.

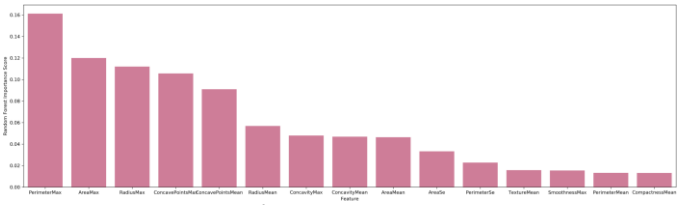

Fig. 1 RFC top 15 feature importance to target label *Diagnosis*

B. *Feature Correlation*

The subset of top 15 ranked features was checked for the presence of the multi-collinearity problem, where input features used to predict an output target are statistically highly associated. This can cause issues in logistic regression models, with highly correlated features affecting coefficient estimation and p-values, while for FIS they are simply redundant and unnecessary in the formulation of a rules knowledge base.

Best possible performance should always be prioritised over FIS complexity given the criticality of correct tumour classification, this is clearly understood. However, to constrain study scope, the feature subset is further reduced, removing features where correlation > 85%. This results in the proposed final subset of 5 key features available to classify breast tumours as malignant or benign (Fig. 2, R), which in ranked order of importance are *PerimeterMax*, *ConcavePointsMax*, *AreaSe*, *TextureMean* and *SmoothnessMax* (*Appendix B, Final Rank*). Going forward only these features form the universe of discourse upon which further analysis and FIS modelling is based.

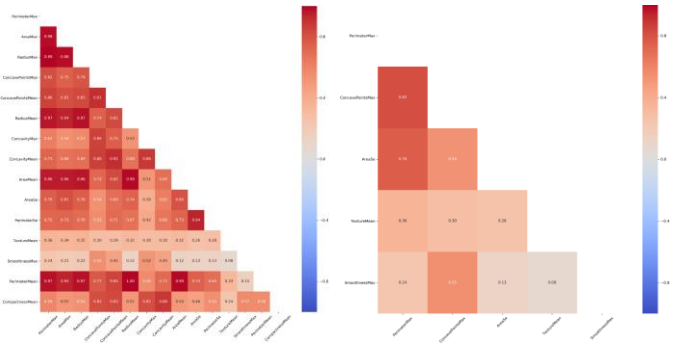

Fig. 2 Correlated features before (L) and after (R) removing highly correlated

C. *Dataset Partitioning*

Without a breast cancer domain expert, this data-driven approach will further mine the selected features to understand distribution and automatically determine parameters of membership functions (MF) which transform inputs into fuzzy grades. The dataset is split into 2 partitions, drawing a random 70% of the 569 samples for training, with the remainder allocated to testing the FIS classification accuracy. This partitioning is especially important to MF shaping which uses only the training subset, as it prevents leaking of bias into testing and thus ensures MFs will grade unseen data with a level of generalisation capability. Where appropriate, data analysis will indicate source is full dataset (FD), training partition (TRP) or test partition (TEP).

D. *Linear Separability*

It is good practice and due diligence to understand if the problem at hand is linear or non-linear. If linear, less complex FIS configuration, or indeed simpler model types such as linear regression may suffice. As each sample is binary classified, 2 features taken as x and y are plotted on a 2D scatter graph with their data point blue if labelled benign (0) or red if malignant (1). A convex hull (CH) is then drawn to connect the outermost points in each class.

Any overlap suggests the problem is non-linear, as is the case here. Figs. 3 and 4 clearly show each red convex hull contains blue points, while blue hulls contain red points. No single straight line can separate red from blue. Therefore, MFs grading a feature as "*is benign*" or "*is malignant*" will need to overlap, a design feature that combined with an array of available MF shapes enable FIS to handle non-linearity effectively.

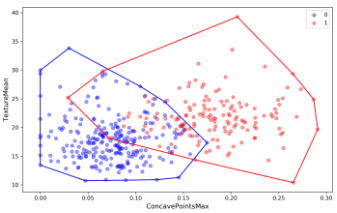

Fig. 3 CH *ConcavePointsMax* vs *TextureMean*

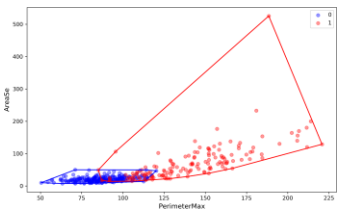

Fig. 4 CH *PerimeterMax* vs *AreaSe*

E. *Clustering*

This study uses *k*-means clustering as an unsupervised learning method to detect any presence of distinct dataset grouping patterns. *K* represents the number of clusters, tested in the range 2-4 with a selection of feature pairings to facilitate easier scatter graph visualisation. As evidenced by Figs. 5 and 6, there is no clear intra-cluster separation, further supporting the claim this problem is one of a non-linear nature.

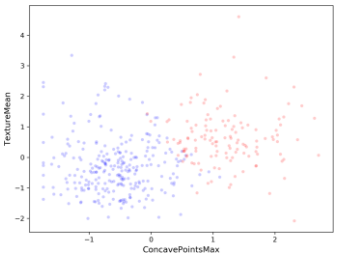

Fig. 5 k=2 *ConcavePointsMax* vs *TextureMean*

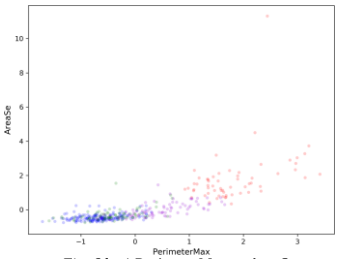

Fig. 6 k=4 *PerimeterMax* vs *AreaSe*

F. *Feature Distribution*

Appendix B summarises the feature distribution while Fig. 7 presents a scaled representation of selected features from the FD, using sklearn's *MinMaxScaler* [12] for comparative illustration. All features exhibit an effectively unimodal distribution with one main peak. *AreaSe* is the most distinctive, given the highest positive skew and kurtosis of all features. Characterised by its tall and thin curve there is little variation, yet with tail to right (mean to right of median) the average tends towards higher values. This is clearly observed in Fig. 8 which plots the distribution as discrete bins in the histogram, with the population smoothed into a kernel density estimate over the top which helps visualise the essence of the distribution shape.

A negative kurtosis of -0.536 signals a flatter, shorter curve than a normal distribution for *ConcavePointsMax* (Figs. 7, 9). It is slightly positively skewed to the right with a bump in population density between 0.16 and 0.22. *SmoothnessMax* with the lowest skew and kurtosis at ~0.46 has the most normal distribution and plots as a smooth bell curve (Fig. 10), closely followed by *TextureMean* measured at ~0.7. *PerimeterMax* is also slightly positively skewed to the right and a little peaky with a kurtosis of 1.07.

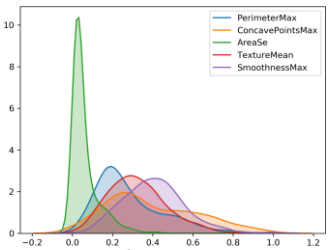

Fig. 7 Selected feature scaled distribution (FD)

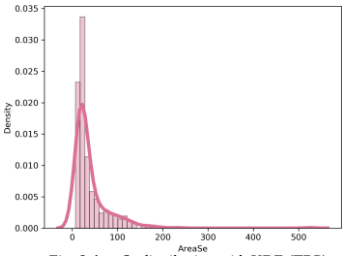

Fig. 8 *AreaSe* distribution with KDE (TRP)

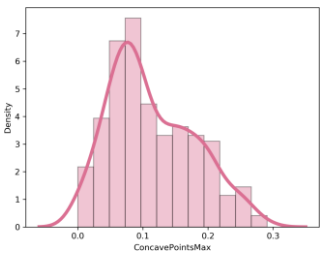

Fig. 9 *ConcavePointsMax* distr. with KDE (TRP)

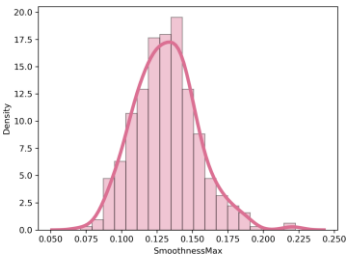

Fig. 10 *SmoothnessMax* distr. with KDE (TRP)

The main purpose of this distribution analysis is the proposal of MF shapes, guided by empirical data and the histograms and KDEs drawn from it. This should best map out and grade each input feature as associated with a benign or malignant tumour, the objective being highest possible classification accuracy. Although the distribution suggests use of gaussian MF functions, others will be tested and compared for classification accuracy. For e.g., Fig. 7 suggests a triangular MF (TRIMF) may be an alternative to GAUSSMF, suited to the tall, peaky *AreaSe* feature. Splitting *AreaSe* KDE by target class does indeed suggest TRIMF could fit and grade "*is benign*" well, while a gaussian MF (GAUSSMF) looks a good fit for "*is malignant*" (Fig. 11). Note the small area of class density intersection, an indicator of little fuzzy overlap.

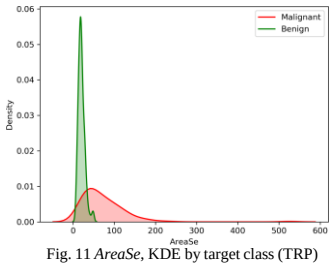

Fig. 11 *AreaSe*, KDE by target class (TRP)

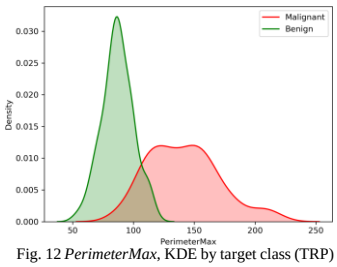

Fig. 12 *PerimeterMax*, KDE by target class (TRP)

For feature *PerimeterMax* the density intersection area is greater here, suggesting a greater fuzzy overlap in MFs. Again, TRIMF may be suitable for the tall, narrow "*is benign*", however trapezoidal (TRAPMF) could work for "*is malignant*" (Fig. 12). Alternatively, overlaid z (ZMF) and s (SMF) shaped functions fuzzifying benign and malignant to the left and right respectively could also be suitable choices that also ensure inclusion of outliers as their foot and ceiling can be set by distribution minimum and KDE peak. Feature *SmoothnessMax* exhibits extensive class overlap (Fig. 40 *Appendix C*), suggesting a fuzziness naturally inherent in nature and further supporting the use of FIS in such problem domains. *Appendix C* includes distribution graphs with KDE split by target class for all features from both FD and TRP. Results testing a variety of relevant MFs are presented in *Section IV*.

A secondary outcome of this specific analysis is the suggested utilisation of only those features exhibiting independent distributions and test to determine if high performance classification is attainable with the smallest knowledge base of antecedent features and rules possible. There are no distinct KDE by target class distributions, and therefore will be no distinct fuzzy sets on the universe of discourse of any used feature, all will be partially overlapping fuzzy sets.

G. *Feature Scaling*

Unlike other machine learning techniques that require dataset normalisation for best results, it is not needed for the proposed FIS. Each tested feature exists within its own min-max range universe within which MFs map out "*how benign*" and "*how malignant*" between 0 and 1, as guided by scaling and distribution characteristics specific to the feature.

III. FIS IMPLEMENTATION

A. *Overview*

A type-1 Mamdani FIS for the classification of breast cancer tumour samples as benign or malignant is implemented with the Python Skfuzzy package [13]. Python [14] was chosen as an alternative to Matlab given it is open source and therefore free of any licensing costs. As a Python-based FIS can be deployed on cost-effective, mobile hardware including raspberry Pi's it is extremely portable and readily located in any clinical environment. As the 4th most popular programming language according to Stack Overflow [15], a large developer base is available to interpret and support any developed solution. With the language's popularity in ML, FIS classification performance can be easily compared with other models. Skfuzzy is the most popular fuzzy logic toolkit for Python based on the number of GitHub stars (395) and repository forks (159) [16]. It has 2 API versions, with the newer used here despite a lack of documentation other than an example of the classic "*tipping*" problem [17].

As presented in Fig. 13, precise, crisp dataset feature values are input into the FIS. MFs specifically shaped to each feature then fuzzify the value to a membership class grade between 0 and 1.The fuzzy rule base defines relationships between sample input features and the diagnosis output in the form of *IF...THEN* class membership rules, used by the inference engine to determine the diagnosis membership class grade which is defuzzified to a precise value for binary classification.

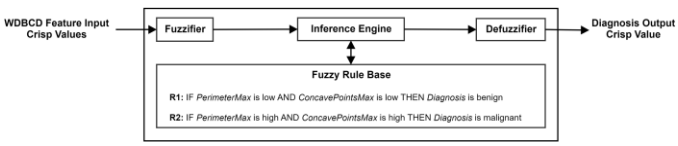

Fig. 13 Breast cancer tumour classification type-1 FIS

Where appropriate a tumour sample with *PerimeterMax* = 76.51 and *ConcavePointsMax* = 0.086 will be used to support illustration of the inference process now detailed in the following sub-sections.

B. *Fuzzy Terms*

The adjectives *low* and *high* are used as linguistic terms to fuzzily represent how each antecedent input feature is graded by a membership function shaped to a distribution filtered by benign or malignant classification considered "*how low*" or "*how high*" the risk of breast cancer is. Only 2 terms are used because the problem is binary classification with only 2 target classes. The naming of the consequent terms inherits naming of the target labels benign and malignant, with the consequent named *Diagnosis* to represent in human language terms the process of classifying the tumour, so "*Diagnosis benign*" as an output membership class defining how benign the sample is.

C. *Antecedent Universe*

The dataset of 569 tumour samples with 5 selected features (*Appendix B, Final Rank*) is loaded into the FIS and randomly split into TRP (70%, 398) and TEP (30%, 171) partitions. An antecedent universe is created for each feature *x*[min, max] of TRP, with boundaries adjusted by an SD "*gain*" to ensure the space can include extreme, boundary or outlier values from the FD universe of *x*. As an example, antecedent *PerimeterMax* is created with a resolution of 200 points bound between 13.79 and 257.41.

```
ant[feat] = ct.Antecedent(np.linspace(
X_train[feat].min() - (X_train[feat].std() * 1.1),
X_train[feat].max() + (X_train[feat].std() * 1.1),
num=200), feat, defuzzify_method=defuzzify_method)
```

D. *Consequent Universe*

A consequent universe with 200 points is created so that when inferring a conclusion from antecedents there is no loss of resolution.

```
diagnosis = ct.Consequent(np.arange(0, 200, 1),
'diagnosis', defuzzify_method=defuzzify_method
```

E. *Feature Statistics*

As domain expertise is provided by data mining, statistics are calculated for each feature from the TRP, filtered by target class, in readiness for shaping the antecedent MFs. This includes deriving mean and SD for GAUSSMF, the *min*, *25th/75th quartiles* and *max* for the 4 vector TRAPMF, and KDE peak for the apex of TRIMF. As the MFs are defined only by TRP distribution, the min/max stats are adjusted to ensure capture and grading of any unseen, extreme values.

```
s['std0']  = feature_std(a, X_y_train, 0)
s['min0']  = feature_min(a, X_y_train, 0) - (s['std0'] * 1.1)
s['mean0'] = feature_mean(a, X_y_train, 0)
s['q251']  = feature_quantile(a, X_y_train, 1, 0.25)
s['pke1']  = feature_kde_peak(a, X_y_train, 1)

def feature_std(feat, df, target):
    return df.loc[df['Diagnosis'] == target, feat].std()

def feature_mean(feat, df, target):
    return df.loc[df['Diagnosis'] == target, feat].mean()

def feature_quantile(feat, df, target, q):
    return df.loc[df['Diagnosis'] == target, feat].quantile(q)

def feature_kde_peak(feat, df, target):
    kernel = stats.gaussian_kde(df.loc[df['Diagnosis']
       == target, feat])
    universe = np.linspace(df[feat].min(), df[feat].max(), num=200)
    kernel = kernel(universe)
    return universe[np.argsort(kernel)[-1]]
```

F. *Antecedent Membership Functions*

A membership function maps each input feature *x* crisp value to a grade in the interval [0, 1] to represent membership of fuzzy linguistic terms *x-low* and *x-high*. The mapping between crisp value and grade is defined by the shape of the MF assigned to test scenario antecedents, configured using JSON-like syntax as shown below:

```
{'Ant': [{'PerimeterMax': {'mf': {'low': 'gaussmf',
                                  'high': 'gaussmf'}}},
         {'ConcavePointsMax': {'mf': {'low': 'gaussmf',
                                      'high': 'gaussmf'}}}]}
```

This study supports shaping of GAUSSMF, TRIMF, TRAPMF, SMF and ZMF MFs [18] using collected statistics. As an example, the

assignment of TRAPMF to created antecedent term *PerimeterMax-low* begins with:

```
ant[a][v[0]] = getattr(self, 'mf_' + v[1])(a, v[0], s)
```

which dynamically invokes a method named after the configured MF, with *mf_* prefix, as shown below. The skfuzzy MF generator TRAPMF is called to shape a trapezoid using *PerimeterMax* class 0 statistics as vectors. An array of membership grade elements for each of the 200 universe points is returned.

```
def mf_trapmf(a, t, s):
    t = str(transform_class_to_target(t))
    return fz.trapmf(ant[a].universe,[s['min' + t],
      s['q25' + t], s['q75' + t],s['max' + t]])
```

Fig. 14 shows feature *ConcavePointsMax* KDE by target class *benign* represented as MF term *low* using TRIMF (as KDE peak determines apex) modelling the linguistic term "*ConcavePointsMax for benign samples should be lower values around 0.06*". Term *high* by GAUSSMF shaped by mean and SD represents linguistic term "*ConcavePointsMax for malignant samples should be higher values around 0.19*". Similarly, Fig. 15 shows TRAPMF used to represent the linguistic term "*PerimeterMax for malignant samples should be higher values approximately between 110 and 160*". The MFs bear a resemblance to *PerimeterMax* distribution, although first glance suggests a greater intersection of MF terms than corresponding target classes (Fig. 12). It should be re-emphasised that it is the TRP distribution and statistics drawn from it that defines the overlap and fuzziness between terms.

As a generalisation, this mapping of input value on the x axis to membership grade on the y axis for each term better supports dealing with complex problems such as high accuracy classification thanks to this non-linear modelling capability, although empirical benchmarking of accuracy and sensitivity will best measure this configuration and overall approach. Fig. 15 is a particularly interesting scenario for grading lower *PerimeterMax* values between 45 and 110, with the left boundaries of both terms almost parallel, warranting further test investigation of rule assembly and MF choice. *Appendix D* provides plots for all antecedent MFs tested.

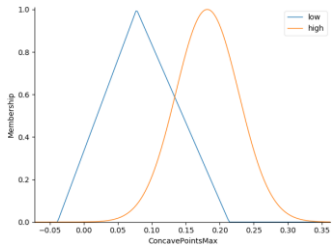

Fig. 14 *ConcavePointsMax* term MF representation

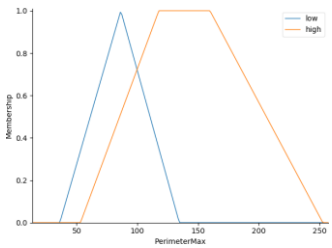

Fig. 15 *PerimeterMax* term MF representation

G. Fuzzifier

The fuzzifier uses antecedent MFs to define how raw, crisp input values are graded between 0 and 1 to indicate the fuzziness of antecedent terms, in this FIS *low* and *high*. As an example, consider *A* as the universe of discourse of *ConcavePointsMax* where x is 0.086 and $\mu_{\alpha-low}(x)$ formally notates the degree of membership of x in *A* term *low*. Recalling *A* consists of 200 points between min and max, to fuzzify crisp value 0.086 the point closest in value to x is interpolated with the MF grade on the y axis. Therefore, with the MFs shown in Fig. 14, the membership grade of *ConcavePointsMax low* = 0.93 and *high* = 0.12. For *PerimeterMax* where x is 76.51, *low = 0.80* and *high = 0.36* (Fig. 15). Fuzzifying the full *PerimeterMax* universe would result in the return of a fuzzy set as a "*class of objects with a continuum of grades of membership*" [7] according to the MF specified.

H. Consequent Membership Functions

The consequent MF grading a tumour sample as "*how benign*" or "*how malignant*" through FIS diagnosis is created in similar fashion to antecedent MFs via a call to the skfuzzy MF generator. However, the shape is not data driven by feature distribution, but fixed in configuration as follows, and applies to all test scenarios.

```
{'Consequent': [{'Diagnosis':
        {'mf': [{'benign': ['zmf', [10, 170]]},
                {'malignant': ['smf', [100, 200]]}]}}]}
```

Many variations of consequent term MF shaping were tested, including with gaussian and trapezoidal functions, scoring their performance on TEP classification accuracy after crisp to binary transformation. The final configuration uses z and s functions to grade benign and malignant respectively (Fig. 16), their choice, shape and point of overlap around *diagnosis*=126 determined by initial TEP prediction performance. As supported by testing results in *Section IV*, the author believes the areas of term separation and overlap fit well with observed feature distribution where, for example, *PerimeterMax* grouped by target exhibits less class intersection than *SmoothnessMax*. The full universe of discourse for diagnosis is covered by these z and s curves, ensuring extreme outliers are captured. Fig. 17 shows an example fuzzy grading of the consequent terms where *benign* has a membership value of 0.80 and *malignant* 0.12, with defuzzification resulting in a crisp output of 62.61 as indicated by the black line.

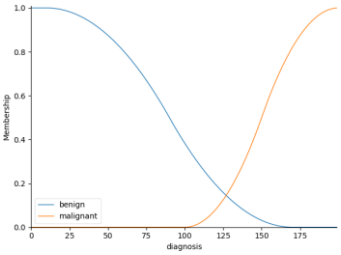

Fig. 16 *Diagnosis* consequent MF terms

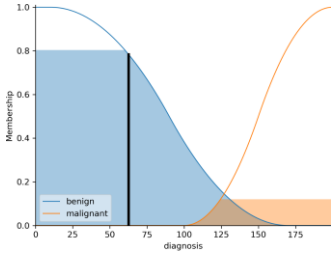

Fig. 17 *Diagnosis* term decision grading (centroid)

I. Fuzzy Rule Base

The knowledge base is compiled as a collection of fuzzy implication rules, each beginning with a specification of the *IF* antecedent condition, listing the included features and their fuzzy term (e.g. *PerimeterMax-low*). Although this FIS implementation supports antecedent relationship by intersection (logical *AND*) or union (logical OR), only intersection is used given comprehensive feature pruning done to remove highly correlated inputs and substantial distribution analysis supports informed simultaneous feature inclusion. The *IF* specification determines the *THEN* consequent fuzzy term to be graded, *Diagnosis-benign* or Diagnosis-malignant. The concept was first introduced by Mamdani in 1974 as a "*set of rules expressed as fuzzy conditional statements*" [19], originally tested in the control of plant systems including model steam engines.

The 5 rule sets (RS) presented in Table I form the core of scenarios designed to test classification accuracy. Each rule set comprises 2 rules, one to grade each consequent term. Observe that *benign* is determined only by *low* antecedent terms, and *malignant* only by *high*. This again is due to the nature of the feature distribution where KDE plots by target class show benign density on the lower end of the value range for every selected feature.

RS1 pairs the 2 features most important to target label *Diagnosis* and aims to provide a classification accuracy reference by which other RS can be benchmarked. It should indicate whether modifying this simple *IF* condition with additional input(s) improves predictive performance or only adds degenerative noise.

RS2 is designed to test the effectiveness of the feature selection process by using all in the antecedent. Performance worse than RS1 may identify features that remain irrelevant to the target, are noisy, or that feature and/or target data anomalies are present.

RS3 looks to supplement RS1 with the 3rd strongest feature *AreaSe*, to see whether performance can be optimised further with classification signals that may be present, as it has the most distinctive KDE of all (Fig. 11), especially for benign tumours.

RS4 includes all features except the weakest in terms of feature importance to target, *SmoothnessMax*, and will provide a useful comparison both with the full complement RS2 and going one feature more than RS3.

It is particularly interesting to include only *TextureMean* and *SmoothnessMax* in RS5, as despite being the 2 weakest features with regard to target importance they both exhibit the greatest KDE by

target density overlay, and therefore could well be features better suited to fuzzy modelling over other ML models.

TABLE I
CORE RULE SETS

Rule Set	If Perimeter Max	And if Concave Points Max	And if AreaSe	And if Texture Mean	And if Smoothness Max	Then Diagnosis
1	low high	low high				benign malignant
2	low high	low high	low high	low high	low high	benign malignant
3	low high	low high	low high			benign malignant
4	low high	low high	low high	low high		benign malignant
5				low high	low high	benign malignant

The authors of [20] stated that the "*human-like representation of rules gives FLSs their subjectivity and also their interpretability*". The rules in the above table clearly exhibit these 2 characteristics as their concepts are easy to understand, and with feature boundaries not defined by precise numerical values but described by words with *low* and *high* terms inheriting the shaping qualities of non-linear MFs. As Zadeh stated in his seminal paper on fuzzy sets that "*such a framework provides a natural way of dealing with problems in which the source of imprecision is the absence of sharply defined criteria of class membership"* [7]. Further supported by the ability to visualise MF terms, this FIS solution is easily explainable unlike other machine learning solutions often considered black box.

An example configuration for RS1 is given below, which is parsed then added to the rule base by instantiation of the rule with associated antecedent and consequent, via the Skfuzzy framework controller.

```
{'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low',
    '-op': '&', 'Diagnosis': 'benign'},
           {'PerimeterMax': 'high', 'ConcavePointsMax': 'high',
              '-op': '&',
            'Diagnosis': 'malignant'}]}
..
r = ct.Rule(antecedents, consequent=consequent, label=label)
```

J. *Inference Engine*

As all configured rules link terms with *AND*, the Skfuzzy inference engine uses Mamdani logic to evaluate and truncate the combined antecedent terms using the t-norm *min(x,y)* implication operator, implemented in Python with function *numpy.fmin* [21]. With the example fuzzy grades of terms *low* and *high* for *ConcavePointsMax* and *PerimeterMax* outlined earlier in *G*, the consequent terms defined in the 2 rules are reduced to single values:

$$Diagnosis\text{-}benign = min(0.93, 0.80) = 0.80$$
$$Diagnosis\text{-}malignant = min(0.12, 0.36) = 0.12$$

These rule outputs are then aggregated into a *Diagnosis* fuzzy set then applied to the consequent MFs during defuzzification.

K. *Defuzzifier*

Typically, in FIS implementations a crisp output is required, whether it is to drive robot motor velocity, control traffic signals or support binary classification. The process of transforming membership grades to a crisp output is known as defuzzification. With the standard Skfuzzy API (new) the method controlling defuzzification is fixed to centroid, therefore the author made minor coding enhancements to Skfuzzy as highlighted in *Appendix G.* This allowed alternatives to be tested that includes *bisector*, *mom* (mean of maximum), *som* (min of maximum) and *lom* (max of maximum). The centroid method takes the centre of area under the curve, bisector divides it into 2 equal regions, while mom, som and lom "*key off the maximum value assumed by the aggregate MF*" [22].

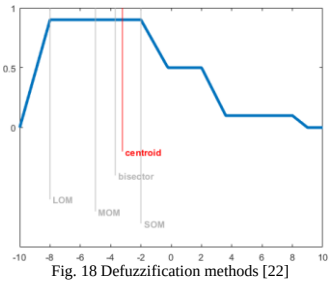

Fig. 18 Defuzzification methods [22]

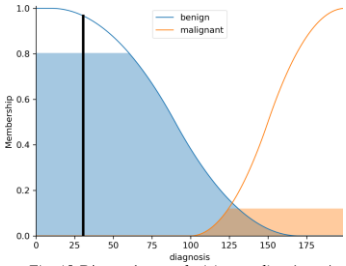

Fig. 19 *Diagnosis* term decision grading (mom)

The example tumour sample is *benign* to degree of 0.80, and *malignant* to degree of 0.12. These degrees are mapped to the consequent MFs, where Mamdani logic takes the *max(0.80, 0.12)* and passes it through a defuzzification method to generate a crisp output, for example using centroid (Fig. 17), or mom (Fig. 19, crisp = 30.48) to slice the aggregate *Diagnosis* set as shown by the vertical line.

L. *Diagnosis Output*

The FIS, named *diagnose* as it emulates a real clinical assessment, is designed to decide categorically if a breast tumour sample is benign or malignant. It is this binary classification that gives unambiguous meaning to its output which medical practitioners can use in next steps of patient care. The selected features of each unseen TEP sample are used to make a crisp diagnosis which if below a crisp threshold (CT) is classified as benign, otherwise malignant.

```
diagnose = ct.ControlSystemSimulation(system)
..
for di, dr in X_test.iterrows():
  for si, sv in dr.iteritems():
    diagnose.input[si] = sv
..
diagnose.compute()
..
crisp_to_binary = 0 if diagnose.output['diagnosis'] <
crisp_threshold else 1
```

A range of CT were tested as detailed in Section IV. The crisp output may be of value as indicators of the level of tumour development and support prioritised care, although this would absolutely need to be validated by domain expertise.

IV. EVALUATION

A. *Method*

The TRP of 398 samples with known target tumour classification is used to supervise the developing and fitting of a range of FIS variants to the data, in addition to default configuration and fitting of LR, DT and RFC models. The 171 samples in TEP, unseen during model assembly, are used to evaluate classifier performance, with each model's prediction of tumour class compared with known ground truth (64 malignant, 107 benign) to support empirical predictive measurement. True positives (TP) is the number of correctly predicted malignant tumours. True negatives (TN) is the number of correctly predicted benign tumours. False positives (FP) is the number of benign predicted as malignant. False negatives (FN) is the number of malignant predicted as benign.

To support the reduction of mortality due to death by breast cancer, with consequences of incorrect classification of malignant as benign far higher for the patient than benign as malignant, this study will score and prioritise ranking of model performance by sensitivity (SEN). This is a ratio of the number of malignant samples correctly predicted, in other words the proportion of TP. However, a secondary consideration must be given to overwhelmed medical systems and prevent, if possible, the allocation of treatment resources where it is not required. Therefore accuracy (ACC), the ratio of correct predictions to total number of predictions, will assess a model's ability to correctly differentiate between benign and malignant samples. ACC follows SEN in model ranking order due both to identification of TP a critical priority and class malignant being the underrepresented, minority class, meaning bias towards TN if ACC performance was 1st priority.

Finally, 3rd in model ranking order is the metric specificity (SPE) which proportionally measures the ability to correctly predict benign samples, or the proportion of TN, and further supports evaluation as it is the opposite of SEN. The metrics are calculated as follows:

$$SEN = TP / (TP+FN)$$
$$ACC = (TP+TN) / (TP+TN+FP+FN)$$
$$SPE = TN / (TN+FP)$$

B. *Tuning*

Model tuning to extract best possible classification performance is critical given that misclassification of tumour risks death by breast cancer due to treatment only at late-stage. This motivates optimisation of the FIS design by extensive testing of varying configurations

including different antecedent MF shapes, defuzzification methods (DM) and CT.

Table II presents the core test configuration. It comprises each of the 5 knowledge base RS antecedent feature combinations presented as the core rule set earlier, with 3 variations on MF shape assignment to *low* and *high* terms. As feature distribution analysis shows a primarily gaussian and normal distribution this MF is first tested. The 2nd MF variant is overlaid z and s shaped functions that guarantee inclusion of extreme outliers, with foot and ceiling set by distribution min and KDE peak. The 3rd variant per RS takes a more heuristic approach, guiding MF selection by way of taking learnings from feature distribution analysis and fitting functions that look to be good alternatives to pure gaussian, for example TRIMF for the highly peaked *AreaSe* class benign.

TABLE II
CORE TEST CONFIGURATION

Test Id	Rule Set	Term	If Perimeter Max	And if Concave Points Max	And if AreaSe	And if Texture Mean	And if Smoothness Max
1	1	low	gaussmf	gaussmf			
		high	gaussmf	gaussmf			
2		low	zmf	zmf			
		high	smf	smf			
3		low	trimf	trimf			
		high	trapmf	gaussmf			
4	2	low	gaussmf	gaussmf	gaussmf	gaussmf	gaussmf
		high	gaussmf	gaussmf	gaussmf	gaussmf	gaussmf
5		low	zmf	zmf	zmf	zmf	zmf
		high	smf	smf	smf	smf	smf
6		low	trimf	trimf	trimf	trimf	trimf
		high	trapmf	gaussmf	gaussmf	trimf	trimf
7	3	low	gaussmf	gaussmf	gaussmf		
		high	gaussmf	gaussmf	gaussmf		
8		low	zmf	zmf	zmf		
		high	smf	smf	smf		
9		low	trimf	trimf	trimf		
		high	trapmf	gaussmf	gaussmf		
10	4	low	gaussmf	gaussmf	gaussmf	gaussmf	
		high	gaussmf	gaussmf	gaussmf	gaussmf	
11		low	zmf	zmf	zmf	zmf	
		high	smf	smf	smf	smf	
12		low	trimf	trimf	trimf	trimf	
		high	trapmf	gaussmf	gaussmf	trimf	
13	5	low				gaussmf	gaussmf
		high				gaussmf	gaussmf
14		low				zmf	zmf
		high				smf	smf
15		low				trimf	trimf
		high				trimf	trimf

These 15 core MF configurations were each tested with the 5 available DM (centroid, bisector, mom, som, lom), totalling 75 FIS implementation variants. Initial testing of consequent *Diagnosis*, with a universe of 200 points mapped by z and s functions to grade *benign* and *malignant*, indicated a CT sweet point of ~126. However, to further fine tune the consequent MF overlap with this additional "*synthetic*" conversion a CT in range 90-130 was tested, resulting in a final total of 3075 FIS configurations tested. An additional 15 tests for the LR, DTC and RFC ML models with default configuration covering the 5 feature combination variants were included.

C. Results

A condensed set of 90 results from 3075 tests of FIS variants and alternative ML classifier models, computed in 555s, is presented in table III. The highest ranking FIS variant per RS, MF group (MFgrp) and DM is included. Column *Test_Ref* identifies the RS number, MF assignment (1/4/7/10/13 for gaussian, 2/5/8/11/14 for zmf and smf, 3/6/9/12/15 for heuristic mix), classifier type, CT number and DM. As stated, ranking of classifier performance is prioritised by SEN/ACC/SPE.

The highest performing models with equal scoring metrics are *rs1_id1_FIS_ct103_bisector* and *rs1_id1_FIS_ct103_centroid* with SEN=1, ACC=0.936 and SPE=0.897. They were able to correctly predict all malignant and 89.7% of benign tumours, misclassifying 11 of 107 benign as malignant as shown in the confusion matrix (Fig. 20). This is an improvement of 0.028 on SPE than the next best FIS variant with same RS (rank 40) and 0.047 better than the top ranking FIS with alternative RS (*rs3_id7_FIS_ct103_bisector*, rank 98).

TABLE III
TEST RESULTS (CONDENSED)

Rank	Clf	RS	Test Ref	MFgrp	DM	Classification Table				Score		
						TP	FP	TN	FN	SEN	ACC	SPE
1	FIS	1	rs1_id1_FIS_ct103_bisector	gauss	bisector	64	11	96	0	1	0.936	0.897
13	FIS	1	rs1_id1_FIS_ct103_centroid	gauss	centroid	64	11	96	0	1	0.936	0.897
40	FIS	1	rs1_id1_FIS_ct90_mom	gauss	mom	64	14	93	0	1	0.918	0.869
81	FIS	1	rs1_id1_FIS_ct90_som	gauss	som	64	14	93	0	1	0.918	0.869
98	FIS	3	rs3_id7_FIS_ct103_bisector	gauss	bisector	64	16	91	0	1	0.906	0.85
110	FIS	3	rs3_id7_FIS_ct103_centroid	gauss	centroid	64	16	91	0	1	0.906	0.85
115	FIS	4	rs4_id10_FIS_ct109_bisector	gauss	bisector	64	17	90	0	1	0.901	0.841
122	FIS	4	rs4_id10_FIS_ct105_centroid	gauss	centroid	64	17	90	0	1	0.901	0.841
135	FIS	1	rs1_id3_FIS_ct92_lom	mix	lom	64	18	89	0	1	0.895	0.832
151	FIS	1	rs1_id1_FIS_ct130_lom	gauss	lom	64	20	87	0	1	0.883	0.813
152	FIS	3	rs3_id7_FIS_ct90_mom	gauss	mom	64	20	87	0	1	0.883	0.813
193	FIS	3	rs3_id7_FIS_ct90_som	gauss	som	64	20	87	0	1	0.883	0.813
215	FIS	2	rs2_id4_FIS_ct103_bisector	gauss	bisector	64	21	86	0	1	0.877	0.804
234	FIS	2	rs2_id4_FIS_ct102_centroid	gauss	centroid	64	21	86	0	1	0.877	0.804
246	FIS	4	rs4_id10_FIS_ct90_mom	gauss	mom	64	21	86	0	1	0.877	0.804
287	FIS	4	rs4_id10_FIS_ct90_som	gauss	som	64	21	86	0	1	0.877	0.804
332	FIS	3	rs3_id9_FIS_ct92_lom	mix	lom	64	24	83	0	1	0.86	0.776
338	FIS	2	rs2_id4_FIS_ct90_mom	gauss	mom	64	25	82	0	1	0.854	0.766
379	FIS	2	rs2_id4_FIS_ct90_som	gauss	som	64	25	82	0	1	0.854	0.766
400	FIS	4	rs4_id12_FIS_ct92_lom	mix	lom	64	27	80	0	1	0.842	0.748
403	FIS	3	rs3_id7_FIS_ct130_lom	gauss	lom	64	27	80	0	1	0.842	0.748
435	FIS	4	rs4_id10_FIS_ct130_lom	gauss	lom	64	31	76	0	1	0.819	0.71
448	FIS	2	rs2_id6_FIS_ct92_lom	mix	lom	64	34	73	0	1	0.801	0.682
468	FIS	2	rs2_id4_FIS_ct130_lom	gauss	lom	64	38	69	0	1	0.778	0.645
550	FIS	1	rs1_id2_FIS_ct129_som	zs	som	64	67	40	0	1	0.608	0.374
551	FIS	2	rs2_id5_FIS_ct129_som	zs	som	64	67	40	0	1	0.608	0.374
552	FIS	3	rs3_id8_FIS_ct129_som	zs	som	64	67	40	0	1	0.608	0.374
553	FIS	4	rs4_id11_FIS_ct129_som	zs	som	64	67	40	0	1	0.608	0.374
605	FIS	1	rs1_id2_FIS_ct130_centroid	zs	centroid	64	84	23	0	1	0.509	0.215
609	FIS	1	rs1_id2_FIS_ct119_bisector	zs	bisector	64	85	22	0	1	0.503	0.206
689	FIS	1	rs1_id2_FIS_ct90_mom	zs	mom	64	89	18	0	1	0.48	0.168
768	FIS	1	rs1_id2_FIS_ct130_lom	zs	lom	64	92	15	0	1	0.462	0.14
778	FIS	3	rs3_id8_FIS_ct127_bisector	zs	bisector	64	95	12	0	1	0.444	0.112
782	FIS	3	rs3_id8_FIS_ct115_centroid	zs	centroid	64	95	12	0	1	0.444	0.112
871	FIS	3	rs3_id8_FIS_ct90_mom	zs	mom	64	97	10	0	1	0.433	0.093
943	FIS	5	rs5_id14_FIS_ct130_centroid	zs	centroid	64	99	8	0	1	0.421	0.075
944	FIS	3	rs3_id8_FIS_ct130_lom	zs	lom	64	99	8	0	1	0.421	0.075
949	FIS	4	rs4_id11_FIS_ct91_bisector	zs	bisector	64	100	7	0	1	0.415	0.065
989	FIS	4	rs4_id11_FIS_ct93_centroid	zs	centroid	64	100	7	0	1	0.415	0.065
1047	FIS	4	rs4_id11_FIS_ct90_mom	zs	mom	64	100	7	0	1	0.415	0.065
1110	FIS	4	rs4_id11_FIS_ct130_lom	zs	lom	64	101	6	0	1	0.409	0.056
1114	FIS	5	rs5_id14_FIS_ct122_bisector	zs	bisector	64	102	5	0	1	0.404	0.047
1162	FIS	2	rs2_id5_FIS_ct127_centroid	zs	centroid	64	103	4	0	1	0.398	0.037
1187	FIS	5	rs5_id14_FIS_ct126_som	zs	som	64	103	4	0	1	0.398	0.037
1192	FIS	2	rs2_id5_FIS_ct90_bisector	zs	bisector	64	104	3	0	1	0.392	0.028
1296	FIS	2	rs2_id5_FIS_ct90_mom	zs	mom	64	104	3	0	1	0.392	0.028
1375	FIS	5	rs5_id14_FIS_ct102_lom	zs	lom	64	105	2	0	1	0.386	0.019
1404	FIS	2	rs2_id5_FIS_ct126_lom	zs	lom	64	105	2	0	1	0.386	0.019
1414	FIS	5	rs5_id14_FIS_ct90_mom	zs	mom	64	105	2	0	1	0.386	0.019
1734	FIS	1	rs1_id3_FIS_ct92_centroid	mix	centroid	62	10	97	2	0.969	0.93	0.907
1738	FIS	3	rs3_id9_FIS_ct92_centroid	mix	centroid	62	11	96	2	0.969	0.924	0.897
1739	FIS	4	rs4_id12_FIS_ct92_centroid	mix	centroid	62	11	96	2	0.969	0.924	0.897
1787	FIS	2	rs2_id6_FIS_ct93_centroid	mix	centroid	62	14	93	2	0.969	0.906	0.869
1937	FIS	1	rs1_id3_FIS_ct90_bisector	mix	bisector	60	8	99	4	0.938	0.93	0.925
1939	FIS	3	rs3_id9_FIS_ct91_bisector	mix	bisector	60	8	99	4	0.938	0.93	0.925
1940	FIS	4	rs4_id12_FIS_ct91_bisector	mix	bisector	60	8	99	4	0.938	0.93	0.925
1947	FIS	1	rs1_id3_FIS_ct90_mom	mix	mom	60	8	99	4	0.938	0.93	0.925
1988	DTC	2	rs2_DTC	n/a	n/a	60	8	99	4	0.938	0.93	0.925
1989	RFC	1	rs1_RFC	n/a	n/a	60	8	99	4	0.938	0.93	0.925
1990	RFC	4	rs4_RFC	n/a	n/a	60	8	99	4	0.938	0.93	0.925
1991	FIS	1	rs1_id3_FIS_ct90_som	mix	som	60	8	99	4	0.938	0.93	0.925
2029	FIS	3	rs3_id9_FIS_ct90_mom	mix	mom	60	9	98	4	0.938	0.924	0.916
2030	FIS	4	rs4_id12_FIS_ct90_mom	mix	mom	60	9	98	4	0.938	0.924	0.916
2111	FIS	3	rs3_id9_FIS_ct90_som	mix	som	60	9	98	4	0.938	0.924	0.916
2112	FIS	4	rs4_id12_FIS_ct90_som	mix	som	60	9	98	4	0.938	0.924	0.916
2143	DTC	4	rs4_DTC	n/a	n/a	60	10	97	4	0.938	0.918	0.907
2144	FIS	2	rs2_id6_FIS_ct91_bisector	mix	bisector	60	11	96	4	0.938	0.912	0.897
2159	FIS	2	rs2_id6_FIS_ct90_mom	mix	mom	60	12	95	4	0.938	0.906	0.888
2200	FIS	2	rs2_id6_FIS_ct90_som	mix	som	60	12	95	4	0.938	0.906	0.888
2277	DTC	1	rs1_DTC	n/a	n/a	59	12	95	5	0.922	0.901	0.888
2316	RFC	2	rs2_RFC	n/a	n/a	58	5	102	6	0.906	0.936	0.953
2328	RFC	3	rs3_RFC	n/a	n/a	58	6	101	6	0.906	0.93	0.944
2377	DTC	3	rs3_DTC	n/a	n/a	58	12	95	6	0.906	0.895	0.888
2564	FIS	5	rs5_id15_FIS_ct90_centroid	mix	centroid	57	49	58	7	0.891	0.673	0.542
2664	LR	1	rs1_LR	n/a	n/a	54	2	105	10	0.844	0.93	0.981
2670	FIS	5	rs5_id13_FIS_ct90_centroid	gauss	centroid	54	37	70	10	0.844	0.725	0.654
2677	LR	2	rs2_LR	n/a	n/a	53	6	101	11	0.828	0.901	0.944
2678	LR	3	rs3_LR	n/a	n/a	53	6	101	11	0.828	0.901	0.944
2679	LR	4	rs4_LR	n/a	n/a	53	6	101	11	0.828	0.901	0.944
2680	FIS	5	rs5_id13_FIS_ct94_lom	gauss	lom	53	53	54	11	0.828	0.626	0.505
2701	FIS	5	rs5_id13_FIS_ct90_mom	gauss	mom	51	31	76	13	0.797	0.743	0.71
2742	FIS	5	rs5_id13_FIS_ct90_som	gauss	som	51	31	76	13	0.797	0.743	0.71
2812	FIS	5	rs5_id15_FIS_ct90_lom	mix	lom	49	33	74	15	0.766	0.719	0.692
2816	FIS	5	rs5_id13_FIS_ct90_bisector	gauss	bisector	48	27	80	16	0.75	0.749	0.748
2841	FIS	5	rs5_id15_FIS_ct90_mom	mix	mom	48	29	78	16	0.75	0.737	0.729
2882	FIS	5	rs5_id15_FIS_ct90_som	mix	som	48	29	78	16	0.75	0.737	0.729
2957	DTC	5	rs5_DTC	n/a	n/a	42	28	79	22	0.656	0.708	0.738
2966	RFC	5	rs5_RFC	n/a	n/a	39	25	82	25	0.609	0.708	0.766
3018	FIS	5	rs5_id15_FIS_ct91_bisector	mix	bisector	27	11	96	37	0.422	0.719	0.897
3025	LR	5	rs5_LR	n/a	n/a	23	12	95	41	0.359	0.69	0.888

To give these metric variances introduced by changes in FIS configuration real world context, that's a difference of 3 and 5 patients respectively where care can be effectively focused on needs and critical tumour treatment resources made available only for those with a malignant diagnosis. This is considered a very satisfactory result given such accuracy in both SEN and SPE was achieved with just 2 antecedent features, albeit *ConcavePointsMax* and *PerimeterMax* previously identified as most important.

Furthermore, it is proposed they also work well forming antecedent terms as they are normally distributed with the gaussian bell shape a good MF fit that plays a crucial role in extracting the very best performance. There is good separation of classes at higher KDE densities yet enough MF overlap to incorporate and grade uncertainty where lower and higher values of both features are present in both tumour types (Figs. 32 & 48, 36 & 51 *Appendices C/D*). These

normally distributed features mapped by gaussian MF also explains why bisector and centroid defuzzification methods achieve best, and comparable results where they are used i.e. RS 1-4. This result highlights how robust distribution analysis can contribute to the design of a simple, minimised rule base that meets the demands of the problem domain without replication of rules or use of correlated and therefore redundant features.

For comparison, the non-FIS models using the same 2 features in RS1 misclassified true benign 2, 12 and 8 times, and misclassified true malignant 10, 5 and 4 times, for LR (rank 2664), DTC (rank 2277) and RFC (rank 1989) respectively. The top FIS delivers superior performance on the critical classification of malignant tumours.

Adding *AreaSe* antecedent terms to *ConcavePointsMax* and *PerimeterMax* in RS3 (rank 98) achieved SEN=1 but increased FP rates to 16, proposed due to the high skew and kurtosis of this feature causing a significant overlap of term *high* over *low* (Fig. 47 *Appendix D*). It is believed that by incorporating *AreaSe* in RS2-4, a detrimental impact on *SPE* occurs, preventing improvement over RS1.

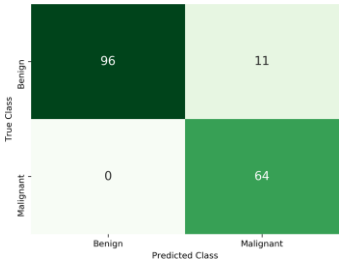

Fig. 20 CM test rs1_id1_FIS_ct103_bisector

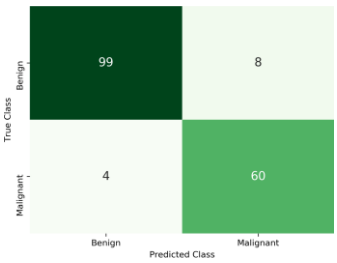

Fig. 21 CM test rs2_DTC

The lom DM attains respectable results with SEN=1 and SPE=0.832 when mapping *ConcavePointsMax* and *PerimeterMax* to a mix of triangular, trapezoidal, and gaussian functions (*rs1_id3_FIS_ct92_lom*, rank 135), classifying all malignant samples correctly but 7 more benign as malignant over the top ranking models. As a gaussian MF maps *ConcavePointsMax* term *high* in both the fully gaussian, id1 variant and here in this id3 MF mix, the suggested cause of higher misclassification is greater MF overlap of the *PerimeterMax* antecedent *high* term TRAPMF (Fig. 53, *Appendix D*) that maps larger *high* term grades for a given input value than is interpolated with a gaussian function and therefore results in the *high* term grade winning during *Diagnosis* fuzzy set aggregation. For e.g., a *PerimeterMax* value of 100 is graded as ~0.75 with TRAPMF, and ~0.35 with GAUSSMF. This further supports the earlier claim gaussian MFs fit well and support both distinct class separation and grade the uncertainty of value-class crossover in a smoother, non-linear curve. The results of some FIS models achieving maximum SPE=1 using an MF mix (e.g. *rs1_id3_FIS_ct116_centroid*, rank 2565) which uses TRIMF for *low* terms does suggest possible opportunities to improve on the misclassification of 11 FP by the top model using gaussian, by further tuning of MF shape and parameters.

Despite all FIS test variants configured with z and s antecedent MFs achieving SEN=1, they do so at an unacceptable cost to SPE rate. For e.g. *rs1_id2_FIS_ct129_som* (rank 550) misclassifies 67 benign samples as malignant, despite using RS1 to leverage the 2 most important features. Such a high error rate can be explained by a review of the MF mapping for *PerimeterMax* (Fig. 52 *Appendix D*). There is very little overlap in terms. The universe of discourse for *PerimeterMax-low* is too narrow with the foot of the z function stopping short, pointing to a configuration error in the ZMF function using KDE peak (rather than max for e.g.). A *PerimeterMax* value of 100 results in an empty set for *low*, which is not the case for gaussian and triangular MFs.

RS5 uses the weakest features, *TextureMean* and *SmoothnessMax*, with greatest density by target overlay. Excluding z and s MFs, the best performing model is *rs5_id15_FIS_ct90_centroid*, using TRIMF/TRAPMF and ranking 2564th. With SEN=0.891 and SPE=0.542 the model misclassifies 49 true benign and 7 true malignant samples. While unacceptable, it does however outperform the fully gaussian *rs5_id13_FIS_ct90_centroid* variant that ranks 2670th with SEN=0.844, SPE=0.654, FP=37 and FN=10. The author would hypothesize TRAPMF for both *low* and *high* terms better manages the extreme low values of *TextureMean* present in both benign and malignant samples (Fig. 59 *Appendix D*), and hence increases the fuzziness of this uncertainty over the greater separation of gaussian grading at the low end (Fig. 57 Appendix D).

There is of course a clear correlation between best configured CT and combination of MFgrp/DM. CT 102/103 works best for gaussian with bisector/centroid, CT 90 for gaussian with mom, CT 130 for gaussian with lom, and CT 92 for *mix* with lom.

The best non-FIS, *rs2_DTC*, requiring all 5 fields to rank 1988th, obtains a higher SPE of 0.925 than the top FIS, however with a SEN rate of 0.938 critically misclassifies 4 malignant tumours as benign (Fig. 21). It's ACC of 93% is now used as a benchmarking threshold to statistically plot the success of configuration settings where FIS model variants attain at least this level of accuracy. The model rank is subtracted from 3090 (number of tests) to assign a rank credit i.e. *rs1_id1_FIS_ct103_bisector* = 3089, which awards configurations making the cut off and penalising those that don't. The credit is then summed for each setting. Fig. 22 shows RS1 is by far the most successful combination of input antecedents, whereas RS3 and RS4 are subjectively similar, and RS2 detrimentally impacted by the weakest feature *SmoothnessMax*. RS5 doesn't make the cut.

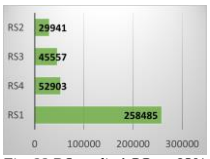

Fig. 22 RS credit ACC >= 93%

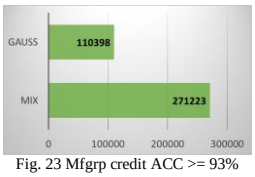

Fig. 23 Mfgrp credit ACC >= 93%

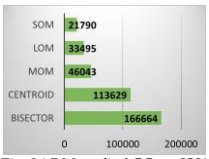

Fig. 24 DM credit ACC >= 93%

Fig. 23 is particularly interesting, presenting sum of rank credits by MF group. Z and s function tests don't make the ACC cut. Reemphasising all MF shaping is data driven, while gaussian MFs are used in the top ranking model, the heuristic mixing of MFs according to gut feel shows generally much higher accuracy success. 549 models using gaussian MFs achieve SEN of 1, whereas only 12 manage this for an MF mix and perform poorer than their gaussian counterparts in SPE. Model *rs3_id9_FIS_ct119_bisector* achieve what some may consider a highly performant SEN of 0.891, SPE of 0.991 and overall ACC of 0.953, resulting in FP of 1 and FN of 7. It is this misclassification of 7 patients with malignant tumours that justifies the penalisation of the model in this specific problem. It does however suggest promising and improved results are possible with further refinement of the mixed MF shaping properties. Rank credits summed by DM (Fig. 24) suggests bisector, and to a lesser extent centroid, are good default choices.

FIS computational runtime does not appear to be significantly impacted by any particular configuration variant e.g. bisector or centroid, with sub-second classification of TEP for all tests.

V. Conclusion

In this paper a FIS was proposed for the binary classification of breast tumours as benign (class 0) or malignant (class 1). The stated primary objective was maximum FIS model performance measured by sensitivity rate, or number of correctly predicted TP malignant samples. A secondary objective considers the efficient use of medical resources under constant strain and therefore also measures overall accuracy and specificity rate to ensure a model can effectively differentiate between classes. In total 3075 tests varying FIS configuration of antecedent MF, implication rules, DM and CT were made to extract optimal performance from the data and system. The best model using the antecedent features *ConcavePointsMax* and *PerimeterMax* mapped with gaussian MFs, defuzzified by bisector and crisp output transformed to binary at a threshold of 103 showed very effective prediction of tumour type, scoring a maximum SEN of 1, ACC of 0.936 and SPE of 0.897. This model shows accuracy comparable with other WDBCD classification work [10], and outperforms the best non-FIS variant with a 4 TP improvement.

The predictive performance comes from the ability to associate each antecedent and consequent term with non-linear functions that, by overlapping, can better work with the uncertainty of disease. A DTC also uses a set of *IF...THEN* rules, however it is not fuzzy. As the authors of [23] put it, "*Unlike conventional modelling, where a single*

model is used to describe the global behaviour of a system, fuzzy rule-based modelling is essentially a multi-model approach in which individual rules (where each rule acts like a 'local model') are combined to describe the global behaviour of the system" Other advantages include the transparency, simplicity and interpretability of its linguistic knowledge base. MFs are easy to visualize and change their shape.

The presented results justify the use of the data driven approach to feature set reduction and MF shaping. The system was able to correctly infer the tumour type in the TEP with maximum SEN and acceptable SPE, in sub-second response times, using a small knowledge base of 2 antecedents and 2 rules that did not sacrifice the primary objective of prediction performance for simplicity given its high classification accuracy across all metrics. As the expert knowledge including features and linguistic term shaping was guided by empirical data, the author did not need to be a problem domain expert, nor was the system design influenced by the varying subjectivity of real experts.

Varying the FIS with MF shaping and DM alternatives, even when using the same feature RS, e.g. RS1, can significantly impact performance, with SEN ranging 1 - 0.797, and ACC 0.936 – 0.392. This shows the importance of testing different configurations to find the best combination, which is challenging and thus automated here.

As the FIS is implemented in Python using the open source Skfuzzy package, and has low computing demands, it could be made available as an expert system running on inexpensive hardware. Furthermore, as the system is modelled in human terms, a web front-end to enable interaction with clinical resources would be straight-forward and might include taking advantage of the fuzzy internal nature of the system by presenting consequent term gradings and crisp output as risk levels. It should be noted that Skfuzzy has a significantly smaller user community with far fewer resources including examples and documentation available than for Matlab's fuzzy logic toolbox. It was satisfying but challenging to use, requiring modification to standard code, however the ability to debug the inference process does offer the opportunity for further understanding of FIS in general.

Maximum SPE rates merits further analysis of the heuristic MF mix approach to see if this helps resolve the FP misclassification of the top model using gaussian antecedent mapping. A new RS variant, explicitly excluding *AreaSe* due to its distribution characteristics, but including *ConcavePointsMax*, *PerimeterMax* and *TextureMean* is proposed, to see if more predicative performance can be extracted with more than 2 features. Fixing the shaping of z and s antecedent MFs for larger term overlap should also be done. Further work would also include hyperparameter optimisation of non-FIS models, for example increasing the number of tree estimators in RFC, to have a fairer like-for-like tuned model comparison.

Exploring the availability of Python FIS type-2 packages might offer greater performance still, as "*especially in modelling, using type-2 counterparts provides an extra degree of freedom which may, more often than not, improve the accuracy and the generalization capabilities of models*" [20]. Modelling the problem with an ANFIS (Adaptive Neuro-Fuzzy Inferencing System) approach, taking advantage of deep learning where errors are minimised through back propagation feedback, would provide a quick and useful alternative accuracy benchmark. For information the Python code used in dataset analysis is provided in *Appendix E*. The developed FIS solution is given in *Appendix F*. Full solution source code can also be found at:

https://github.com/corticalstack/fuzzy-system-breast-cancer-wisconsin

REFERENCES

[1] F. Basciftci and E. Avuclu, "An expert system design to diagnose cancer by using a new method reduced rule base," *Computer Methods and Programs in Biomedicine,* vol. 157, pp. 113-120, 2018.

[2] W. H. Organisation, "Breast Cancer," [Online]. Available: https://www.who.int/cancer/detection/breastcancer/en/index1.html. [Accessed 13 12 2019].

[3] W. C. R. Fund, "Breast Cancer Statistics," [Online]. Available: https://www.wcrf.org/dietandcancer/cancer-trends/breast-cancer-statistics. [Accessed 13 12 2019].

[4] C. R. UK, "Breast Cancer Statistics," [Online]. Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/breast-cancer. [Accessed 13 12 2019].

[5] K. Lie, Z. Chen, J. Wu, Y. Tan, L. Wang, Y. Yan, H. Zhang and J. Long, "Big Medical Data Decision-Making Intelligent System Exploiting Fuzzy Inference Logic for Prostate Cancer in Developing Countries," *IEEE Access,* vol. 7, pp. 2348-2363, 2018.

[6] UCI, "Breast Cancer Wisconsin (Diagnostic) Dataset," [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic). [Accessed 13 12 2019].

[7] L. Zadeh, "Fuzzy Sets," *Information Control,* vol. 8, no. 3, pp. 338-353, 1965.

[8] I. Scrobota, G. Baciut, A. G. Filip, B. Todor, F. Blaga and M. F. Baciut, "Application of Fuzzy Logic in Oral Cancer Risk Assessment," *Iran Journal of Public Health,* vol. 46, no. 5, pp. 612-619, 2017.

[9] A. Torres and J. J. Nieto, "Fuzzy Logic in Medicine and Bioinformatics," *Journal of Biomedicine and Biotechnology,* vol. 2006, pp. 1-7, 2006.

[10] M. Nilashi, O. Ibrahim, H. Ahmadi and L. Shahmoradi, "A knowledge-based system for breast cancer classification using fuzzy logic method," *Telematics and Informatics,* vol. 34, no. 4, pp. 133-144, 2017.

[11] J.-P. Boyd, "A neural network model for detecting intrusions or attacks on a computer network," 2018.

[12] scikit-learn, "sklearn.preprocessing.MinMaxScaler," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html. [Accessed 15 12 2019].

[13] Skfuzzy, "Skfuzzy," [Online]. Available: https://pythonhosted.org/scikit-fuzzy/. [Accessed 13 12 2019].

[14] Python.org, "Welcome to Python," [Online]. Available: https://www.python.org/.

[15] S. Overflow, "Stack Overflow Survey 2019," [Online]. Available: https://insights.stackoverflow.com/survey/2019. [Accessed 20 12 2019].

[16] G. scikit-fuzzy, "scikit-fuzzy," [Online]. Available: https://github.com/scikit-fuzzy/scikit-fuzzy. [Accessed 20 12 2019].

[17] Skfuzzy, "Skfuzzy Tipping Problem New API," [Online]. Available: https://scikit-fuzzy.readthedocs.io/en/latest/auto_examples/plot_tipping_problem_newapi.html.

[18] skfuzzy, "skfuzzy membership," [Online]. Available: https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.membership.html. [Accessed 19 12 2019].

[19] E. Mamdani, "Application of fuzzy algorithms for control of simple dynamic plant," *Proceedings Of The Institution Of Electrical Engineers,* vol. 121, no. 12, pp. 1585 - 1588, 1974.

[20] O. Obajemu, M. Mahfouf and J. W. F. Catto, "A New Fuzzy Modeling Framework for Integrated Risk Prognosis and Therapy of Bladder Cancer Patients," *IEEE Transactions on Fuzzy Systems ,* vol. 26, no. 3, pp. 1565-1577, 2018.

[21] SciPy.org, "numpy.fmin - NumPy v1.17 Manual," [Online]. Available: https://docs.scipy.org/doc/numpy/reference/generated/numpy.fmin.html. [Accessed 29 12 2019].

[22] Mathworks, "Defuzzification Methods - MATLAB," [Online]. Available: https://ch.mathworks.com/help/fuzzy/defuzzification-methods.html.

[23] D. Soria, J. M. Garibaldi, A. R. Green, D. G. Powe, C. C. Nolan, C. Lemetre, G. R. Ball and I. O. Ellis, "A quantifier-based fuzzy classification system for breast cancer patients," *Artificial Intelligence in Medicine,* vol. 58, no. 3, pp. 175-184, 2013.

APPENDIX A – EXPERIMENTATION SETUP

Software Windows 10 Professional (v18.3), PyCharm Professional 2018.3

Hardware Intel i7-5820K, Nvidia GeForce GTX 1080, 16GB RAM

Main Python Modules skfuzzy 0.4.1, numpy 1.16.1, pandas 0.24.1, scikit-learn 0.20.2, scipy 1.2.0, matplotlib 3.1.2, seaborn (0.9.0), network 2.0

APPENDIX B – FEATURE DISTRIBUTION STATISTICAL ANALYSIS AND SELECTION RANKING

Column	Type	0 Count	Distinct	Min	Mean	25%	50%	75%	Max	Std	Skew	Kurt	UniChi Rank	RFC Rank	Final Rank
ID	Integer	0	569	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a			
Diagnosis	String	0	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a			
RadiusMean	Float	0	456	6.98	14.12	11.70	13.37	15.78	28.11	3.524	0.942	0.846	1	6	
TextureMean	Float	0	479	9.71	19.29	16.17	18.84	21.80	39.28	4.30	0.65	0.75	2	12	4
PerimeterMean	Float	0	522	43.79	91.96	75.17	86.24	104.10	188.50	24.29	0.99	0.97	3	14	
AreaMean	Float	0	539	143.5	654.88	420.30	551.10	782.70	2501.0	351.91	1.64	3.65	4	9	
SmoothnessMean	Float	0	474	0.05	0.09	0.08	0.09	0.10	0.16	0.01	0.45	0.85			
CompactnessMean	Float	0	537	0.01	0.10	0.06	0.09	0.13	0.34	0.05	1.19	1.65		15	
ConcavityMean	Float	13	537	0.00	0.08	0.03	0.06	0.13	0.42	0.08	1.40	1.99	5	8	
ConcavePointsMean	Float	13	542	0.00	0.04	0.02	0.03	0.07	0.20	0.03	1.17	1.06		5	
SymmetryMean	Float	0	432	0.10	0.18	0.16	0.17	0.19	0.30	0.02	0.72	1.28			
FractalDimensionMean	Float	0	499	0.04	0.06	0.05	0.06	0.06	0.09	0.00	1.304	3.00			
RadiusSe	Float	0	540	0.11	0.40	0.23	0.32	0.47	2.87	0.27	3.08	17.68	6		
TextureSe	Float	0	519	0.36	1.21	0.83	1.10	1.47	4.88	0.55	1.64	5.34			
PerimeterSe	Float	0	533	0.75	2.86	1.60	2.28	3.35	21.98	2.02	3.44	21.40	7	11	
AreaSe	Float	0	528	6.80	40.33	17.85	24.53	45.19	542.2	45.49	5.44	49.20	8	10	3
SmoothnessSe	Float	0	547	0.00	0.00	0.00	0.00	0.00	0.03	0.00	2.31	10.47			
CompactnessSe	Float	0	541	0.00	0.02	0.01	0.02	0.03	0.13	0.01	1.90	5.10			
ConcavitySe	Float	13	533	0.00	0.03	0.01	0.02	0.04	0.39	0.03	5.11	48.86			
ConcavePointsSe	Float	13	507	0.00	0.01	0.00	0.01	0.01	0.05	0.00	1.44	5.12			
SymmetrySe	Float	0	498	0.00	0.02	0.01	0.01	0.02	0.07	0.00	2.19	7.89			
FractalDimensionSe	Float	0	545	0.00	0.00	0.00	0.00	0.00	0.02	0.00	3.92	26.28			
RadiusMax	Float	0	457	7.93	16.26	13.01	14.97	18.79	36.04	4.83	1.10	0.94	9	3	
TextureMax	Float	0	511	12.02	25.67	21.08	25.41	29.72	49.54	6.14	0.49	0.22	10		
PerimeterMax	Float	0	514	50.41	107.26	84.11	97.66	125.40	251.20	33.60	1.12	1.07	11	1	1
AreaMax	Float	0	544	185.20	880.58	515.30	686.50	1084.00	4254.00	569.35	1.85	4.39	12	2	
SmoothnessMax	Float	0	411	0.07	0.13	0.11	0.13	0.14	0.22	0.02	0.41	0.51		13	5
CompactnessMax	Float	0	529	0.02	0.25	0.14	0.21	0.33	1.05	0.15	1.47	3.03	13		
ConcavityMax	Float	13	539	0.00	0.27	0.11	0.22	0.38	1.25	0.20	1.15	1.61	14	7	
ConcavePointsMax	Float	13	492	0.00	0.11	0.06	0.10	0.16	0.29	0.06	0.49	-0.536	15	4	2
SymmetryMax	Float	0	500	0.15	0.29	0.25	0.28	0.31	0.66	0.06	1.43	4.44			
FractalDimensionMax	Float	0	535	0.05	0.08	0.07	0.08	0.09	0.20	0.01	1.66	5.24			

Rounded to 2 decimal places. 25%, 50% and 75 % represent % quartile.

APPENDIX C – FEATURE DISTRIBUTION WITH KERNAL DENSITY (KDE) AND SPLIT BY TARGET

Feature *AreaSe*

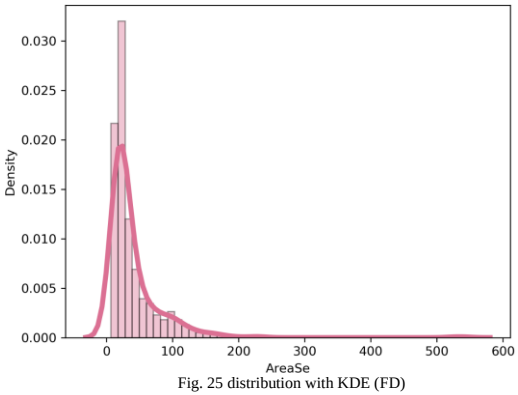

Fig. 25 distribution with KDE (FD)

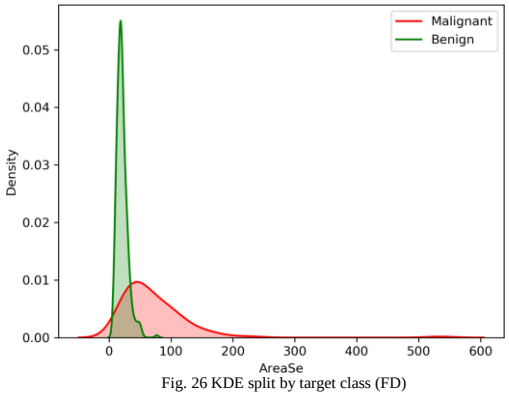

Fig. 26 KDE split by target class (FD)

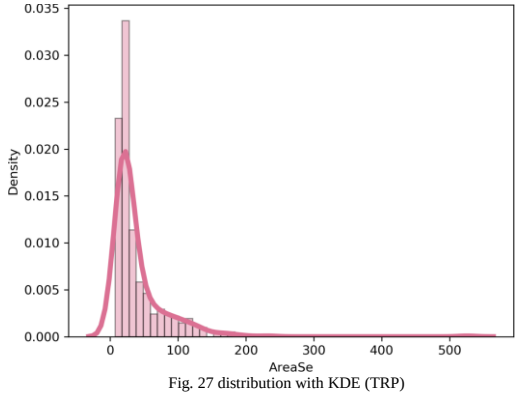

Fig. 27 distribution with KDE (TRP)

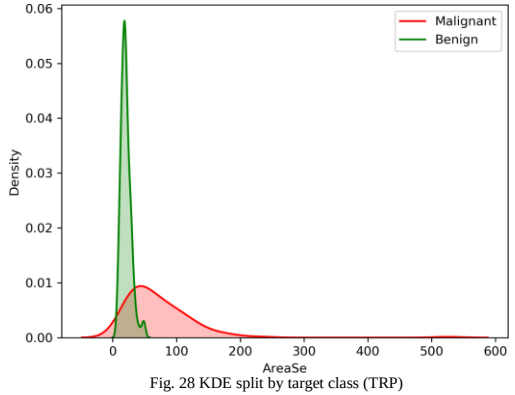

Fig. 28 KDE split by target class (TRP)

Feature *ConcavePointsMax*

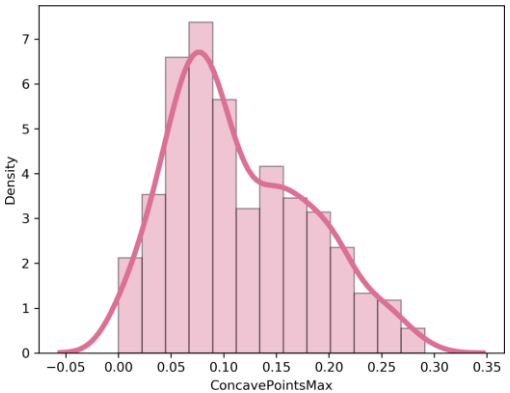

Fig. 29 distribution with KDE (FD)

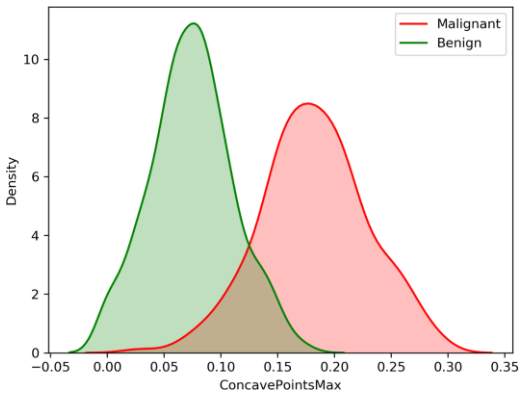

Fig. 30 KDE split by target class (FD)

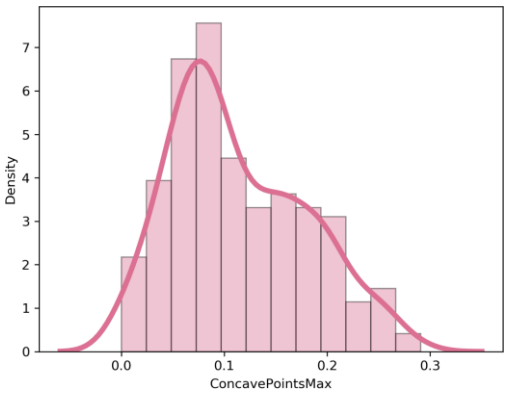

Fig. 31 distribution with KDE (TRP)

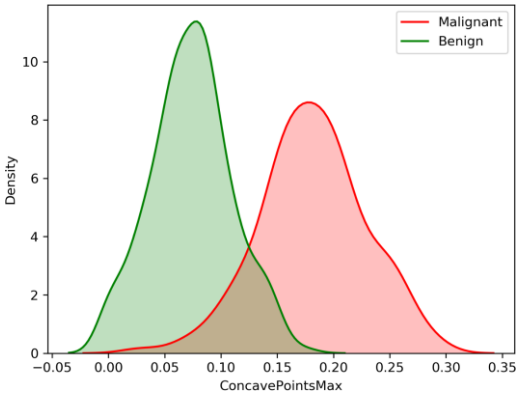

Fig. 32 KDE split by target class (TRP)

Feature *PerimeterMax*

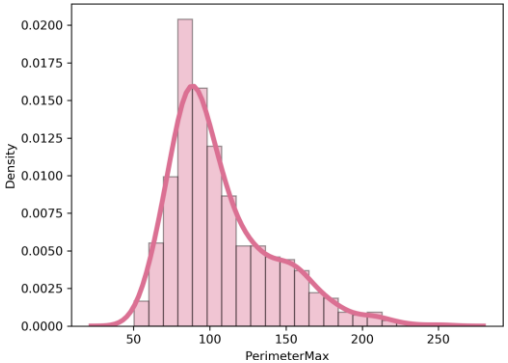

Fig. 33 distribution with KDE (FD)

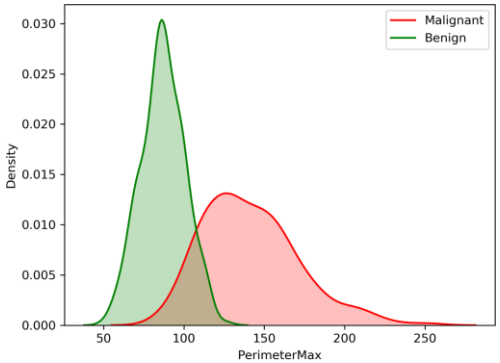

Fig. 34 KDE split by target class (FD)

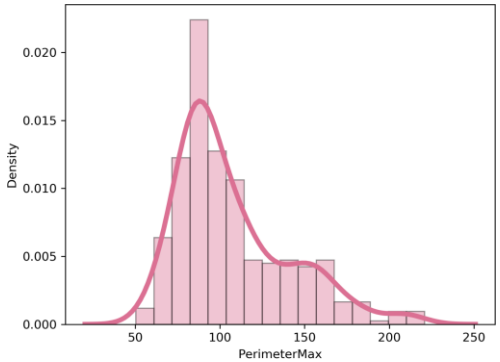

Fig. 35 distribution with KDE (TRP)

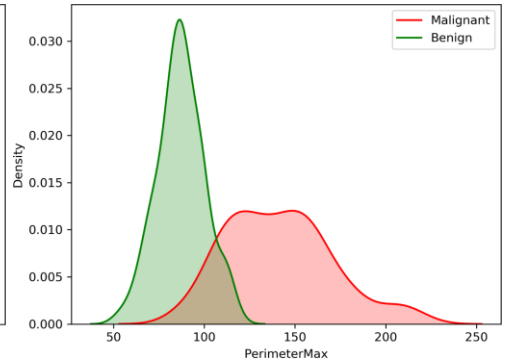

Fig. 36 KDE split by target class (TRP)

Feature *SmoothnessMax*

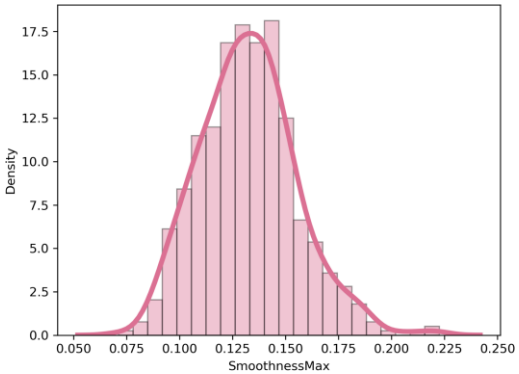

Fig. 37 distribution with KDE (FD)

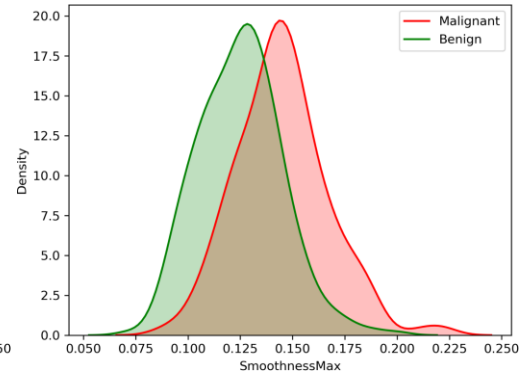

Fig. 38 KDE split by target class (FD)

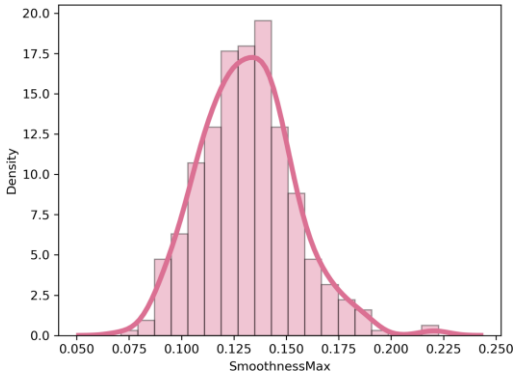

Fig. 39 distribution with KDE (TRP)

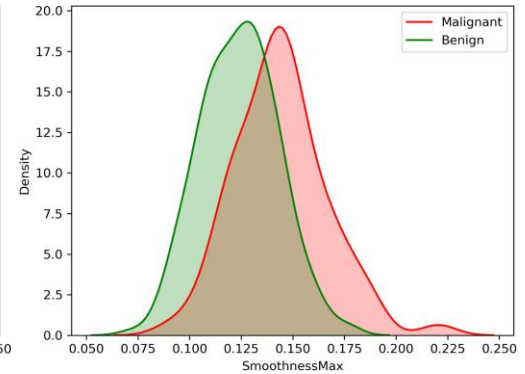

Fig. 40 KDE split by target class (TRP)

Feature *TextureMean*

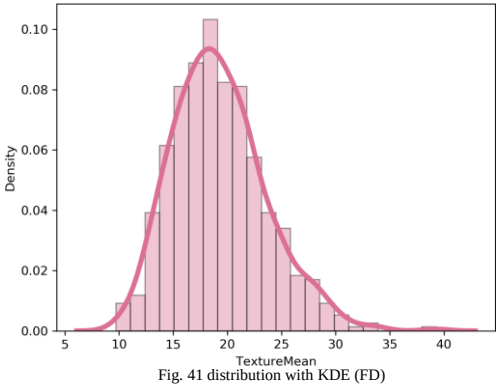

Fig. 41 distribution with KDE (FD)

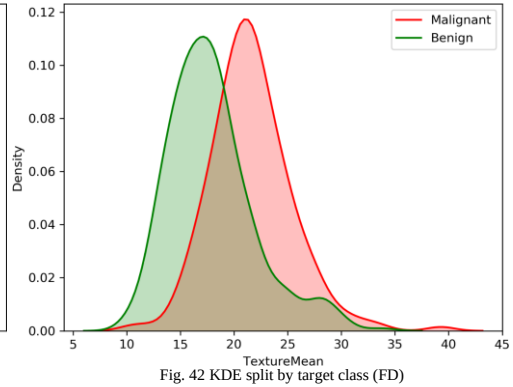

Fig. 42 KDE split by target class (FD)

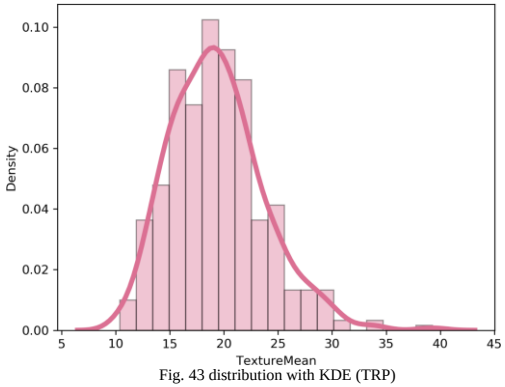

Fig. 43 distribution with KDE (TRP)

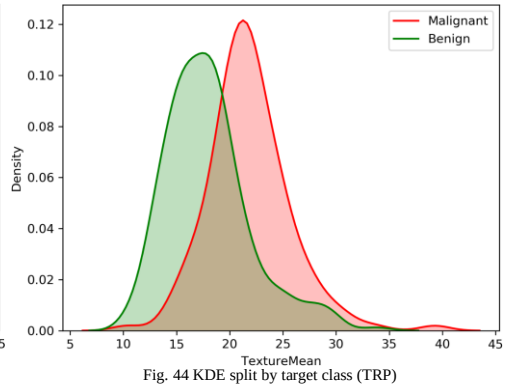

Fig. 44 KDE split by target class (TRP)

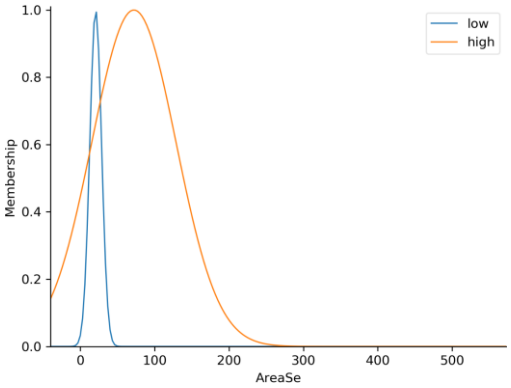

Fig. 45 *AreaSe* low GAUSSMF, high GAUSSMF

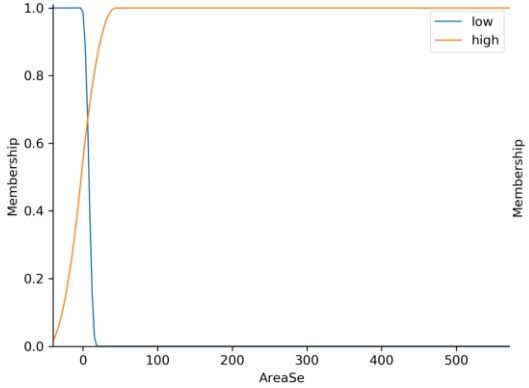

Fig. 46 *AreaSe* low ZMF, high SMF

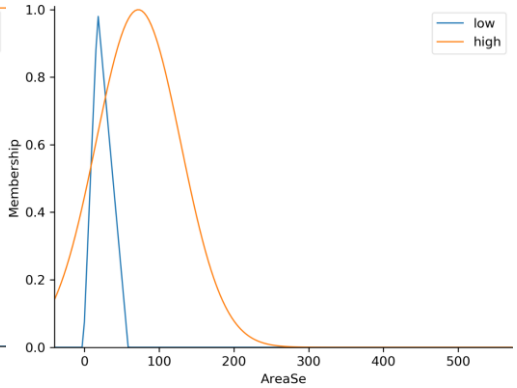

Fig. 47 *AreaSe* low TRIMF, high GAUSSMF

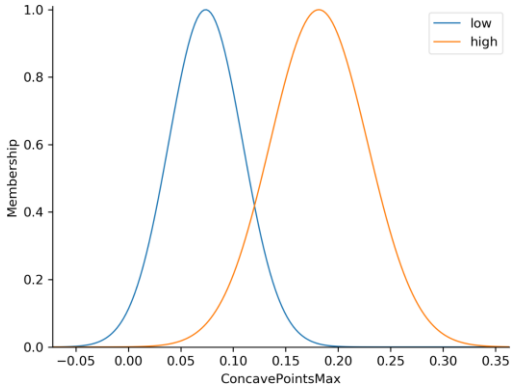

Fig. 48 *ConcavePointsMax* low GAUSSMF, high GAUSSMF

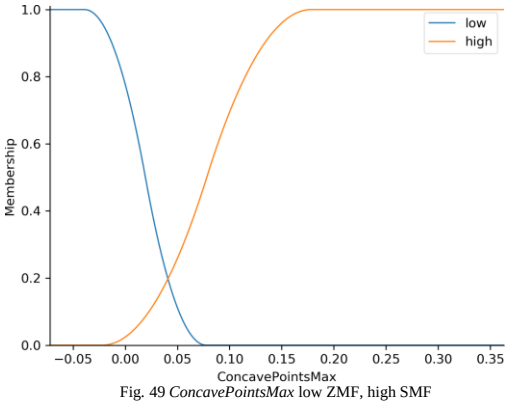

Fig. 49 *ConcavePointsMax* low ZMF, high SMF

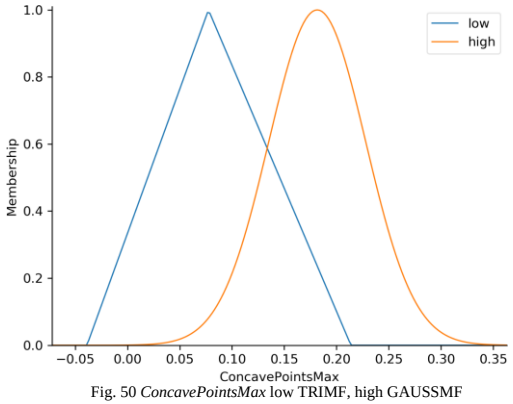

Fig. 50 *ConcavePointsMax* low TRIMF, high GAUSSMF

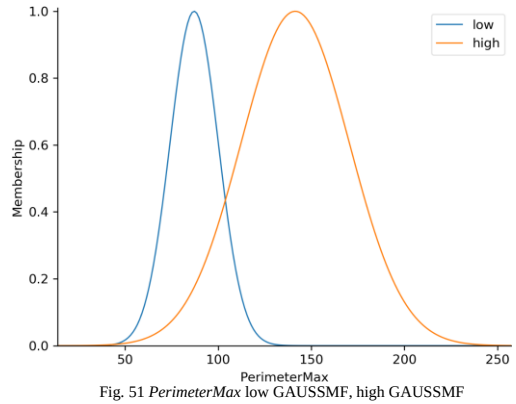

Fig. 51 *PerimeterMax* low GAUSSMF, high GAUSSMF

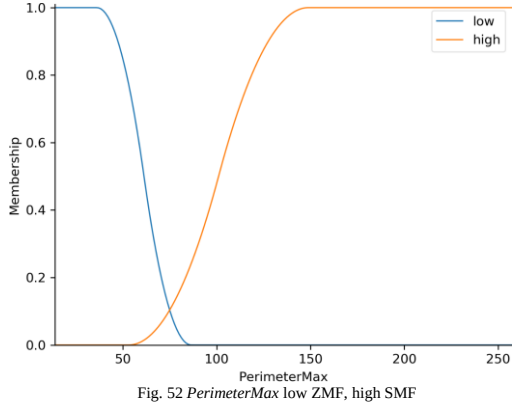

Fig. 52 *PerimeterMax* low ZMF, high SMF

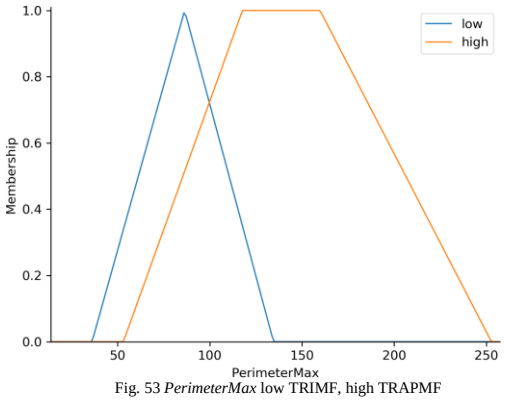

Fig. 53 *PerimeterMax* low TRIMF, high TRAPMF

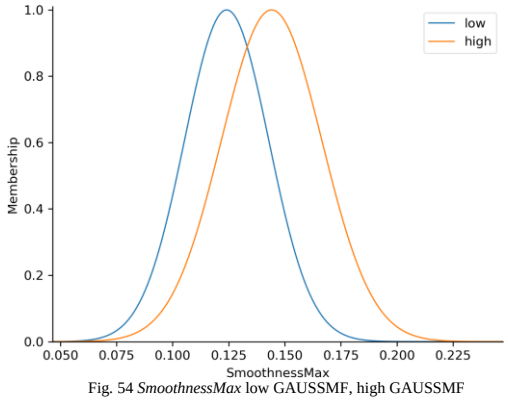

Fig. 54 *SmoothnessMax* low GAUSSMF, high GAUSSMF

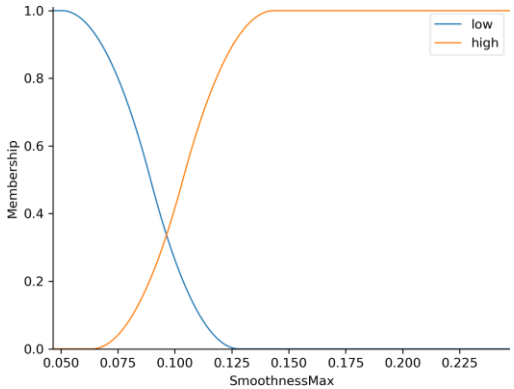

Fig. 55 *SmoothnessMax* low ZMF, high SMF

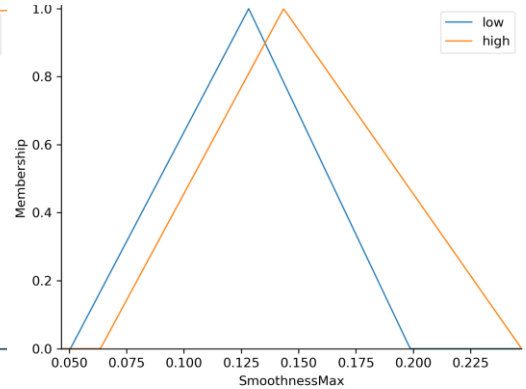

Fig. 56 *SmoothnessMax* low TRIMF, high TRIMF

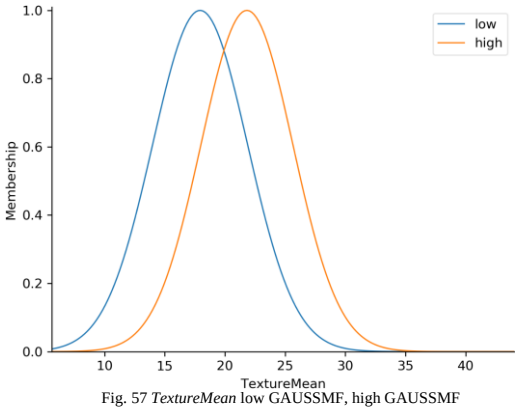

Fig. 57 *TextureMean* low GAUSSMF, high GAUSSMF

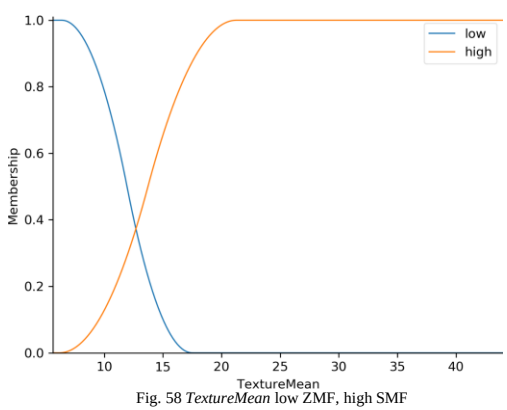

Fig. 58 *TextureMean* low ZMF, high SMF

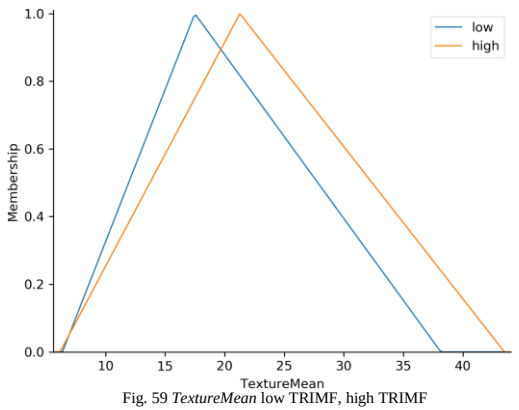

Fig. 59 *TextureMean* low TRIMF, high TRIMF

APPENDIX E – DATASET_ANALYSIS.PY

```
import time
from contextlib import contextmanager
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import chi2, SelectKBest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from scipy.spatial import ConvexHull
from sklearn.preprocessing import StandardScaler, MinMaxScaler


@contextmanager
def timer(title):
    t0 = time.time()
    yield
    print('{} - done in {:.0f}s'.format(title, time.time() - t0))


class WDBCAnalysis:
    def __init__(self):
        self.random_state = 20
        self.column_stats = {}
        self.wdbc_full = None
        self.X = None
        self.y = None
        self.X_scaled = None

        self.X_train = None
        self.X_test = None
        self.y_train = None
        self.y_test = None

        self.label_map_string_2_int = {'M': 1, 'B': 0}

        self.features_selected = []
        self.highly_correlated_features = []
        self.cols = []
        self.class_colours = np.array(['blue', 'red', 'green', 'darkviolet', 'lime', 'darkorange', 'goldenrod',
                                       'cyan', 'silver', 'deepskyblue', 'mediumspringgreen', 'gold'])
        self.clusters_stop = 5
        self.feature_idx = {0: 0, 1: 0, 2: 0}
        self.cluster_cols = [('PerimeterMax', 'AreaSe'),
                             ('ConcavePointsMax', 'TextureMean')]

        with timer('\nLoad dataset'):
            self.load_data()
            self.encode_target()

        with timer('\nInitial Statistics'):
            self.column_statistics()
            self.row_count_by_target('Diagnosis')

        with timer('\nSetting X and y'):
            self.set_X_y()

        with timer('\nFeature Selection'):
            self.univariate_feature_selection()
            self.random_forest_classifier()

        with timer('\nCondensing X to selected features'):
            self.X = self.X[self.features_selected]

        with timer('\nDistributions - Full Dataset'):
            self.correlation_heatmap()
            self.update_X_drop_highly_corr()
            self.scale(self.X)
            self.distribution_multi_kde_scaled('full')
            self.distplot(self.X, 'full')
            self.kdeplot_with_target(self.wdbc_full, 'full')
            self.linearity(self.wdbc_full, 'full')
```

```
        self.cluster(self.wdbc_full, 'full', idx=self.feature_idx)

    with timer('\nOutput Wrangled Dataset'):
        self.write_csv()

    with timer('\nDistributions - Training Subset'):
        self.cols = ['ID'] + list(self.X.columns) + ['Diagnosis']
        self.X = self.wdbc_full[self.cols]
        self.set_y()
        self.train_test_split()
        self.scale(self.X_train)
        self.distribution_multi_kde_scaled('train')
        self.distplot(self.X_train, 'train')
        self.kdeplot_with_target(self.X_train, 'train')
        self.linearity(self.X_train, 'train')
        self.cluster(self.X_train, 'train', idx=self.feature_idx)

def load_data(self):
    self.wdbc_full = pd.read_csv('data/wdbc.data', header=None)

    self.wdbc_full.columns = ['ID', 'Diagnosis', 'RadiusMean', 'TextureMean', 'PerimeterMean', 'AreaMean',
                              'SmoothnessMean', 'CompactnessMean', 'ConcavityMean', 'ConcavePointsMean',
                              'SymmetryMean', 'FractalDimensionMean', 'RadiusSe', 'TextureSe', 'PerimeterSe',
                              'AreaSe', 'SmoothnessSe', 'CompactnessSe', 'ConcavitySe', 'ConcavePointsSe',
                              'SymmetrySe', 'FractalDimensionSe', 'RadiusMax', 'TextureMax', 'PerimeterMax',
                              'AreaMax', 'SmoothnessMax', 'CompactnessMax', 'ConcavityMax', 'ConcavePointsMax',
                              'SymmetryMax', 'FractalDimensionMax']

    print('\n', '_' * 40, 'Shape After Data Load', '_' * 40)
    self.print_shape()

# Encode string class label to binary
def encode_target(self):
    self.wdbc_full['Diagnosis'] = self.wdbc_full['Diagnosis'].map(self.label_map_string_2_int)
    self.wdbc_full['Diagnosis'].astype(int)

def column_statistics(self):
    print('\n', '_' * 40, 'Column Statistics', '_' * 40)
    for col in self.wdbc_full:
        self.column_stats[col + '_dtype'] = self.wdbc_full[col].dtype
        self.column_stats[col + '_zero_num'] = (self.wdbc_full[col] == 0).sum()
        self.column_stats[col + '_zero_num'] = self.column_stats[col + '_zero_num'] + (self.wdbc_full[col] == '?').sum()
        self.column_stats[col + '_zero_pct'] = (((self.wdbc_full[col] == 0).sum() / self.wdbc_full.shape[0]) * 100)
        self.column_stats[col + '_nunique'] = self.wdbc_full[col].nunique()

        print('\n- {} ({})'.format(col, self.column_stats[col + '_dtype']))
        print('\tzero {} ({:.2f}%)'.format(self.column_stats[col + '_zero_num'],
                                          self.column_stats[col + '_zero_pct']))
        print('\tdistinct {}'.format(self.column_stats[col + '_nunique']))

        # Numerical features
        if self.wdbc_full[col].dtype != object:
            self.column_stats[col + '_min'] = self.wdbc_full[col].min()
            self.column_stats[col + '_mean'] = self.wdbc_full[col].mean()
            self.column_stats[col + '_quantile_25'] = self.wdbc_full[col].quantile(.25)
            self.column_stats[col + '_quantile_50'] = self.wdbc_full[col].quantile(.50)
            self.column_stats[col + '_quantile_75'] = self.wdbc_full[col].quantile(.75)
            self.column_stats[col + '_max'] = self.wdbc_full[col].max()
            self.column_stats[col + '_std'] = self.wdbc_full[col].std()
            self.column_stats[col + '_skew'] = self.wdbc_full[col].skew()
            self.column_stats[col + '_kurt'] = self.wdbc_full[col].kurt()
            print('\tmin {}'.format(self.column_stats[col + '_min']))
            print('\tmean {:.3f}'.format(self.column_stats[col + '_mean']))
            print('\t25% {:.3f}'.format(self.column_stats[col + '_quantile_25']))
            print('\t50% {:.3f}'.format(self.column_stats[col + '_quantile_50']))
            print('\t75% {:.3f}'.format(self.column_stats[col + '_quantile_75']))
            print('\tmax {}'.format(self.column_stats[col + '_max']))
            print('\tstd {:.3f}'.format(self.column_stats[col + '_std']))
            print('\tskew {:.3f}'.format(self.column_stats[col + '_skew']))
            print('\tkurt {:.3f}'.format(self.column_stats[col + '_kurt']))

def set_X_y(self):
    self.X = self.wdbc_full.copy()

    # Remove individual sample ID and target label
    self.X.drop(['ID', 'Diagnosis'], axis=1, inplace=True)
```

```
    # Set y as target class label
    self.y = pd.Series(self.wdbc_full.Diagnosis)

# Get feature importance by chi-squared
def univariate_feature_selection(self):
    print('\n', '_' * 40, 'Univariate Feature Selection With Chi-squared', '_' * 40)
    k = 15
    kbest = SelectKBest(score_func=chi2, k=15)
    fit = kbest.fit(self.X, self.y)
    print(fit.scores_)
    cols = kbest.get_support()
    features_selected = self.X.columns[cols]
    print(features_selected)

# Get feature importance by RFC
def random_forest_classifier(self):
    print('\n\n', '_' * 40, 'Random Forest Classifier', '_' * 40)
    data = pd.DataFrame(columns=['Feature', 'Random Forest Importance Score'])
    k = 15
    model = RandomForestClassifier(n_estimators=100, random_state=self.random_state)
    model.fit(self.X, self.y)
    ranked = list(zip(self.X.columns, model.feature_importances_))
    ranked_sorted = sorted(ranked, key=lambda x: x[1], reverse=True)
    print(ranked_sorted)

    for f in ranked_sorted[:k]:
        self.features_selected.append(f[0])
        data = data.append({'Feature': f[0], 'Random Forest Importance Score': f[1]},
                           ignore_index=True)

    fig, ax = plt.subplots(figsize=(25, 7))
    ax = sns.barplot(x='Feature', y='Random Forest Importance Score', data=data, color='palevioletred')
    plt.savefig(fname='plots/full - feature selection RFC.png', dpi=300, format='png')
    plt.show()

def scale(self, dataset):
    df_temp = dataset.copy()
    try:
        df_temp = df_temp.drop(columns=['ID', 'Diagnosis'])
    except KeyError:
        pass
    scaler = MinMaxScaler()
    self.X_scaled = scaler.fit_transform(df_temp)
    self.X_scaled = pd.DataFrame(self.X_scaled, columns=df_temp.columns)

def row_count_by_target(self, target):
    print('\n\n', '_' * 40, 'Row Count By {}'.format(target), '_' * 40)
    series = self.wdbc_full[target].value_counts()
    for idx, val in series.iteritems():
        print('\t{}: {} ({:6.3f}%)'.format(idx, val, ((val / self.wdbc_full.shape[0]) * 100)))

def print_shape(self):
    print('\tRow count:\t', '{}'.format(self.wdbc_full.shape[0]))
    print('\tColumn count:\t', '{}'.format(self.wdbc_full.shape[1]))

def distribution_multi_kde_scaled(self, dataset_name):
    for col in self.X_scaled:
        sns.kdeplot(self.X_scaled[col], shade=True)

    plt.savefig(fname='plots/' + dataset_name + ' - distplot.png', dpi=300, format='png')
    plt.show()

# Plot feature correlation
def correlation_heatmap(self, title='Correlation Heatmap', drop=False):
    # Top x selected features
    df_corr = self.wdbc_full[self.features_selected].copy()
    corr = df_corr.corr()
    plt.clf()
    fig, ax = plt.subplots(figsize=(15, 15))
    ax.set_title(title, size=16)
    drop_self = np.zeros_like(corr)  # Drop self-correlations
    drop_self[np.triu_indices_from(drop_self)] = True
    g = sns.heatmap(corr, cmap='coolwarm', vmin=-1, vmax=1, center=0, annot=True, fmt=".2f", mask=drop_self, cbar=True)
    g.set_yticklabels(g.get_yticklabels(), rotation=0)
    g.set_xticklabels(g.get_yticklabels(), rotation=45)
```

```
    plt.savefig(fname='plots/full - corr heatmap top features.png', dpi=300, format='png')
    plt.show()

    # Drop highly correlated for 2nd heatmap
    upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(np.bool))
    self.highly_correlated_features = [column for column in upper.columns if any(upper[column] > 0.85)]
    df_corr = df_corr.drop(df_corr[self.highly_correlated_features], axis=1)
    corr = df_corr.corr()
    plt.clf()
    fig, ax = plt.subplots(figsize=(15, 15))
    ax.set_title(title, size=16)
    drop_self = np.zeros_like(corr)  # Drop self-correlations
    drop_self[np.triu_indices_from(drop_self)] = True
    g = sns.heatmap(corr, cmap='coolwarm', vmin=-1, vmax=1, center=0, annot=True, fmt=".2f", mask=drop_self,
                    cbar=True)
    g.set_yticklabels(g.get_yticklabels(), rotation=0)
    g.set_xticklabels(g.get_yticklabels(), rotation=45)
    plt.savefig(fname='plots/full - corr heatmap top feat after dropping highly corr.png', dpi=300, format='png')
    plt.show()

def update_X_drop_highly_corr(self):
    self.X = self.X.drop(self.X[self.highly_correlated_features], axis=1)
    self.features_selected = list(self.X)

def distplot(self, dataset, dataset_name):
    for col in self.features_selected:
        if col == 'ID' or col == 'Diagnosis':
            continue
        plt.ylabel('Density')
        sns.distplot(dataset[col], hist=True, color='palevioletred', kde=True, hist_kws={'edgecolor':'black'},
                     kde_kws={'linewidth': 4})
        plt.savefig(fname='plots/' + dataset_name + ' - dist - col ' + col + '.png', dpi=300, format='png')
        plt.show()

def kdeplot_with_target(self, dataset, dataset_name):
    for col in self.features_selected:
        sns.kdeplot(dataset.loc[dataset['Diagnosis'] == 1, col], shade=True, color='r')
        sns.kdeplot(dataset.loc[dataset['Diagnosis'] == 0, col], shade=True, color='g')
        plt.xlabel(col)
        plt.ylabel('Density')
        plt.legend(labels=['Malignant', 'Benign'])
        plt.savefig(fname='plots/' + dataset_name + ' - kdeplot target - ' + col + '.png', dpi=300, format='png')
        plt.show()

# Determine linear separability
def linearity(self, dataset, dataset_name):
    buckets = [0, 1]
    self.convex_hull(dataset, buckets, cola='ConcavePointsMax', colb='TextureMean', target='Diagnosis',
                     dataset_name=dataset_name)
    self.convex_hull(dataset, buckets, cola='PerimeterMax', colb='AreaSe', target='Diagnosis',
                     dataset_name=dataset_name)

# Draw convex hull around 2 buckets of feature points
def convex_hull(self, df, buckets, cola, colb, target, dataset_name):
    cmap = plt.get_cmap('Set1')
    plt.clf()
    plt.figure(figsize=(10, 6))
    title = '{} vs {} - Label {}'.format(cola, colb, target)
    plt.title(title, fontsize=16)
    plt.xlabel(cola, fontsize=12)
    plt.ylabel(colb, fontsize=12)
    for i in range(len(buckets)):
        bucket = df[df[target] == buckets[i]]
        bucket = bucket.iloc[:, [df.columns.get_loc(cola), df.columns.get_loc(colb)]].values
        hull = ConvexHull(bucket)
        hull_color = self.class_colours[i]
        plt.scatter(bucket[:, 0], bucket[:, 1], label=buckets[i], c=self.class_colours[i], alpha=0.4)
        for j in hull.simplices:
            plt.plot(bucket[j, 0], bucket[j, 1], color=hull_color)
    plt.legend()
    plt.savefig(fname='plots/' + dataset_name + ' - convex hull - ' + cola + ' ' + colb + '.png', dpi=300,
                format='png')
    plt.show()

# K-means clustering
def cluster(self, dataset, dataset_name, idx):
```

```
        df_temp = dataset.copy()
        try:
            df_temp = df_temp.drop(columns=['ID', 'Diagnosis'])
        except KeyError:
            pass
        sc = StandardScaler()
        df_temp_cols = list(df_temp.columns)
        df_temp = sc.fit_transform(df_temp)
        df_temp = pd.DataFrame(df_temp, columns=df_temp_cols)
        for cola, colb in self.cluster_cols:
            for c in range(2, self.clusters_stop):
                self.set_indexes(cola, colb, df_temp)
                kmeans = KMeans(n_clusters=c, random_state=self.random_state)
                kmeans.fit(df_temp)
                y_km = kmeans.fit_predict(df_temp)
                self.scatter_clusters(df_temp, dataset_name, c, y_km, idx)

    def scatter_clusters(self, df, dataset_name, n_clusters, y_clusters, col_idx):
        fig = plt.figure(figsize=(8, 6))
        ax = fig.add_subplot(111)
        df_x = df

        if isinstance(df_x, pd.DataFrame):
            df_x = df_x.values

        title = str(n_clusters) + ' Clusters - ' + df.columns[col_idx[0]] + ' vs ' + df.columns[col_idx[1]]
        plt.title(title, fontsize=12)

        xlabel = df.columns[col_idx[0]]
        ylabel = df.columns[col_idx[1]]
        ax.set_xlabel(xlabel, fontsize=12)
        ax.set_ylabel(ylabel, fontsize=12)

        # 2 clusters minimum
        for c in range(n_clusters):
            ax.scatter(df_x[y_clusters == c, col_idx[0]], df_x[y_clusters == c, col_idx[1]], alpha=0.2,
                       edgecolors='none', s=20, c=self.class_colours[c])

        plt.savefig(fname='plots/' + dataset_name + ' - ' + str(n_clusters) + ' Clusters - ' + xlabel + ' vs ' + ylabel
                          + '.png', dpi=300, format='png')
        plt.show()

    def set_indexes(self, cola, colb, dataset):
        self.feature_idx[0] = dataset.columns.get_loc(cola)
        self.feature_idx[1] = dataset.columns.get_loc(colb)

    def set_y(self):
        self.y = self.X['Diagnosis']

    def remove_target_from_X(self):
        self.X.drop('Diagnosis', axis=1, inplace=True)

    def train_test_split(self):
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(self.X, self.y, test_size=0.30,
                                                                                 random_state=self.random_state)

    def write_csv(self):
        cols = ['ID'] + list(self.X.columns) + ['Diagnosis']
        self.wdbc_full[cols].to_csv('data/wdbc_selected_cols.csv', header=True, index=False)


wdbcAnalysis = WDBCAnalysis()
```

```
import time
from contextlib import contextmanager
import collections
import operator
import pandas as pd
import numpy as np
import skfuzzy as fz
from skfuzzy import control as ct
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt


@contextmanager
def timer(title):
    t0 = time.time()
    yield
    print('{} - done in {:.0f}s'.format(title, time.time() - t0))


class WDBCFis:
    def __init__(self):
        self.random_state = 20

        self.X = None
        self.y = None
        self.X_train = None
        self.X_y_train = None
        self.X_test = None
        self.y_train = None
        self.y_test = None

        self.predict_log = []

        # Globals for test enabled and test config
        self.t_enabled = None
        self.t_id = None
        self.t_rs = None
        self.t_mfgrp = None

        # Support running non-fis models only once per feature rule set
        self.last_rs = None

        # FIS components
        self.ant = {}
        self.diagnosis = None
        self.rules = []
        self.ops = {
            '&': operator.and_,
            '|': operator.or_
        }
        self.system = None
        self.crisp_binary_threshold = 0

        # Map cycling through range to defuzzification methods to actual mode name
        self.defuzzify_switcher = {1: 'centroid', 2: 'bisector', 3: 'mom', 4: 'som', 5: 'lom'}

        # Support plotting of unique MF only once
        self.mf_plotted = {}

        # Suite of FIS tests that include antecedents and their configured term MFs, consequent terms and Mfs,
        # and rules linking the 2
        self.tests = [[{'Config': [{'Id': 1, 'Rs': 1, 'Mfgrp': 'gauss', 'Enabled': True}]},
                       {'Ant': [{'PerimeterMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
                                {'ConcavePointsMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}}]},
                       {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                                      {'malignant': ['smf', [100, 200]]}]}}]},
                       {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', '-op': '&', 'Diagnosis': 'benign'},
```

```
           {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', '-op': '&',
            'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 2, 'Rs': 1, 'Mfgrp': 'zs', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'zmf', 'high': 'smf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', '-op': '&', 'Diagnosis': 'benign'},
            {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', '-op': '&',
             'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 3, 'Rs': 1, 'Mfgrp': 'mix', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'trimf', 'high': 'trapmf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'trimf', 'high': 'gaussmf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', '-op': '&', 'Diagnosis': 'benign'},
            {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', '-op': '&',
             'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 4, 'Rs': 2, 'Mfgrp': 'gauss', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'AreaSe': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'TextureMean': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'SmoothnessMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', 'TextureMean':
     'low', 'SmoothnessMax': 'low', '-op': '&', 'Diagnosis': 'benign'}, {'PerimeterMax': 'high',
     'ConcavePointsMax': 'high', 'AreaSe': 'high', 'TextureMean': 'high', 'SmoothnessMax': 'high',
     '-op': '&', 'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 5, 'Rs': 2, 'Mfgrp': 'zs', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'AreaSe': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'TextureMean': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'SmoothnessMax': {'mf': {'low': 'zmf', 'high': 'smf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [
     {'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', 'TextureMean': 'low',
      'SmoothnessMax': 'low', '-op': '&', 'Diagnosis': 'benign'},
     {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', 'AreaSe': 'high', 'TextureMean': 'high',
      'SmoothnessMax': 'high', '-op': '&', 'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 6, 'Rs': 2, 'Mfgrp': 'mix', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'trimf', 'high': 'trapmf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'trimf', 'high': 'gaussmf'}}},
          {'AreaSe': {'mf': {'low': 'trimf', 'high': 'gaussmf'}}},
          {'TextureMean': {'mf': {'low': 'trimf', 'high': 'trimf'}}},
          {'SmoothnessMax': {'mf': {'low': 'trimf', 'high': 'trimf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [
     {'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', 'TextureMean': 'low',
      'SmoothnessMax': 'low', '-op': '&', 'Diagnosis': 'benign'},
     {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', 'AreaSe': 'high', 'TextureMean': 'high',
      'SmoothnessMax': 'high', '-op': '&', 'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 7, 'Rs': 3, 'Mfgrp': 'gauss', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'AreaSe': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', '-op': '&',
             'Diagnosis': 'benign'},
            {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', 'AreaSe': 'high', '-op': '&',
             'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 8, 'Rs': 3, 'Mfgrp': 'zs', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'zmf', 'high': 'smf'}}},
```

```
         {'AreaSe': {'mf': {'low': 'zmf', 'high': 'smf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', '-op': '&',
             'Diagnosis': 'benign'},
            {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', 'AreaSe': 'high', '-op': '&',
             'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 9, 'Rs': 3, 'Mfgrp': 'mix', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'trimf', 'high': 'trapmf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'trimf', 'high': 'gaussmf'}}},
          {'AreaSe': {'mf': {'low': 'trimf', 'high': 'gaussmf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', '-op': '&',
             'Diagnosis': 'benign'},
            {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', 'AreaSe': 'high', '-op': '&',
             'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 10, 'Rs': 4, 'Mfgrp': 'gauss', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'AreaSe': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'TextureMean': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', 'TextureMean':
     'low', '-op': '&', 'Diagnosis': 'benign'},
            {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', 'AreaSe': 'high', 'TextureMean':
                'high', '-op': '&', 'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 11, 'Rs': 4, 'Mfgrp': 'zs', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'AreaSe': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'TextureMean': {'mf': {'low': 'zmf', 'high': 'smf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', 'TextureMean':
     'low', '-op': '&', 'Diagnosis': 'benign'},
            {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', 'AreaSe': 'high', 'TextureMean':
                'high', '-op': '&', 'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 12, 'Rs': 4, 'Mfgrp': 'mix', 'Enabled': True}]},
 {'Ant': [{'PerimeterMax': {'mf': {'low': 'trimf', 'high': 'trapmf'}}},
          {'ConcavePointsMax': {'mf': {'low': 'trimf', 'high': 'gaussmf'}}},
          {'AreaSe': {'mf': {'low': 'trimf', 'high': 'gaussmf'}}},
          {'TextureMean': {'mf': {'low': 'trimf', 'high': 'trimf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'PerimeterMax': 'low', 'ConcavePointsMax': 'low', 'AreaSe': 'low', 'TextureMean':
     'low', '-op': '&', 'Diagnosis': 'benign'},
            {'PerimeterMax': 'high', 'ConcavePointsMax': 'high', 'AreaSe': 'high', 'TextureMean':
                'high', '-op': '&', 'Diagnosis': 'malignant'}]}],

[{'Config': [{'Id': 13, 'Rs': 5, 'Mfgrp': 'gauss', 'Enabled': True}]},
 {'Ant': [{'TextureMean': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}},
          {'SmoothnessMax': {'mf': {'low': 'gaussmf', 'high': 'gaussmf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                 {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'TextureMean': 'low', 'SmoothnessMax': 'low', '-op': '&', 'Diagnosis': 'benign'},
            {'TextureMean': 'high', 'SmoothnessMax': 'high', '-op': '&', 'Diagnosis':
                'malignant'}]}],

[{'Config': [{'Id': 14, 'Rs': 5, 'Mfgrp': 'zs', 'Enabled': True}]},
 {'Ant': [{'TextureMean': {'mf': {'low': 'zmf', 'high': 'smf'}}},
          {'SmoothnessMax': {'mf': {'low': 'zmf', 'high': 'smf'}}}]},
 {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                {'malignant': ['smf', [100, 200]]}]}}]},
 {'Rules': [{'TextureMean': 'low', 'SmoothnessMax': 'low', '-op': '&', 'Diagnosis': 'benign'},
            {'TextureMean': 'high', 'SmoothnessMax': 'high', '-op': '&', 'Diagnosis':
                'malignant'}]}],

[{'Config': [{'Id': 15, 'Rs': 5, 'Mfgrp': 'mix', 'Enabled': True}]},
 {'Ant': [{'TextureMean': {'mf': {'low': 'trimf', 'high': 'trimf'}}},
          {'SmoothnessMax': {'mf': {'low': 'trimf', 'high': 'trimf'}}}]},
```

```
                {'Con': [{'Diagnosis': {'mf': [{'benign': ['zmf', [10, 170]]},
                                               {'malignant': ['smf', [100, 200]]}]}}]},
                {'Rules': [{'TextureMean': 'low', 'SmoothnessMax': 'low', '-op': '&', 'Diagnosis': 'benign'},
                           {'TextureMean': 'high', 'SmoothnessMax': 'high', '-op': '&', 'Diagnosis':
                               'malignant'}]}]
              ]

    with timer('\nLoad dataset'):
        self.load_data()

    with timer('\nSplit dataset'):
        self.set_y()
        self.remove_target_from_X()

    with timer('\nTesting Scenarios'):
        for i, t in enumerate(self.tests):
            self.set_test_global_cfg(t)

            if not self.t_enabled:
                continue

            with timer('\nTest ' + str(self.t_id) + '_' + str(self.t_rs)):

                # Clear any previous FIS test config
                self.ant_cfg = []
                self.con_cfg = []
                self.rule_cfg = []

                self.set_test_ant_con_cfg(t)

            with timer('\nPreparing dataset for test'):

                # Partition dataset and set partition cols as per FIS test antecedents
                self.train_test_split()
                self.set_X_train_test_cols()
                self.set_X_y_train()

            # Execute non-FIS model predict and score only once per RS
            if self.last_rs != self.t_rs:
                self.last_rs = self.t_rs
                with timer('\nLogistic Regression ML Model Prediction'):
                    self.lr_model_predict_score()
                with timer('\nDecision Tree ML Model Prediction'):
                    self.dtc_model_predict_score()
                with timer('\nRandom Forest Classification ML Model Prediction'):
                    self.rfc_model_predict_score()

            # For each defuzzification method
            for d in range(1, 6):
                self.ant = {}
                self.diagnosis = None
                self.rules = []

                # Prepare FIS components
                self.create_antecendents_universe(defuzzify_method=self.defuzzify_switcher[d])
                self.create_consequent_universe(defuzzify_method=self.defuzzify_switcher[d])
                self.set_antecendents_mfs()
                self.set_consequent_mfs()
                self.set_rules(t)

                self.system = ct.ControlSystem(rules=self.rules)
                self.diagnose = ct.ControlSystemSimulation(self.system)

                # For each crisp to binary threshold limit in required range
                for c2b in range(90, 131):
                    with timer('\nMaking FIS Model Predictions with defuzzify method ' + self.defuzzify_switcher[d]
                               + ' & threshold ' + str(c2b)):
                        # Compute FIS and score classification
                        self.fis_model_predict_score(self.defuzzify_switcher[d], c2b)

        self.download_predict_log()

# Suite of methods for feature distribution statistics that support MF shaping
@staticmethod
def feature_std(feat, df, target):
    return df.loc[df['Diagnosis'] == target, feat].std()
```

```
@staticmethod
def feature_mean(feat, df, target):
    return df.loc[df['Diagnosis'] == target, feat].mean()

@staticmethod
def feature_min(feat, df, target):
    return df.loc[df['Diagnosis'] == target, feat].min()

@staticmethod
def feature_max(feat, df, target):
    return df.loc[df['Diagnosis'] == target, feat].max()

@staticmethod
def feature_quantile(feat, df, target, q):
    return df.loc[df['Diagnosis'] == target, feat].quantile(q)

@staticmethod
def feature_kde_peak(feat, df, target):
    kernel = stats.gaussian_kde(df.loc[df['Diagnosis'] == target, feat])
    universe = np.linspace(df[feat].min(), df[feat].max(), num=200)
    kernel = kernel(universe)
    return universe[np.argsort(kernel)[-1]]

@staticmethod
def transform_class_to_target(t):
    return 0 if t == 'low' else 1

@staticmethod
def specificity_score(test, pred):
    tn, fp, fn, tp = confusion_matrix(test, pred).ravel()
    return tn / (tn + fp)

def load_data(self):
    self.X = pd.read_csv('data/wdbc_selected_cols.csv')

    print('\n', '_' * 40, 'Shape After Data Load', '_' * 40)
    self.print_shape(self.X)

def set_y(self):
    self.y = self.X['Diagnosis']

def remove_target_from_X(self):
    self.X.drop('Diagnosis', axis=1, inplace=True)

# Split dataset into train (70%) and test (30%) partitions
def train_test_split(self):
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(self.X, self.y, test_size=0.30,
                                                                             random_state=self.random_state)
    print('\n', '_' * 40, 'X_train Shape After Split', '_' * 40)
    self.print_shape(self.X_train)
    print('\n', '_' * 40, 'X_test Shape After Split', '_' * 40)
    self.print_shape(self.X_test)

def set_X_y_train(self):
    # Used for MF shaping when feature distribution filtered by target
    self.X_y_train = pd.concat([self.X_train, self.y_train], axis=1)

def set_test_global_cfg(self, t):
    self.t_enabled = t[0]['Config'][0]['Enabled']
    self.t_id = t[0]['Config'][0]['Id']
    self.t_rs = t[0]['Config'][0]['Rs']
    self.t_mfgrp = t[0]['Config'][0]['Mfgrp']

# Parse test config grabbing MF term and shape
def set_test_ant_con_cfg(self, t):
    for cfg in t:
        if 'Ant' in cfg:
            for f in cfg['Ant']:
                for fk, fv in f.items():
                    if 'mf' in f[fk]:
                        for mfclass, mfshape in f[fk]['mf'].items():
                            self.ant_cfg.append({fk: (mfclass, mfshape)})
        if 'Con' in cfg:
            for f in cfg['Con']:
                for fk, fv in f.items():
```

```
                    if 'mf' in f[fk]:
                        for terms in f[fk]['mf']:
                            for mfclass, mfshape in terms.items():
                                self.con_cfg.append({fk: (mfclass, mfshape)})

def set_X_train_test_cols(self):
    cols = []
    for c in self.ant_cfg:
        for k, v in c.items():
            cols.append(k)
    cols = list(set(cols))
    self.X_train = self.X_train[cols]
    self.X_test = self.X_test[cols]

def print_shape(self, df):
    print('\tRow count:\t', '{}'.format(df.shape[0]))
    print('\tColumn count:\t', '{}'.format(df.shape[1]))

def create_antecendents_universe(self, defuzzify_method='centroid'):
    # Set universe boundary for each feature
    for feat in self.X_train:
        self.ant[feat] = ct.Antecedent(np.linspace(self.X_train[feat].min() - (self.X_train[feat].std() * 1.1),
                                                   self.X_train[feat].max() + (self.X_train[feat].std() * 1.1),
                                                   num=200), feat, defuzzify_method=defuzzify_method)

def create_consequent_universe(self, defuzzify_method='centroid'):
    self.diagnosis = ct.Consequent(np.arange(0, 200, 1), 'diagnosis', defuzzify_method=defuzzify_method)

def set_antecendents_mfs(self):
    # Diagnosis: Benign = class 0, Malignant = Class 1
    for a in self.ant:
        s = self.set_antecedents_stats(a)
        self.set_antecedent_mfs(a, s)

def set_antecedent_mfs(self, a, s):
    for c in self.ant_cfg:
        for k, v in c.items():
            if k == a:
                # Execute MF function associated with configured antecedent
                self.ant[a][v[0]] = getattr(self, 'mf_' + v[1])(a, v[0], s)
    if a + v[1] not in self.mf_plotted: # If not done then plot term MFs for antecedent
        self.mf_plotted[a + v[1]] = True
        self.ant[a].view()

def set_antecedents_stats(self, a):
    s = {}

    s['std0'] = self.feature_std(a, self.X_y_train, 0)
    s['std1'] = self.feature_std(a, self.X_y_train, 1)
    s['min0'] = self.feature_min(a, self.X_y_train, 0) - (s['std0'] * 1.1)
    s['min1'] = self.feature_min(a, self.X_y_train, 1) - (s['std1'] * 1.1)
    s['max0'] = self.feature_max(a, self.X_y_train, 0) + (s['std0'] * 1.1)
    s['max1'] = self.feature_max(a, self.X_y_train, 1) + (s['std1'] * 1.1)
    s['mean0'] = self.feature_mean(a, self.X_y_train, 0)
    s['mean1'] = self.feature_mean(a, self.X_y_train, 1)
    s['q250'] = self.feature_quantile(a, self.X_y_train, 0, 0.25)
    s['q251'] = self.feature_quantile(a, self.X_y_train, 1, 0.25)
    s['q750'] = self.feature_quantile(a, self.X_y_train, 0, 0.75)
    s['q751'] = self.feature_quantile(a, self.X_y_train, 1, 0.75)
    s['pke0'] = self.feature_kde_peak(a, self.X_y_train, 0)
    s['pke1'] = self.feature_kde_peak(a, self.X_y_train, 1)

    return s

# Suite of calls to skfuzzy functions to shape term MFs to feature distributions
def mf_gaussmf(self, a, t, s):
    t = str(self.transform_class_to_target(t))
    return fz.gaussmf(self.ant[a].universe, s['mean' + t], s['std' + t])

def mf_trimf(self, a, t, s):
    t = str(self.transform_class_to_target(t))
    return fz.trimf(self.ant[a].universe, [s['min' + t], s['pke' + t], s['max' + t]])

def mf_trapmf(self, a, t, s):
    t = str(self.transform_class_to_target(t))
    return fz.trapmf(self.ant[a].universe, [s['min' + t], s['q25' + t], s['q75' + t], s['max' + t]])
```

```
def mf_zmf(self, a, t, s):
    t = str(self.transform_class_to_target(t))
    return fz.zmf(self.ant[a].universe, s['min' + t], s['pke' + t])

def mf_smf(self, a, t, s):
    t = str(self.transform_class_to_target(t))
    return fz.smf(self.ant[a].universe, s['min' + t], s['pke' + t])

def set_consequent_mfs(self):
    for c in self.con_cfg:
        for k, v in c.items():
            self.diagnosis[v[0]] = getattr(fz, v[1][0])(self.diagnosis.universe, v[1][1][0], v[1][1][1])

    if 'diagnosis' not in self.mf_plotted:
        self.mf_plotted['diagnosis'] = True
        self.diagnosis.view()

def set_rules(self, test):
    for r in test:
        if 'Rules' in r:
            self.add_rules(r['Rules'])

# Parse configured test rules
def add_rules(self, rules):
    for r in rules:
        antecedent = None
        consequent = None
        label = None
        op_func = None
        od = collections.OrderedDict(sorted(r.items()))  # Required to ensure operator determined first
        for arg in od.items():
            if arg[0] == 'Diagnosis':
                consequent = self.diagnosis[arg[1]]
                label = arg[1]
            elif arg[0] == '-op':
                op_func = self.ops[arg[1]]
            else:
                if antecedent is None:
                    antecedent = self.ant[arg[0]][arg[1]]
                else:
                    antecedent = op_func(antecedent, self.ant[arg[0]][arg[1]])  # Link antecedent terms
        r = ct.Rule(antecedent, consequent=consequent, label=label)
        self.rules.append(r)

def fis_model_predict_score(self, defuzzify_method, crisp_threshold):
    y_pred = []

    # Set to true to support debugging and output analysis
    dv = False
    ychk = False

    # For each sample in test partition
    for di, dr in self.X_test.iterrows():
        # Take each feature value in test sample and pass it to previously configured antecedent
        for si, sv in dr.iteritems():
            self.diagnose.input[si] = sv
        try:
            self.diagnose.compute()  # Now execute the FIS
            #self.diagnose.print_state()
            if dv:
                self.diagnosis.view(sim=self.diagnose)
        except ValueError:
            print(self.diagnose.input)
            continue

        # Transform crsip to binary using threshold
        crisp_to_binary = 0 if self.diagnose.output['diagnosis'] < crisp_threshold else 1
        if ychk:  # If set to true, supports checking terms MF shape
            if self.y_test.loc[di] != crisp_to_binary:
                print('Ground truth is ', self.y_test.loc[di], ' and pred is ', crisp_to_binary)
                print('Crisp output is ', crisp_to_binary)
                print('Inputs are')
                for si, sv in dr.iteritems():
                    print(si, ' = ', sv)
```

```
                    self.diagnosis.view(sim=self.diagnose)

            # Add to array of binary predictions for test set
            y_pred.append(crisp_to_binary)

        # Log test prediction results
        ref = 'rs' + str(self.t_rs) + '_id' + str(self.t_id) + '_FIS_ct' + str(crisp_threshold) + '_' + defuzzify_method
        self.log_prediction(ref, self.t_rs, self.t_id, self.t_mfgrp, 'FIS', defuzzify_method, crisp_threshold,
                            self.y_test, y_pred)

        # Plot confusion matrix
        self.plot_cm(self.y_test, y_pred, ref)

    def lr_model_predict_score(self):
        lr = LogisticRegression(random_state=self.random_state)
        lr.fit(self.X_train, self.y_train)
        y_pred = lr.predict(self.X_test)
        ref = 'rs' + str(self.t_rs) + '_LR'
        self.log_prediction(ref, self.t_rs, 'n/a', 'n/a', 'LR', 'n/a', 'n/a', self.y_test, y_pred)
        self.plot_cm(self.y_test, y_pred, ref)

    def dtc_model_predict_score(self):
        dtc = DecisionTreeClassifier(random_state=self.random_state)
        dtc.fit(self.X_train, self.y_train)
        y_pred = dtc.predict(self.X_test)
        ref = 'rs' + str(self.t_rs) + '_DTC'
        self.log_prediction(ref, self.t_rs, 'n/a', 'n/a', 'DTC', 'n/a', 'n/a', self.y_test, y_pred)
        self.plot_cm(self.y_test, y_pred, ref)

    def rfc_model_predict_score(self):
        rfc = RandomForestClassifier(random_state=self.random_state)
        rfc.fit(self.X_train, self.y_train)
        y_pred = rfc.predict(self.X_test)
        ref = 'rs' + str(self.t_rs) + '_RFC'
        self.log_prediction(ref, self.t_rs, 'n/a', 'n/a', 'RFC', 'n/a', 'n/a', self.y_test, y_pred)
        self.plot_cm(self.y_test, y_pred, ref)

    def plot_cm(self, y, y_pred, ref):
        cm = confusion_matrix(y, y_pred)
        ax = plt.subplot()
        sns.heatmap(cm, annot=True, ax=ax, annot_kws={"size": 14}, fmt='d', cmap='Greens', cbar=False)
        ax.set_xlabel('Predicted Class')
        ax.set_ylabel('True Class')
        ax.set_title('Confusion Matrix - ' + ref)
        ax.xaxis.set_ticklabels(['Benign', 'Malignant'])
        ax.yaxis.set_ticklabels(['Benign', 'Malignant'])
        plt.savefig(fname='plots/cm/CM - ' + ref + '.png', dpi=300, format='png')
        plt.close()
        #plt.show()

    def log_prediction(self, ref, trs, tid, mfgrp, clf, dm, cbt, y_test, y_pred):
        acc = round(accuracy_score(y_test, y_pred), 3)
        sen = round(recall_score(y_test, y_pred), 3)
        spe = round(self.specificity_score(y_test, y_pred), 3)
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
        self.predict_log.append([ref, trs, tid, mfgrp, clf, dm, cbt, tp, fp, tn, fn, sen, acc, spe])

    # Save full and summarised prediction log to csv for analysis
    def download_predict_log(self):
        df_pred_log = pd.DataFrame(self.predict_log, columns=['Ref', 'Rs', 'Id', 'Mfgrp', 'Clf', 'Dm', 'Ct', 'Tp', 'Fp',
                                                              'Tn', 'Fn', 'Sen', 'Acc', 'Spe'])
        df_pred_log.sort_values(by=['Sen', 'Acc', 'Spe', 'Dm', 'Ct', 'Clf',  'Rs', 'Id'],
                                ascending=[False, False, False, True, True, True, True, True], inplace=True)
        df_pred_log['Rank'] = np.arange(1, len(df_pred_log) + 1)
        df_pred_log.to_csv('data/predict_log.csv', header=True, index=False)

        df_log_sum = df_pred_log[['Rank', 'Clf', 'Ref', 'Rs', 'Mfgrp', 'Dm', 'Tp', 'Fp', 'Tn', 'Fn', 'Sen', 'Acc', 'Spe']].copy()
        df_log_sum.drop_duplicates(subset=['Clf', 'Rs', 'Mfgrp', 'Dm'], keep='first', inplace=True)
        df_log_sum.to_csv('data/predict_log_sum.csv', header=True, index=False)


wdbcFis = WDBCFis()
```

APPENDIX G – SKFUZZY PACKAGE MODIFICATIONS

Added argument *defuzzify_method* with default ‘centroid’ to *__init__* method of class *Antecedent* in *control/antecedent_consequent.py*. Added this argument to call of super. Same done for class Consequent in same .py file.

```
def __init__(self, universe, label, defuzzify_method='centroid'):
  super(Antecedent, self).__init__(universe, label, defuzzify_method)
```

This modification allowed overwriting of default centroid defuzzification method which is set in *__init__* method of class *FuzzyVariable* in *control/fuzzyvariable.py*. Example call specifying defuzzification method from custom FIS solution below.

```
# Prepare FIS components
self.create_antecendents_universe(defuzzify_method=self.defuzzify_switcher[d])
self.create_consequent_universe(defuzzify_method=self.defuzzify_switcher[d])
```