DAT410 - Assignment 5

Eric Johansson & Max Sonnelid

February 22, 2021

Group number: 70	
Module number: 5	
Eric Johansson	**Max Sonnelid**
970429-2854	960528-5379
MPDSC	MPDSC
johaeric@student.chalmers.se	sonnelid@student.chalmers.se
25 hours spent	25 hours spent

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions.

Introduction

1 Summary of articles

1.1 Short summary of "Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates"

The article *Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates* by Wolberg, Street & Mangasarian (1993) revolves around the fine-needle aspirates (FNA) method, which is used as a diagnostics tool for breast cancer. Previous manual diagnostics with this method has reported an accuracy of over 90 %, but this accuracy was also strongly characterized by high standard deviations and thus strong subjectivity in diagnosis. The article authors therefore aimed to *increase consistency in diagnostics* with this already efficient *FNA method* by using *computer-based image analysis* and *machine learning techniques*.

Images of viscous fluid aspirated from breast masses were captured with camera and microscope. With the help of a graphical user interface, a rough outline of the boundaries of each visible cell nucleus was drawn manually. This rough outline was then more exactly defined by an *active contour model*, also called *snake model*, which aims to minimize discontinuities in the curvature.

After all the cell nucleus were clearly defined, ten different nuclear features were calculated for each cell, for example radius, perimeter and compactness, where higher values of these features are typically associated with malignant cells. Mean value, worst value and standard error for each feature were also calculated, resulting in *30 extracted features* in total.

The classification into malignant and benign cells was then done by the *multisurface method*, known as *MSM-Tree*, which iteratively uses linear programming to place separating planes in the feature space of the training data. If the two groups are not linearly separable, the MSM-T constructs a plane that minimizes an average distance of misclassified points. In order to avoid overfitting, the classifier aims to minimize both the number of separating planes and also the number of features used. This because simpler classifiers in general perform better than more complex ones on new test data.

The best found single-plane classifier was based on mean texture, worst area and worst smoothness and separated 97.3 % of the samples successfully. When tested on 54 new samples (not previously included in training data), the machine learning algorithm made a correct diagnosis for all samples.

Successfully applying computer-based image analysis for breast cancer diagnostics had been done previously by other researchers, but the article authors argue that their method elevates the previous

research by better addressing multiple and possibly less experienced observers, because of the minimal training needed together with improved image processing as well as robustness of the machine learning algorithm.

1.2 Short summary of "The Mythos of Model Interpretability"

When using supervised machine-learning models to make classifications, a common issue is that of interpretability. The definition of interpretability is still vague, and the article tries to further define it, and compare the interpretability of a machine learning model and decisions based on professionals (e.g, a pathologist). The reason why interpretability is of great importance is because important decisions often needs to be justified by some explanation. To refer a decision to a black box model might not please all stakeholders. The problem with current models is thus that they can not reason about their decisions. If people do not know the reason behind a decision, the model could include unethical parameters into the calculation (such as racial biases).

The authors tries to define the desiderata of interpretability. To characterize a model as interpretable, it must be *transparent* and *post hoc explainable*, which means that the human decision makers need to know how the model works, and what the model can tell beyond the classification. The authors further defines the criteria of transparency in three ways; simulability, decomposability and algorithmic transparency. Moreover, post hoc interpretability is split up in text explanations, visualization, and local explanations.

Demanding a model that is both transparent and post hoc explainable puts some limitation to its performance, no matter if it consist of a decision tree or a neural network. It is also interesting to compare the transparency of a model with the transparency of a human being, In the same way a model refers their decisions on available data, pathologists make their decision based on previous research papers.

To conclude, choosing a model that is interpretable is not an easy thing do. It often depends on how interpretability is defined. A neural network can be advantageous if decomposability is emphasized but a linear model could be better if transparency is of higher importance.

2 Implementing a Diagnostic System

In order to diagnose the patient data created by the FNA method as either malignant or benign, three different classifiers were created: a rule-based classifier, a Random Forest classifier, and finally also a classifier by the RuleFit model. The implementation of these three classifiers is described below and in the end they are compared with each other.

2.1 Implementing a rule based classifier

To start with this implementation, we first wanted to know more about the given data, and what each parameter meant. That was done by reading the short descriptions given at UCL's website and by plotting histograms of the different columns. The classifier was supposed to define a persons cell as either malignant or benign by looking at four given parameters: *size, shape, texture* and *arrangement/similarity/homogeneity*. This was done by going through a number of if-statements.

In order to create these if-statements, we first needed to define thresholds that decided whether a parameter is considered to be abnormal. Before creating the classifier, we first split up the original data into two dataframes, one with all the malignant tumors and one with the benign tumors. We then plotted the different columns that we thought might be of interest to see whether one could differentiate the bad and good tumors based on this parameter.

For example, to define the size threshold, we plotted the graph shown below in Figure 1 of the data from column `area_0`. By looking at the graph, we noticed that there existed difference in size based on their classification. To decide where to put the threshold, we used the `describe` function to find the mean, standard deviation and more, and use that as a way of generalizing the threshold. Note that `area_0` represents the mean value of area and that there also exists a column representing the standard deviation of area for a particular sample (`area_1`). However, it is still relevant to talk about mean and standard deviation for the mean values `area_0` as these measures concerns the distribution of values for all patients, while `area_1` only concerns the distribution of values for a single patient. In the example below, the threshold was calculated by finding the value of the largest 25% of the

mean value of the benign tumors and then adding two standard deviations (standard deviation of mean value).

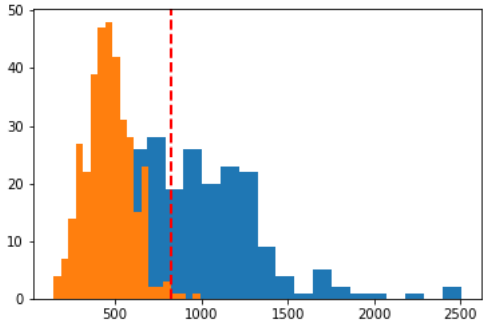


Figure 1: Histogram of the sizes of the malignant (blue) tumors and the benign (red).

This procedure was then repeated for the other given parameters. However, to calculate the thresholds of *shape* and *arrangement/similarity/homogeneity*, we first had to define those parameters. We did not find any additional descriptions of what these parameters meant, and thus made our own assumptions.

To define shape, we wanted some kind of measurements of its form based on the available measures. To get a value for this, we tried two different approaches.

$$\text{Approach 1: } shape = \frac{perimeter}{area}$$

$$\text{Approach 2: } shape = \frac{radius}{area}$$

Both ways of calculating shape give a high value to an irregular shape, since an irregular shape takes a lot of area, but the perimeter could in theory stay the same. Plotting these two different measures yielded similar results. We chose to go with approach 1, since radius seemed a bit more arbitrary for an irregular volume (since the radius varies depending based on where you measure) compared to using the perimeter.

Choosing a measure for *arrangement/similarity/homogeneity* was also difficult. After reading the descriptions at UCL's website, we though that homogeneity could be defined by measuring the concave points. To check that concave points could differentiate good and bad tumors, we plotted a histogram in the same manner as described above. Since concave points generated a clear distinction between good and bad tumors, we decided to use that as our last parameter. After some additional discussion, we also wanted to include irregularities in some other way to further define homogeneity. We therefore plotted the standard deviations of `radius` and `perimeter` to see if there existed any possibility to differentiate benign and malignant tumors based on these parameters. It turned out that is was possible. Since this definition of homogeneity was based on multiple if-statements we decided to set high threshold here, since a cell needs to pass all thresholds to be classified as benign, but only needs to fail at one to be classified as malignant.

As the suffixes on the features `_0`, `_1` and `_2` represent mean value, standard deviation and worst value of each feature for each individual patient, the initial idea was to handle these different measures in some advanced way. However, after some more thought, it was concluded that e.g. for size it is as relevant to include the mean value of the size as it is relevant to include the standard deviation of size and thus the size rule takes into account both the actual size of the cells (mean value) and the variation in size of cells (standard deviation). As it was possible to reach high accuracy by setting thresholds by the same method for all three different measures, it was not attempted to handle these three different measures in any more advanced way.

2.1.1 Creating the classifier

The fit method took the trainData and trainLabels as inputs, and based on the above-mentioned reasoning created thresholds based on the values of the trainData and trainLabels. One dataFrame consisting of all good samples, and one with all bad samples was created in order calculate the thresholds.

When all threshold were set, we looped through the entire X_test dataframe and made a classification based on the if statements.

Finally, the score function simply compared the test set with the predictions made above.

2.1.2 Results of rule based classifier

The accuracy turned out to be satisfying - at 94.7%. The confusion matrix below shows the resulting accuracy, as well as the false predictions. Strangely enough, the model made worse predictions on the training set, compared to the test set. It would have been interesting to try the model on a bigger data set, and preferably one that is created on another occasion to learn more about the true robustness of the model.

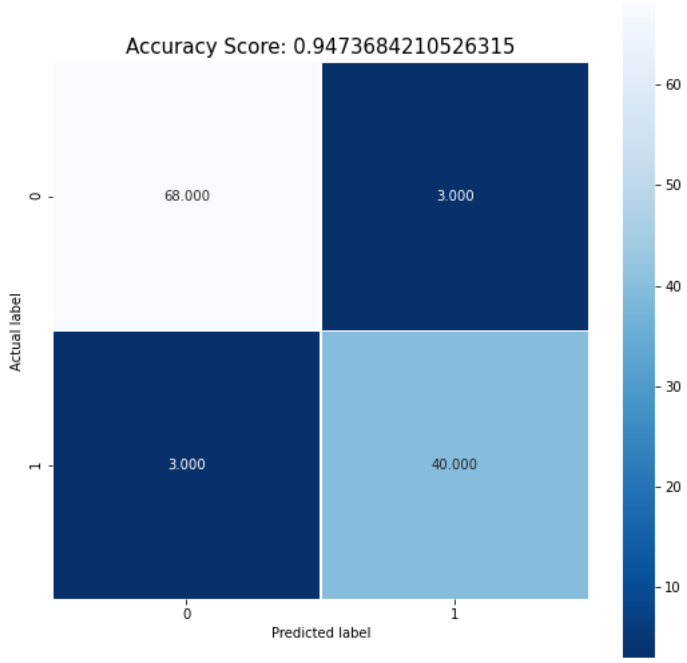


Figure 2: Confusion matrix of the result from the rule based classifier.

2.2 Implementing a random forest classifier

In this step, Scikit-learn's Random Forest Classifier was implemented. As a starting point, no pre-processing of the data was made.

By training the data on only default values for the hyperparameters, it was possible to reach a mean accuracy of 0.963 when performing a 5-fold cross validation on the data. However, the mean accuracy is varying slightly when running the 5-fold cross validation several times.

It was attempted to use GridSearchCV for optimizing the hyperparameters, where the best possible combination of values selected from the following hyperparameters was searched for: `n_estimators`, `max_depth`, `bootstrap`, `min_impurity_decrease` and `weight_fraction_leaf`. The best combination returned was `n_estimators = 25`, `max_depth = 15` and `bootstrap = False`. However, when using these hyperparameters and running the 5-fold cross validation, the mean accuracy did not improve from 0.963 (but did not decrease either).

It was also attempted to use Scikit-learn's method `SelectFromModel` for finding the most relevant features from the data. After running `SelectFromModel` on the Random Forest Classifier and the unprocessed data, the method suggested that the following features are the most

relevant ones: `'area_0', 'concavity_0', 'concave points_0', 'radius_2', 'perimeter_2', 'area_2', 'concavity_2', 'concave points_2'`. However, after only training the Random Forest Classifier with these features, the 5-fold cross validation mean accuracy decreases to 0.943.

If it was not successful to select a few of the most important features, it is possibly more sucessful to drop the least important features. All the feature importances calculated by `SelectFromModel` were sorted in descending order, as can be shown in Figure 3, and then the five features with the lowest feature importances (`fractal dimension_0, smoothness_0, smoothness_1, texture_1, compactness_0`) were dropped from the data set. However, this resulted in a 5-fold cross validation mean accuracy of 0.956, which is less than without the feature drop.

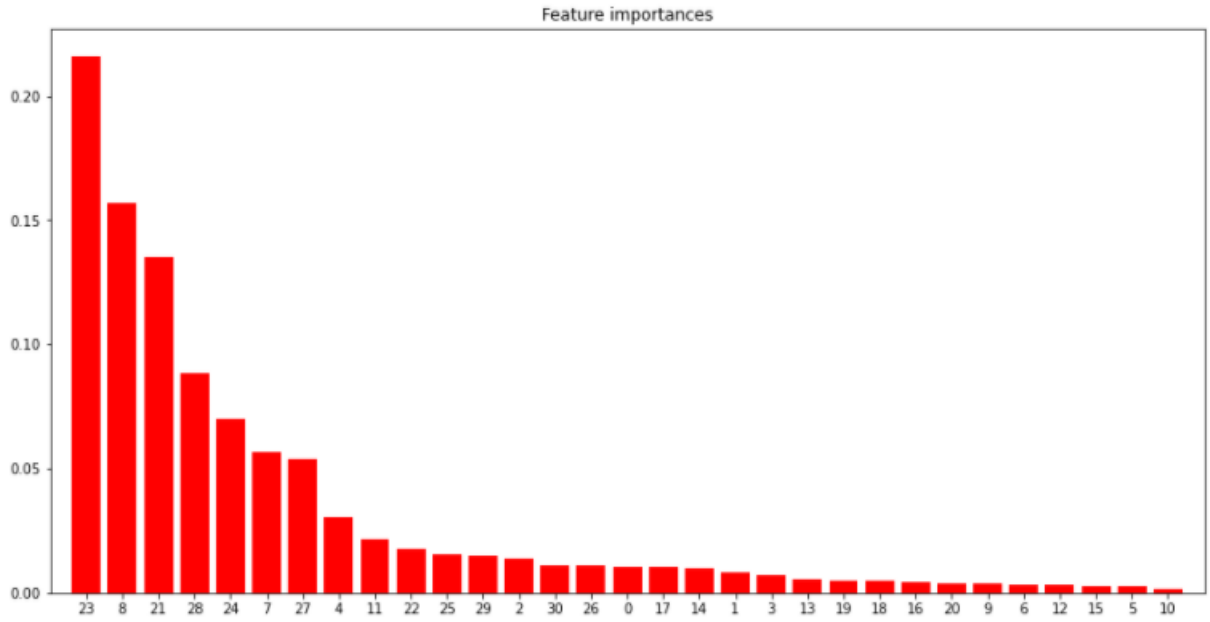


Figure 3: Feature importances by Random Forest Classifier computed by `SelectFromModel`.

Finally, a confusion matrix was created from the unprocessed data with Random Forest Classifier with optimized hyperparameters, which can be seen in Figure 4.

2.3 Implement a classifier of own choice

As mentioned by Lipton (2018), there often exists a trade off between interpretability and the performance of a model. In the best possible world, the decision of a model could be easily understood by anyone, and the output would be highly accurate. Unfortunately, this is not often the case.

To find a model that yields a high accuracy, and is yet understandable, we searched around to find articles talking about this subject and discussed pros and cons with different models. After some research, we stumbled upon a classifier called rule fit. The idea originates from Friedman and Popescu (2005), who describes the model as a perfect match between linear combinations of simple rules (which are interpretable) and more advanced models that considers interactions between the original features (which contributes to an accurate model). One could say that there is some resemblance between this model and the rule based model, since both consist of rules in a tree-like structure. The RuleFit model consists of two parts, one that generates rules using decision trees and one that fits a linear model that takes as input the initial parameters and then creates rules.

There does not yet exists any implementation of this model in the Scikit-Learn library, but luckily enough, an interpretable machine learning researcher named Christoph Molnar has created his own library that implements this classifier. His model was available for free on Github. Due to the clear benefits of this model, we decided to try it out.

The model implemented by Molnar initially generates a large number of rules, but by using Lasso when calculating the loss function for each rule, a lot of weights are set to zero and can thus easily be ignored. This leads to a final prediction based on few rules that at the same time is highly accurate. The results of the classifiers is presented in the confusion matrix in Figure 5.

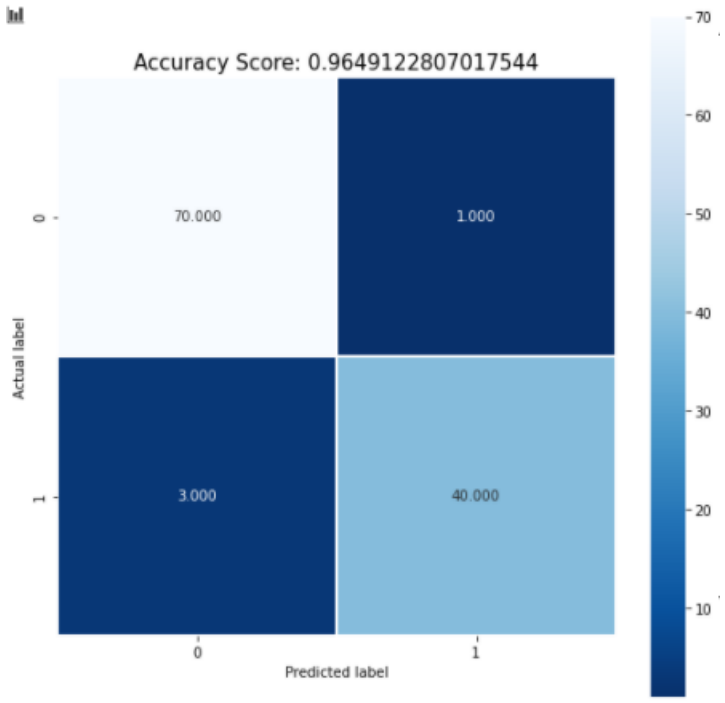


Figure 4: Confusion matrix for Random Forest Classifier.

At the same time as the figure is very accurate it is also possible to get an idea of how the classification is made. The rules made from the model can easily be retrieved. Below are the ten most important rules from the classifier:

- area_1 > 16.36 & concave points_2 > 0.092 & radius_2 > 16.789
- concavity_0 <= 0.119 & perimeter_2 > 109.45 & texture_0 <= 16.369
- concave points_2 <= 0.1429 & area_1 > 54.739
- area_2 > 718.649 & concave points_0 > 0.0512
- perimeter_2 > 105.150
- radius_2 > 16.795 & concavity_2 <= 0.2186 & texture_0 <= 19.8599
- concave points_0 > 0.0516 & texture_2 > 20.755
- texture_2 > 20.324 & concave points_0 > 0.0559 & concavity_1 <= 0.098
- fractal dimension_2 > 0.0790 & radius_2 <= 16.7899 & texture_2 > 32.614 & concave points_2 <= 0.1589
- concavity_2 > 0.2613 & concave points_0 > 0.05127

2.4 Comparison between the classifiers

Below is a table of the three classifiers and their respective accuracy, sensitivity and specificity.

Classifier	**Accuracy**	**Sensitivity**	**Specificity**
Rule Based Classifier	0.9473	0.930	0.958
Random Forest Classifier	0.9649	0.930	0.986
Rule Fit Classifier	0.9736	0.953	0.986

It is very interesting to see that the rule fit model yields the highest accuracy. With that said, the difference between the performance of the three models are minor and all models perform well.

Looking at the metrics sensitivity and specificity, it is probably more desirable with a high sensitivity of the test from the patient's point of view as a high sensitivity implies that a high share of the patients who actually have breast cancer are detected in the test. A high specificity, which implies that a high share of the actually healthy patients are given negative test results, is probably not

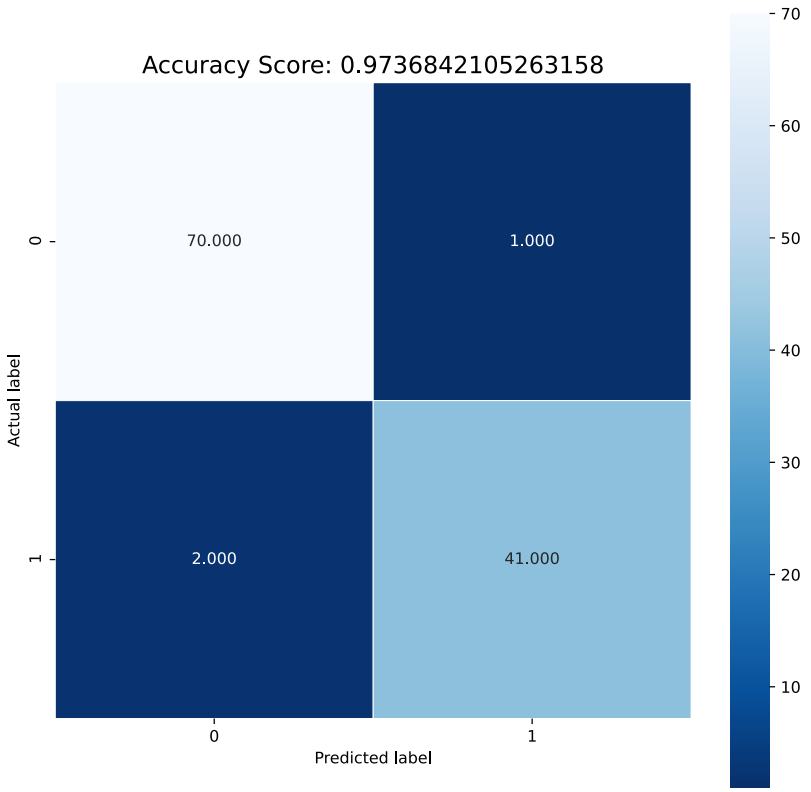


Figure 5: Confusion matrix based on the Rule Fit model

equally important as the only consequence is that the patient gets to visit the doctor once more. Comparing the different classifiers with respect to these metrics, it is clear that the Rule Fit Classifiers yields the most desirable results here as both the sensitivity and specificity values are highest among all classifiers.

Regarding interpretability, the rule based classifier is the easiest model to understand, since it only consists of pre-defined rules that originates from human insights. The rule fit model is also relatively easily to interpret, since it only consist of a some few rules. For a deeper understanding of its classifications, more time must be put in to fully learn how the model works. The random forest was the classifier that, according to us, is the least interpretable model. In order to gain some insight about its decisions, we printed some of its decision trees in Figure 6. But since the forest consist of a large number of trees, it is hard to draw any conclusion by looking at a few sample trees. Moreover, since the maximum depth was set to 15, each tree becomes quite intricate.

In the article "The Mythos of Model Interpretability", Lipton (2018) highlights the desiderata of interpretability, one of them being causality. In the models described above, it is hard to draw any conclusion on how the parameters causes other to change. Figure 5 shows the most important parameters of the rule fit model. On sixth place comes id. That means that the decision of whether a tumor is benign or malignant is partly based on what id a certain patient is assign with. This is an example of correlation without causation. When the model is relatively small and is based on few parameters, it is possible to discover such an error. However, if this approach was implemented on a larger scale, this could easily be missed. And let us say that is was not id that showed some random importance but race or gender? This problem further highlights the importance of interpretability of a machine learning model.

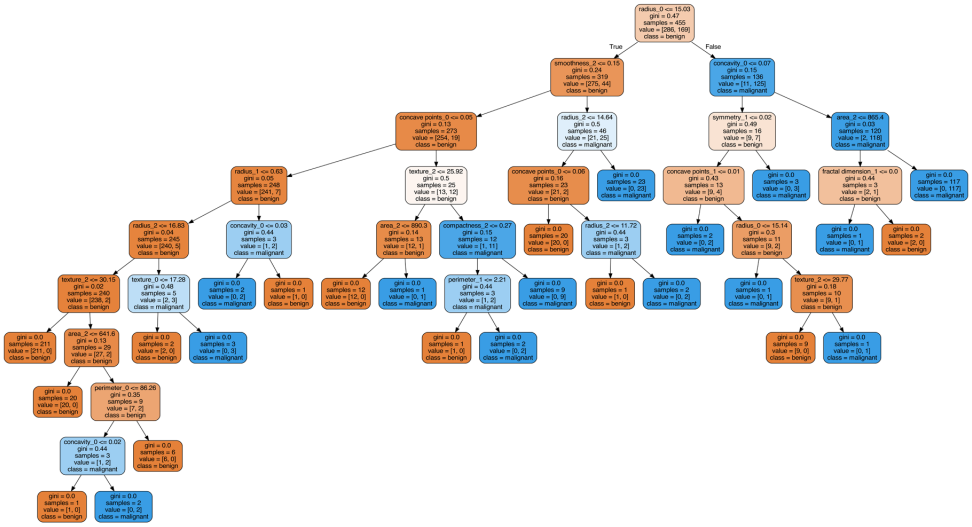


Figure 6: One tree from the random forest

	0	Importance
0	fractal dimension_1	6.486579
1	concave points_2	4.433193
2	concavity_2	4.372342
3	area_1	3.309823
4	area_2	3.115748
5	compactness_2	3.076998
6	id	2.741779
7	compactness_1	2.467120
8	compactness_0	2.366987
9	concave points_0	2.193401

Figure 7: Feature importance from Rule Fit model

3 Discussion

Interpretability of systems, models and predictions is important in many applications. Healthcare is one of these. Discuss the meaning of interpretability in the context of this week's assignment, its potential benefits and drawbacks, and how interpretability may be defined. Be as specific as you can and use mathematical notation where appropriate. Take inspiration from e.g., "The Mythos of Model Interpretability".

3.1 Max Sonnelid

Implementing Artificial Intelligence systems in healthcare applications can both lead to improved decision-making for physicians by letting AI systems be an augmentation of physicians' professional knowledge and also to more efficient patient flows as AI systems are often able to recommend decisions faster than humans. However, as healthcare often involves situations where an incorrect decision might lead to death, it is crucial that both physicians and patients are able to trust the AI systems used in health care.

Connecting this topic to this week's assignment, it is an interesting question whether trust necessarily needs to equal interpretability. For example, if an AI-based breast cancer diagnosis system has been proved over years to always make 100 % accurate diagnoses, is there an actual purpose of trying to understand the diagnostics process? Probably not in such a case, but in most practical applications

AI systems are not 100 % accurate, as is the fact for our implementation. Probably our breast cancer diagnosis system would be able to bring value to both physicians and patients, but not by only providing the sole diagnosis of cancer or not cancer as there is a rather high uncertainty connected to this diagnosis. The value creation from this system lies in understanding how the breast cancer is diagnosed and possibly the AI system has then identified new causal relationships that human physicians have not been able to identify before.

In order to understand how the diagnoses are made, it is necessary to be able to interpret the ML model in some way. An interesting method for creating interpretability for ML classifiers is called LIME (Local Interpretable Model-agnostic Explanations), described by Sharma (2018), which for any individual prediction made is able to learn an interpretable explanation model locally around the prediction. This is done by LIME minimizing the following explanation model equation:

$$E(x) = \underset{g \in G}{\operatorname{argmax}} \, \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{1}$$

In the equation above, x represents the original features, f the original predictor, g the explanation model and π the proximity measure for defining locality around x. The first term is called the locality-aware loss is measures how well g approximates f in the locality defined by π, while the last term measures the model complexity of g.

An application of LIME to breast cancer detection has been done by Jansen et al. (2020), where the researchers were able to assign features weights by LIME to individual patients diagnosed for breast cancer by a ML model. As the explanation models made sense, the researchers concluded that LIME could be an important contribution for explaining complex ML models. Being able to use automatic explanation models as LIME is a big step towards better understanding how ML models used in healthcare settings make diagnoses and could possibly contribute to discover brand new causal relationships for different diseases.

3.2 Eric Johansson

Interpretable models are always useful, no matter what areas machine learning is implemented. In some cases, the importance of a model that can be understood is not only useful, but of great essence. One of the areas where interpretabililty is a critical factor is in healthcare. In this weeks assignment, we were supposed to determine weather a tumor is malignant or benign. The following treatment that a patient will get as a result of the classification will be widely different depending on what class the tumors gets. Even if a machine learning model would outperform any human pathologist, the model would not be trusted if it makes its decisions in a black box manner. In the cases where such a model classifies a malignant tumor as benign, who is there to blame when that patients cancer grows beyond rescue?

In "The Mythos of Model Interpretability", Lipton (2018) talks about three ways for a model to be interpretable, namely simulability, decomposability, and algorithmic transparency. A model that is simulatable is a model that can be understood all at once. In the above-mentioned case, I wouldn't say that this should be a criteria for tumor classifier, since such a model would need to be very incomplex and include few calculations. If such a model is used, the chance that it would outperform current systems would be relatively low. The second notion of transparency is decomposability that needs each segment of the model (input, parameters and calculations) to be understood in separate. A model that is decomposable could potentially be useful and trusted in a real world situation, but it is hard to determine how well it would perform compared to humans. Finally, an algorithmic transparent model is a model were one can understand how an algorithm learns something from the data and what relationships it can learn from new data. These models could probably yield high accuracy and, at the same time be understood by people with the proper knowledge in computer science.

According to Robnik-Sikonja and Bohanec (2018), there exist many way of evaluating the interpretability of model. Some of these parameters are accuracy (performance on previously unseen data), consistency (how much does the explanation differs between each evaluation) and comprehensibility (how well does a human understands the given justifications given by the model). Robnik-Sikonja and Bohanec (2018) mentions several additional parameters that one can consider when evaluating a model, but lets focus on these three when looking at our original problem. Accuracy could in our case be defined as the number of correct classifications divided by the total number

of classifications. Though, since there will be more serious outcomes of false-negative (sick patients does not get proper treatment) compared to false-positive (patient gets excessive treatment) it would be interesting to look at the rate of miss-classifications. This could for example be done by plotting a confusion matrix or a ROC-curve to determine the proper sensitivity of a model. Consistency could be evaluated if the model can explain its decisions through for example a decision tree or by rules. If it is able to explain its way of classifying data, one could use new sets of data as input and see how the importance of each parameter changes with each new data set. The final parameter, comprehensibility, can be evaluated simply by trying to understand the model, and potentially try to replicate the classifications based on its given rules.

4 Summary of lectures

4.1 Max Sonnelid

4.1.1 Lecture 5 - Diagnostic Systems

- A technically simple version of a diagnostic system is called *knowledge-based expert system*, which use logic for inference based on a knowledge base, i.e. gives instructions to doctors on how to diagnose diseases based on specific symptoms. Two examples of such systems are *Mycin* and the probabilistic system *INTERNIST-1*.
- A drawback of knowledge-based expert systems are that they rely on manually curated knowledge bases, which is time-consuming to program and thus makes it hard to scale such systems.
- *Diagnostics* can be defined as finding out the cause of some phenomenon, e.g. human disease or a broken phone, by observing symptoms or other noticeable features. Diagnostics is always data-based, regardless whether a human doctor or a computer makes the diagnosis.
- Three different types of diagnostics are: *clinical diagnosis* (signs and symptoms), *laboratory diagnosis* (laboratory tests) and *radiology diagnosis* (imaging).
- A suitable classifier for diagnostics is a *Naïve Bayes classifiers*, where the probabilities for symptoms are conditionally independent of the disease. Then Bayes rules can be used for calculating the probability for disease given symptoms.
- Benefits of using a Naïve Bayes classifier is that probabilities of symptoms given diseases can be easily estimated from data, it handles lack of data rather smoothly and can easily be generalized to many different kind of diseases.
- Diagnostic tests will almost never have a 100 % accuracy and therefore the *trade-off between specificity and sensitivity* is important to consider as well as the related costs to these trade-off.
- A big opportunity for Machine Learning in diagnostics is to *improve consistency* in e.g. pathology as many pathologists today disagree significantly in their interpretation of different conditions.
- Another opportunity is that Machine Learning is *not limited to known explanations* to diseases as features are learned from data. Totally new explanations and causes, which traditional medical research has not been able to discover, could therefore be discovered by Machine Learning models.

4.1.2 Follow-up lecture on module 4

- An important similarity between rule-based translation systems and state-of-the-art neural systems is that both of these systems use a sort of *interlingua* as an abstraction of the source language sentence before it is translated into the target language.
- One approach for solving the decoding task was to *extract the transition probabilities* between source and target languages for all words in a given sentence from the source language, then calculate *all possible permutations of the most likely translations* for each word and then *maximize the probability of a sentence* in the target language. However, this only works for short sentences and is a rather slow algorithm.
- One approach of computing word frequencies for very large data sets is to *process the data files sequentially*, not load all of the data simultaneously into the working memory and then calculate

the word frequencies sequentially. A similar method is to split the data into smaller batches, *distribute these batches over several machines* which each calculates the word frequencies for their respective batch and then all individual word frequencies are eventually combined.

- An interesting application of language models is in *time series analysis*, where a language model can help predicting what will happen next in a certain time series based on previous event and also help identify whether a current event is an anomaly or not.
- The technical reason why a gender-neutral pronoun in one language is translated to a certain gender-specific pronoun in another language is a *consequence of sentence frequencies in the training data*. However, this probabilistic approach risk reinforcing undesirable stereotypes from the current society.
- Evaluation of translation systems can be done by *manually looking at the output* and assess whether the translation is useful or not. This manual method is more accurate, but is rather costly. An *automatic method will compare some kind of reference translation with the machine translation* and then calculate the error rate by comparing string by string. However, often there exists several ways of translating a sentence, which the automatic method struggles to handle.

4.2 Eric Johansson

4.2.1 Lecture 5 - Diagnostic Systems

Before AI were coined, something called a knowledge-based expert system was used in healthcare. They used logic for inference based on knowledge base. One of the first models where Mycin, develop at Stanford which yielded good result in 65% of times it made prediction (which was good in this case). Later on, the INTERNIST-1 managed to associate deceases with symptoms. Here, a lot of pre-defined rules were used in order to make classifications. The first time a neural network in medicine was used was in the 1990's where it diagnosed people with potential breast cancer. Some problem with this technique at the time was partly lack of data and that the model were manually curated on knowledge base.

Diagnosis is the task of finding a cause to some phenom om (e.g., a decease based on symptoms). In medicine, there exists several types of diagnostics, such as clinical-, laboratory-, and radiology diagnosis. When making classification, there often exist false positive, and false negative classification. The lower amount of the false classifications, the better the test is. The relative share between false positive and false negatives can be tweaked by changing thresholds. A low threshold could lead to fewer false negatives, but more false positives. This is known as the sensitivity and specificity trade off. A commonly used model when diagnosing symptoms is the Naive Bayes model.

To summarize, diagnostics is never right 100% of the time, but are still usefill. We must also think about the specificity/sensitivity trade off in each case since the costs is different from time to time.

There are multiple reasons why machine learning usage is currently growing. A very important reason is data is being collected in all sectors and much more often than before. For example, medical health records are now stored electronically, whereas it was written down on paper before. Moreover, the collected data is being standardized, which facilitates comparison between different data sets. The large amount of data is also useful since the computing power is increasingly getting better.

The opportunities that comes out of ML is threefold. Firstly, using ML can improve consistency, which leads to more accurate diagnoses. Secondly, the speed of diagnoses can be increased. For example, AI can narrow the scope, so that a doctor only needs to focus on a more narrow part of an image or a list of symptoms. Thirdly, using ML, one can explore new patterns that are not yet identified. This is called exploration. With all these pros, it seems obvious that ML shoudl be used as often as possible. It is however important to note that there exist some problems with ML, such as reproducibility. For a model to be useful, it has to be reproducible, which often is a big hurdle.

4.2.2 Follow-up lecture

Even though neural networks generate accurate predictions, it might not always be the optimal option. There exist problems that can be better solved by using "old-school" rule based solutions,

for example when the system needs to be fully understood in order to trust the output. Another example is when there doesn't exist any training data.

Another important lesson is that of time complexity. It is essential to be aware of that the data samples given in the assignments are very small compared to real life problem. When tackling a real problem, a task such as reading the data can be an issue if the set is large enough. One way of solving this is to use map reduce, which uses a divide and conquer approach to solve the problem.

Moreover, the previous assignment is further discussed and common problem are highlighted as well as potential solution. For example, a problem that we encountered was that it was hard to calculate the probability for longer sentences. This could be solved by calculating the logarithmic probability of the word sequence. Another interesting take away from the reflection lecture was that building a translation model (or any kind of AI) by using existing data made by humans can unintentionally become biased towards different groups or ethnicity's. This is a problem than currently doesn't have a clear solution and thus, it is even more important to be aware of the risks of AI. One thing that can be done to mititgate the risk of a racsist AI is to be cautious when selecting what data to train the model on (tweets may not favourable).

5 Reflection on the previous module

5.1 Max Sonnelid

Implementing a somewhat fully functioning machine translation system by implementing the IBM model 1 was a challenging, but also a rewarding task. Compared to the other assignments, this one required the most advanced programming skills as all the modules of the IBM model 1 were pre-defined and there was little room for adapting the complexity of the model with regard to our own programming skills, which to some extent has been the case for the two previous assignments. However, having to implement a rather complex system forced us to step out of our comfort zone in programming and thus forced us to elevate our structure in programming, by e.g. divide the program into smaller methods and use more efficient data structures such as dictionaries.

Furthermore, it was interesting to implement a program which was divided into rather separate modules. This let us divide the work more efficiently by me taking more responsibility for the language model, while Eric took more responsibility for the translation model. By this division of work, little overlapping work was done and we could each gain a deep understanding of our own respective modules and eventually be able to get reasonable outputs from the language respectively translation models. As both the input and output from these two models was well-defined from start, it was later easy to integrate both of these models into one machine translation program. A module-based work method, with well defined input and output from each module, proved to be very efficient and we should attempt to use this work method in future assignments and projects.

Finally, it was interesting that the output of the model was not measurable in such a way that the output from classifiers or regressors is. The reflection about suitable evaluation metrics for machine translation was interesting as there are many different ways of defining what a good translation is and equally many ways of evaluating whether that definition of a good translation is achieved or not. This discussion share similarities with the discussion about suitable evaluation metrics for recommender systems. For both machine translation and recommender systems, a good translation respectively a good recommendation is something very subjectively, depending on each person's personal taste, and therefore fascinating areas to learn more about.

5.2 Eric Johansson

The purpose of this module was to learn more about the history of machine translation, the different techniques that have been used over the years and their respective pros and cons. Moreover, the practical implementation of a translation system led to deeper knowledge about machine translation.

This module was the, so far, most challenging one. I liked that the module was introduced by an article that covered the history of machine translation, to give a better understanding of where we are in the development of this subject and what led to this development. The second part of the module, which was the implementation of a model, was fun but took a long time to complete. This time, we quickly realised that this could be a challenging task and thus, it would be hard to do all the tasks side by side. What we instead did was to split up the task in smaller parts and try to

solve them individually. If one person managed to solve a part, we stop and discussed that part until both members understood the problem. This was a more efficient way of solving the task, but not as fun since I find it easier to digest a problem by discussing it in-depth with someone else.

Similar to the previous module, we once again saw the importance of being aware of the time complexity of a model. This time we put some more thought into it before implementing the most challenging part of the lab, which was the EM-algorithm. A decision that led to a better model was to store our words in a map structure using defaultdicts. This led to fewer loops compared to using a list structure, and thus a faster model. A takeaway from this lab was the importance of understanding the idea behind an algorithm before implementing it. We had a lot of issues with implementing the EM-algorithm. Eventually, it was solved by changing the indentation of a for-loop, something that could have been spotted much faster if more time was put on understanding the algorithm more thoroughly

References

Friedman, J. H. and Popescu, B. E. (2005). Predictive learning via rule ensembles.

Lipton, Z. C. (2018). The mythos of model interpretability. *acmqueue*.

Robnik-Sikonja, M. and Bohanec, M. (2018). Perturbation-based explanations of prediction models.

Design of AI systems

Assignment 5, Eric Johansson & Max Sonnelid

In [1]:

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
```

In [2]:

```
with open('data/wdbc.pkl', 'rb') as f:
    data = pickle.load(f)
```

In [3]:

```
data.head()
```

Out[3]:

	id	malignant	radius_0	texture_0	perimeter_0	area_0	smoothness_0	compactne
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.2
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.0
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.1
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.2
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.1

5 rows × 32 columns

In [4]:

```
data.columns
```

Out[4]:

```
Index(['id', 'malignant', 'radius_0', 'texture_0', 'perimeter_0', 'area_
0',
       'smoothness_0', 'compactness_0', 'concavity_0', 'concave points_0',
       'symmetry_0', 'fractal dimension_0', 'radius_1', 'texture_1',
       'perimeter_1', 'area_1', 'smoothness_1', 'compactness_1', 'concavit
y_1',
       'concave points_1', 'symmetry_1', 'fractal dimension_1', 'radius_
2',
       'texture_2', 'perimeter_2', 'area_2', 'smoothness_2', 'compactness_
2',
       'concavity_2', 'concave points_2', 'symmetry_2', 'fractal dimension
_2'],
      dtype='object')
```

In [5]:

```
radius_cols = ['radius_0', 'radius_1', 'radius_2']
radius = data[radius_cols]
```

In [6]:

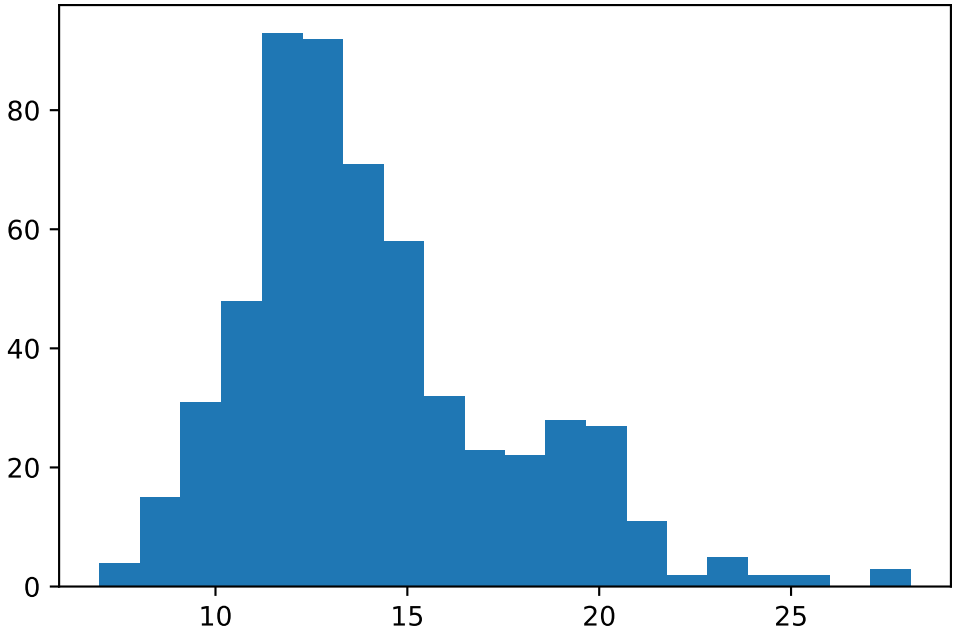
```
radius0 = radius[radius.columns[0]]
radius1 = radius[radius.columns[1]]
radius2 = radius[radius.columns[2]]
```

In [7]:

```
plt.hist(radius0, bins = 20)
#plt.hist(radius1, bins = 20)
#plt.hist(radius2, bins = 20)
```

Out[7]:

```
(array([ 4., 15., 31., 48., 93., 92., 71., 58., 32., 23., 22., 28., 27.,
        11.,  2.,  5.,  2.,  2.,  0.,  3.]),
 array([ 6.981  ,  8.03745,  9.0939 , 10.15035, 11.2068 , 12.26325,
        13.3197 , 14.37615, 15.4326 , 16.48905, 17.5455 , 18.60195,
        19.6584 , 20.71485, 21.7713 , 22.82775, 23.8842 , 24.94065,
        25.9971 , 27.05355, 28.11   ]),
 <a list of 20 Patch objects>)
```



In [8]:

```
radius1.describe()
```

Out[8]:

```
count    569.000000
mean       0.405172
std        0.277313
min        0.111500
25%        0.232400
50%        0.324200
75%        0.478900
max        2.873000
Name: radius_1, dtype: float64
```

In [9]:

```
badCells = data[data['malignant']==1]
goodCells = data[data['malignant']==0]
radius_badCells = badCells[radius_cols]
```

Size thresholds

In [87]:

```
area_good = goodCells['area_0']
area_bad = badCells['area_0']
```

In [88]:

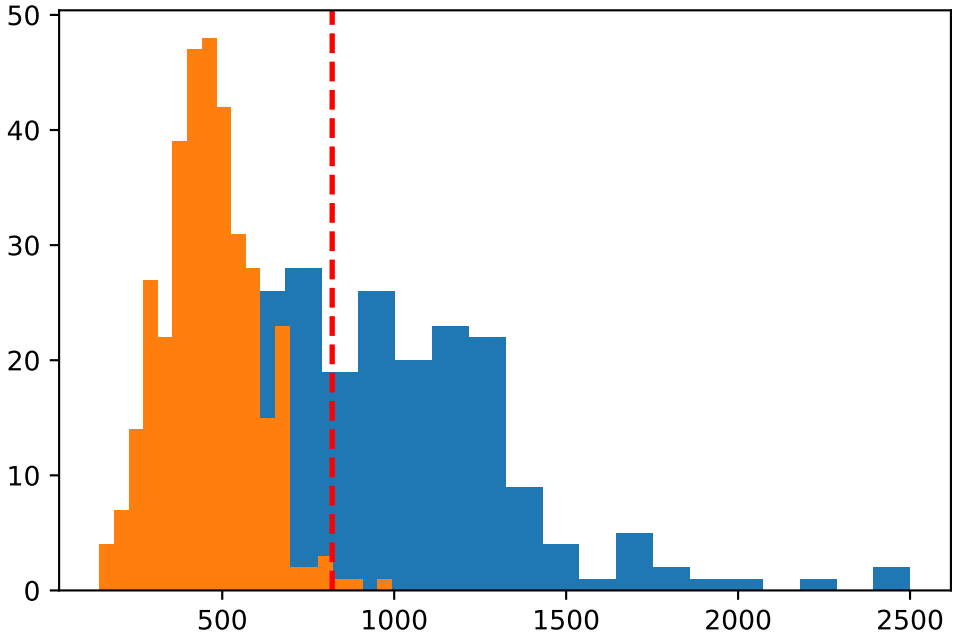
```
# Choosing a threshhold for what is abnormal. Area above threshold is considered a bad
 sign
threshold_area = area_good.describe(include='all').loc['75%']+area_good.describe(includ
e='all').loc['std']*2
threshold_area
```

Out[88]:

```
819.6742362941192
```

In [89]:

```
plt.hist(area_bad, bins=20)
plt.hist(area_good, bins = 20)
plt.axvline(x=threshold_area, color='r', linestyle='dashed', linewidth=2)
plt.savefig('size_plot.png')
```



Shape thresholds

Checking if shape can be described as perimeter/area

In [13]:

```
shape_cols = ['perimeter_0', 'area_0', 'perimeter_1', 'area_1', 'perimeter_2', 'area_2'
]
shape_good = goodCells[shape_cols]
shape_good['shape_0'] = shape_good['perimeter_0']/shape_good['area_0']
shape_good['shape_1'] = shape_good['perimeter_1']/shape_good['area_1']
shape_good['shape_2'] = shape_good['perimeter_2']/shape_good['area_2']

shape_bad = badCells[shape_cols]
shape_bad['shape_0'] = shape_bad['perimeter_0']/shape_bad['area_0']
shape_bad['shape_1'] = shape_bad['perimeter_1']/shape_bad['area_1']
shape_bad['shape_2'] = shape_bad['perimeter_2']/shape_bad['area_2']
```

In [14]:

```
print(shape_bad['shape_0'].describe())
print(' ')
print(shape_good['shape_0'].describe())
```

```
count    212.000000
mean       0.125965
std        0.023412
min        0.074730
25%        0.108704
50%        0.121338
75%        0.141991
max        0.202710
Name: shape_0, dtype: float64

count    357.000000
mean       0.176477
std        0.028502
min        0.115513
25%        0.156008
50%        0.171288
75%        0.189386
max        0.305157
Name: shape_0, dtype: float64
```

In [15]:

```
# Shape below threshold is considered a bad sign
threshold_shape = shape_good['shape_0'].describe(include='all').loc['75%']-shape_good[
'shape_0'].describe(include='all').loc['std']/2
threshold_shape
```

Out[15]:

```
0.17513535964746452
```

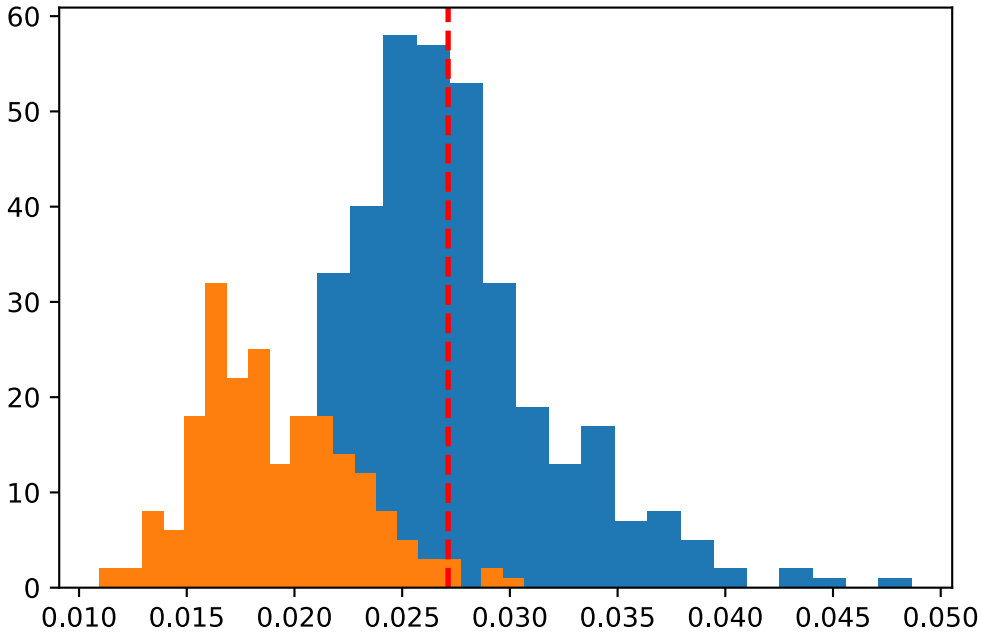
In [68]:

```
print(f"Mean of shape at bad tumours {shape_bad['shape_0'].mean()}")
print(f"Mean of shape at good tumours {shape_good['shape_0'].mean()}")

plt.hist(shape_good['shape_0'], bins = 20)
plt.hist(shape_bad['shape_0'], bins = 20)
plt.axvline(x=threshold_shape, color='r', linestyle='dashed', linewidth=2)
```

```
Mean of shape at bad tumours 0.01909188424660167
Mean of shape at good tumours 0.027492179030757978
```

Out[68]:

```
<matplotlib.lines.Line2D at 0x7f9dc8ba3d00>
```



In [ ]:

Comparing above with shape = radius/area

In [17]:

```
shape_cols = ['radius_0', 'area_0', 'radius_1', 'area_1', 'radius_2', 'area_2']
shape_good = goodCells[shape_cols]
shape_good['shape_0'] = shape_good['radius_0']/shape_good['area_0']
shape_good['shape_1'] = shape_good['radius_1']/shape_good['area_1']
shape_good['shape_2'] = shape_good['radius_2']/shape_good['area_2']

shape_bad = badCells[shape_cols]
shape_bad['shape_0'] = shape_bad['radius_0']/shape_bad['area_0']
shape_bad['shape_1'] = shape_bad['radius_1']/shape_bad['area_1']
shape_bad['shape_2'] = shape_bad['radius_2']/shape_bad['area_2']
```

In [18]:

```
threshold_shape = shape_good['shape_0'].describe(include='all').loc['75%']-shape_good[
'shape_0'].describe(include='all').loc['std']/2
threshold_shape
```

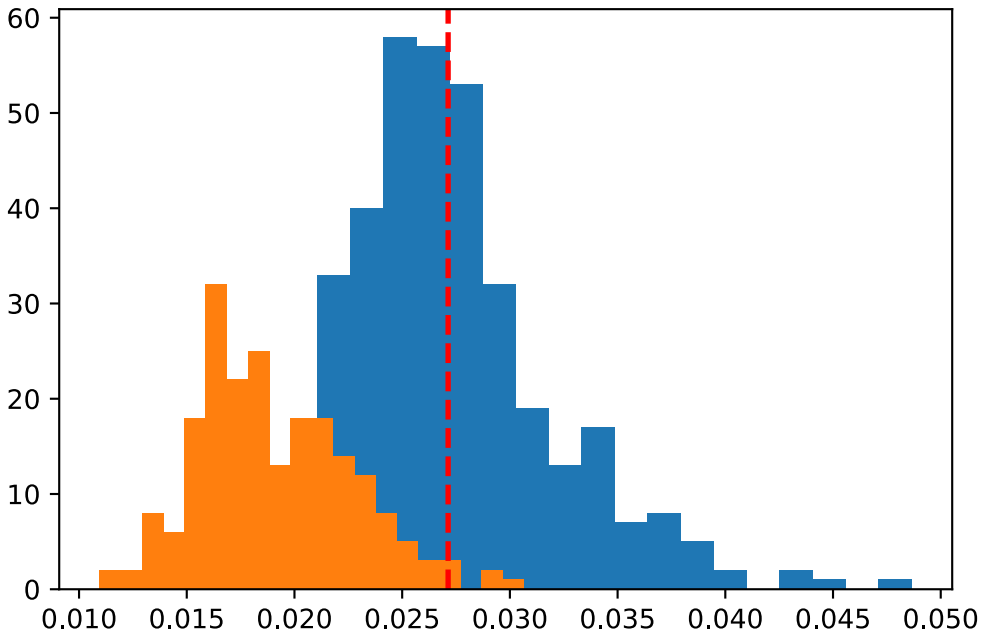
Out[18]:

```
0.02713289708901503
```

In [19]:

```
print(f"Mean of shape at bad tumours {shape_bad['shape_0'].mean()}")
print(f"Mean of shape at good tumours {shape_good['shape_0'].mean()}")
plt.hist(shape_good['shape_0'], bins = 20)
plt.hist(shape_bad['shape_0'], bins = 20)
plt.axvline(x=threshold_shape, color='r', linestyle='dashed', linewidth=2)
plt.show()
```

```
Mean of shape at bad tumours 0.01909188424660167
Mean of shape at good tumours 0.027492179030757978
```



Texture thresholds

In [137]:

```
texture_bad = badCells['texture_0']
texture_good = goodCells['texture_0']
```

In [138]:

```
print(texture_bad.describe())
print(' ')
print(texture_good.describe())
```

```
count    212.000000
mean      21.604906
std        3.779470
min       10.380000
25%       19.327500
50%       21.460000
75%       23.765000
max       39.280000
Name: texture_0, dtype: float64

count    357.000000
mean      17.914762
std        3.995125
min        9.710000
25%       15.150000
50%       17.390000
75%       19.760000
max       33.810000
Name: texture_0, dtype: float64
```

In [139]:

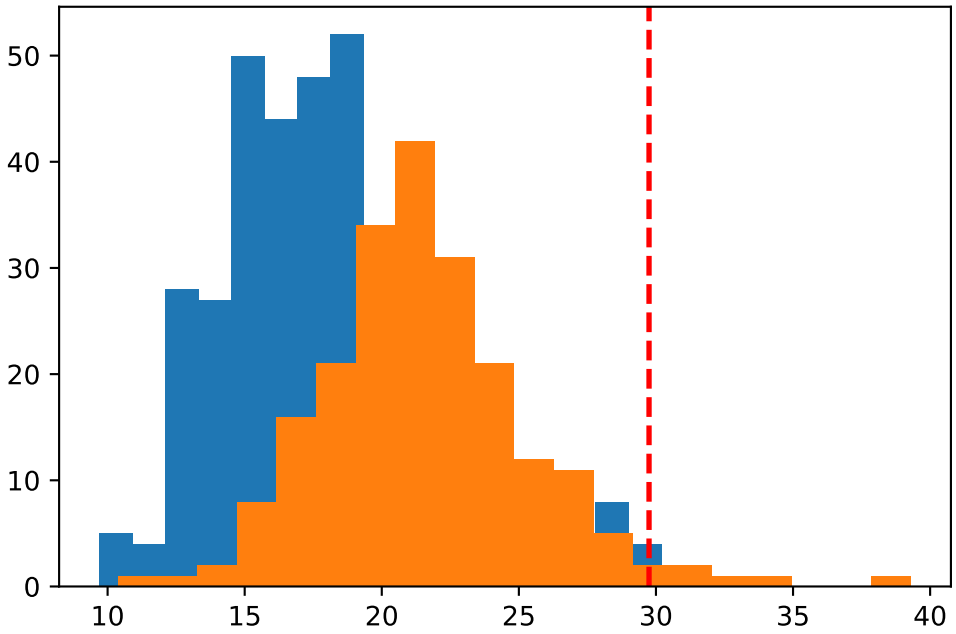
```
# Texure above threshold is considered a bad sign
threshold_texture = texture_good.describe(include='all').loc['75%']+texture_good.descri
be(include='all').loc['std']*5/2
threshold_texture
```

Out[139]:

```
29.747811484189782
```

In [140]:

```
plt.hist(texture_good, bins = 20)
plt.hist(texture_bad, bins=20)
plt.axvline(x=threshold_texture, color='r', linestyle='dashed', linewidth=2)
plt.show()
```



Homogenity thresholds

In [24]:

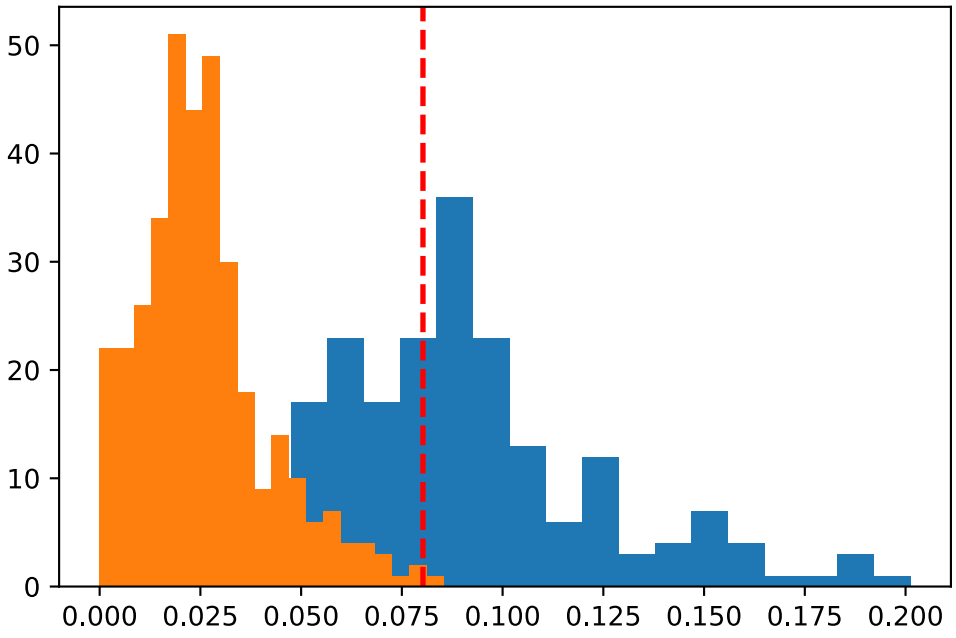
```
threshold_homogenity = goodCells['concave points_0'].describe(include='all').loc['75%']
+goodCells['concave points_0'].describe(include='all').loc['std']*3
threshold_homogenity
```

Out[24]:

```
0.08023633513482431
```

In [25]:

```
# Using concave points as a measure (which is the number of concave portions of the con
tour)
plt.hist(badCells['concave points_0'], bins = 20)
plt.hist(goodCells['concave points_0'], bins = 20)
plt.axvline(x=threshold_homogenity, color='r', linestyle='dashed', linewidth=2)
plt.show()
```

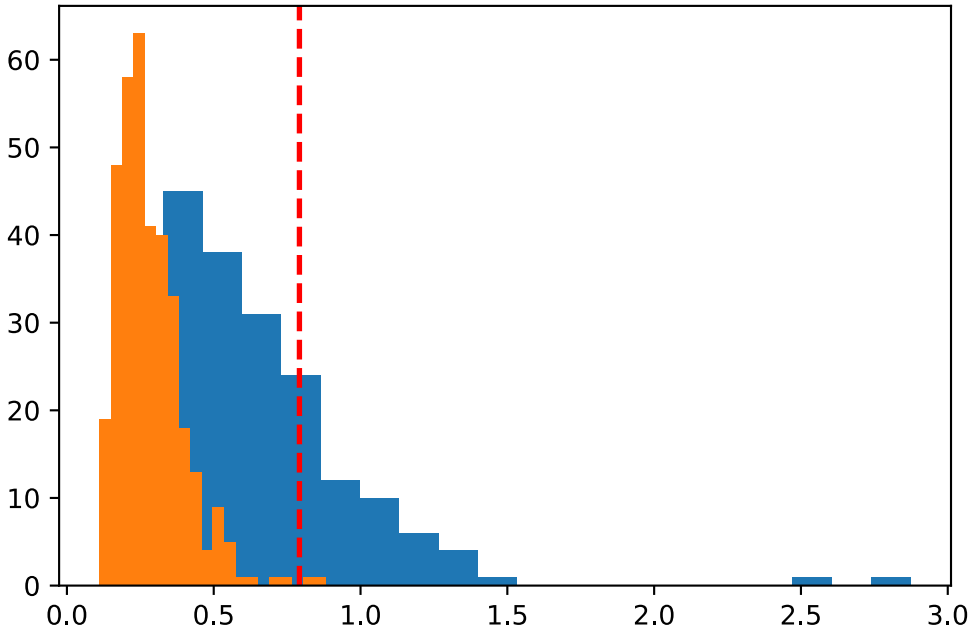


In [119]:

```
# Trying to check if the variance of radius, perimeter and area can say anything about
 homogenity
threshold_homogenity2 = goodCells['radius_1'].describe(include='all').loc['75%']+goodCe
lls['radius_1'].describe(include='all').loc['std']*4
```

In [120]:

```
# Trying to check if the variance of radius, perimeter and area can say anything about
 homogenity
plt.hist(badCells['radius_1'], bins = 20)
plt.hist(goodCells['radius_1'], bins = 20)
#plt.hist(badCells['perimeter_1'], bins = 20)
#plt.hist(goodCells['perimeter_1'], bins = 20)
plt.axvline(x=threshold_homogenity2, color='r', linestyle='dashed', linewidth=2)
plt.show()
```



Using the standard deviation a meassure of homogenity seems like a good option. Since we use both radius and perimeter as two separate if statements, we use high thresholds in order to mitigate the risk of false-negatives.

Splitting the data

In [26]:

```
target = data['malignant']
data = data.drop(columns=['malignant'])
```

In [27]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(data,target, test_size = .2, random
_state = 42)
```

In [28]:

```
data.columns
```

Out[28]:

```
Index(['id', 'radius_0', 'texture_0', 'perimeter_0', 'area_0', 'smoothness
_0',
       'compactness_0', 'concavity_0', 'concave points_0', 'symmetry_0',
       'fractal dimension_0', 'radius_1', 'texture_1', 'perimeter_1', 'are
a_1',
       'smoothness_1', 'compactness_1', 'concavity_1', 'concave points_1',
       'symmetry_1', 'fractal dimension_1', 'radius_2', 'texture_2',
       'perimeter_2', 'area_2', 'smoothness_2', 'compactness_2', 'concavit
y_2',
       'concave points_2', 'symmetry_2', 'fractal dimension_2'],
      dtype='object')
```

Creating rule based classifier

In [134]:

```
class rbClassifier:

    def __init__(self):
        self.text = 'test'


    def fit(self, X, Y):
        self.trainData = X
        self.trainData['label'] = Y

        # Splitting up the df into goodCells and badCells in order to define useful thr
esholds
        self.goodCells = self.trainData[self.trainData['label']==0]
        self.badCells = self.trainData[self.trainData['label']==1]

        # Creating thresholds for area
        self.threshold_area_0 = self.goodCells['area_0'].describe(include='all').loc['7
5%']+self.goodCells['area_0'].describe(include='all').loc['std']*2
        self.threshold_area_1 = self.goodCells['area_1'].describe(include='all').loc['7
5%']+self.goodCells['area_1'].describe(include='all').loc['std']*2
        self.threshold_area_2 = self.goodCells['area_2'].describe(include='all').loc['7
5%']+self.goodCells['area_2'].describe(include='all').loc['std']*2

        # Adding the column shape
        self.goodCells['shape_0'] = self.goodCells['perimeter_0']/goodCells['area_0']
        #self.goodCells['shape_1'] = self.goodCells['perimeter_1']/goodCells['area_1']
        self.goodCells['shape_2'] = self.goodCells['perimeter_2']/goodCells['area_2']

        # Creating thresholds for shape
        self.threshold_shape_0 = self.goodCells['shape_0'].describe(include='all').loc[
'75%']-5*self.goodCells['shape_0'].describe(include='all').loc['std']/2
        #self.threshold_shape_1 = self.goodCells['shape_1'].describe(include='all').loc
['75%']-5*self.goodCells['shape_1'].describe(include='all').loc['std']/2
        self.threshold_shape_2 = self.goodCells['shape_2'].describe(include='all').loc[
'75%']-5*self.goodCells['shape_2'].describe(include='all').loc['std']/2

        # Creating threshold for texture
        self.threshold_texture_0 = self.goodCells['texture_0'].describe(include='all').
loc['75%']+self.goodCells['texture_0'].describe(include='all').loc['std']*5/2
        #self.threshold_texture_1 = self.goodCells['texture_1'].describe(include='al
l').loc['75%']+self.goodCells['texture_1'].describe(include='all').loc['std']*5/2
        self.threshold_texture_2 = self.goodCells['texture_2'].describe(include='all').
loc['75%']+self.goodCells['texture_2'].describe(include='all').loc['std']*5/2

        # Creating threshold for homogenity
        self.threshold_homogenity_0 = self.goodCells['concave points_0'].describe(inclu
de='all').loc['75%']+self.goodCells['concave points_0'].describe(include='all').loc['st
d']*3
        self.threshold_homogenity_1 = self.goodCells['concave points_1'].describe(inclu
de='all').loc['75%']+self.goodCells['concave points_1'].describe(include='all').loc['st
d']*3
        self.threshold_homogenity_2 = self.goodCells['concave points_2'].describe(inclu
de='all').loc['75%']+self.goodCells['concave points_2'].describe(include='all').loc['st
d']*3

        # Creating threshold for homogenity
        self.threshold_homogenity_peri = self.goodCells['perimeter_1'].describe(include
='all').loc['75%']+self.goodCells['perimeter_1'].describe(include='all').loc['std']*4
        self.threshold_homogenity_rad = self.goodCells['radius_1'].describe(include='al
```

```
l').loc['75%']+self.goodCells['radius_1'].describe(include='all').loc['std']*4



    def predict(self, X_test):
        res = [] * 0
        for index in range(len(X_test)):
            if (X_test.iloc[index]['area_0']>self.threshold_area_0):
                res.append(1)
            elif (X_test.iloc[index]['area_1']>self.threshold_area_1):
                res.append(1)
            elif (X_test.iloc[index]['area_2']>self.threshold_area_2):
                res.append(1)

            elif((X_test.iloc[index]['perimeter_0']/X_test.iloc[index]['area_0'])<self.
threshold_shape_0):
                res.append(1)
            #elif((X_test.iloc[index]['perimeter_1']/X_test.iloc[index]['area_1'])<sel
f.threshold_shape_1):
              #  res.append(1)
            elif((X_test.iloc[index]['perimeter_2']/X_test.iloc[index]['area_2'])<self.
threshold_shape_2):
                res.append(1)

            elif (X_test.iloc[index]['texture_0']>self.threshold_texture_0):
                res.append(1)
           # elif (X_test.iloc[index]['texture_1']>self.threshold_texture_1):
             #   res.append(1)
            elif (X_test.iloc[index]['texture_1']>self.threshold_texture_2):
                res.append(1)

            elif (X_test.iloc[index]['perimeter_1']>self.threshold_homogenity_peri):
                res.append(1)
            elif (X_test.iloc[index]['radius_1']>self.threshold_homogenity_rad):
                res.append(1)



            elif (X_test.iloc[index]['concave points_0']>self.threshold_homogenity_0):
                res.append(1)
                print('1')
            elif (X_test.iloc[index]['concave points_1']>self.threshold_homogenity_1):
                res.append(1)
                print('2')
            elif (X_test.iloc[index]['concave points_2']>self.threshold_homogenity_2):
                res.append(1)
                print('3')

            else:
                res.append(0)

        return res


    def score(self,predict,y_test):
        sum = 0
        for i in range(len(predict)):
            if (predict[i] == y_test.iloc[i]):
                sum = sum + 1

        acc = sum / len(predict)
```

```
        return acc
```

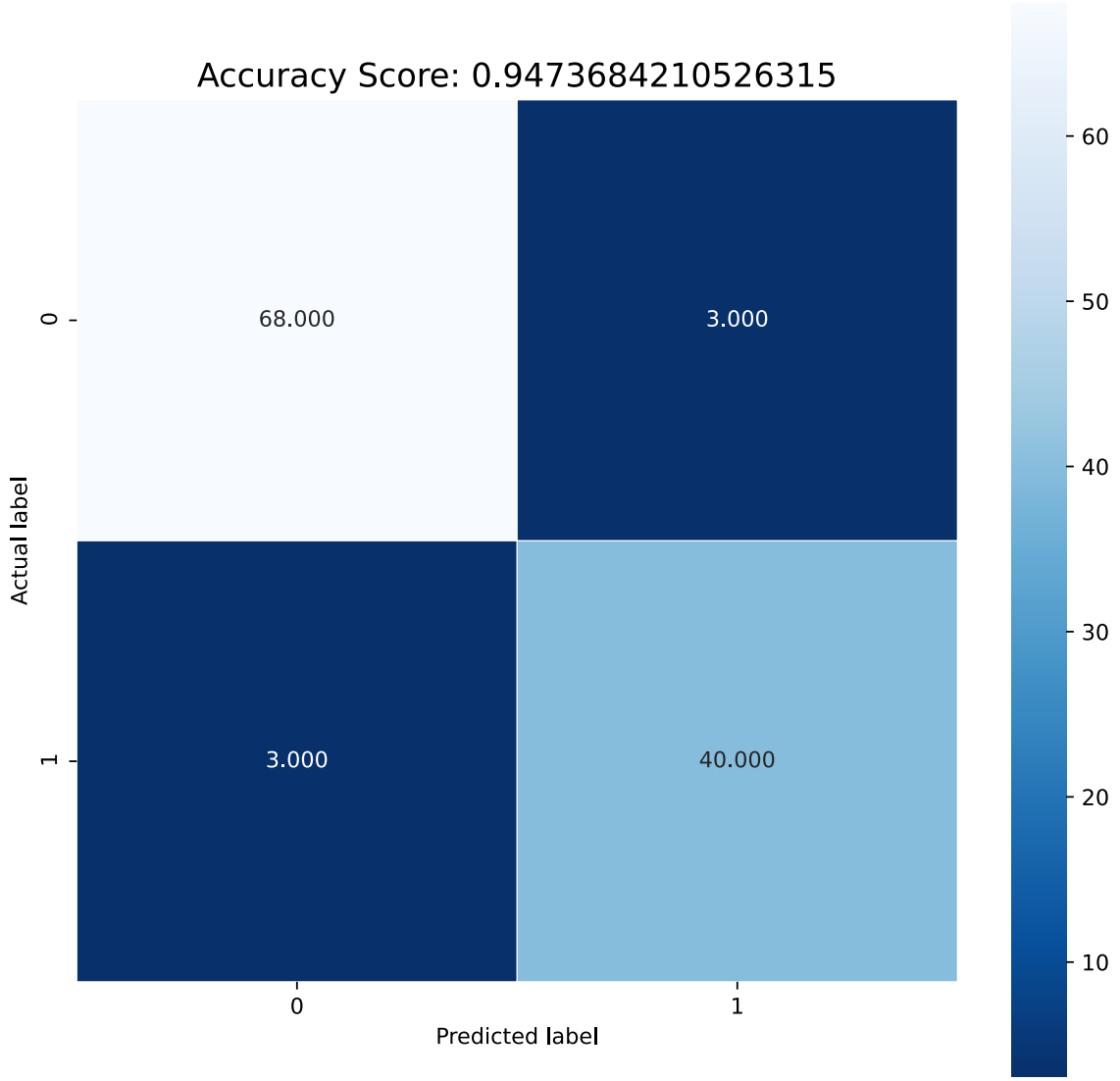
In [135]:

```
clf = rbClassifier()
clf.fit(X_train, Y_train)
predict_train = clf.predict(X_train)
predict_test = clf.predict(X_test)
print(clf.score(predict_train, Y_train))
accuracy = clf.score(predict_test, Y_test)
print(accuracy)
```

```
2
1
1
1
1
1
1
1
1
1
1
3
2
1
1
1
3
1
3
1
0.9010989010989011
0.9473684210526315
```

In [136]:

```
import seaborn as sns
from sklearn import metrics

cm = metrics.confusion_matrix(Y_test, predict_test)

plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(accuracy)
plt.title(all_sample_title, size = 15);
plt.savefig('confusion_ruleBased.png')
#plt.show();
```



Using a random forest classifier

In [32]:

```
var = target.sum()
percentage = var/(len(target))
print(percentage)
```

```
0.37258347978910367
```

In [ ]:

```
data_selected_1 = data[['area_0', 'concavity_0', 'concave points_0', 'radius_2', 'perim
eter_2','area_2', 'concavity_2', 'concave points_2']] #Only keeping columns selected by
SelectFromModel
data_selected_2 = data.drop(columns=['fractal dimension_0','smoothness_0','smoothness_
1','texture_1','compactness_0']) #Dropping columns with lowest feature importance
```

In [33]:

```
X_train, X_test, Y_train, Y_test = train_test_split(data,target, test_size = .2, random
_state = 42)
```

In [34]:

```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
rfClf=RandomForestClassifier(n_estimators=25,max_depth=15,bootstrap=False)

#Train the model using the training sets y_pred=clf.predict(X_test)
rfClf.fit(X_train,Y_train)

y_pred=rfClf.predict(X_test)
```

In [35]:

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
```

In [ ]:

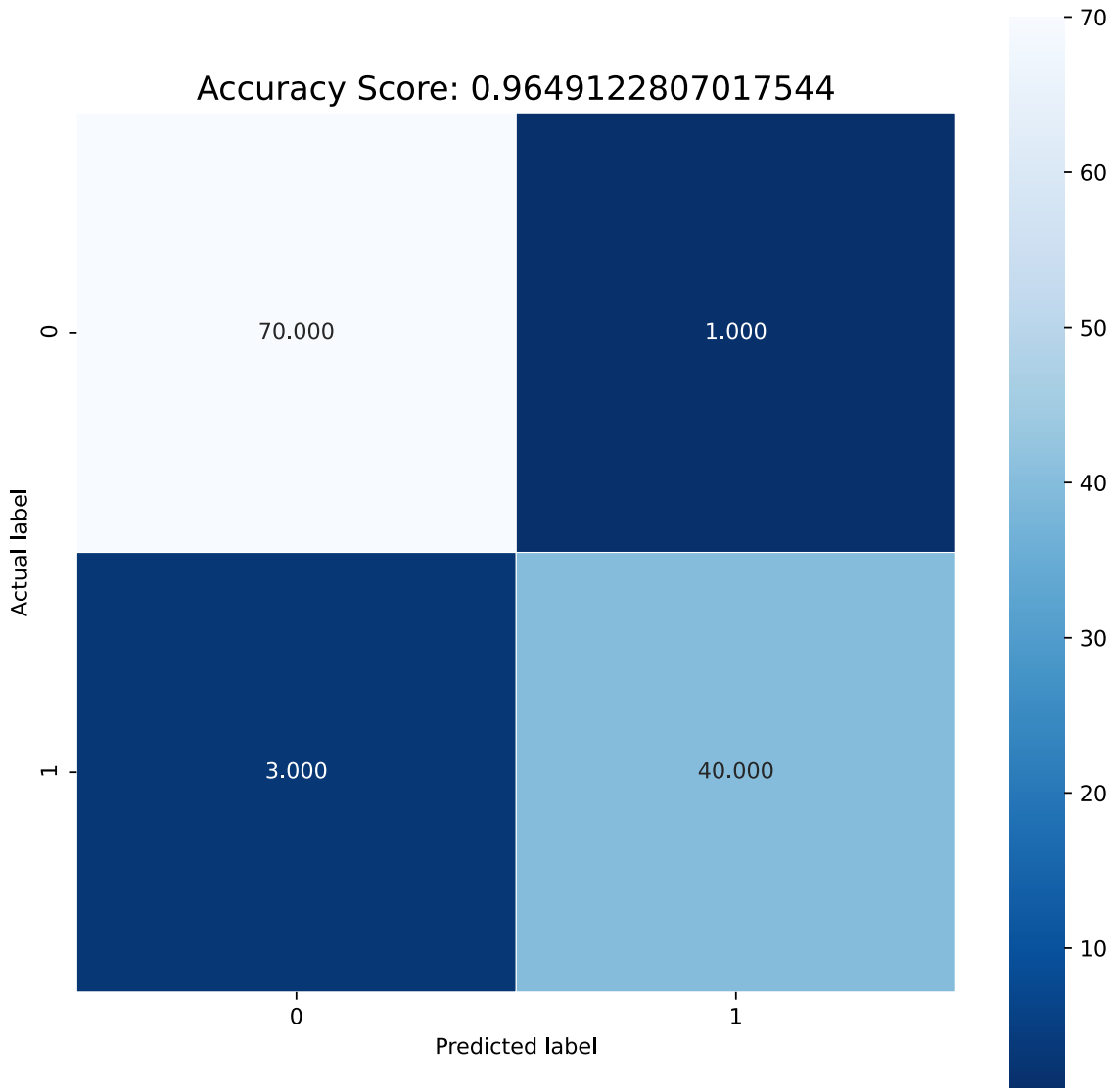
```
from sklearn.model_selection import cross_val_score

#5 fold cross-validation score
RF_cross_val = cross_val_score(rfClf, X_train, Y_train, cv=5)
print(RF_cross_val.mean())
```

In [36]:

```
from sklearn.model_selection import GridSearchCV
gs_clf_rf = GridSearchCV(estimator=RandomForestClassifier(), param_grid={'n_estimators'
: (10, 25, 50, 75, 100), 'max_depth': (3,5,7,11,15), 'bootstrap': (False, True), 'min_i
mpurity_decrease': (0.0, 0.1, 0.5, 1.0), 'min_weight_fraction_leaf': (0.0, 0.1, 0.5, 1.
0)})
#gs_clf_rf = gs_clf_rf.fit(X_train, Y_train)
#print(gs_clf_rf.best_score_)
#print(gs_clf_rf.best_params_)
```

In [37]:

```
import seaborn as sns
from sklearn import metrics

cm = metrics.confusion_matrix(Y_test, y_pred)

plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score(Y_test, y_pred))
plt.title(all_sample_title, size = 15);
#plt.show();
```



In [38]:

```
import pandas as pd
from sklearn.feature_selection import SelectFromModel

sel = SelectFromModel(RandomForestClassifier(n_estimators=25,max_depth=15,bootstrap=Fal
se))
sel.fit(X_train, Y_train)
```

Out[38]:

```
SelectFromModel(estimator=RandomForestClassifier(bootstrap=False, max_dept
h=15,
                                                 n_estimators=25))
```

In [39]:

```
selected_feat= X_train.columns[(sel.get_support())]
len(selected_feat)
```

Out[39]:

```
9
```

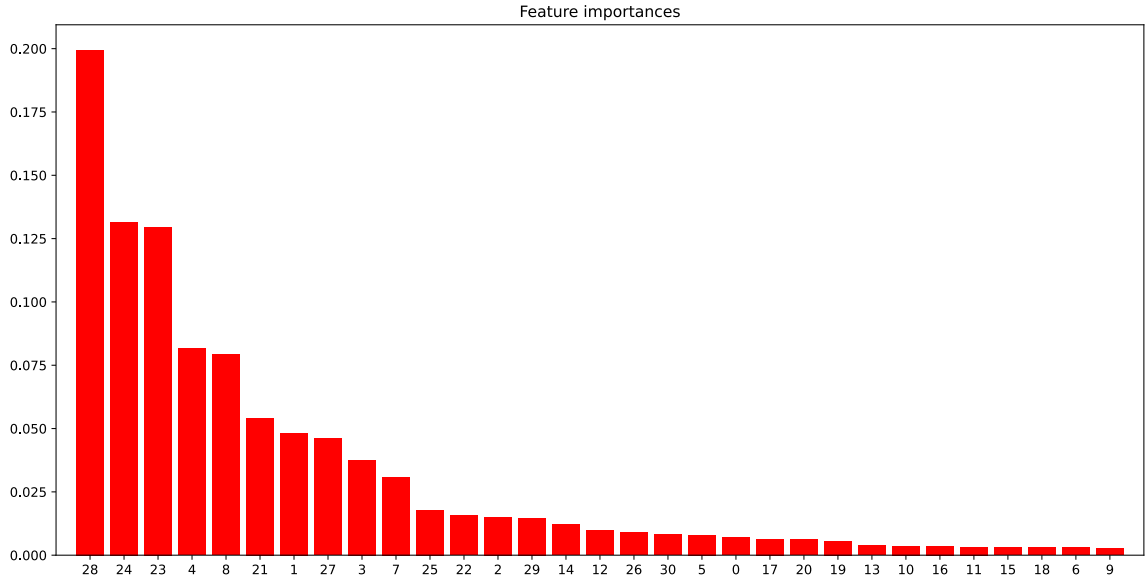
In [40]:

```
print(selected_feat)
```

```
Index(['radius_0', 'perimeter_0', 'area_0', 'concave points_0', 'radius_
2',
       'perimeter_2', 'area_2', 'concavity_2', 'concave points_2'],
      dtype='object')
```

In [41]:

```
importances = sel.estimator_.feature_importances_
indices = np.argsort(importances)[::-1]
# X is the train data used to fit the model
plt.figure(figsize=(15,7.5))
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", align="center")
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```



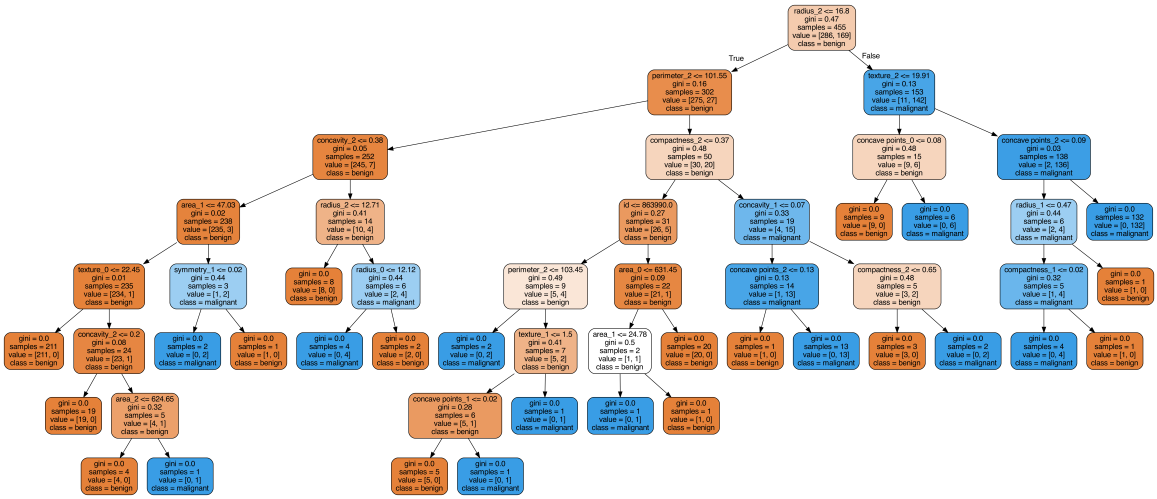
In [42]:

```
feature_names = list(data.columns.values)
estimator = rfClf.estimators_[5]
```

In [43]:

```
from sklearn.tree import export_graphviz
# Export as dot file
export_graphviz(estimator, out_file='tree.dot',
                feature_names = feature_names,
                class_names = ['benign','malignant'],
                rounded = True, proportion = False,
                precision = 2, filled = True)

# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'tree.png')
```

Out[43]:



Creating our own classifier

In [44]:

```
# pip install git+git://github.com/christophM/rulefit.git
```

Testing rule fit

In [45]:

```
X_train, X_test, Y_train, Y_test = train_test_split(data,target, test_size = .2, random
_state = 42)
```

In [46]:

```
X = X_train.to_numpy()
x2 = X_test.to_numpy()
```

In [47]:

```
from rulefit import RuleFit

rf = RuleFit(tree_size=4, sample_fract='default', max_rules=30,
             memory_par=0.01,
             tree_generator=None,
             rfmode='classify', lin_trim_quantile=0.025,
             lin_standardise=True, exp_rand_tree_size=True, random_state=1)
rf.fit(X, Y_train, feature_names=feature_names)

#predict
y_pred = rf.predict(x2)
y_proba = rf.predict_proba(X)
```

In [48]:

```
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Accuracy: 0.9736842105263158
```

In [49]:

```
rules = rf.get_rules()
rules = rules[rules.coef != 0].sort_values("support", ascending=False)
num_rules_rule=len(rules[rules.type=='rule'])
num_rules_linear=len(rules[rules.type=='linear'])

print(num_rules_rule)
print(num_rules_linear)

rules_importance = rules.sort_values(by='importance', ascending=False)
rules_importance
```

```
9
14
```

Out[49]:

	rule	type	coef	support	importance
11	radius_1	linear	1.088780e+01	1.000000	2.499442
22	texture_2	linear	2.884364e-01	1.000000	1.659031
27	concavity_2	linear	6.584365e+00	1.000000	1.289213
16	compactness_1	linear	-5.960557e+01	1.000000	1.015738
10	fractal dimension_0	linear	-1.496501e+02	1.000000	0.965949
29	symmetry_2	linear	1.528244e+01	1.000000	0.883777
57	concave points_2 <= 0.1465499997138977	rule	-1.575166e+00	0.660793	0.745747
9	symmetry_0	linear	-2.840331e+01	1.000000	0.737803
12	texture_1	linear	-1.410400e+00	1.000000	0.679410
40	area_2 > 718.6499938964844 & concave points_0 ...	rule	1.350153e+00	0.356828	0.646809
28	concave points_2	linear	9.323104e+00	1.000000	0.596450
37	concave points_0 <= 0.05127999931573868	rule	-1.184242e+00	0.603524	0.579290
31	area_2 <= 957.4500122070312 & concave points_0...	rule	-1.158280e+00	0.603524	0.566590
34	area_2 <= 927.1000061035156 & concave points_2...	rule	-1.064423e+00	0.612335	0.518605
44	area_2 <= 884.5499877929688	rule	-8.853422e-01	0.674009	0.414999
38	concave points_0 > 0.05127999931573868 & conca...	rule	8.557071e-01	0.334802	0.403826
2	texture_0	linear	5.303152e-02	1.000000	0.211403
25	smoothness_2	linear	6.395804e+00	1.000000	0.139716
1	radius_0	linear	-3.035018e-02	1.000000	0.100156
32	perimeter_2 <= 114.45000076293945 & concave po...	rule	-7.116854e-01	0.017621	0.093636
19	symmetry_1	linear	-1.208938e+01	1.000000	0.088768
0	id	linear	-2.561115e-09	1.000000	0.077641
42	concave points_0 > 0.05127999931573868 & fract...	rule	-9.031472e-02	0.039648	0.017623

In [50]:

```
rules_importance.iloc[2]['rule']
```

Out[50]:

```
'concavity_2'
```

In [51]:

```
rules_rule_importance = rules_importance[rules_importance['type']=='rule']
rules_linear_importance = rules_importance[rules_importance['type']=='linear']
```

In [52]:

```
rules_rule_importance
```

Out[52]:

	rule	type	coef	support	importance
57	concave points_2 <= 0.1465499997138977	rule	-1.575166	0.660793	0.745747
40	area_2 > 718.6499938964844 & concave points_0 ...	rule	1.350153	0.356828	0.646809
37	concave points_0 <= 0.05127999931573868	rule	-1.184242	0.603524	0.579290
31	area_2 <= 957.4500122070312 & concave points_0...	rule	-1.158280	0.603524	0.566590
34	area_2 <= 927.1000061035156 & concave points_2...	rule	-1.064423	0.612335	0.518605
44	area_2 <= 884.5499877929688	rule	-0.885342	0.674009	0.414999
38	concave points_0 > 0.05127999931573868 & conca...	rule	0.855707	0.334802	0.403826
32	perimeter_2 <= 114.45000076293945 & concave po...	rule	-0.711685	0.017621	0.093636
42	concave points_0 > 0.05127999931573868 & fract...	rule	-0.090315	0.039648	0.017623

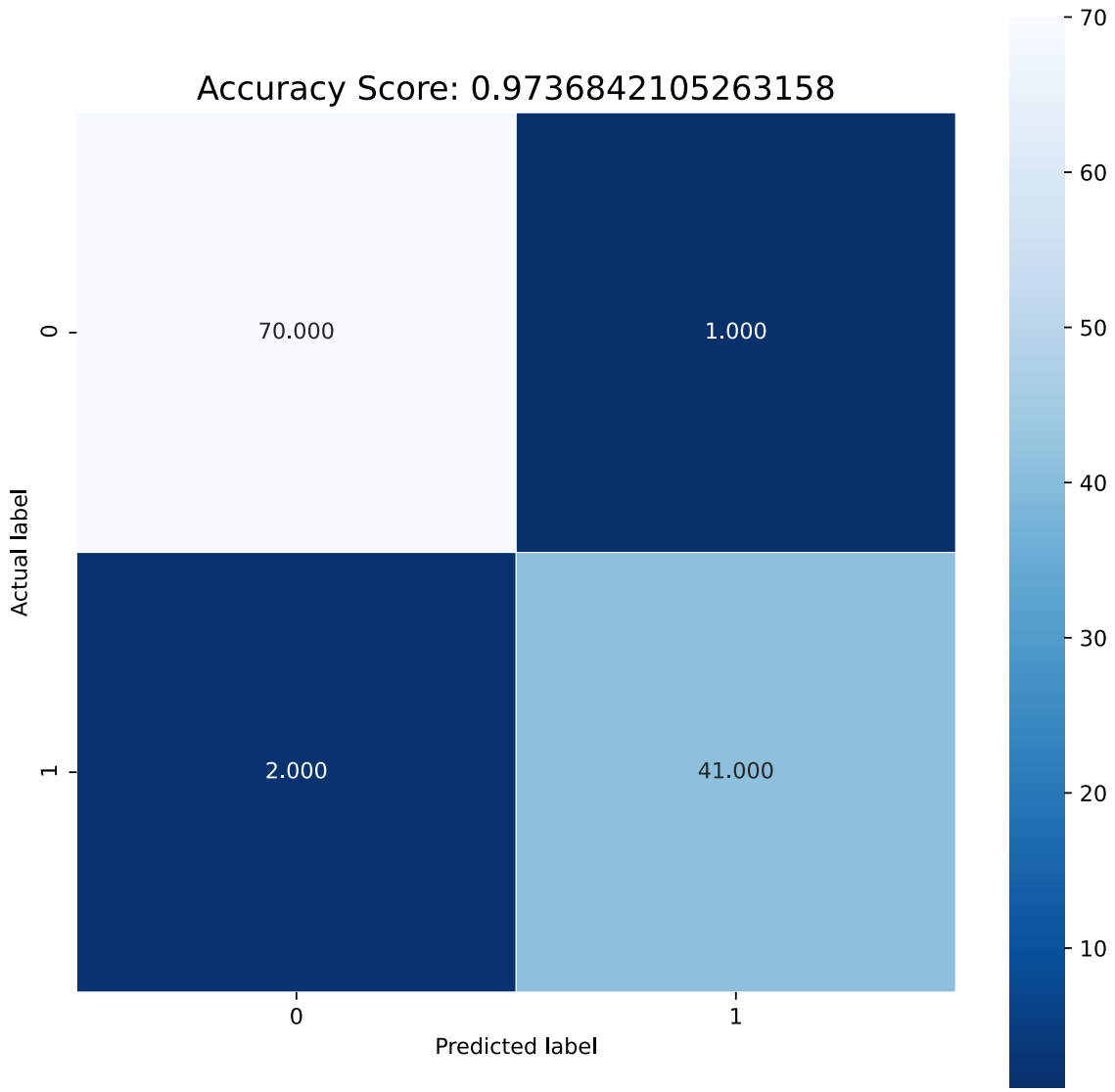
In [53]:

```
#rules_linear_importance.head()
print('Most important rules: \n')
for i in range(9):
    print(rules_rule_importance.iloc[i]['rule'])
```

```
Most important rules:

concave points_2 <= 0.1465499997138977
area_2 > 718.6499938964844 & concave points_0 > 0.05127999931573868
concave points_0 <= 0.05127999931573868
area_2 <= 957.4500122070312 & concave points_0 <= 0.055800000205636024
area_2 <= 927.1000061035156 & concave points_2 <= 0.14124999940395355
area_2 <= 884.5499877929688
concave points_0 > 0.05127999931573868 & concavity_2 > 0.26135000586509705
perimeter_2 <= 114.45000076293945 & concave points_2 > 0.14705000072717667
& texture_2 <= 26.28499984741211
concave points_0 > 0.05127999931573868 & fractal dimension_0 > 0.060990000
143647194 & concavity_2 <= 0.26135000586509705
```

In [54]:

```
import seaborn as sns
from sklearn import metrics

cm = metrics.confusion_matrix(Y_test, y_pred)

plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(metrics.accuracy_score(Y_test, y_pred))
plt.title(all_sample_title, size = 15);
#plt.show();
```

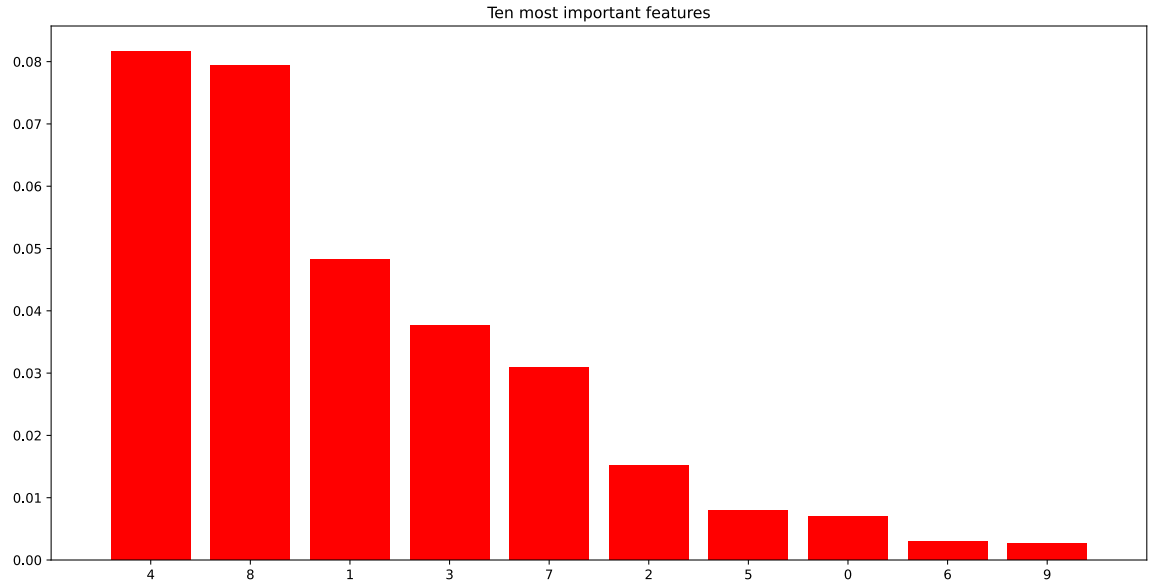


Looking at the ten most important features

Random forest

In [55]:

```
importances = sel.estimator_.feature_importances_[0:10]
indices = np.argsort(importances)[::-1][0:10]
# X is the train data used to fit the model
plt.figure(figsize=(15,7.5))
plt.title("Ten most important features")
plt.bar(range(len(indices)), importances[indices],
       color="r", align="center")
plt.xticks(range(len(indices)), indices)
plt.xlim([-1, len(indices)])
plt.show()
```



In [56]:

```
print('Most important features: \n ')
best_features_rf = []

for i in indices:
    best_features_rf.append(feature_names[i])
    print(feature_names[i])
```

```
Most important features:

area_0
concave points_0
radius_0
perimeter_0
concavity_0
texture_0
smoothness_0
id
compactness_0
symmetry_0
```

Rule Fit model

In [57]:

```
ruleFit_feature_importances_ = rules_linear_importance['importance'].to_numpy()
ruleFit_feature_names_ = rules_linear_importance['rule'].to_numpy()
```

In [209]:

```

```

Out[209]:

```
array(['radius_1', 'perimeter_1', 'perimeter_2', 'compactness_1', 'id',
       'concave points_0', 'concave points_2', 'texture_1',
       'smoothness_2', 'area_2', 'concavity_1', 'concavity_2',
       'compactness_2', 'area_0', 'symmetry_1', 'area_1',
       'fractal dimension_1', 'fractal dimension_2', 'concave points_1',
       'smoothness_1', 'radius_2', 'compactness_0', 'concavity_0',
       'perimeter_0', 'smoothness_0', 'symmetry_0', 'texture_0',
       'texture_2', 'radius_0', 'symmetry_2', 'fractal dimension_0'],
      dtype=object)
```

In [58]:

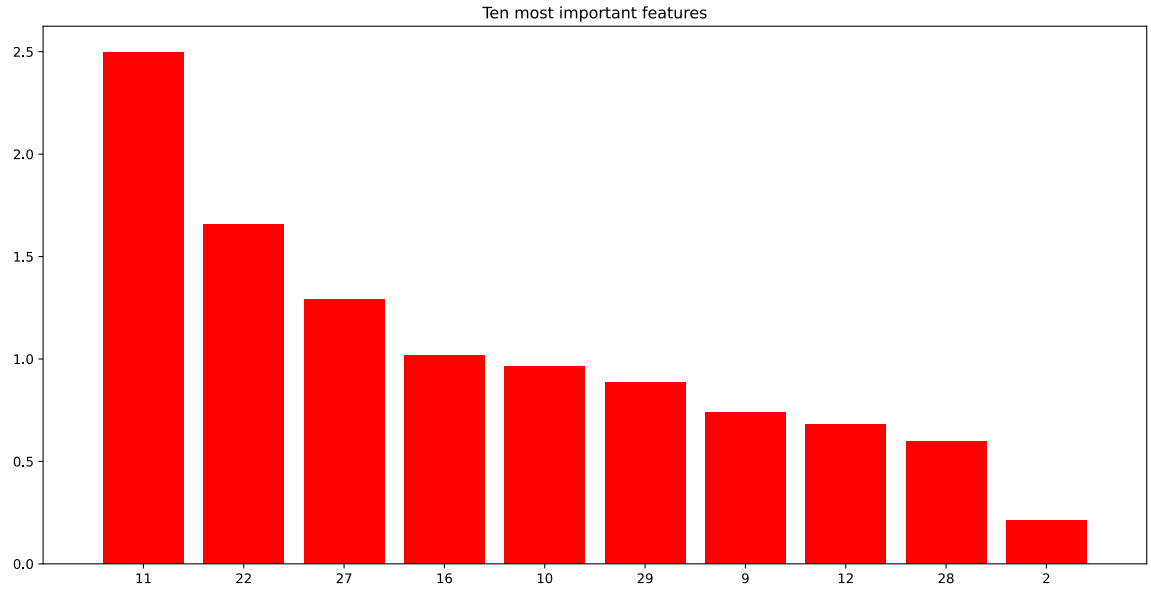
```
def retrieveIndex(name):
    for i in range(len(feature_names)):
        if feature_names[i]==name:
            return i
```

In [59]:

```
feature_indicies = []
for feature in ruleFit_feature_names_:
    index = retrieveIndex(feature)
    feature_indicies.append(retrieveIndex(feature))
```

In [60]:

```
importances = ruleFit_feature_importances_[0:10]
indices = feature_indicies[0:10]

# X is the train data used to fit the model
plt.figure(figsize=(15,7.5))
plt.title("Ten most important features")
plt.bar(range(len(indices)), importances,
       color="r", align="center")
plt.xticks(range(len(indices)), indices)
plt.xlim([-1, len(indices)])
plt.savefig('importance_ruleFit.png')
plt.show()
```



In [62]:

```
print('Most important features: \n ')
best_features_ruleFit = []
p = 0
for i in indices:
    p += 1
    best_features_ruleFit.append(feature_names[i])
    print(p, ': ', feature_names[i])
```

```
Most important features:

1 :  radius_1
2 :  texture_2
3 :  concavity_2
4 :  compactness_1
5 :  fractal dimension_0
6 :  symmetry_2
7 :  symmetry_0
8 :  texture_1
9 :  concave points_2
10 :  texture_0
```

In [63]:

```
df = pd.DataFrame(best_features_ruleFit)
df['Importance'] = importances
df
```

Out[63]:

	0	Importance
0	radius_1	2.499442
1	texture_2	1.659031
2	concavity_2	1.289213
3	compactness_1	1.015738
4	fractal dimension_0	0.965949
5	symmetry_2	0.883777
6	symmetry_0	0.737803
7	texture_1	0.679410
8	concave points_2	0.596450
9	texture_0	0.211403

Now, for the two latter classifiers, lets try to only use the 10 most important features and see what happens with the accuracy

In [64]:

```
X_train, X_test, Y_train, Y_test = train_test_split(data,target, test_size = .2, random
_state = 42)
```

In [65]:

```
X_train_rf = X_train[best_features_rf]
X_test_rf = X_test[best_features_rf]
X_train_ruleFit = X_train[best_features_ruleFit]
X_test_ruleFit = X_test[best_features_ruleFit]
```

Random forest

In [66]:

```
rfClf2=RandomForestClassifier(n_estimators=25,max_depth=15,bootstrap=False)

#Train the model using the training sets y_pred=clf.predict(X_test)
rfClf2.fit(X_train_rf,Y_train)

y_pred=rfClf2.predict(X_test_rf)
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Accuracy: 0.9385964912280702
```

Rule fit

In [67]:

```
X = X_train_ruleFit.to_numpy()
x2 = X_test_ruleFit.to_numpy()

rf2 = RuleFit(tree_size=4, sample_fract='default', max_rules=30,
              memory_par=0.01,
              tree_generator=None,
              rfmode='classify', lin_trim_quantile=0.025,
              lin_standardise=True, exp_rand_tree_size=True, random_state=1)
rf2.fit(X, Y_train, feature_names=best_features_ruleFit)

#predict
y_pred = rf2.predict(x2)
y_proba = rf2.predict_proba(X)

print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Accuracy: 0.9298245614035088
```

In [ ]:

```

```