Application of Machine Learning Techniques to Classify Web Services

Medha Chippa
Computer Science & Engineering, Keshav Memorial Institute of Technolgy Hyderabad, India.
medha6271@gmail.com

Aishwarya Priyadarshini
Computer Science and Engineering IIIT, Bhubaneswar Bhubaneswar, India.
aishpriya8s@gmail.com

Ramakanta Mohanty
Computer Science & Engineering, Keshav Memorial Institute of Technology Hyderabad, India.
ramakanta@kmit.in

***Abstract*— The architects and designers of some of the most recent web applications are resorting to the use of a variety of different languages and platforms. Each of these applications are required to communicate among one another to make a common unifying service. Communication may involve the sending and receiving information or the transformation and simplification of it. Each of these heterogeneous applications serves a different intent and are therefore taken to share a different architecture. Different languages and descriptions are so used to leverage the same. It therefore becomes difficult for these applications to communicate with and exchange information amongst one another. Web service technologies are independent of the design and architecture of the underlying applications. Thus, these technologies can tackle this issue by standardizing the way these applications communicate with one another. This paper proposes machine learning models that could be used to classify these web services viz., K-Nearest Neighbors (KNN), Naïve Bayes (Gaussian Naïve Bayes Classifier), Kernel Support Vector Machine Classifier (Kernel SVM), linear SVM, Decision Trees and Random Forests. The QWS (Quality Web Services) dataset has been used for classification and analysis**

Keywords — Web Services, K-Nearest Neighbors (KNN), Naïve Bayes (Gaussian Naïve Bayes Classifier), Kernel Support Vector Machine Classifier (Kernel SVM), Linear SVM, Decision Trees, Random Forests

I. Introduction

Web services are blocks of managed code that serve as the basis for communication between heterogeneous applications that are designed using different languages and platforms [1]. This relieves the application designer of the overhead that comes with writing application specific code. Thus web services pave way for efficient communication to be carried out between heterogeneous applications. Web services are ideally employed in a client server environment. The client and the web service are characterized by a loosely coupled architecture, making heterogeneous system integration simple and manageable. Web services help carry out communication between application using standards like HTML, XML, WSDL, SOAP. Web services extensively make use XML. WSDL, SOAP and UDDI make use of XML-based messaging. WSDL is driven by XML and is used for describing web services. WSDL helps clients access the location, learn about the operations the service is capable of performing and an acceptable format for sending messages to the service. The WSDL documents function as a contract between the web service client and the server. By abiding by this contract, the service provider and the consumer agree to communicate in a standardized way, irrespective of the architectures and platforms that they are using.

SOAP offers a standardized method for exchanging XML messages between the node and host applications. The SOAP messages constitute either requests or responses. The format of these messages is specified in the WSDL definition. UDDI is a standard that is used for the creation of an XML based registry. The registry contains information about businesses and the corresponding web services that they offer. In this way, the UDDI provides for a standard and uniform way of listing the web services offered by businesses, thereby making it easy for the clients to discover the same [2]. UDDI uses the WSDL documentation to identify the services and SOAP messaging to ease communication. To discover a web service, the UDDI registry is queried and the WSDL for the service requested by the client can be obtained. Thus web services' platform independence, robustness, cost effectiveness, etc. makes them an ideal choice for communication in a heterogeneous environment.

There are even a few inherent challenges that Web services arrive with. Sometimes, their growth and implementation could become complex, particularly when multiple operations are required. Thereby causing the generation of WSDL code relatively complex. The primary step of using web services involves designing the WSDL document followed by the construction of the web service based on this definition. This would, however, prevent designers from encompassing language-specific constructs and types in their definition [3]. Likewise, the evolution of more or less of the most simplest web services would require its designers to write great and extended code that would frequently turn out to be arduous and time consuming making it error prone. For the purpose of overcoming these barriers, designers will have to come up with new tools and methodologies that would make the process of designing web services less tedious.

The rest of the paper is organized in the following manner. A brief discussion about Literature Survey is presented in section II. Section III presents an overview of machine learning techniques, Section IV presents a detailed discussion of the results and discussions. Finally, in sections VI conclusion of the paper.

II. Literature Servey

Many machine learning techniques has been employed from time to time to analyze and classify web services. Fokaefs et al. [4] employed the VTracker tool to represent a WSDL file by calculating the minimum distance between two trees as a WSDL component plays a very important role to classify the web services and from that they found that if they excluded the WSDL component from the data, the simulated results accuracies varies significantly. The outcome of the tool is the percentage of interface changes such as added, changed, and removed elements among the XML models of two WSDL interfaces. Romano et al. [5] chose a similar techniques called WSDLDiff that can analyze the evolution of a WSDL interface without manually inspecting the XML changes. Aversano et al. [6] compare the relationships between the different attributes and also extract the relationship among attribute by use of machine learning to extract highly dominant attributes by using feature selection. Xing et al. [7] selected UMLDiff to represent different UML diagrams and they found out that the role of each attribute for classification of web services. Zarras et al. [8] focused on Amazon Web services about the evolution of web services in the real world for classification and prediction of suitability of web services for a particular purpose. Kral et al. [9] found several SOA anti patterns that excludes the principle of SOA and also addressed many anti pattern in SOA architecture.. Recently, Moha et al. [10] have employed a rule-based approach viz. SODA for SCA systems (Service Component Architecture). Later, Palma et al. [11] extended this work for Web service antipatterns in SODA-W is used declarative rule specification based a domain-specific language (DSL) to specify/identify the key symptoms that characterize an antipattern using a set of WSDL metrics. Rodriguez et al. [12] and Mateos et al. [13] provided a set of guidelines for service providers to avoid bad practices while writing WSDLs based on eight bad practices in the writing of WSDL for Web services. Recently, Ouni et al. [14] proposed a search-based approach based on standard GP to find regularities, from examples of Web service antipatterns, to be translated into detection rules. Most recently, Mohanty et al.[2, 15,16] employed neural networks, decision Tree and support Vector Machine to classify the quality of web services and also proposed data envelopment analysis and FMADMA to classify different web services based on the quality of web services.

III. Overview Of Techniques Employed

Supervised machine learning algorithms have been used in this report for the purpose of classification and analysis of web services. The following classification models have been used viz. K-Nearest Neighbors (KNN), Naïve Bayes (Gaussian Naïve Bayes Classifier), Support Vector Machine Classifier (Kernel SVM), Linear SVM, Decision Trees and Random Forests. The quality of web service (QWS) dataset has been used AL-Masari et al. [1]. Al-Masri and Mahmood had created the dataset by observing real web service implementations. This dataset was collected from Web Service Crawler Engine (WSCE). AL-Masari et al.[3]. A majority of the web services were obtained from public sources on the web, including UDDI registries, search engines and service portals. The dataset consists of 364 web services (samples) each of which are characterized by 10 QWS attributes**.** The different attributes in QWS datasets are Response Time, Availability, Throughput, Successability, Reliability, Compliances, Best Practices, Latency, Documentation and WSRF.

A. K-Nearest Neighbor

The K-Nearest Neighbor method is one of the earliest methods to have been applied for the purpose of sorting. The classifier proceeds by treating every sample is the dataset as a point in n-dimensional distance. Every sample (point) can be symbolized as a n-dimensional feature vector. Every sample (point) in the training set is plotted in the n - dimensional space. When a sample (point) in the test set is to be categorized into one of the classes it is initially piloted in n-dimensional space and then categorized into one of classes by examining the n-dimensional space for the k samples that are situated close to the test sample under study Callahan et al. [17]. "Nearest" is set based on a distance metric like the Euclidean space. The duration between the incoming test sample and the training samples plotted in the n-dimensional space is measured and the test sample is concluded to belong to a class that the k-nearest training samples [18]. The determination of a suitable K value constitutes a really significant step in the algorithm propagation. Various methods like the elbow method, hyper parameter tuning, etc. can be used to find the best value for K [19]. The performance of the model largely depends on the value of K. Therefore an appropriate value for K would yield the best consequences and an inappropriate value for K on the other hired man would give bad results Cost et al., [20].

KNN:

Algorithm:

- *T - Training Data($\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$), P – Feature Vectors/Predictors in the testing set/samples($\boldsymbol{t_1}, \ldots, \boldsymbol{t_n}$), D- Distance metric*
- *Load the Training data T and the testing data P*
- *Associate every sample in the training set T with its corresponding class label*

Compute the distance (distance metric) D between the training data point and the testing data point

- *Arrange the distances of the first K data points and their corresponding classes in a specific order*
- *FOR each datapoint $\boldsymbol{x_i}$, such that i<=K*
- *IF D($\boldsymbol{x_i}$,$\boldsymbol{t_i}$)<D($\boldsymbol{x_j}$,$\boldsymbol{t_i}$)*
- *THEN*
- *Class Label of $\boldsymbol{t_i}$ = Class Label of $\boldsymbol{x_i}$*
- *END IF*
- *END FOR*

B. Naïve Bayes (Gaussian)

The Naïve Bayes Classifier is a model of Bayesian Classifiers. This model is centered on posterior probability estimation. Every sample in the dataset is characterized by a n-dimensional attribute vector and a grade label. The model is primarily trained using some training data. The performance of the model is later measured using the test data set. Post training, the model proceeds by calculating the posterior probability of every sample in the test data set. The sample is supposed to belong to a particular class if the class probability of that sample belonging to that class passed a lot of attribute conditions is more prominent than the class probability of it belonging to another class[21-25]. In other words a sample with the greatest class probability is grouped under that class. It is assumed that each of the predictors in the dataset is independent of one another [26-28].

The Gaussian Naïve Bayes Classifier is a special category of Naïve Bayes Classifiers. An extension of the Naïve Bayes Classifier applied to real-valued attributes are termed as the Gaussian Naïve Bayes Classifier. The mean and standard deviation for each of the samples is calculated to summarize the distribution. The Gaussian Probability density function (PDF) is used to calculate the new probabilities. The samples the classified based on these new probability values. The likelihood of features is assumed to be Gaussian:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma^2 y}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma^2 y}\right) \quad (1)$$

Gaussian Naïve Bayes:
Algorithm:

- *T - Training Data P – Feature Vectors/Predictors in the testing set(samples)*
- *Load the training data T*
- *Compute the mean and standard deviation of each sample in the training set*
- *REPEAT*
- *Compute the probability of every predictor variable in the testing set using the gaussian density equation in each target class*
- *UNTIL*
- *The probability of all the features has been computed*
- *Compute the likelihood for each target class*
- *Label the sample with the target class that has the greatest likelihood*

C. SVM and Linear SVC

Support Vector Machine Classifiers are an example of supervised machine learning models consisting of related learning algorithms that analyze data and distinguish patterns that constitute the base for classification and regression analysis. This model proceeds by taking labeled data as input. The SVM model works by classifying each of the samples into one or more given class labels. Every sample in the dataset is characterized by a n-dimensional attribute vector plotted in n-dimensional space such the categories of data points are separated by a clear gap that is as wide as possible. When a new incoming datapoint is to be classified under one of these categories, a hyper plane is constructed and the data point is plotted. Established on the comparative location of the data point with respect to the hyper plane, it is classified appropriately Chang and Lin [29].

It is nevertheless not always really easy to construct hyperplanes. This is particularly genuine when the data are randomly dispersed. The Kernel method of SVM is applied in such scenarios. When the Kernel method of SVM is used as part of the SVM it is called the SVM. The Kernel function could be any of the following viz. linear, polynomial, rbf and sigmoid. The type of kernel being used largely affects the performance of the model and is thus of crucial importance. It is a multiclass model that uses the one-vs-rest strategy.

Kernel Functions can mathematically be described as follo ws Vapnik and Corinna [30] and Pedregosa *et al[31].*

$$Linear : (x, x')$$

$$Polynomial: (\gamma(x, x') + r)^d \ where\ d = degree, r = coef$$

Polynomial: $(\boldsymbol{\gamma}(\boldsymbol{x}, \boldsymbol{x}') + \boldsymbol{r})^{\boldsymbol{d}}$ *where d= degree, r = coef*

$$rbf: \exp\left(-\gamma - \gamma' \left|\left|x - x'\right|\right|^2\right) where\ \gamma > 0 \quad (2)$$

$$Sigmoid: (\tanh(\gamma(x, x') + r))\ where\ r = coef$$

These models can be described as follows

SVC: ½||w||^2+C SUM $\boldsymbol{x_{i\_i}}$

$$VC: \frac{1}{2}\ ||w||^2 + C\ SUM\ x_{i\_i} \quad (3)$$

General Algorithm for SVM:

- *T - Training Data P – Feature Vectors/Predictors in the testing set (samples) K- Kernel function*
- *Load the training data T*
- *Perform Feature Scaling for the entire dataset*
- *Select the appropriate kernel K*
- *Adjust the Classification hyperplane according to the nature of the data*
- *Predict the class label for the testing data P*
- *END*

D DECISION TREE:

Decision Trees are used for the purpose of classification. They belong to the category of three methods. A decision tree is a tree like data structure that is composed of nodes and arcs. Every node in the tree represents a decision and every arc in the tree represents the consequence of that decision Colin [32]. A decision tree is primarily constructed using a top down approach. Attribute Selection measures are used as the basis for selecting the best possible split. The attribute values of every sample in the test dataset is tested against the decision tree. The leaf node that is arrived upon post testing is anticipated to be the class label of that sample..

E RANDOM FOREST

Random Forests are an example of Ensemble methods that are used for the purpose of classification. To ameliorate the performance of a good example, if several decision trees are employed with a random sample of characteristics, it makes a random forest. A new random sample from among the features is chosen for every single tree at every single split [33].

For example, if there is a really strong feature in the dataset. When using bagged, trees, most of the trees would use that feature as the top split. This would result in a collection of highly correlated trees. Averaging such highly correlated quantities will not significantly reduce the discrepancy. By randomly leaving out features from each split, random forests attempt to decorate trees, thus proving to be advantageous [34].

Algorithm:

- *DecisionTree (T,P,y) T - Training Data, P-Feature Vectors/Predictors, y - Target Class*
- *Initialize a new tree DT with a single root node.*
- *IF one of the Stopping Criteria is met THEN label the root node in DT as a leaf with the most predominant value of y in DT.*
- *ELSE search for a distinct function f(P) of the input attributes values such that splitting DT according to f(P)'s outcomes (*$\boldsymbol{v_i}$*,...,vn) gains the best splitting metric with the greatest amount of accuracy.*
- *IF best splitting metric > threshold THEN*
- *Label* $\boldsymbol{t'}$ *with f(P)*

- *FOR each outcome $\boldsymbol{v_i}$ of f(P):*
- *Set $\boldsymbol{ST_I}$= DecisionTree (BS(f(P))=($\boldsymbol{y_i}$DT,P,y) where BS= Best Splitting Metric*
- *Connect the root node of $\boldsymbol{t'_{DT}}$ to Subtreei with an edge that is labelled as $\boldsymbol{v_i}$*
- *END FOR*
- *ELSE Mark the root node in T as a leaf with the most common value of y in S as a label.*
- *END IF*
- *END IF*
- *RETURN DT*
- *Random Forest:*
- *Algorithm:*
- Select a random subset of P' features from a set of P features, where $P' \in$ P
- From among the $\boldsymbol{P'}$ features, calculate the node n using the best splitting criteria.
- Split the node into child nodes using the best possible split.
- Replicate **steps 1 through 3** until the leaf node has been reached.
- Construct a forest by replication steps **1 through 4** for n' number times to create n'**number of trees.**

V. Results And Discussions

In this paper, we collected QWS dataset from literature [1]. The QWS dataset consists of different samples of web service implementations and their corresponding attributes/features. The attributes from x1 through x10 have been used as predictors and the attribute x11 has been used as a response variable. Attributes, x12 and x13 have been ignored because they do not contribute to the analysis**.** The web services (samples) in the dataset have been classified into four categories, namely (i) Platinum (high quality) (ii) gold (iii) silver and (iv) bronze (low quality).

The classification is measured on the basis of the overall quality rating provided by WSRF. Web services have been categorized into a particular class based on the classification. The functionality of web services could be used to differentiate between various services AL-Masri et al. [1,3].

Feature Scaling is a preprocessing technique that is used to standardize or normalize the features of a dataset. Data or useful information used for the purpose of data analysis is usually gathered up from a diversity of authors. The characteristics of the dataset would thus contain values that are open after a varying set of magnitudes, units or ranges. Certain machine learning models fail to acknowledge these changes, making it ultimately prone to errors or incompatibilities. Thus, Feature scaling or normalization is used to regularize or standardize the values of a dataset prior to using models that cannot handle values that exhibit high variance Bengio and LeCun [35]. Good examples that employ the calculation of a distance metric as part of their algorithm would require scaling. Such models include KNN, SVM, etc. Other models like Principal Component Analysis would also take the data to be surmounted. In certain scenarios, scaling can also be applied to improve the accuracy of the mannequin. Nevertheless, models like Naïve Bayes or tree based methods are known to be capable of handling features with varying ranges and therefore do not require feature scaling.

In this composition, feature scaling has been applied as a constituent of the KNN, SVM and Linear SVM models. The scikit learn library has been utilized for the purpose scaling. The Standard Scaler estimator object from the preprocessing family has been used to implement scaling.

The performance of a machine learning model largely depends on a set of parameters that effect the architecture of a model. Such parameters are called hyper parameters. The hyper parameters of a model are thus used to describe how the model is structured. If the hyper parameters of a model are incorrectly chosen, it would cause the accuracy of the exemplar to be lowered or the error rate rise. For instance k, the algorithm is some hyper parameters that would bear on the KNN model. The process of iterating through a set of all possible parameters effecting the model, with the motive of finding the best possible combination that would produce the greatest possible accuracy is called hyper parameter tuning James Bergstra and Bengio [36].

Quint of the models documented in this paper (K-Nearest Neighbors (KNN), Naïve Bayes (Gaussian Naïve Bayes Classifier), Kernel Support Vector Machine Classifier (Kernel SVM), Linear SVM (Lib SVM), Decision Trees) use hyper parameter tuning to hold the best classification accuracy. The scikit learn library has been utilized for the purpose scaling. The Grid Search CV estimator object from the model selection family has been used to implement hyper parameter tuning Pedregosa et al. [31].

We developed the python code for each machine learning techniques and simulated as shown in appendix 1. Therefore, prior to using the KNN, SVM and Linear SVC models, the data are standardized using the Standard Scaler estimator object from the preprocessing library. Hyper parameter tuning is then used to pick out the best combination of hyperparameters that would ensue in the greatest possible accuracy. 10-Fold Cross Validation is performed and the average accuracy of all the 10 folds is computed. The individual Fold accuracies obtained for each of the models are documented in Table 1.

The Random Forest Classifier has resulted in an average accuracy of 99.44%. The SVM model provides an average accuracy of 98.07%. Decision Trees resulted in an accuracy of 99.72%. The Linear SVC model has resulted in an accuracy of 85.43%. The Gaussian Naïve Bayes model had resulted in an accuracy of 82.70%. The KNN model resulted in an accuracy of 80.29%. The average accuracies of all the models have been presented in Table 1. Feature scaling was applied to normalize the feature values of the dataset prior to applying the KNN, SVM and Linear SVC models. Hyper parameter Tuning was used to come up with the best possible combination of hyper parameters. The stages of machine learning models inclusive of feature scaling have been illustrated in Fig 1. The hyper parameters have been shown in Table 2. The best accuracies had resulted from the utilization of the Decision Tree Classifier and the Random Forest Classifiers. They produced accuracies 99.72% and 99.44% respectively. The Support Vector machine Classifier

(SVM) had resulted in the second highest accuracy of 98.07.

Overall, we compare all four techniques, we found in fold1 that the SVM, Random Forest and Decision Tree resulted 100% accuracies respectively, followed by linear SVM and Naïve base. In fold2, we found that Decision tree and Random forest provided 100% and SVM resulted 97.29% accuracies followed by linear SVC, Naïve base and KNN respectively. However, we observe that Random forest and Decision Tree resulted 100% accuracies in most of the folds except fold4 which leads to average accuracy of 99.44% and 99.72% respectively. We also found that SVM resulted 100% accuracies in fold 1, fold 6, fold 7, fold 8 and average accuracy of 98.07%, which also the second best results in our simulation.

In our simulation, we tuned parameter for different machine learning techniques are presented in Table 3.We compare our results with the results provided by Mohanty et al. [2]. We found that our results compare to the techniques employed by them outperformed most of the techniques which is presented in Table 2.

V. Conclusion

With the ubiquitous use of Web Services, the need to record and analysis the Quality of Web Services has become imperative. Through the classification of these Web Services on the basis of quality, it becomes relatively easy to analyze their performance. Through this report, we have documented certain machine learning models (techniques) that could be utilized for the function of classifying these Web Services into four major divisions based on certain quality metrics. Analysis has been carried out on the QWS dataset through the use of several supervised Machine Learning Techniques namely: Nearest Neighbors (KNN), Naïve Bayes (Gaussian Naïve Bayes Classifier),Support Vector Machine Classifier (Kernel SVM), Linear SVM , Decision Trees and Random Forests. 10 Fold Cross Validation has been carried out using each of these techniques. The scikit learn library has been utilized for analysis. A few models have performed exceedingly well while a few others have done reasonably. Feature Scaling and Hyper parameter Tuning have been applied wherever necessary to improve the accuracy of the mannequin. The stages of machine Learning modelling from the splitting of data into the train and test sets (only used for the purpose of hyper parameter tuning) through the computation of average accuracies for all the 10 folds have been graphically represented. Lastly, we conclude that this study could be used to classify a new Web Service into one of the four major classes, given the set of quality attributes.

References

[1] AL-Masri, E., & Mahmood, Q." H.QoS based discovery and ranking of web services", In IEEE 16th international conference on computer communications and networks (ICCCN), pp. 529–534, 2007.

[2] **Ramakanta Mohanty**, V. Ravi and M. R.Patra, (2010) 'Web Services Classification using intelligent Techniques'. Elsevier, Expert Systems with Applications 37, PP. 5484-5490.

[3] AL- Masri, E., & Q. H. Mahmood, " Investigating web services on the World Wide Web", In 17th International conferences on World Wide Web, PP. pp. 795–804, 2008, Beijing.

[4] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia and A. Lau, "An Empirical Study on Web Service Evolution," in IEEE International Conference on Web Services, pp. 49-56,2011.

[5] D. Romano and M. Pinzger, "Analyzing the Evolution of Web Services Using Fine-Grained Changes," in IEEE 19th International Conference on Web Services (ICWS),pp. 392-399, Honolulu, 2012

[6] L. Aversano, M. Bruno, M. Di Penta, A. Falanga and R. Scognamiglio, "Visualizing the evolution of Web services using formal concept analysis," in IWPSE '05 Proceedings of the Eighth International Workshop on Principles of Software Evolution, pp. 57-60, Lisbon, Portugal, 2005.

[7] Z. Xing and E. Stroulia, "UMLDiff: an algorithm for object-oriented design differencing," in ASE '05 Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, pp. 54-65, Long Beach, California , 2005.

[8] L. Dinos, P. Vassiliadis and A. V. Zarras, "Keep Calm and Wait for the Spike! Insights on the Evolution of Amazon Services," in Advanced Information Systems Engineering, vol. 9694, Ljubljana, Springer International Publishing, 2016, pp. 444-458.

[9] J. Král and M. Žemlicka, "Popular SOA Antipatterns," Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, pp. 271-276, Athens, Greece, 2009.

[10] Y.-G. Guéhéneuc, G. Tremblay, N. Moha and F. Palma, "Specification and Detection of SOA Antipatterns in Web Services," in Software Architecture, Vienna, Springer International Publishing, 2014, pp. 58-73.

[11] J.-M. Jézéquel, B. Baudry, Y.-G. Guéhéneuc, B. J. Conseil, M. Nayrolles, [11]F. Palma and N. Moha, "Specification and Detection of SOA Antipatterns," in Service-Oriented Computing, Shanghai, Springer Berlin Heidelberg, 2012, pp. 1-16.

[12] J. M. Rodriguez, M. Crasso, A. Zunino and M. Campo, "Automatically Detecting Opportunities for Web Service Descriptions Improvement," in Software Services for eWorld, pp. 139-150, Buenos Aires, Argentina, 2010.

[13] C. Mateos, J. M. Rodriguez and A. Zunino, "A tool to improve code-first Web services discoverability through text mining techniques," Software: Practice and Experience, vol. 45, no. 7, p. 925–948, 2014.

[14] A. Ouni, R. G. Kula, M. Kessentini and K. Inoue, "Web Service Antipatterns Detection Using Genetic Programming," in GECCO '15 Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation,PP. 1351-1358, Madrid, Spain, 2015.

[15] V. Ravi, **Ramakanta Mohanty**, M. R. Patra, (2009) 'Application of Data Envelopment Analysis to Rank Web Services', The 3rd International conferences on Global Interdependence and Decision Sciences (ICGIDS '09), pp. 287 – 296, McMillan, India. ASCI, Hyderabad.

[16] **Ramakanta Mohanty**, V. Ravi, M R Patra, "Application of Fuzzy Multi Attribute Decision making Analysis to Rank Web Services", The 6th **IEEE** International Conference on next generation web services Practice (NWeSP, 2010), pp.52-57,2010, Gwalior, India.

[17] R. Agrawal, A. Gupta, Y. Prabhu, M. Varma, "Multi-Label Learning with Millions of Labels: Recommending Advertiser Bid Phrases for Web Pages", WWW '13 Proceedings of the 22nd international conference on World Wide Web, pp. 13 – 24, 2013.

[18] P. B Callahan and S. R.Kosaraju, "A decomposition of multi-dimensional pointsets with applications to k-nearest-neighbors and n-body potential fields", In Proc. 24th Ann. ACM Sympos. Theory Comput, pp. 546–556, 1992.

[19] K. L. Clarkson, K. L. "Fast algorithms for the all nearest neighbors problem", In Proc. 24th Ann. IEEE Symposia. on the Found. Comput. Sci., pp. 226–232, 1983

[20] J. G. Cleary, "Analysis of an algorithm for finding nearest neighbors in Euclidean space", ACM Transactions on Mathematical Software 5, 2, 183–192, 1979.

[21] S. Cost, and S. Salzberg, "A weighted nearest neighbor algorithm for learning with symbolic features", Machine Learning 10, 57–78, 1993.

[22] P. Langley, W. Iba, K. Thompson, "An Analysis of Bayesian Classifiers. Proc. 10th Nat. Conf. on Artificial Intelligence, AAAI Press and MIT Press,,PP. 223-228, 1992, USA.

[23] D. Lewis, "Naive Bayes at forty: The independence assumption in information re-Trieval", In Proceedings of European Conference on Machine Learning, pp. 4–15, 1998.

[24] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. Machine Learning, 29:103–130, 1997.

[25] Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of Bayesian classifiers. In AAAI-92, 1992.

[26] N. Friedman, D. Geiger, and Goldszmidt M. Bayesian network classifiers. Machine Learning, 29, pp. 131–163, 1997.

[27] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. "A Practical Guide to Support Vector Classification". Dept. of Computer Sci.National Taiwan Uni, Taipei, 106, Taiwan http://www.csie.ntu.edu.tw/~cjlin 2007.

[28] I.Guyon,B.Boster, ,V.Vapnik "Automastic Capacity Tuning of Very Large VC-dimension Classifiers",Advances in neural information Processing, 1993

[29] C. Chang and C. Lin, "LIBSVM: A Library for Support Vector Machines", 2001.

[30] Vladimir Vapnik, Corinna Cortes"Support vector networks," Machine Learning, vol. 20, pp. 273-297, 1995.

[31] Scikit-learn:Machine Learning in Python, Pedregosa *et al.*, JMLR12, pp. 2825-2830, 2011

[32] Andrew Colin, "Building Decision Trees with the ID3 Algorithm", Dr. Dobbs Journal, 1996

[33] R. Quinlan, "Induction of decision trees," Machine Learning, vol.1, No.1, pp. 81-106, 1986

[34] Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32, 2001.

[35] Bengio, Y. and LeCun, Y. Scaling learning algorithms towards AI. In Large-Scale Kernel Machines, 2007.

[36] James Bergstra, Yoshua Bengio, "Random Search for yperparameter ization", 2012.

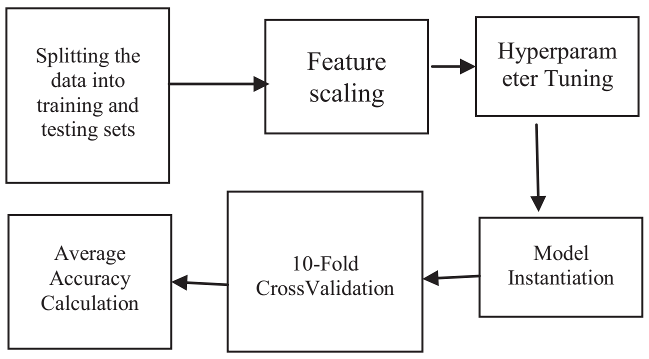

Fig.1 Phases of machine learning modelling carried out with feature scaling

TABLE 1 ACCURACIES OF DIFFERENT MACHINE LEARNING TECHNIQUES

Technique	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**Average Accuracy**
KNN	73.68	73.68	73.68	73.68	73.68	73.68	73.68	73.68	73.68	73.68	73.68
Naïve Bayes	81.57	81.08	81.08	83.33	80.55	91.66	77.77	80.55	77.77	91.66	82.70
SVM	**100**	97.29	97.29	97.22	97.22	**100**	**100**	**100**	94.44	97.22	**98.07**
Linear SVC	86.84	89.18	81.08	83.33	88.88	80.55	80.55	88.88	83.33	91.66	85.43
Decision Tree	**100**	**100**	**100**	97.22	**100**	**100**	**100**	**100**	**100**	**100**	**99.72**
Random Forest	**100**	**100**	**100**	94.44	**100**	**100**	**100**	**100**	**100**	**100**	**99.44**

TABLE 2 RESULTS OBTAINED BY MOHANTY et.al.. [2]

Techniques	**Accuracy**
PNN	98.71
BPNN	97.22
GMDH	98.32
J48	99.72
TREENET	99.72
CART	98.61
SVM	63.61

TABLE 3 PARAMETER TUNED FOR DIFFERENT MACHINE LEARNING ALGORITHM

Technique	**Hyper Parameters**
KNN	n_neighbor = 7, algorithm = 'auto'
Decision Tree	criterion ='gini', max_depth = 3, min_samples_lea= 7, splitter = 'best'
Random Forest	n_estimators = 200
SVM (kernel SVM)	c=1000.0, kernel='linear'
Linear SVM	c=10.0

APPENDIX 1

General Preprocessing Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
df=pd.read_csv('data.csv')
x=df.drop('Target_Class',axis=1)
y=df['Target_Class']
```

KNN:

Preprocessing Code:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size
=0.2,random_state=100)
from sklearn.preprocessing import StandardScaler
#splitting into train-test data and transforming x-train and
y-train for the purpose of performing gridSearchCV
scaler = StandardScaler()
x1_train = scaler.fit_transform(x1_train)
x1_test = scaler.transform(x1_test)
from sklearn.neighbors import KNeighborsClassifier
from random import randint
param_grid=[{'algorithm':['auto','ball_tree','kd_tree','brut
e']}]
b=GridSearchCV(KNeighborsClassifier(n_neighbors=7),
param_grid,cv=10,scoring='accuracy')
b.fit(x1_train,y_train)
b.best_params_
scaled_dt=scaler.fit_transform(x)
fin_dt=pd.DataFrame(scaled_dt,columns=df.columns[:-
1])
model1=KNeighborsClassifier(n_neighbors=7,algorithm
='auto')
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
validate_score_m1=cross_validate(model1,fin_dt,y,cv=10
,scoring='accuracy',return_train_score=False)
sorted(validate_score_m1.keys())
print(validate_score_m1)
fold_score_knn=cross_val_score(model1,fin_dt,y,cv=10,s
coring='accuracy')
print(fold_score_knn)
score_knn=fold_score_knn.mean()
print(score_knn)
```

Gaussian Naïve Bayes:

```
from sklearn.naive_bayes import GaussianNB,
BernoulliNB, MultinomialNB
model2= GaussianNB(priors=None)
validate_score_m2=cross_validate(model6,x,y,cv=10,scor
ing='accuracy',return_train_score=False)
sorted(validate_score_m2.keys())
print(validate_score_m2)
```

```
fold_score_gnb=cross_val_score(model6,x,y,cv=10,scorin
g='accuracy')
print(fold_score_gnb)
score_gnb=fold_score_gnb.mean()
print(score_gnb)
```

SVM:
Preprocessing Code:

```
from sklearn.svm import SVC
param_grid=[{'kernel':['rbf'],'gamma':[1e-3,1e-
4],'C':[1.0,10.0,100.0,1000.0]},
{'kernel':['linear'],'C':[1.0,10.0,100.0,1000.0]},
{'kernel':['rbf','linear','poly','sigmoid'],'gamma':[1e-3,1e-
4],'C':[1.0,10.0,100.0,1000.0]}]
clf=GridSearchCV(SVC(),param_grid,scoring='accuracy'
)
clf.fit(x1_train,y_train)
clf.best_params_
model3=SVC(C=1000.0,kernel='linear')
model3=SVC(C=1000.0,kernel='linear')
validate_score_m3=cross_validate(model3,fin_dt,y,cv=10
,scoring='accuracy',return_train_score=False)
sorted(validate_score_m3.keys())
print(validate_score_m3)
fold_score_svc=cross_val_score(model3,fin_dt,y,cv=10,sc
oring='accuracy')
print(fold_score_svc)
score_svc=fold_score_svc.mean()
print(score_svc)
```

Linear SVC:
Preprocessing Code:

```
from sklearn.svm import LinearSVC
param_grid=[{'C':[1.0,10.0,100.0,1000.0]}]
clf=GridSearchCV(LinearSVC(random_state=123),param
_grid,scoring='accuracy')
clf.fit(x1_train,y_train)
clf.best_params_
model4=LinearSVC(C=10.0,random_state=123)
validate_score_m4=cross_validate(model4,fin_dt,y,cv=10
,scoring='accuracy',return_train_score=False)
sorted(validate_score_m4.keys())
print(validate_score_m4)
fold_score_lsvc=cross_val_score(model4,fin_dt,y,cv=10,s
coring='accuracy')
print(fold_score_lsvc)
score_lsvc=fold_score_lsvc.mean()
print(score_lsvc)
```

DECISION TREE
Preprocessing Code:

```
from sklearn.tree import DecisionTreeClassifier
param_grid=[{'max_depth':[3,None],'min_samples_leaf':
[randint(1,9)],'criterion':['gini','entropy'],'splitter':['best','
random']}]
trial=GridSearchCV(DecisionTreeClassifier(random_stat
e=123),param_grid,cv=10,scoring='accuracy')
trial.fit(x_train,y_train)
print(trial.best_params_)
model5=DecisionTreeClassifier(criterion='gini',max_dept
h=3,min_samples_leaf=2,splitter='best',random_state=12
3)
validate_score_m5=cross_validate(model5,x,y,cv=10,scor
ing='accuracy',return_train_score=False)
sorted(validate_score_m5.keys())
print(validate_score_m5)
fold_score_dtrees=cross_val_score(model5,x,y,cv=10,sco
ring='accuracy')
print(fold_score_dtrees)
fold_score_dtrees=cross_val_score(model5,x,y,cv=10,sco
ring='accuracy')
print(fold_score_dtrees)
```

Random Forest:

```
from sklearn.ensemble import RandomForestClassifier
model6=RandomForestClassifier(n_estimators=200,rand
om_state=123)
validate_score_m6=cross_validate(model6,x,y,cv=10,scor
ing='accuracy',return_train_score=False)
sorted(validate_score_m6.keys())
print(validate_score_m6)
fold_score_rfc=cross_val_score(model6,x,y,cv=10,scorin
g='accuracy')
print(fold_score_rfc)
score_rfc=fold_score_rfc.mean()
print(score_rfc)
```