# CHAPTER 4
# ANALYSIS AND DESIGN

## 4.1. Analysis

The dataset used in this study was taken from the UCI Machine Learning Repository website. This dataset consists of 12330 rows and 18 columns with attribute details can be seen in the following table. Attributes such as “Administrative”, “Administrative Duration”, “Informational”, “Informational Duration”, “Product Related” and “Product Related Duration” are about visitors opening multiple pages on an online store and how long it took them to open each one. . These values are derived from the URL information of the pages visited by the users and are updated in real time when the user performs an action, such as moving from one page to another. Then the attributes "Bounce Rate", "Exit Rate" and "Page Value" are the metric values measured by "Google Analytics" for each page in the online store. The value of "Bounce Rate" is the percentage of visitors who enter the site from that page and immediately leave ("bounce") without taking any action. The "Exit Rate" value is the percentage of visitors leaving the site after visiting several pages in their entirety. Then "Page Value" is the average value of visitors visiting a specific page before making a transaction. The attribute "Special Day" is a value that indicates the closeness of time to certain special days such as Hari Raya, Valentine's Day, Independence Day, etc. where visitors are more likely to make transactions. Then other attributes like "Operating Systems", "Browser", "Region", "Traffic Type", "Visitor Type" are values that represent supporting descriptions for visitor data. "Weekend" and "Month" are adverbs of the time when a visitor visits the online store site. And the last is the "Revenue" attribute, which indicates a boolean value whether the visitor makes a transaction or not on the online store site.

**Table 4.1.** Dataset Columns Details

| No | Attribute | Description |
|---|---|---|
| **1** | Administrative | The number of pages of this type (administrative) that the user visited |
| **2** | Administrative_Duration | The amount of time spent in this category of pages |

| 3 | Informational | This is the number of pages of this type (informational) that the user visited |
|---|---|---|
| 4 | Informational_Duration | The amount of time spent in this category of pages |
| 5 | ProductRelated | The number of pages of this type (product related) that the user visited |
| 6 | ProductRelated_Duration | The amount of time spent in this category of pages |
| 7 | BounceRates | The percentage of visitors who enter the website through that page and exit without triggering any additional tasks |
| 8 | ExitRates | The percentage of pageviews on the website that end at that specific page |
| 9 | PageValues | The average value of the page averaged over the value of the target page and/or the completion of an eCommerce transaction |
| 10 | SpecialDay | Represents the closeness of the browsing date to special days or holidays (eg Mother's Day or Valentine's day) in which the transaction is more likely to be finalized |
| 11 | Month | Contains the month the pageview occurred, in string form |
| 12 | OperatingSystems | Representing the operating system that the user was on when viewing the page |
| 13 | Browser | Representing the browser that the user was using to view the page |
| 14 | Region | Representing which region the user is located in |
| 15 | TrafficType | Representing what type of traffic the user is categorized into |
| 16 | VisitorType | Representing whether a visitor is New Visitor, Returning Visitor, or Other |
| 17 | Weekend | Representing whether the session is on a weekend |
| 18 | Revenue | Representing whether or not the user completed the purchase |

In the data, in the "Month" and "VisitorType" columns there are data types as strings that will not be processed, as well as “Weekend” and “Revenue” boolean data types. So the conversion will be done first to change the data type to int. The conversion process is carried out by changing the name of the month to a number according to the sequence, for example January becomes

number 1, February becomes number 2, and so on. For the boolean data type use the LabelEncoder process from sklearn to convert the data into int.

## 4.2. Algorithm

There are two algorithms used to process the data, namely Stochastic Gradient Descent (SGD) and Support Vector Machine (SVM). SGD and SVM have been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features. The advantages of Stochastic Gradient Descent are efficiency and ease of implementation (lots of opportunities for code tuning), but it has disadvantages that requires a number of hyperparameters such as the regularization parameter and the number of iterations, and also SGD is sensitive to feature scaling. On the other side, SVM has advantages such as Effective in high dimensional spaces. Still effective in cases where number of dimensions is greater than the number of samples. Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. Both algorithms are then re-tuned their hyperparameters using RandomizedSearchCV. RandomizedSearchCV implements a “fit” and a “score” method. It also implements “score_samples”, “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used. The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings. In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n_iter. If all parameters are presented as a list, sampling without replacement is performed. If at least one parameter is given as a distribution, sampling with replacement is used. It is highly recommended to use continuous distributions for continuous parameters.

### *4.2.1. Stochastic Gradient Descent (SGD)*

This algorithm in plain terms means slope or slant of a surface. So gradient descent literally means descending a slope to reach the lowest point on that surface. For classification it implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties for classification. The decision boundary of a SGD trained with the hinge loss, equivalent to a linear SVM. As other classifiers, SGD has to be fitted with two arrays: an array X of shape (n_samples, n_features) holding the training samples, and an array y of shape (n_samples,) holding the target values (class labels) for the training samples.

The parameters used in this algorithm are more, namely there is penalty, alpha, max_iter, loss, learning_rate, class_weight, eta0. The penalty (aka regularization term) to be used. Defaults to 'l2' which is the standard regularizer for linear SVM models. 'l1' and 'elasticnet' might bring sparsity to the model (feature selection) not achievable with 'l2'. The alpha is constant that multiplies the regularization term. The higher the value, the stronger the regularization. Also used to compute the learning rate when set to learning_rate is set to 'optimal'. The max_iter is the maximum number of passes over the training data (aka epochs). The loss function describes how well the model will perform given the current set of parameters (weights and biases), and gradient descent is used to find the best set of parameters. 'hinge' gives a linear SVM. 'log_loss' gives logistic regression, a probabilistic classifier. 'modified_huber' is another smooth loss that brings tolerance to outliers as well as probability estimates. 'squared_hinge' is like hinge but is quadratically penalized. 'perceptron' is the linear loss used by the perceptron algorithm. The eta0 is the initial learning rate for the 'constant', 'invscaling' or 'adaptive' schedules. The default value is 0.0 as eta0 is not used by the default schedule 'optimal'. 'constant': eta = eta0. 'optimal': eta = 1.0 / (alpha * (t + t0)) where t0 is chosen by a heuristic proposed by Leon Bottou. 'invscaling': eta = eta0 / pow(t, power_t). 'adaptive': eta = eta0, as long as the training keeps decreasing. The class_weight can indeed help increasing the ROC AUC or f1-score of a classification model trained on imbalanced data.

Stochastic Gradient Descent is an optimization method for unconstrained optimization problems. In contrast to (batch) gradient descent, SGD approximates the true gradient of $\boldsymbol{E}(\boldsymbol{w}, \boldsymbol{b})$ by considering a single training example at a time. The class ***SGDClassifier***

implements a first-order SGD learning routine. The algorithm iterates over the training examples and for each example updates the model parameters according to the update rule given by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{\eta}\left[\boldsymbol{\alpha}\frac{\partial \boldsymbol{R}(\boldsymbol{w})}{\partial \boldsymbol{w}} + \frac{\partial \boldsymbol{L}(\boldsymbol{w}^T \boldsymbol{x}_i + \boldsymbol{b}, \boldsymbol{y}_i)}{\partial \boldsymbol{w}}\right]$$

**Figure 4.1** SGD Classifier

where $\boldsymbol{\eta}$ is the learning rate which controls the step-size in the parameter space. The intercept $\boldsymbol{b}$ is updated similarly but without regularization. The learning rate $\boldsymbol{\eta}$ can be either constant or gradually decaying. For classification, the default learning rate schedule ***(learning_rate='optimal')*** is given by

$$\boldsymbol{\eta}^{(t)} = \frac{\mathbf{1}}{\boldsymbol{\alpha}(\boldsymbol{t}_0 + \boldsymbol{t})}$$

**Figure 4.2** Learning Rate for Classification

where $\boldsymbol{t}$ is the time step (there are a total of ***n_samples * n_iter time*** steps), $\boldsymbol{t}_0$ is determined based on a heuristic proposed by Léon Bottou such that the expected initial updates are comparable with the expected size of the weights (this assuming that the norm of the training samples is approx. 1).

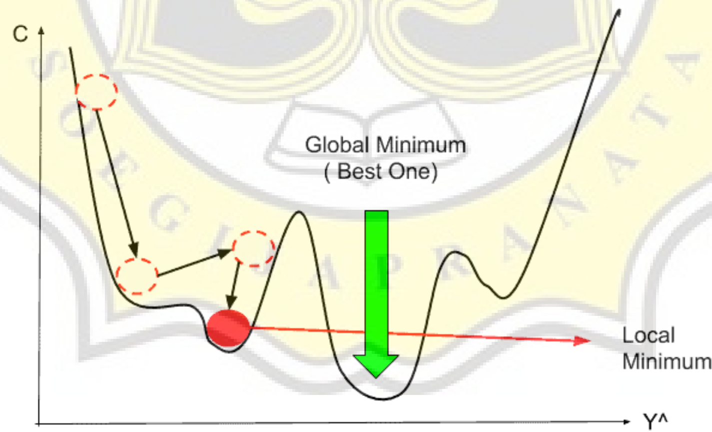


**Figure 4.3** SGD Example

### *4.2.2. Support Vector Machine (SVM)*

A support vector machine constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. The figure below shows the decision function for a linearly separable problem, with three samples on the margin boundaries, called ***"support vectors"***.

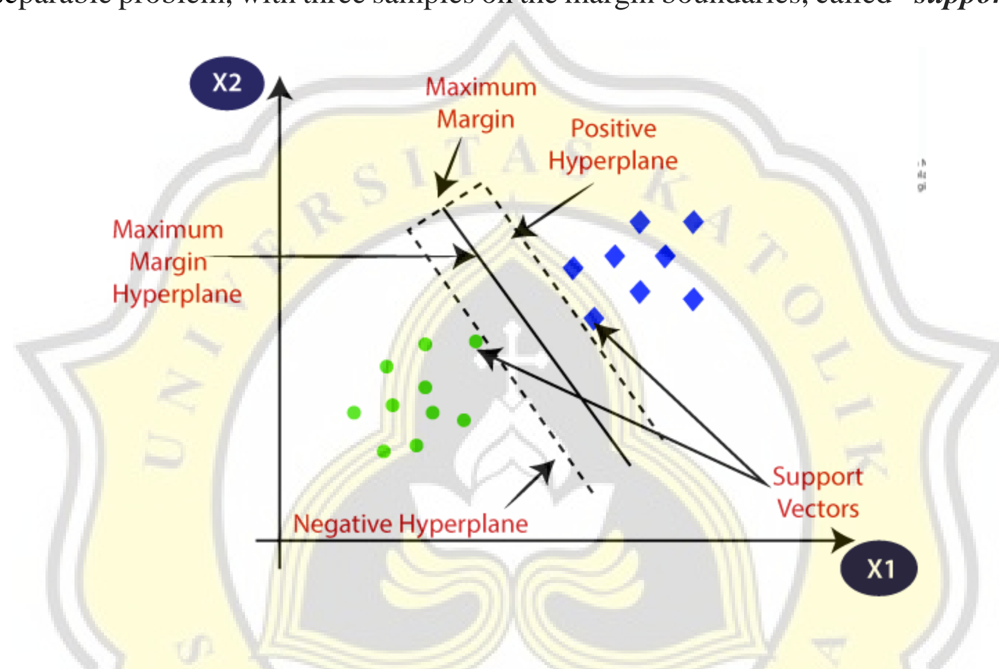


**Figure 4.1** SVM Example

In this Support Vector Machine, there are two kernels that will be used and tuned again to find which one is more optimal, namely the RBF and Sigmoid kernels. RBF or Radial Basis Function kernel works by mapping the data into a high-dimensional space by finding the dot products and squares of all the features in the dataset and then performing the classification.

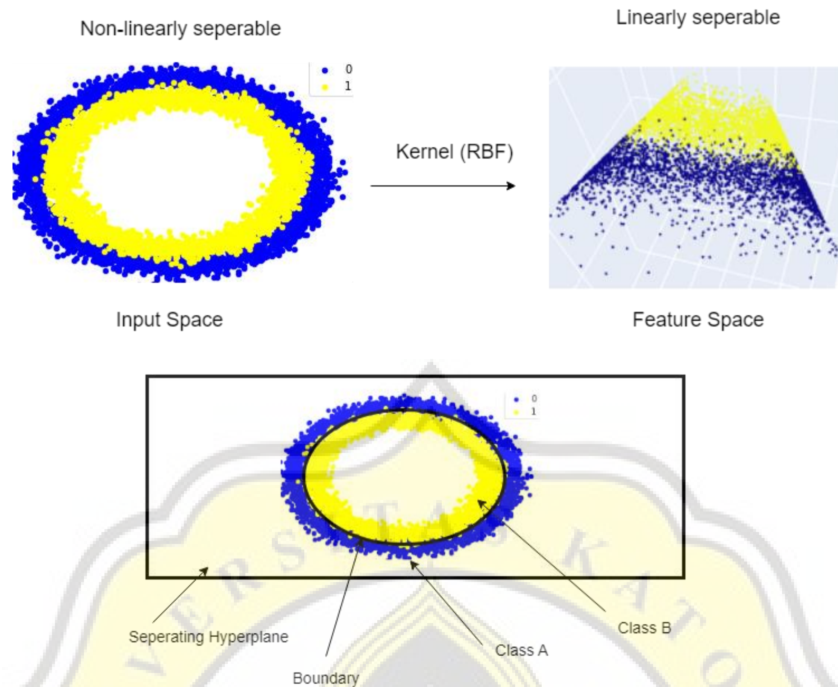


**Figure 4.2** RBF Kernel

Next is Sigmoid, The Sigmoid kernel is widely applied in neural networks for classification processes. The SVM classification with the sigmoid kernel has a complex structure and it is difficult for humans to interpret and understand how the sigmoid kernel makes classification decisions. Interest in these kernels stems from their success in classifying with the neural netwotk and logistic regression, specific properties, linearity and cumulative distribution. The sigmoid kernel is generally problematic or invalid because it is difficult to have positive parameters. The sigmoid function is now not widely used in research because it has a major drawback, namely that the output value range of the sigmoid function is not centered on zero. This causes the backpropagation process to occur which is not ideal, so that the weight of the ANN is not evenly distributed between positive and negative values and tends to approach the extreme values 0 and 1.

Parameter Cost or commonly referred to as C is a parameter that works as an SVM optimization to avoid misclassification in each sample in the training dataset. When the value of C is too large, the algorithm tries to reduce misclassifications or misclassifications as much as possible. This will cause the loss of generalization properties of the classifier (algorithm). Simply put, if C is too large, it will cause the decision boundary to be very small. When the value of C is too small, misclassification of data points will occur due to a wider decision boundary. Wider decision boundaries generalize well to training and test data but may classify some records incorrectly. Conclusion: the higher the value of C, the smaller the possibility of errors in determining the solution. Conversely, the lower the value of C, the higher the proportion of errors that occur in determining the solution.

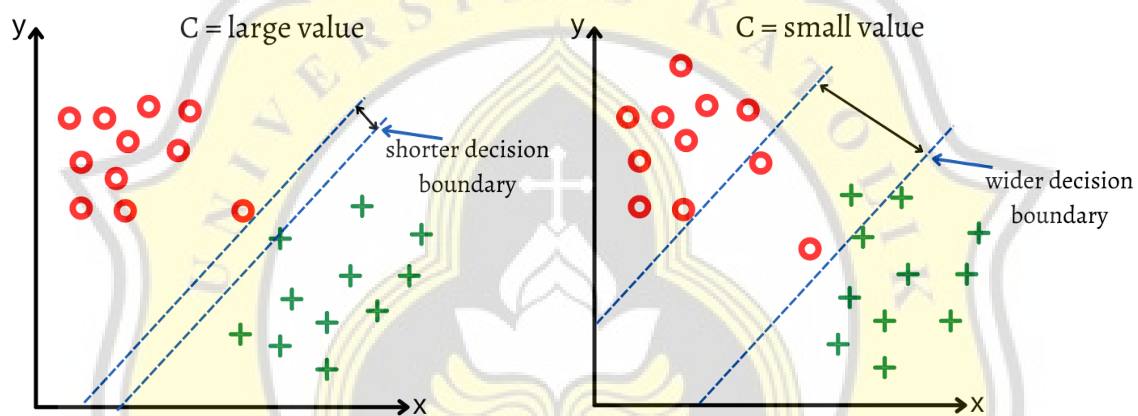


**Figure 4.3** C Value

The Gamma parameter determines how far the influence of one sample training dataset is. A low value means "far", and a high value means "close". When the gamma value is high, the exact breakdown of the decision boundary will depend only on the points very close to it. When the Gamma value is low it indicates that even distant points are considered when we want to decide where the decision boundary should be. In conclusion, when taking a high Gamma value, it means that the points around the line will be considered in the calculations. Meanwhile, when the Gamma value is low, points that are far from the dividing line are considered in the calculation. Therefore it is necessary to find the optimum value of C and Gamma value.

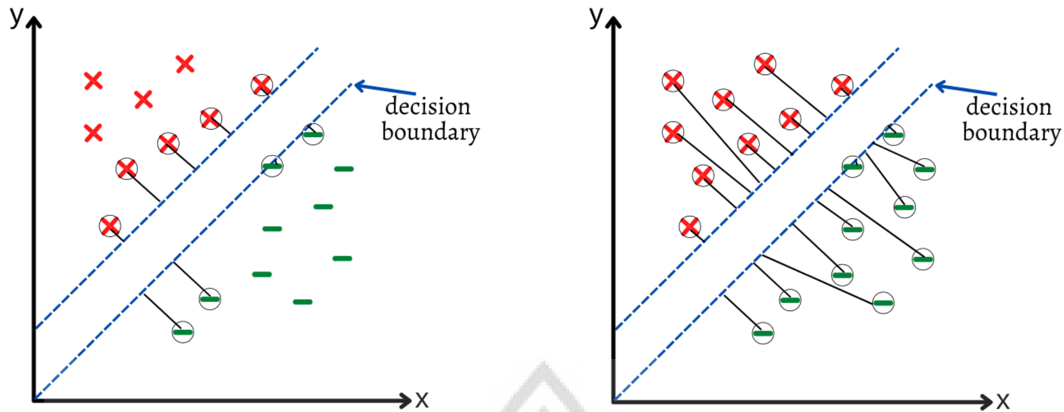


**Figure 4.4** Gamma Value

In general, when the problem isn't linearly separable, the support vectors are the samples within the margin boundaries. The two-dimensional linearly separable data can be separated by a line. The function of the line is $y = ax + b$. We rename x with $x_1$ and y with $x_2$ and we get:

$$\boldsymbol{a x_1 - x_2 + b = 0}$$

**Figure 4.5** SVM Hyperplane I

If we define $\boldsymbol{x} = (\boldsymbol{x_1}, \boldsymbol{x_2})$ and $\boldsymbol{w} = (\boldsymbol{a}, \boldsymbol{-1})$, we get:

$$\boldsymbol{w \cdot x + b = 0}$$

**Figure 4.6** SVM Hyperplane II

This equation is derived from two-dimensional vectors. But in fact, it also works for any number of dimensions. This is the equation of the hyperplane. Once we have the hyperplane, we can then use the hyperplane to make predictions. We define the hypothesis function h as:

$$\boldsymbol{h}(\boldsymbol{x_i}) = \begin{cases} +1 \boldsymbol{if}\ \boldsymbol{w \cdot x + b} \geq 0 \\ -1 \boldsymbol{if}\ \boldsymbol{w \cdot x + b} < 0 \end{cases}$$

**Figure 4.7** SVM Classifier

The point above or on the hyperplane will be classified as class +1, and the point below the hyperplane will be classified as class -1. The goal of the SVM learning algorithm is to find a hyperplane which could separate the data accurately. There might be many such hyperplanes and needed to find the best one, which is often referred as the optimal hyperplane.

### *4.2.3. Hyperparameter Tuning with RandomizedSearchCV*

The hyperparameters tuning in this project use RandomizedSearchCV. It has parameters for tuning the target parameters algorithm such as estimator, param_distributions, n_jobs, verbose, scoring. The estimator is an object of that type is instantiated for each grid point. This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function, or scoring must be passed. The param_distributions is the dictionary with parameters names (str) as keys and distributions or lists of parameters to try. Distributions must provide a rvs method for sampling (such as those from scipy.stats.distributions). If a list is given, it is sampled uniformly. If a list of dicts is given, first a dict is sampled uniformly, and then a parameter is sampled using that dict as above. The verbose controls the verbosity: the higher, the more messages. >1 : the computation time for each fold and parameter candidate is displayed; >2 : the score is also displayed; >3 : the fold and candidate parameter indexes are also displayed together with the starting time of the computation. The scoring Strategy to evaluate the performance of the cross-validated model on the test set. If scoring represents a single score, one can use: a single string or a callable that returns a single value. If scoring represents multiple scores, one can use: a list or tuple of unique strings or a callable returning a dictionary where the keys are the metric names and the values are the metric scores or a dictionary with metric names as keys and callables a values. If None, the estimator's score method is used. The parameter n_jobs refers to the number of these jobs that will be executed in parallel. With n_jobs=1, the jobs will be executed sequentially. With n_jobs=4, 4 jobs will be executed in parallel, potentially making better use of multiple cpu cores.

### *4.2.4. Data Balancing with Adaptive Synthetic (ADASYN)*

Adaptive Synthetic Sampling is an oversampling technique used in imbalanced dataset to balance the class distribution by generating synthetic samples for the minority class. The main idea behind ADASYN is to increase the weight of the samples that are difficult to classify and decrease the weight of the samples that are easily classifiable.

ADASYN works in following steps:

1. Calculate the density of each sample in the minority class.

2. Assign a weight to each sample in the minority class based on its density. Higher density samples will have higher weight and lower density samples will have lower weight.
3. Select k-nearest neighbors of each sample in the minority class.
4. For each sample in the minority class, generate a synthetic sample by interpolating between the selected sample and one of its k-nearest neighbors. The interpolation weight is proportional to the density of the neighbors.
5. Combine the original minority class samples and the synthetic samples to form the new balanced dataset.

ADASYN aims to balance the class distribution by generating synthetic samples for the minority class that are similar to the real samples and close to the decision boundary between the classes. This way, it aims to improve the classification performance on imbalanced datasets.

### *4.2.5. Dimensionality Reduction with Principal Component Analysis (PCA)*

Principal Component Analysis is a dimensionality reduction technique used in machine learning and other data analysis fields. It works by transforming the original data into a new coordinate system, where the first principal component has the highest variance, the second principal component has the second highest variance, and so on. The idea behind PCA is to preserve as much of the original data's information as possible in these new components, and to eliminate noise and redundant information in the process. PCA is performed by first computing the covariance matrix of the original data, and then calculating the eigenvectors and eigenvalues of the matrix. The eigenvectors corresponding to the largest eigenvalues are selected as the principal components, and the original data is transformed into the new coordinate system defined by these components. The transformed data is then projected onto a lower-dimensional subspace, which reduces the size of the data while preserving as much of its structure as possible.

The implementation of PCA is often performed using linear algebra and matrix operations, and can be easily done using libraries such as scikit-learn in Python. The steps for implementing PCA in code are as follows:

1. Import the PCA class from the scikit-learn library

2. Initialize the PCA class, specifying the number of components you want to keep (or the amount of explained variance you want to preserve)
3. Fit the PCA model to the original data
4. Transform the original data into the new PCA-defined coordinate system
5. Optionally, inverse transform the transformed data back into the original coordinate system, if necessary.