CEG5103 Wireless and Sensor Networks for IoT

CA2 report

Anomaly Detection and Prediction

Supervisor: Prof. Tham, Chen Khong

Author:

Student No:

April 2023

Table of Contents

Window Size & Step Size for the Sliding Mean 1
Maximum Likelihood Gaussian Classifier 2
MLE as Parameter Estimation 2
Gaussian MLE[5] 2
Gaussian Classifier 3
Naive Bayesian 4
Support Vector Machine 6
Random Forest (Decision tree) 8
Logistic regression 10
Comparison & Conclusion: 11
Reference 13

Window Size & Step Size for the Sliding Mean

Since sensor measurements are often noisy, features from a sliding window are used to capture a better trend of the underlying data. A reasonable window size and step size can reduce noise as much as possible while keeping the fine-grain feature of the original data. There are various techniques to evaluate the quality of the feature, such as data visualization, statistical analysis, and feature selection. Here I use statistical methods such as mean, median, standard deviation, skewness, kurtosis etc. to help disclose the characteristics of the feature. As Signal-to-Noise-Ratio is one of the most popular evaluation metrics in electrical engineering, I define the SNR for feature evaluation as follows:

$$SNR = \frac{CV}{mean} = \frac{(x_{max} - x_{min})/\sigma}{\bar{x}}$$

Where x is the samples of the data, $\bar{x}$ is the average of the samples, $x_{max}\ and\ x_{min}$ are the maximum and minimum of the samples respectively, σ is the standard deviation of the samples. CV stands for Coefficient of Variation, which is a statistical measure that expresses the degree of variation of a set of data points relative to their mean. It is calculated as the ratio of the standard deviation to the mean. In this project, the SNR of 4 sensor measurement data is computed independently and the average value of 4 SNR is used for evaluation. I went through the window size from 2 to 50. For each window size, I went through the step size from 1 to the value of the window size. The result is shown in Fig.1

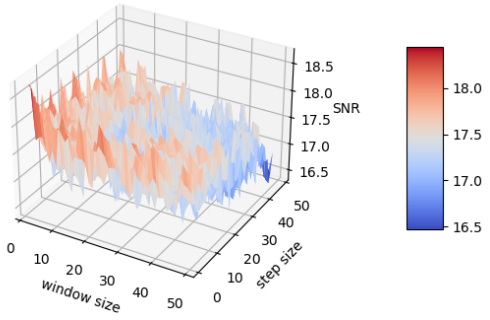


Fig.1 SNR distribution as window size and step size variation[1]

The maximum 10 SNR are shown in following table:

SNR	18.72	18.67	18.44	18.43	18.43	18.38	18.38	18.36	18.29	18.25
Window size	2	3	4	17	5	8	23	23	30	24
Step size	1	1	1	1	1	1	1	1	1	1

From the result, the value of SNR is close to each other. On the other hand, if the window size is too small, the filter is useless. In general, the longer the window size, the better the filter performance. Therefore, here I choose the window size 30 and step size 1 as the parameter setting for our group.

Maximum Likelihood Gaussian Classifier

MLE as Parameter Estimation

MLE is one flavor of parameter estimation in machine learning, and in order to perform parameter estimation, we need:

- some data X
- some hypothesized generating function of the data f(X, θ)
- a set of parameters from that function θ
- some evaluation of the goodness of our parameters (an objective function)

In MLE, the objective function (evaluation) we chose is the likelihood of the data given our model. This intuitively makes sense if you keep in mind that we don't get to change our data, and we have to make some assumptions about the form of our model, but we can adjust the parameterization of our model. So, we are limited to adjusting θ, and we might as well choose the best θ for our data. To find the best θ then, we need to find the θ which maximizes our evaluation function (the likelihood). Therefore, in its general form, the MLE is:

$$\boldsymbol{\theta}_{\mathrm{MLE}} = \mathbf{argmax}_{\boldsymbol{\theta}}\mathbf{p}(\mathbf{X}|\boldsymbol{\theta})$$

Gaussian MLE[5]

We assume the data we're working with was generated by an underlying Gaussian process in the real world. As such, the likelihood function (L) is the Gaussian itself.

$$\mathbf{L} = \mathbf{p}(\mathbf{X}|\boldsymbol{\theta}) = \mathbf{N}(\mathbf{X}|\boldsymbol{\theta}) = \mathbf{N}(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Therefore, for MLE of a Gaussian model, we will need to find good estimates of both parameters: μ and

[1] Since it is meaningless if step size bigger than window size, SNR is not computed in such situation. It is symmetry from the step size less than window size for figure ploting.

Σ:

$$\boldsymbol{\mu_{\mathrm{MLE}} = \mathbf{argmax}_{\boldsymbol{\mu}} \mathbf{N}(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}; \quad \boldsymbol{\Sigma_{\mathbf{MLE}} = \mathbf{argmax}_{\boldsymbol{\Sigma}} \mathbf{N}(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma})}$$

Before we can get to the point where we can find our best μ and Σ, we need to do some algebra, and to make that algebra easier, instead of just using the likelihood function as our evaluation function, we're going to use the log-likelihood. This makes the math easier and it doesn't run any risks of giving us worse results. That's because the log function is monotonically increasing, and therefore

$$\mathbf{argmax}_{\boldsymbol{\theta}} \mathbf{log}(\mathbf{f}(\boldsymbol{\theta})) == \mathbf{argmax}_{\boldsymbol{\theta}} \mathbf{f}(\boldsymbol{\theta})$$

Since in reality, our dataset X is a set of labelled data $X = [x_1, x_2, x_3 \dots x_n]$, to evaluate our parameters on the entire dataset, we need to sum up the log-likelihood for each data point.

$$\log\big(N(X|\theta)\big) = \sum_{n=1}^{N} \log\left(N(x_n|\theta)\right); \qquad \theta = [\mu, \boldsymbol{\Sigma}]$$

Plug in the formula and the value of θ for the Gaussian distribution.

$$\sum_{n=1}^{N} \log\left(N(x_n|\,\mu, \sigma^2)\right) = \sum_{n=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} * e^{-\frac{1(x_n-\mu)^2}{2 \quad \sigma^2}}\right)$$

We want to find the best parameters for our model given our data, so we're going to find the $\mathbf{argmax}_{\boldsymbol{\mu}}$ and $\mathbf{argmax}_{\sigma^2}$. Before we can get to that point, we need to do some simplifications to the log likelihood to make it easier to work with (that is, since we will soon be doing some partial derivatives, the log likelihood in its current form it will lead to some messy math). In the following, LL means log likelihood.

$$LL = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n - \mu)^2$$

To solve $argmax_{\mu} LL(X|\mu, \sigma^2), argmax_{\sigma^2} LL(X|\mu, \sigma^2)$ are equal to solve problemS:

$$\frac{\partial LL}{\partial \mu} = 0; \quad \frac{\partial LL}{\partial \sigma^2} = 0$$

after calculating: $\mu = \frac{1}{N}\sum_{n=1}^{N} x_n; \quad \sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu)^2,$

Gaussian Classifier

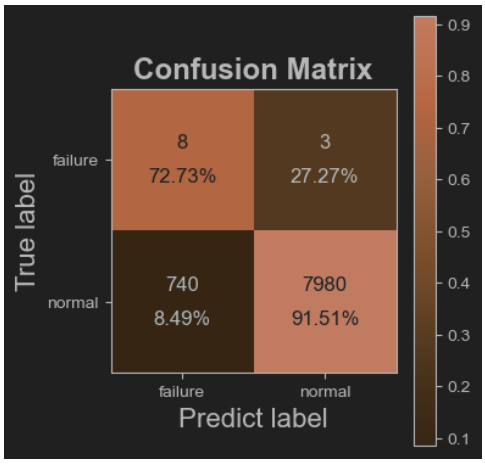
Assume that different measurements $\{[x_1, x_2, x_3 \dots x_K]\}$ are jointly Gaussian and correlated. M possible target classes: $\omega_m \in \Omega$ in this task, there are only two classes 'normal' and 'failure'.

$$\mu_j^c = E_j[x^c], \qquad \boldsymbol{\Sigma}_j^c = E_j[\big(x^c - \mu_j\big)\big(x^c - \mu_j\big)^T],$$

x^c is the train_data in j-th class, μ_j^c and $\boldsymbol{\Sigma}_j^c$ are the mean and covariance of the j-th class, respectively. And C is the classifier assigns one of the classes to x :

$$P\big(x\big|\omega_j\big) = \frac{1}{\sqrt{2\pi\boldsymbol{\Sigma}_j^c}} \times e^{-\frac{1\left(x-\mu_j^c\right)^2}{2 \quad \boldsymbol{\Sigma}_j^c}}, \qquad C(x) = argmax_{j=1,\dots,M} P\big(x\big|\omega_j\big),$$

The performance of the Gaussian Maximum Likelihood Classifier in anomaly detection and prediction task is shown in the following confusion matrix. The accuracy of the model is 0.915, G-Mean is 0.816, recall is 0.727, specificity is 1.000, precision is 0.011, F1 is 0.021



The Gaussian Likelihood classifier is a practical and computationally efficient method for many applications, especially when processing large datasets. However, its performance may be limited in situations where the data distribution is non-Gaussian or the covariance matrix is ill-conditioned, leading to inaccurate results and poor anomaly detection. Ultimately, the efficacy of the Gaussian Likelihood classifier relies on the data and parameter estimation quality.

The computational complexity[6] of the Gaussian Likelihood classifier depends on the number of features (variables) in the dataset. Given a dataset with n samples and p features, the computational complexity of training the classifier is $O(np^2 + p^3)$, where the first term corresponds to the calculation of the sample means and covariances, and the second term corresponds to the inversion of the covariance matrix. In the case of making predictions for a new sample, the computational complexity is $O(p^2)$, which corresponds to the calculation of the probability density function for each class using the mean and covariance parameters estimated during training.

Naive Bayesian

The naive Bayes classifier, also known as naive Bayes, is an effective and widely used classification algorithm. It is ranked among the top 10 algorithms in data mining [7] and finds applications in various domains such as text categorization [8], spam filtering [9], and data stream classification [10]. Naive Bayes is a generative model-based classifier [11] that offers quick and efficient learning and testing processes.

Bayesian classifiers, work based on the Bayesian rule and probability theorems. It has been proven that learning an optimal Bayesian classifier from training data is an NP-hard problem [12]. A simplified version of the Bayesian classifier called naive Bayes uses two assumptions. The former is that, given the class label, attributes are conditionally independent and the latter is that, no latent attribute affects the label prediction process [13].

Assume, the vector $(x_1, \dots, x_n)$ represents the n attributes of the instance x. Let c represents the class label of the instance x. The probability of observing x given the class label c can be computed by following equation that is relaxed by above assumptions.

$$p(x_1, \dots, x_n|c) = \prod_{i=1}^{n} p(x_i|c)$$

In order to predict the label of instance x, the probability of instance x in each class label is computed. The class with the maximum probability is identify as the class label of the instance x. Following equation defines the label estimation process of instance x.

$$C(x)_{NB} = \underbrace{\operatorname{argmax}}_{c} \; p(c) \prod_{i=1}^{n} p(x_i|c)$$

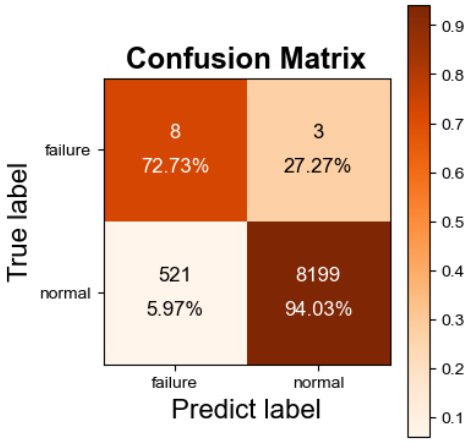
In most real-world problems, the assumption of conditional independence between attributes in naive Bayes is often weak and incorrect, except when the attributes are extracted from independent processes. Various methods have been proposed to enhance the conditional independence assumption in naive Bayes.

Gaussian naive Bayes classification is a case of naive Bayes method with an assumption of having a Gaussian distribution on attribute values given the class label. For example, suppose that ith attribute is continuous and its mean and variance are represented by $\mu_{c,i}$ and $\sigma_{c,i}$ respectively, given the class label c. Hence, the probability of observing the value x_i in ith attribute given the class label c, is computed by following equation that is also called as normal distribution.

$$p(x_i|c) = \frac{1}{\sqrt{2\pi\sigma_{c,i}^2}} \exp\left(-\frac{(x_i-\mu_{c,i})^2}{2\sigma_{c,i}^2}\right)$$

With Gaussian estimation, this method progress its weakness of conditional independence of attributes.

In this section, I use the GaussianNB API in sklearn to train and test the gaussian naïve bayes model. The confusion matrix is shown:



Accuracy	Gmean	Recall	Specificity	Precision	F1
0.94	0.827	0.727	1	0.015	0.03

The performance is shown in the table. It is obvious that the gaussian naïve bayes classifier achieves a relatively high accuracy. However, the precision of the model is only 0.015, which means it misclassify too many false positive. Comparing to the example KNN, the recall of GNB is higher.

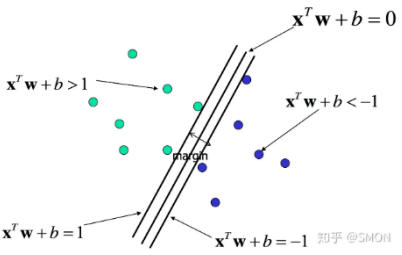
The time complexity of Gaussian naive Bayes classifier for training is O(np), where n is the number of training instances and p is the number of features. The time complexity for classification of a single instance is O(p), since it involves computing the probabilities of each

feature given the class and then multiplying them together. The space complexity of Gaussian naive Bayes classifier is O(cp), where c is the number of classes and p is the number of features. This is because the algorithm needs to store the mean and variance of each feature for each class.

Support Vector Machine

SVM[1] is a binary classification model. Its basic model is defined as the linear classifier with the largest margin in the feature space. The learning strategy of SVM is to maximize the margin. There are many models in Support Vector Machine (SVM) learning methods, ranging from simple to complex.

1. Linearly Separable SVM: When the training data is linearly separable, a linear classifier can be learned through hard margin maximization, which is called hard-margin SVM. Consider a linearly separable training dataset in this form: $(X_1, y_1), (X_2, y_2), \dots, (X_n, y_n), y \in +1, -1$, y = +1 represents positive class, y = -1 represents negative class. A hyperplane is determined by the normal vector W and the intercept b. Its equation is $X^T W + b = 0$. We can specify that the side pointed to by the normal vector is the positive class, and the other side is the negative class. Our goal is to maximum the margin ρ in the figure above. After some mathematical transformations:



$$\min_{W,b} J(W) = \min_{W,b} \frac{1}{2}||W||^2, s.t. y_i(X_i^T W + b) \geq 1, i = 1,2, \dots, n.$$

We refer to the problem described by the equation as the primal problem. We can apply the Lagrange multiplier method to construct the Lagrange function and then obtain the optimal solution for the primal problem by solving its dual problem. The Lagrange function is:

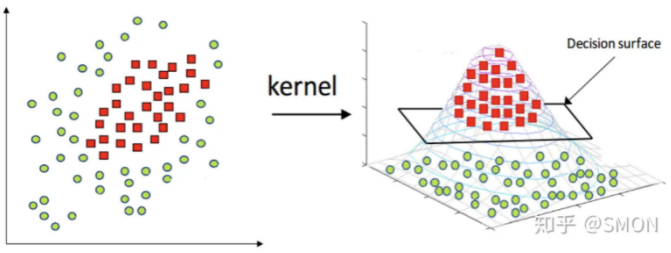
$$L(W, b, \alpha) = \frac{1}{2}||W||^2 - \sum_{i=1}^{n} \alpha_i [y_i(X_i^T W + b) - 1].$$

After some discussion and the application of the Lagrange multiplier method, we need to solve this: $\max_{\alpha} \min_{W,b} L(W, b, \alpha)$. In order to obtain the solution of the dual problem, we first need to find the minimum L with respect to W and b, and then find the maximum L with respect to α. We have $\min_{\alpha} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j X_i^T X_j - \sum_{i=1}^{n} \alpha_i \ (\sum_{i=1}^{n} \alpha_i y_i = 0, \alpha_i \geq 0, i = 1,2, \dots, n)$. It is not difficult to see that this is a quadratic programming problem, and there are existing general algorithms to solve it. The best $\hat{b}$ is: $\hat{b} = y_j - \sum_{i=1}^{n} \hat{\alpha}_i y_i X_j^T X_i$. The obtained separating hyperplane is: $\sum_{i=1}^{n} \hat{\alpha}_i y_i X^T X_i + \hat{b} = 0$, the decision function for classification is: $f(X) = sign(\sum_{i=1}^{n} \hat{\alpha}_i y_i X^T X_i + \hat{b})$.

2. Linear SVM: When the training data is not linearly separable but can be approximately linearly separable, a linear classifier can also be learned through soft margin maximization, which is called soft-margin SVM. By allowing a small number of samples to not satisfy the constraints, we add a penalty term for these points, and the optimization objective becomes:

$\min_{W,b} \frac{1}{2}||W||^2 + C\sum_{i=1}^{n} \max\left(0,1 - y_i\left(X_i^T W + b\right)\right)$. With the similar process, it can be solved again.

3. Non-linear SVM: When the training data is not linearly separable, a non-linear SVM can be learned through the use of kernel techniques and soft margin maximization.



The basic idea of the kernel trick can be divided into two steps: first, use a transformation to map the original space data to a new space (for example, a higher-dimensional or even infinite-dimensional space); then, learn the model from the training data in the new space using linear methods.

Parameters setting:

The window size and step I selected are the same as those of my teammates, which are 30 and 1. The penalty parameter, C, is crucial. The larger the value of C, the greater the penalty on slack variables, and the more severe the punishment for misclassification. This tends to result in high accuracy for the training set but weak generalization ability. When the value of C is small, the penalty for misclassification is reduced, allowing for error tolerance and treating them as noise points, which results in stronger generalization ability. I set C to 0.5 to achieve a neutral performance. The class_weight is set to "balanced," using the value of y to automatically adjust the weights inversely proportional to the class frequencies in the input data.

This model has several strengths and weaknesses. The strengths include: 1. The model has a very high overall accuracy, indicating that it can correctly predict both positive and negative instances in the majority of cases. 2. The model has an excellent ability to correctly classify negative instances, which means it rarely misclassifies negative instances as positive. The weaknesses include: 1. The model has a poor ability to identify positive instances, meaning a significant portion of positive instances are misclassified as negative. 2. The model's precision is moderate, suggesting that a fair number of instances classified as positive are actually negative. 3. The F1 score, which represents the harmonic mean of precision and recall, indicates that the classifier's performance is not well-balanced between these two aspects. 4. The geometric mean of recall and specificity, G-mean, is not very high, suggesting that the classifier's performance in identifying both positive and negative instances is not equally strong. In summary, this classifier performs well in identifying negative instances, as indicated by the high specificity and accuracy. However, it struggles to identify positive instances, leading to a low recall and a poor balance between precision and recall, as shown by the low F1 score. The moderate G-mean also suggests that the classifier's performance is not equally strong across both positive and negative instances.

To improve this classifier, efforts should be made to enhance its ability to identify positive instances. This may involve using techniques such as oversampling the minority class, adjusting class weights, or experimenting with different algorithms or model architectures.

The time complexity is $O(n^2p)$ to $O(n^3p)$, where n is the number of training samples and p is the number of features.

Random Forest (Decision tree)

Random Forest is an ensemble learning method that combines the predictions of multiple decision trees to create a more accurate and robust model. Decision trees are tree-like structures that map out all possible outcomes of a series of decisions based on a set of input features. Random Forest creates a large number of decision trees randomly and then aggregates their predictions to generate the final prediction. Each decision tree is trained on a randomly selected subset of the data and a random subset of the features, where only a random subset of the features is considered for splitting at each node of the decision tree. This randomness generates a diverse set of decision trees, which helps to enhance the accuracy of the model. Furthermore, averaging the predictions of multiple trees enables Random Forest to reduce the variance of the model. This methodology helps to mitigate the risk of overfitting, which occurs when a model is too complex and fits the training data too closely, leading to poor performance on new, unseen data.

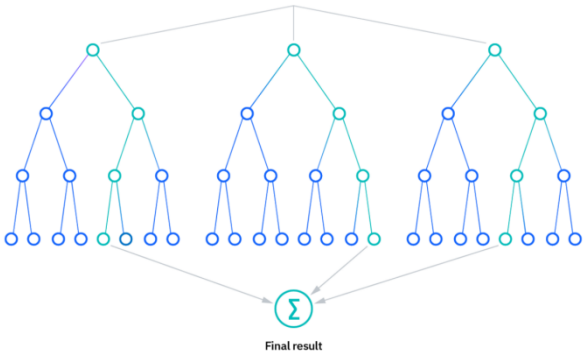


Fig. The illustration of random forest

The common element in all of these procedures is that for the k th tree, a randomo vector θ_k is generated, independent of the past random vector; and a tree is grown using the training set. After a large number of trees is generated, the vote for the most popular class. A random forest is a classifier consisting of a collection of tree-structured classifier $\{h(x, \theta_k), k-1, \ldots\}$, where the $\{(\theta_k)\}$ are independent identically distributed random vectors and each tree cats a unit vote for the most popular class at input x.[12]

For this assignment, *sklearn.ensemble.RandomForestClassifier* is the module used to realize the diagnosis the anomaly.

The main parameters and descriptions are given as below:

- n_estimators: the number of trees, and the parameter decides the accuracy in some extend, however, there is a trade-off between the velocity and the accuracy.
- Max_fearture: the max number of feature that every tree could have in the random forest. This could be set as auto, sqrt, log2, and none. If this is none, the max_fearture equal to n_fearture. And feartures exceeding max will be discarded.
- Random_state: The seed of random number.

The confusion matrix is shown below:

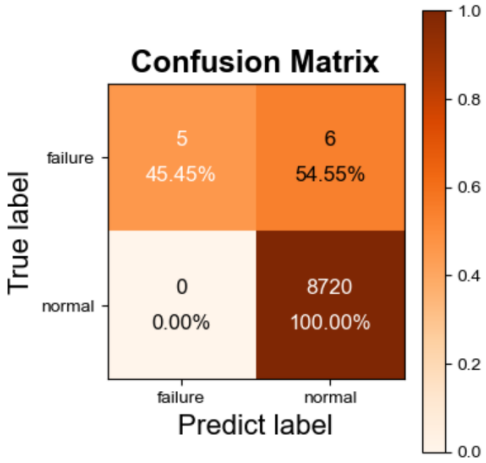


Fig. the confusion matrix

Accuracy	Gmean	Recall	Specificity	Precision	F1
0.999	0.674	0.455	1.000	1.000	0.625

Thus, in the experiment, we could see that the result could be improved further, according to analyzing the parameter and here are some ways to improve the performance of a Random Forest Classifier:

- Increase the number of trees: A larger forest with more trees can improve the accuracy of the classifier. And this could be realized by adjust the value of n_estimators.
- Tune the hyperparameters: such as the number of features to consider at each split and the minimum number of samples required to split a node. Tuning these hyperparameters can significantly impact the performance of the classifier. Grid search or randomized search can be used to find the optimal hyperparameters.
- Feature selection: Feature selection is the process of selecting a subset of relevant features to use in the classifier. By reducing the number of features, the classifier can become more efficient and less prone to overfitting. Random Forest has a built-in feature importance measure that can be used to identify the most important features.
- Ensemble pruning: Ensemble pruning is the process of removing weak or redundant models from the ensemble. This can help to reduce the complexity and improve the generalization of the model. And it could be realized by adjust the parameter max_fearture.

Overall, the performance of a Random Forest Classifier can be improved by carefully tuning the hyperparameters, selecting relevant features, preprocessing the data, and using ensembling or stacking techniques.

The total complexity of the training phase is O(nmlog(m)), where n is the number of training instances and m is the number of features.

Logistic regression

Logistic Regression is a discriminative supervised classifier, it models the posterior $P(y|x)$ directly, estimates the parameters of the model from the training data, and then computes $P(y|x)$, where feature vector $x \in X$ with n features and label $y \in Y$.

Logistic Regression assumes a parametric form for the distribution $P(y|x)$, then directly estimates its parameters from the training data. In this project we have binary classification, i.e., y ∈ {normal, failed}. Assume that $\{normal, failed\} = \{0,1\}$.

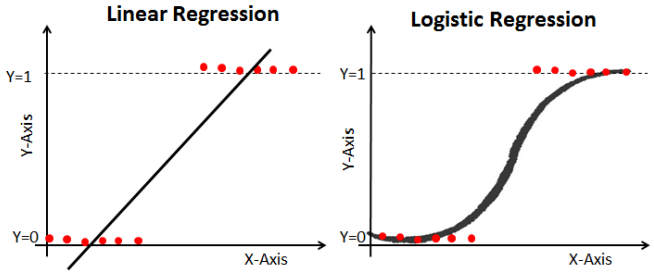
In the binary classification setting with $p(x) = P(y = 1|x)$, the parametric model assumed by Logistic Regression is given by the following probabilities:

$$P(y=1\,|\,x) = p(x) = \frac{\exp(w_0 + \sum_{i=1}^{n} w_i x_i)}{1+\exp(w_0 + \sum_{i=0}^{n} w_i x_i)},\ P(y=0\,|\,x) = 1 - p(x) = \frac{1}{1+\exp(w_0 + \sum_{i=0}^{n} w_i x_i)},$$

Taking the ratio of the above 2 equation, and taking the logarithm of both sides, we have:

$$\log \frac{p(x)}{1-p(x)} = w_0 + \sum_{i=0}^{n} w_i x_i.$$

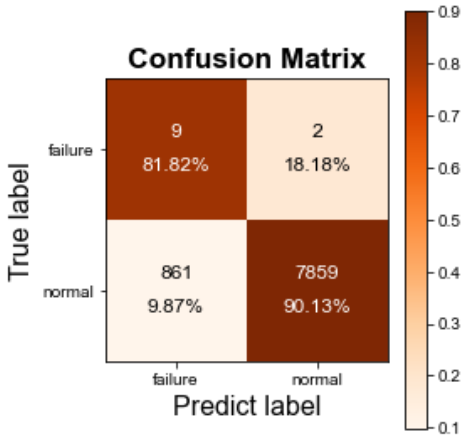
From the above equation, we see that Logistic Regression has a logit that is linear in x. The comparison between Linear Regression and Logistic Regression is shown in the following figure.



Parameter tuning[2]: *sklearn.linear_model.LogisticRegression* is the module used to implement logistic regression in this project. The parameters and descriptions are given in the following table.

Parameter	***Description***	***Value (in this project)***
penalty	This parameter is used to specify the norm (L1 or L2) used in penalization (regularization).	'L1'
tol	It represents the tolerance for stopping criteria.	1e-6
C	It represents the inverse of regularization strength.	0.5
intercept_scaling	It specifies as a constant (bias or intercept) should be added to the decision function.	50
class_weight	It represents the weights associated with classes. The "balanced" mode will use the values of y to automatically adjust weights.	'balanced'
random_state	This parameter represents the seed of the pseudo random number generated which is used while shuffling the data. Followings are the options	5
solver	Algorithm to use in the optimization problem. 'liblinear' is a good choice for small datasets. It also handles L1 penalty.	'liblinear'
max_iter	Maximum number of iterations taken for the solvers to converge.	2000

The performance of Logistic Regression in anomaly detection and prediction task is shown in the following confusion matrix. The accuracy of LR is 0.901, G-Mean of LR is 0.859, recall of LR is 0.010, specificity of LR is 1.000, precision of LR is 0.818, and F1 of LR is 0.020.



The Logistic Regression is efficient and easily interpretable, but it performs worse than SVM and Random Forest since it uses the "logistic model" (sigmold function) to model the probability of a certain class. The decision boundary is a log-like curve, making it much more suitable for linearly separable data. We can't solve the non-linear problem with the logistic regression that is why it requires a transformation of non-linear features. Also, Logistic regression will not perform well with independent variables that are not correlated to the target variable and are very similar or correlated to each other. And Logistic regression is not able to handle a large number of categorical features/variables. It is vulnerable to overfitting.

The time complexity [3][4] is $O(np)$, where n is the number of training samples and p is the number of features. The runtime complexity $O(p)$, and the space complexity while training is $O(np + n + p)$.

Comparison & Conclusion:

With the metrics test above, the comparison table of the metric for the classifiers shows below:

Classifier	Accuracy	G-mean	Recall	Specificity	Precision	F-score
MLE Gaussian	0.915	0.816	0.727	0.915	0.011	0.021
Naïve Bayesian	0.940	0.827	0.727	0.94	0.015	0.030
SVM	0.998	0.674	0.455	0.999	0.385	0.417
Random forest	0.999	0.674	0.455	1.000	1.000	0.625
Logistic regression	0.901	0.859	0.818	0.901	0.010	0.020

Based on the result, he performance of the different classifiers for the anomaly detection and prediction task can be compared as follows:

1. SVM and Random Forest classifiers have the highest accuracy with 0.998 and 0.999 respectively. Naive Bayes also has a high accuracy of 0.940. ML Gaussian and Logistic Regression have lower accuracy values of 0.915 and 0.901 respectively.
2. G-Mean is the metric which is a more appropriate metric for imbalanced datasets. Logistic Regression has the highest G-Mean value of 0.859. Gaussian and Naive Bayes have similar

G-Mean values of 0.816 and 0.827 respectively. SVM and Random Forest have the lowest G-Mean values of 0.674. Thus, for the result, the Logistic Regression may be the most suitable model for some imbalance dataset.

3. The recall metric measures the proportion of actual positive instances that are correctly identified by the classifier. The precision metric measures the proportion of positive predictions that are correctly identified by the classifier. This two metrics are essential for some medical diagnosis and fraud detection, thus comprehensively considering, the Naïve Bayesian perform best according to those two.
4. According to the metric F-score: Random Forest has the highest F-Score value of 0.625. Naive Bayes has the second-highest F-Score value of 0.030. MLE Gaussian and Logistic Regression have similar F-Score values ranging from 0.021 to 0.020.SVM has the lowest F-Score value of 0.417.

The Random Forest classifier performs best. Compared to other classifiers, it achieves the highest score in 4 metrics of 6. Especially, the specificity and precision of it achieves 100%. This is because it has an inherent non-linear decision boundary with can cooperate well with the non-linear features.

The SVM and Logistic Regression have second-ranked performances. They both obtain the non-linear decision boundary by some high order formulas. SVM does have a weakness when it comes to classifying negative cases, particularly when the features for negative cases in the dataset are not distinct or obvious. In such situations, SVM may struggle to identify and accurately classify the negative cases, which could lead to suboptimal performance in certain scenarios. Logistic Regression performs worse than SVM and Random Forest since it uses the "logistic model" and the decision boundary is a log-like curve, making it much more suitable for linearly separable data. Also, that is why it requires a transformation of non-linear features.

The MLE Gaussian and NB do worst in this task because Both of these classifiers are based on specific data distribution assumptions. The MLE Gaussian assumes the data follows a Gaussian distribution, while the NB classifier assumes conditional independence among features. If the actual distribution of the data contradicts these assumptions, the performance of the classifiers may be affected. Besides, NB and MLE Gaussian classifiers are relatively simple, and therefore may not be suitable for handling features with complex relationships. What's more, MLE Gaussian and NB classifiers are sensitive to outliers and noise.

Reference

[1] Jakkula, V. (2006). Tutorial on support vector machine (svm). *School of EECS, Washington State University*, *37*(2.5), 3.

[2] Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830

[3] Lim, TS., Loh, WY. & Shih, YS. (2000). A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms. Machine Learning 40, 203–228.

[4] Mrisho, Zubeda & Ndibwile, Jema David & Sam, Anael. (2021). Low Time Complexity Model for Email Spam Detection using Logistic Regression. International Journal of Advanced Computer Science and Applications. 12. 10.14569/IJACSA.2021.0121215.

[5] Di Zenzo, Silvano, et al. "Gaussian maximum likelihood and contextual classification algorithms for multicrop classification." IEEE Transactions on Geoscience and Remote Sensing 6 (1987): 805-814.

[6] Bishop, C. M. (2006). Pattern recognition and machine learning (Vol. 4).

[7] Wu, Xindong, et al. "Top 10 algorithms in data mining." Knowledge and information systems 14 (2008): 1-37.

[8] Jiang, Liangxiao, et al. "Naive Bayes text classifiers: a locally weighted learning approach." Journal of Experimental & Theoretical Artificial Intelligence 25.2 (2013): 273-286.

[9] Feng, Weimiao, et al. "A support vector machine based naive Bayes algorithm for spam filtering." 2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC). IEEE, 2016.

[10] Hue, Carine, Marc Boullé, and Vincent Lemaire. "Online learning of a weighted selective naive Bayes classifier with non-convex optimization." Advances in Knowledge Discovery and Management: Volume 6 (2017): 3-17.

[11] Ng, Andrew, and Michael Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." Advances in neural information processing systems 14 (2001).

[12] Chickering, Max, David Heckerman, and Chris Meek. "Large-sample learning of Bayesian networks is NP-hard." Journal of Machine Learning Research 5 (2004): 1287-1330.

[13] John, George H., and Pat Langley. "Estimating continuous distributions in Bayesian classifiers." arXiv preprint arXiv:1302.4964 (2013).