Capstone Project - Iceberg Classification

Machine Learning Engineer Nanodegree

YuanFu Yang, Taiwan Semiconductor Manufacturing Company(tsmc), No. 8, Li-Hsin Rd. 6, Hsinchu Science Park,Hsin-Chu 300, Taiwan, R.O.C. email: yfyangd@tsmc.com

Abstract

During the winter and spring, drifting icebergs in the North Atlantic ocean present threats to navigation and activities in areas such as offshore of the east coast of Canada. It poses a risk to people's lives and oil and gas equipment, as well as present challenges to shipping and other activities. The Current method for monitoring iceberg conditions is aerial reconnaissance, supplemented by platform and vessel-based monitoring. However, in remote areas with particularly harsh weather, these methods are not feasible, and the only viable monitoring option is via satellite. The data in this paper was detected by synthetic aperture radar (SAR) from Sentinel-1,which is a space mission funded by the European Union and carried out by the ESA within the Copernicus Programme. After detection, additional processing is needed to distinguish between ships and icebergs. The discrimination between the two classes is carried out through feature extraction and target classification steps.

This paper proposes the application of Convolutional Neural Networks (CNN) for ship-iceberg discrimination in high resolution Sentinel-1 StripMap images. The data was collected from Kaggle - iceberg classifier challenge. The CNN model is compared with a Random Decision Forests/Support Vector Machine (SVM)/Adaptive Boosting/eXtreme Gradient Boosting, and the final results indicate a superior classification performance of the proposed method.

1. Introduction

Sentinel-1 is a space mission funded by the European Union and carried out by the ESA within the Copernicus Programme, consisting of a constellation of two satellites. The payload of Sentinel-1 is a Synthetic Aperture Radar (SAR) in C band that provides continuous imagery (day, night and all weather). Sentinet-1 sat is at about 680 Km above earth. It can send pulses of signals at a particular angle of incidence and then recoding it back with following instruments: (1) A single C band SAR with its electronics (SES); (2) A SDRAM-based mass memory (DSHA), with an active data storage capacity of about 1 443 Gbit (168 GiB), receiving data streams from SAR SES over two independent links gathering SAR_H and SAR_V polarization, with a variable data rate up to 640 Mbit/s on each link, and providing 520 Mbit/s X-band fixed user data downlink capability over two independent channels towards Ground.

Unlike optical images, SAR images are formed by coherent interaction of the transmitted microwave with the targets. Hence, it suffers from the effects of speckle noise which arises from coherent summation of the signals scattered from ground scatterers distributed randomly within each pixel. A radar image appears more of noisy than an optical image. The speckle noise is sometimes suppressed by applying a speckle removal filter on the digital image before display and further analysis. In this paper, I tried to use MaxAbsScaler to normalizing image and reduce noise impact. Then I used Singular Value Decomposition (SVD) to dimensionally reduce the images to just a few features.

Interpreting a radar image is not a straightforward task. It very often requires some familiarity with the ground conditions of the areas imaged. As a useful rule of thumb, the higher the backscattered intensity, the rougher is the surface being imaged. Basically those reflected signals are called backscatter. The data we have been given is backscatter coefficient which is the conventional form of backscatter coefficient given by [7]:

$$\boldsymbol{\sigma o(dB) = \beta o(dB) + 10\log10 \left[\sin(ip) / \sin(ic) \right]}$$

where:

ip: angle of incidence for a particular pixel

ic: angle of incidence for center of the image

We have been given σo directly in the data. Now coming to the features of σo. Basically σo varies with the surface on which the signal is scattered from. For example, for a particular angle of incidence, it varies like as below table. As you can see, the HH component varies a lot but HV doesn't. We don't have the data for scatter from ship, but being a metal object, it should vary differently as compared to ice object.

Table1: the σo of backscatter in each target

Type	Water	Settlements	Agriculture	Barren
HH	-27.001	2.70252	-12.7952	-17.257
HV	-28.035	-20.2665	-21.4471	-20.019

Sentinet-1 only transmits pings in H polarization, and not in V polarization. Those H-pings gets scattered, objects change their polarization and returns as a mix of H and V. Since Sentinel has only H-transmitter, return signals are of the form of HH and HV only. Because Sentinel don't have V-ping transmitter, They don't given signal as VV. Now coming to features, for the purpose of machine learning, I am extracting all two bands and taking avg of them as 3rd channel to create a 3-channel RGB equivalent for CNN (Convolutional Neural Networks). In previous studies [1], intensity and polarimetric parameters are used as features to a Support Vector Machine (SVM) classifier, in order to discriminate ships from icebergs in simulated, dual polarized, medium resolution SAR data. Recently, Convolutional Neural Networks (CNN) have been successfully adopted for demanding SAR classification tasks, as in [2,3]. Both SVM and CNN will be used in this study for iceberg classification.

For model performance calculation, I use three measurements: precision, recall, and F-score. Precision is the number of correct positive results divided by the number of all positive results returned by the classifier, and recall is the number of correct positive results divided by the number of all relevant samples. Both precision and recall are based on an understanding and measure of relevance. Often, there is an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. Therefore, precision and recall scores are not discussed in isolation. A measure that combines precision and recall is the harmonic mean of precision and recall, the F-score:

$$\mathrm{F} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

This measure is approximately the average of the two when they are close, and is more generally the harmonic mean, which, for the case of two numbers, coincides with the square of the geometric mean divided by the arithmetic mean.

It is a special case of the general Fβ measure (for non-negative real values of β):

$$\mathrm{F}\beta = (1 + \beta) \frac{\text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}$$

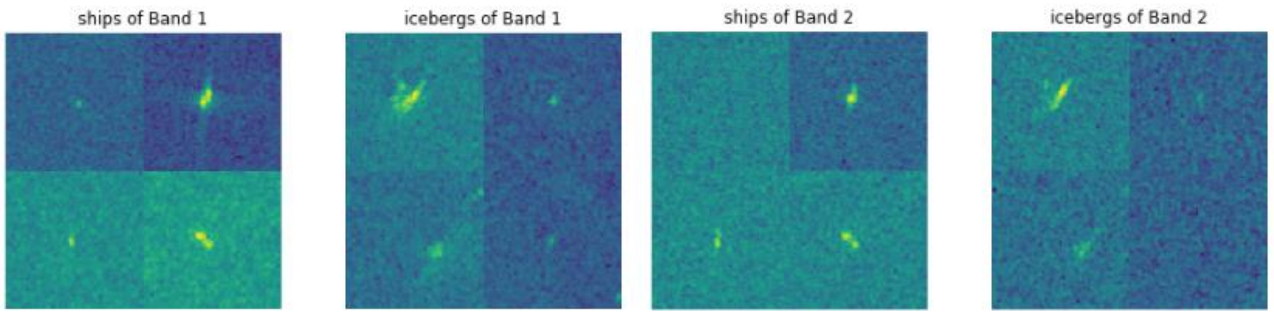
Two other commonly used F measures are the F2 measure, which weights recall higher than precision, and the F0.5 measure, which puts more emphasis on precision than recall. In this paper, we hope the predict model have more precision result, so the final result will be decided by F-score with β = 0.5.

2. Analysis

2.1 Data exploration

The datasets has been provided by the Centre for Cold Ocean Resource Engineering (C-CORE) on Kaggle (https://www.kaggle.com/c/statoil-iceberg-classifier-challenge/data) and presented in json format. It consists of a list of images and other information. In the training file, there are 1604 data with 1 label (is iceberg or not), and 4 inputs (HH/HV/Angle/image_id). The labels are provided by human experts and geographic knowledge on the target. The distribution of labels seems to balance (Is_iceberg: 753, not_iceberg: 851). The images are 75x75 images with two bands (HH/HV). I load the data and then combine the two bands and recombine them into a single image/tensor for review.

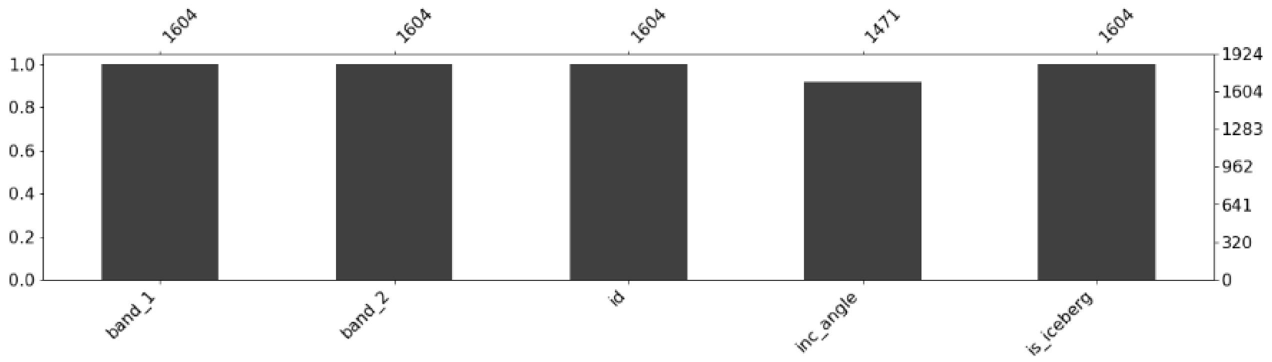
Figure-1: data exploration of two band



Based on the above image, the data look like pretty messier and it appears that the background is not really random noise but rather has some spatial correlations. If the background is dominated by things like waves rather than noise, then spatial correlations would clearly be expected. Besides, the ships seem to have a more regular structure than iceberg, with a pronounced skewness in the blobs for larger signals. Some of these blobs are not that high above noise, so it may be advantageous to first transform the images in some way to enhance the contrast between the signals and the background.

In the input of angle, there are some missing data here that need to be processed. I dropped 133 missing data before model training. As Figure-2, we can see that the data of angle before data processed is 1607. After missing data dropping, the data has been reduced to 1471. It is help for machine learning to avoid error message when model training.

Figure-2: the distribution of datasets



2.2 Evaluation Metrics

I use three index to calculate performance of my model: **precision**, **recall**, and **F-score**. The final result will be decided by **F-score** with β = 0.5.

2.3 Benchmark Model

For compare with final result, I choose " **Random Forest** " as my benchmark model. The score (**precision/recall/F-score**) of first run is **0.7465/0.6839/0.7331**. It seems like solvable because the performance is acceptable in the absence of data extraction.

3. Methodology

3.1 Solution Statement

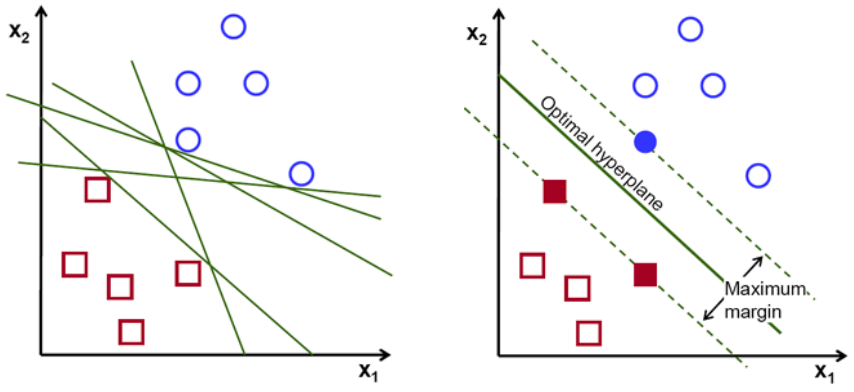
In this datasets, there are 2 types of predictive factors: imagical (HH/HV) and numerical (Angle) data. I will use 3 supervised learning methodology, **Support Vector Machine (SVM)**, **Adaptive Boosting(AdaBoost)**, and **eXtreme Gradient Boosting (XGboost)** with all factors to predict test data. Before doing this, some feature engineering should be done. Then I will use deep learning - **Convolutional Neural Networks (CNN)** to predict test data and compared with previous 3 supervised learning. In CNN, it only can be use imagical factors, so the “angle” will be dropped before model training. Furthermore, CNN model needs three input channels but we only have two (band_1 and band_2) in the datasets. I will take the mean of band_1 and band_2 to be used as the third channel.

3.2 Algorithm Introduction

In this section, I briefly introduce the algorithms, include of describing how algorithms work, how to train and predict data, what's the procedure behind the scenes. The kernel of these algorithms will be explained at next chapter.

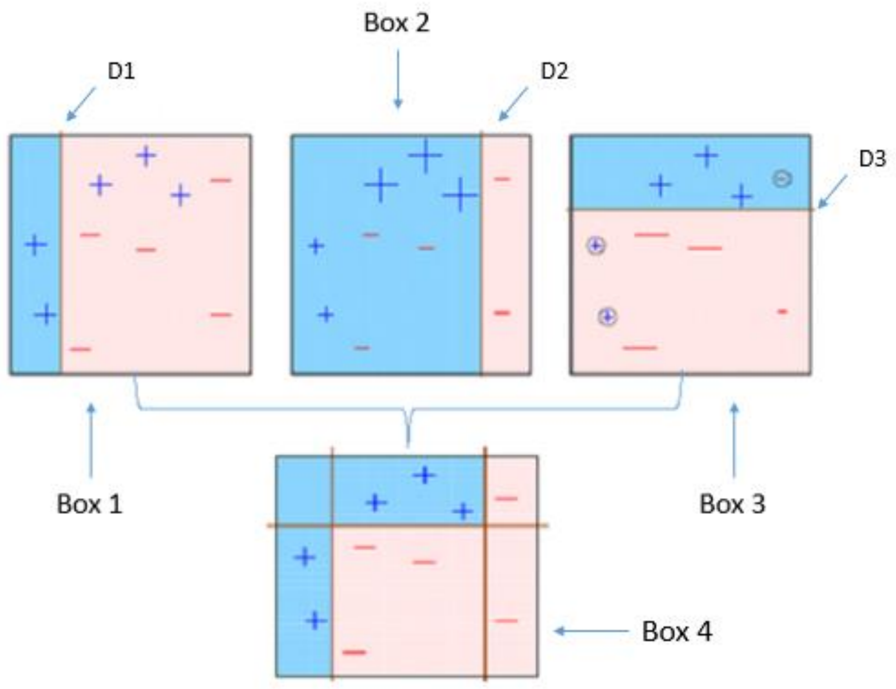
The first algorithm I choice is **Support Vector Machine (SVM)**, which is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples. In the below left picture you can see that there exists multiple lines that offer a solution to the problem. We can intuitively define a criterion to estimate the worth of the lines: A line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly. Therefore, our goal should be to find the line passing as far as possible from all points. Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of margin within SVM’s theory. As above right picture we can see that the optimal separating hyperplane maximizes the margin of the training data [13].

Figure-3: the operation of the SVM



The second algorithm what I choice is **Adaptive Boosting (AdaBoost)**. It fits a sequence of weak learners on different weighted training data. It starts by predicting original data set and gives equal weight to each observation. If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly. Being an iterative process, it continues to add learner(s) until a limit is reached in the number of models or accuracy. The below picture aptly explains Ada-boost. Let's understand it closely [14]:

Figure-4: the operation of the AdaBoost



Box 1: You can see that we have assigned equal weights to each data point and applied a decision stump to classify them as + (plus) or – (minus). The decision stump (D1) has generated vertical line at left side to classify the data points. We see that, this vertical line has incorrectly predicted three + (plus) as – (minus). In such case, we'll assign higher weights to these three + (plus) and apply another decision stump.

Box 2: Here, you can see that the size of three incorrectly predicted + (plus) is bigger as compared to rest of the data points. In this case, the second decision stump (D2) will try to predict them correctly. Now, a vertical line (D2) at right side of this box has classified three mis-classified + (plus) correctly. But again, it has caused mis-classification errors. This time with three -(minus). Again, we will assign higher weight to three – (minus) and apply another decision stump.

Box 3: Here, three – (minus) are given higher weights. A decision stump (D3) is applied to predict these mis-classified observation correctly. This time a horizontal line is generated to classify + (plus) and – (minus) based on higher weight of mis-classified observation.

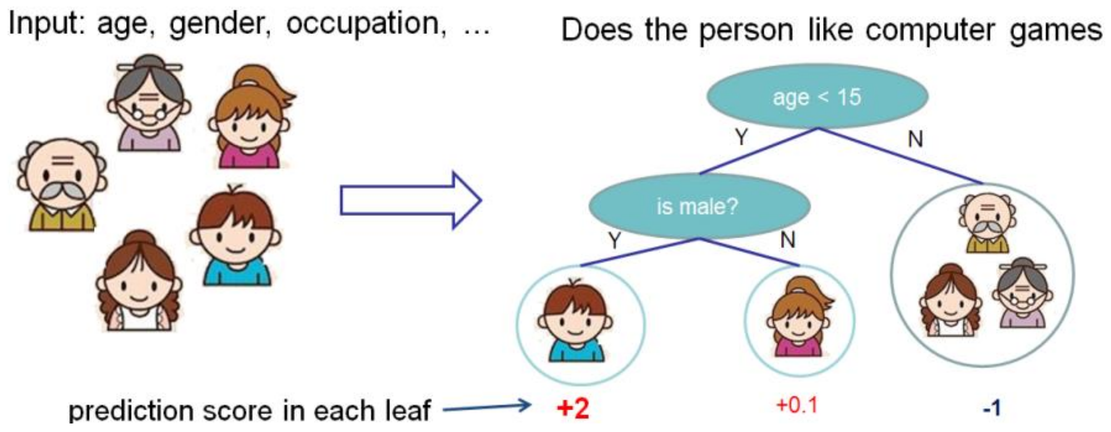
Box 4: Here, we have combined D1, D2 and D3 to form a strong prediction having complex rule as compared to individual weak learner. You can see that this algorithm has classified these observation quite well as compared to any of individual weak learner.

AdaBoost can be used in conjunction with many other types of learning algorithms to improve performance. Mostly, we use decision stamps with AdaBoost. But, we can use any machine learning algorithms as base learner if it accepts weight on training data set. We can use AdaBoost algorithms for both classification and regression problem. In this study, I use the Decision tree with AdaBoost for iceberg classification problem.

The third algorithm what I choice is **eXtreme Gradient Boosting (XGboost)**, which is designed for boosted tree algorithms. It has become a popular machine learning framework among data science practitioners, especially on Kaggle, which is a platform for data prediction competitions where researchers post their data and statisticians and data miners compete to produce the best models.

To begin with, let us first learn about the model of XGboost: tree ensembles. The tree ensemble model is a set of classification and regression trees (CART). Here's a simple example of a CART that classifies whether someone will like computer games [15,16].

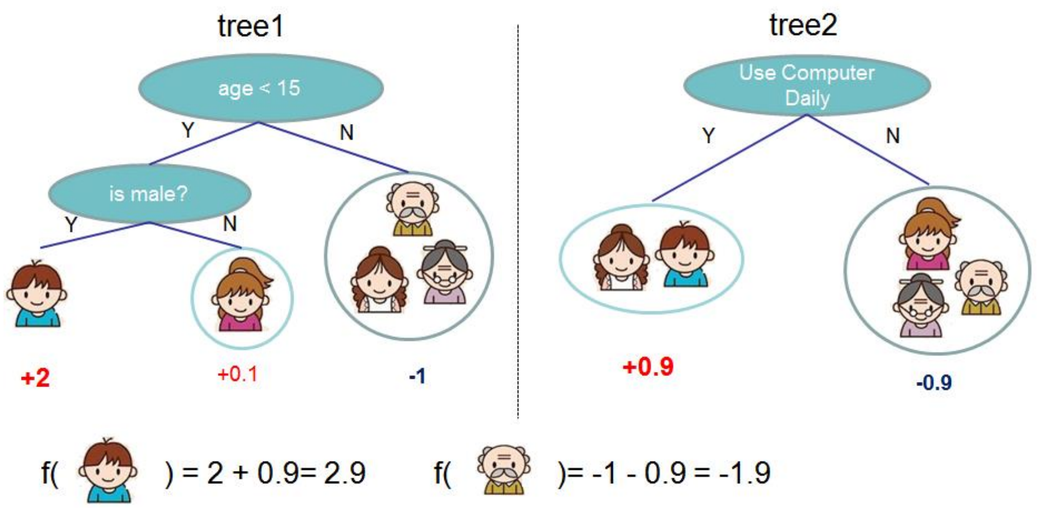
Figure-5: the example of ensemble model


Input: age, gender, occupation, …
Does the person like computer games
age < 15
Y
N
is male?
Y
N
prediction score in each leaf
+2
+0.1
-1

We classify the members of a family into different leaves, and assign them the score on the corresponding leaf. A CART is a bit different from decision trees, where the leaf only contains decision values. In CART, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification. This also makes the unified optimization step easier. Usually, a single tree is not strong enough to be used in practice. What is actually used is the so-called tree ensemble model, which sums the prediction of multiple trees together.

Here is an example of a tree ensemble of two trees. The prediction scores of each individual tree are summed up to get the final score.

Figure-6: the example of ensemble model



If you look at the example, an important fact is that the two trees try to complement each other. Mathematically, we can write our model in the form-1. Where K is the number of trees, f is a function in the functional space F, and F is the set of all possible CARTs. Therefore our objective to optimize can be written as form-2.

Form-1

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

Form-2

$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

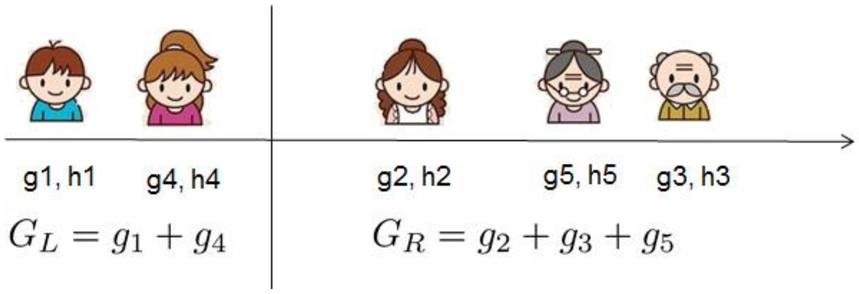
In terms of model, there are no different in Random forests and XGBoost. The difference is how we train them and measure the benefit. Ideally we would enumerate all possible trees and pick the best one. In practice this is intractable, so we will try to optimize one level of the tree at a time. Specifically we try to split a leaf into two leaves, and the score it gains is:

Form-3

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

This formula can be decomposed as (1) the score on the new left leaf (2) the score on the new right leaf (3) The score on the original leaf (4) regularization on the additional leaf. We can see an important fact here: if the gain is smaller than γ, we would do better not to add that branch. This is exactly the pruning techniques in tree based models! By using the principles of supervised learning, we can naturally come up with the reason these techniques work. For real valued data, we usually want to search for an optimal split. To efficiently do so, we place all the instances in sorted order, like the following picture. A left to right scan is sufficient to calculate the structure score of all possible split solutions, and we can find the best split efficiently. That is how XGBoost work.

Figure-7: the example of XGBoost



Finally, I choice **Convolutional Neural Networks (CNN)** for my last algorithm. Before CNN, let me introduce the **Artificial Neural Networks (ANN)** first. ANN is computing system vaguely inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units or nodes called artificial neurons (a simplified version of biological neurons in an animal brain). Each connection (a simplified version of a synapse) between artificial neurons can transmit a signal from one to another.

The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it. In common ANN implementation, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is calculated by a non-linear function of the sum of its inputs. Artificial neurons and connections typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that only if the aggregate signal crosses that threshold is the signal sent. The below picture (Fique-6) presents how different is between six activation functions of threshold.

Figure-8: Basic ANN Structure

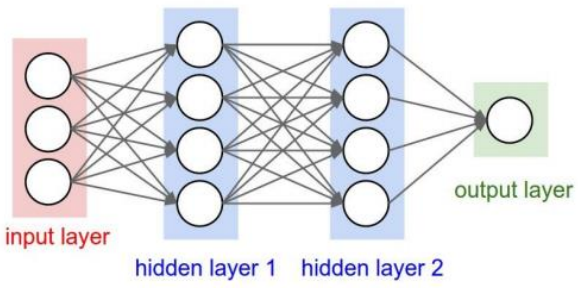
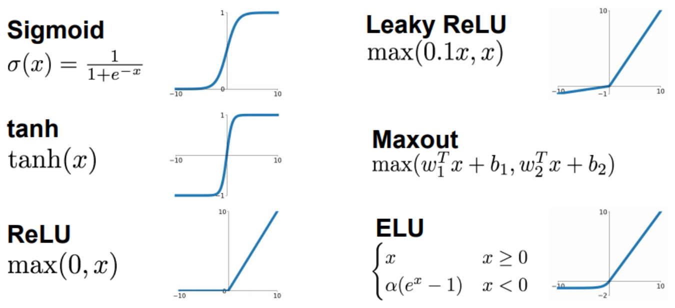


Figure-9: Activation Functions



Typically, artificial neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times. (as Fiqure-5)

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers. Description of the process as a convolution in neural networks is by convention. (1) Convolutional layers: It applies a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field. (2) Pooling layer: Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Another example is average pooling, which uses the average value from each of a cluster of neurons at the prior layer. (3) Fully connected layer: It connects every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network.

3.3 Experiment Design

(1) Data Pre-process: Dropping the missing data, normalizing numerical features, and transforming skewed continuous features.
(2) Initial Model Evaluation: Before data engineering, I will properly evaluate the performance of each model (SVM, Adaboost, XGBoost, CNN).
(3) Feature Extraction: In CNN, the feature can be extracted in convolution process. In supervised learning (SVM, Adaboost, XGBoost), the feature will be extracted by statistics.
(4) Model Tuning: To improve performance, I will use GridSearchCV by Scikit-learn to find the optimization parameters in each supervised learning model.
(5) Data Augmentation: I added more data to train set by simply way including horizontally and vertically flipped data.
Finally, based on the above process, I will evaluate the performance of each model and choose the best as the result of this study.

3.4 Challenges

The most challenge in the process is model tuning. There are about 10~20 parameters in each algorithm, with different type of input, such as float, string, integer, array, etc. First, we must decide the key parameters of algorithm due to we can't control all parameters in each training. Therefore, the study of the algorithm is necessary. It can help us to select key parameters.

Second, for each key parameter, we must decide the experimental value, it is the most difficult of all part. For example, the gamma of SVM, If gamma is too large, it will be able to prevent overfitting. When gamma is very small, the model will not generalize well. By sklearn-GridSearchCV, we can easily estimator the parameters and optimized by cross-validated grid-search over a parameter grid.

4. Supervised Learning Development

4.1 Data Pre-process

I create a numpy matrix with different matrices which includes all features, which enables us to build different combinations of input variables for the model. Then I use MaxAbsScaler to scale each variable in the range of [-1,+1] which is centered by 0, in order to normalizing numerical features

4.2 Initial Model Evaluation

Pre-train model with 3 supervised learning - SVM/Adaptive Boosting/eXtreme Gradient Boosting. Based on F-Score, we can see that Adabooster & XGBooster have a significant improvement after data preprocess compared to benchmark model. But the F-Score of SVM is small than benchmark model (0.6514<0.7331), due to precision is too low (0.6096). In the next step, the performance of SVM will be better after data extraction & model turning.

Table-2: Initial Model Performance

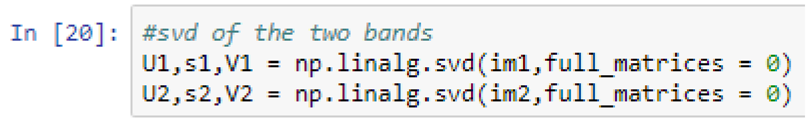
KPI	Benchmark Model	SVM	AdaBooster	XGBooster
Precision	0.7465	0.6096	0.7558	0.7676
Recall	0.6839	0.8968	0.8387	0.9161
F-Score	0.7331	0.6514	0.7711	0.7933

4.3 Feature Extraction

Although SAR provides a massive amount of information, much of the insight might be redundant or useless (noise). Thus, it is significant to recognize the most informative features of data. This will help the analysis of the data by removing the consequences of high dimensionality, in addition of obtaining other advantages of lower dimensional data such as lower computational cost and a less complex model Singular Value Decomposition (SVD) has been successfully used to feature extraction in various area, such as physiological signals [4], Motor faults classification [5]. In this paper, I use SVD to extract the features of training data Band-1 and Band-2, and adopt the SVM/AdaBooster/XGBooster to classify the different image states. By the library of numpy, we can extract orthogonal matrix (U), diagonal matrix (s), unitary matrix (V) from original image:

U, s, V = np.linalg.svd(img, full_matrices = 0)

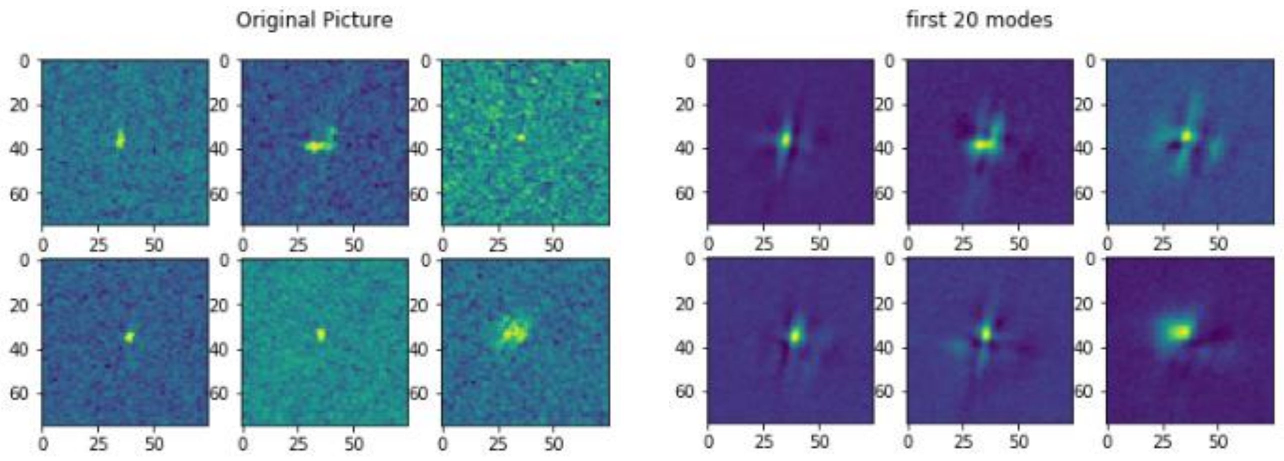
Fill up the training sets with the feature from SVD:



```
In [20]: #svd of the two bands
         U1,s1,V1 = np.linalg.svd(im1,full_matrices = 0)
         U2,s2,V2 = np.linalg.svd(im2,full_matrices = 0)
```

The below picture of the left one is the original image. We can see that there are many noises around the target. After feature extraction by SVD (the below picture of the right one), we can see that the target is much clear and the noise (waves) has almost been removed.

Figure-10: image comparison with feature extraction



Then we used the new training sets and trained the model of three supervised learning. XGBooster has the best performance after feature extraction, the F-Score has been improve to 0.8248 from 0.7933. Adabooster is the second one, the F-Score has been improve to 0.7860 from 0.7711. Besides, SVM does not perform well, it is because that the default parameters of SVM are not suitable here. We will change that in the next section.

Table-3: performance comparison after feature extraction

Methodology	KPI	Benchmark Model	Data Preprocess	Feature Extraction
SVM	Precision	0.7465	0.6096	0.5254
	Recall	0.6839	0.8968	1.0000
	F-Score	**0.7331**	**0.6514**	**0.5805**
AdaBooster	Precision	0.7465	0.7558	0.7738
	Recall	0.6839	0.8387	0.8387
	F-Score	**0.7331**	**0.7711**	**0.7860**
XGBooster	Precision	0.7465	0.7676	0.8011
	Recall	0.6839	0.9161	0.8355
	F-Score	**0.7331**	**0.7933**	**0.8248**

4.4 Model Tuning

In this section, I use GridSearchCV to optimize the classifier by cross-validation. The performance of the selected hyper-parameters and trained model measured on a data set after data extraction. Before model tuning, Let me to study the kernel of below supervised learning model first:

- **SVM – C_range (defaut = 1.0)**

It is the penalty parameter of the error term, critical here, as in any regularization scheme, that a proper value is chosen for C, the penalty factor. If it is too large, we have a high penalty for nonseparable points and we may store many support vectors and overfit. If it is too small, we may have underfitting [6].

- **SVM – gamma (default = 1/n_features)**

This Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting. When gamma is very small, the model is too constrained and cannot capture the complexity or "shape" of the data [8].

- **AdaBooster – n_estimators (default = 50)**

The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early. It controls the number of weak learners. In general, if the n_estimators is too small, it will be easy to underfit. if the n_estimators is too large, it will be easy to overfit. Generally, a moderate value is chosen. The default is 50. Actually, we often consider the n_estimators together with the learning_rate parameter [8].

- **AdaBooster – learning (default = 1)**

Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators [9].

- **XGBooster – learning (default = 0.3)**

Step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and learning rate actually shrinks the feature weights to make the boosting process more conservative [10].

- **XGBooster –max_depth (default = 0.6)**

It is the maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting. 0 indicates no limit, limit is required for depth-wise grow policy [10].

- **XGBooster –gamma (default = 0)**

It is the minimum loss reduction required to make a further partition on a leaf node of the tree. the larger, the more conservative the algorithm will be [10].

Then, I used GridSearchCV to optimize the 3 classifier model. GridSearchCV has been successfully used in many fields [11,12]. The parameters design of this study is as below table:

Table-4: the design of model tuning

Classifier	Parameter	Default	Range
SVM	C_range	1	0.1, 1, 10, 50, 100
	gamma	Auto (1/n_features)	0, 5, 10, 20, 50
AdaBooster	n_estimators	50	10, 50, 100, 200, 1000
	learning_rate	1	0.2, 0,4, 0.6, 0.8, 1.0
XGBooster	learning_rate	0.3	0.2, 0,4, 0.6, 0.8, 1.0
	max_depth	6	6, 10, 20, 50, 100
	gamma	0	0, 5, 10, 20, 50

4.5 Supervised Learning Summary

It is exciting that see the performance is pretty well after data preprocess, feature extraciton, and model tuning. No matter what classifier, It has been improve so much. The final F-score of SVM / Adabooster / XGBooster is 0.8424 / 0.8105 / 0.8501 with 29% / 5% / 7% improvement. The best classifier is **XGBooster** (**F-score: 0.8501**) of all supervised learning in my test. In the next chapter, I developed convolutional neural networks (CNN) to see how good is in computer vision problem.

Table-5: the performance summary

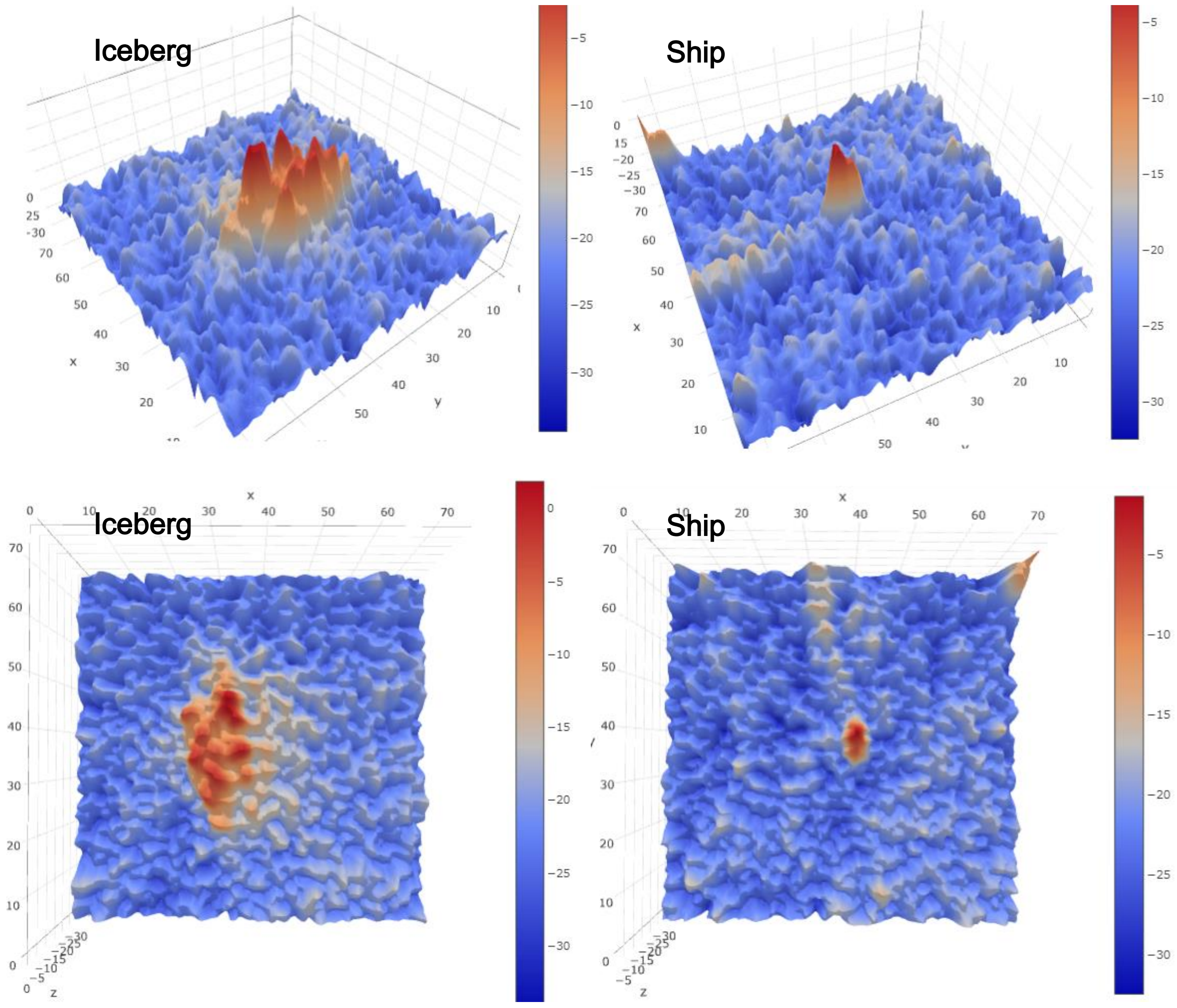
Methodology	KPI	Benchmark Model	Data Preprocess	Feature Extraction	Optimized Model
SVM	Precision	0.7465	0.6096	0.5254	0.8284
	Recall	0.6839	0.8968	1.0000	0.9032
	F-Score	0.7331	0.6514	0.5805	**0.8424**
AdaBooster	Precision	0.7465	0.7558	0.7738	0.7953
	Recall	0.6839	0.8387	0.8387	0.8774
	F-Score	0.7331	0.7711	0.7860	**0.8105**
XGBooster	Precision	0.7465	0.7676	0.8011	0.8324
	Recall	0.6839	0.9161	0.8355	0.9290
	F-Score	0.7331	0.7933	0.8248	**0.8501**

5. Deep Learning Development (CNN)

5.1 Signal Review

The picture as below illustrate the typical SAR signature of an iceberg and a ship(Fiqure-4) in an C Band high resolution image. The high resolution of Sentinet -1 StripMap mode allows the detection of structural components from both floating structures. Even though ships and icebergs can have similar intensity and size values, their structures and shapes typically follow different patterns. The CNN is designed to learn a set of features from the input image in a supervised training process, in a way to capture the differences between the two observed classes, include of shapes and structures. That is the advantage that traditional supervised learning doesn't have.

Fiqure-11: the image of C Band – Iceberg & Ship



5.2 Data Pre-process

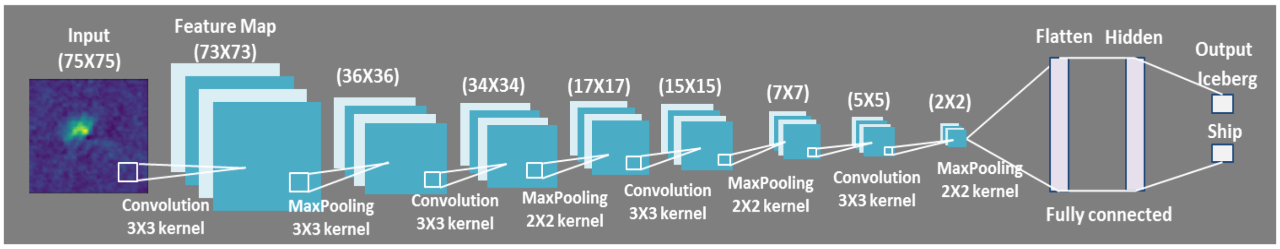
I user Max-min scaler to nomalize data: **(x-mean)/(Max-Min)**. Furthermore, CNN model needs three input channels but we only have two (band_1 and band_2) in the datasets. I will take the mean of band_1 and band_2 to be used as the third channel.

```
In [27]: def get_scaled_imgs(df):
             imgs = []

             for i, row in df.iterrows():
                 band_1 = np.array(row['band_1']).reshape(75, 75)
                 band_2 = np.array(row['band_2']).reshape(75, 75)
                 band_3 = (band_1 + band_2)/2
                 # Normalize
                 a = (band_1 - band_1.mean()) / (band_1.max() - band_1.min())
                 b = (band_2 - band_2.mean()) / (band_2.max() - band_2.min())
                 c = (band_3 - band_3.mean()) / (band_3.max() - band_3.min())
                 imgs.append(np.dstack((a, b, c)))

             return np.array(imgs)
```

5.3 Initial Model Evaluation

The CNN is designed to learn a set of features from the input image in a training process, in a way to capture the differences between the two observed classes. The architecture is presented in Figure 5.

Fiqure-12: Convolution Neural Network architecture



```
# CNN 1
model.add(Conv2D(64, kernel_size=(3, 3),activation='relu', input_shape=(75, 75, 3)))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(Dropout(0.2))
# CNN 2
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu' ))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.2))
# CNN 3
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.2))
#CNN 4
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.2))
# flatten the data for the dense layers
model.add(Flatten())
#Dense 1
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
#Dense 2
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
# Output
model.add(Dense(1, activation="sigmoid"))
```

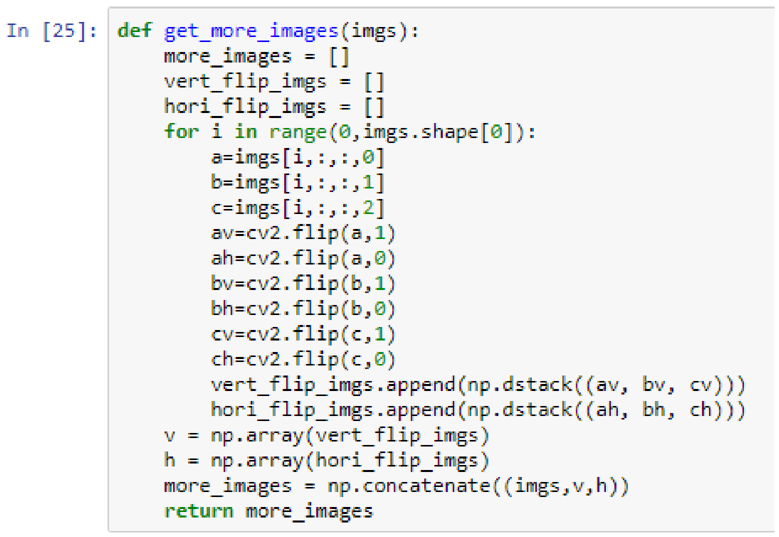
CNN used in our experiments is composed of four convolutional layers (Convolution + Pooling), one flatten layer, two dense layer, and one sigmoid layer to generate the classification probabilities in the final layer. After training of 20 epochs, the F-score of test data is **0.9172**. That is better than below supervised model. In the next section, I used data augmentation to increase datasets and expect to have better performance than initial model.

Table-6: CNN training history and test performance

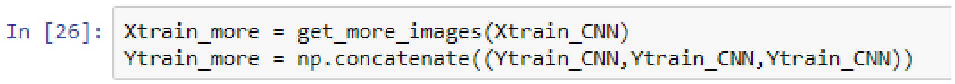
Methodology	KPI	Benchmark Model (RandomForest)	Initial Model of CNN
CNN	Precision	0.7465	0.9708
	Recall	0.6839	0.8692
	F-Score	0.7331	**0.9172**

5.4 Data Augment

I added more data to train set by OpenCV, including horizontally and vertically flipped data. Then, I combine them with original image in one datasets for next training.



```
In [25]: def get_more_images(imgs):
             more_images = []
             vert_flip_imgs = []
             hori_flip_imgs = []
             for i in range(0,imgs.shape[0]):
                 a=imgs[i,:,:,0]
                 b=imgs[i,:,:,1]
                 c=imgs[i,:,:,2]
                 av=cv2.flip(a,1)
                 ah=cv2.flip(a,0)
                 bv=cv2.flip(b,1)
                 bh=cv2.flip(b,0)
                 cv=cv2.flip(c,1)
                 ch=cv2.flip(c,0)
                 vert_flip_imgs.append(np.dstack((av, bv, cv)))
                 hori_flip_imgs.append(np.dstack((ah, bh, ch)))
             v = np.array(vert_flip_imgs)
             h = np.array(hori_flip_imgs)
             more_images = np.concatenate((imgs,v,h))
             return more_images
```



```
In [26]: Xtrain_more = get_more_images(Xtrain_CNN)
         Ytrain_more = np.concatenate((Ytrain_CNN,Ytrain_CNN,Ytrain_CNN))
```

5.5 Deep Learning Summary

After data augmentation, we can see that the precision has been decreased from 0.9708 to 0.9235, but the Recall has good improvement from 0.8692 to 0.9375. Finally, the F-scorer has been improved 1.4% (0.9172 to **0.9304**).

Table-7: the performance summary of CNN

KPI	Benchmark Model	Initial CNN Model	Data Augmentation
Precision	0.7465	0.9708	0.9235
Recall	0.6839	0.8692	0.9375
F-Score	0.7331	0.9172	**0.9304**

6. Experiment Result

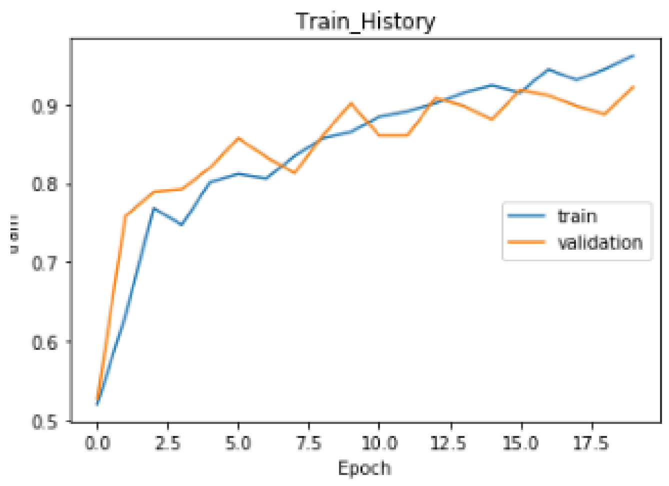
Compare with 4 methodologies, beyond all doubt, the CNN model is the best classifier in the Iceberg classification problem, even that the CNN performance in initial model (0.9172) is good than all of final supervised learning model (around 0.81~0.85). The final test show that the F-score of CNN is **0.9304**, this is higher than what I expected.

Table-8: the performance summary of all model

Methodology	KPI	Benchmark Model	Data Preprocess	Feature Extraction	Optimized Model/ Data Augmentation
SVM	Precision	0.7465	0.6096	0.5254	0.8284
	Recall	0.6839	0.8968	1.0000	0.9032
	F-Score	0.7331	0.6514	0.5805	0.8424
AdaBooster	Precision	0.7465	0.7558	0.7738	0.7953
	Recall	0.6839	0.8387	0.8387	0.8774
	F-Score	0.7331	0.7711	0.7860	0.8105
XGBooster	Precision	0.7465	0.7676	0.8011	0.8324
	Recall	0.6839	0.9161	0.8355	0.9290
	F-Score	0.7331	0.7933	0.8248	0.8501
CNN	Precision	0.7465	0.9708	0.9708	0.9235
	Recall	0.6839	0.8692	0.8692	0.9375
	F-Score	0.7331	0.9172	0.9172	**0.9304**

In the figure-13, the plot clearly indicates that the training curve and validation curve both keeps increase as the epoch increase. It shows that the CNN model generalizes well and not suffered overfitting problem. The model can generalize well to unseen data. Essentially, we can trust this model.

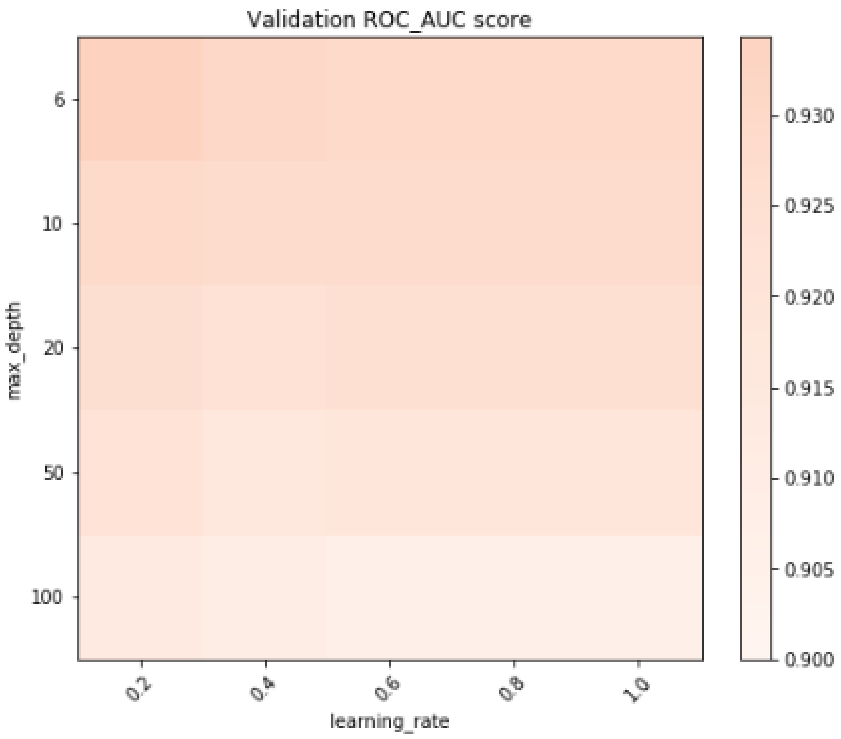
Figure-13: the performance of CNN model



7. Conclusion

In this study, we successfully used data extraction & GridSearchCV to improve the 3 supervised learning (SVM/AdaBooster/XGBooster) performances. The XGBooster is the best classifier of 4 supervised learning model. As below plot is a heatmap of the XGBooster's cross-validation accuracy score as a function of learning rate and gamma. For this example we explore a relatively grid for illustration purposes. The behavior of the model is very sensitive to the parameter of max_depth (the maximum depth of a tree). If max_depth is too large, it will make the model more complex and likely to be overfitting. Limit of max_depth is required for depth-wise grow policy

Figure-14: the heatmap of cross-validation accuracy score.



In the part of CNN, with the data pre-process and the data augmentation, the F-score improved by 2%. From this study, the CNN is still a more applicable method for image classification problems. The reason is that the CNN is able to learn relevant features from the input image, resulting in a better generalized model, which is better than other machine learning methods.

Although CNNs have achieved great success in experimental evaluations, there are still lots of issues that deserve further investigation. Firstly, since the recent CNNs are becoming deeper and deeper, they require large-scale dataset and massive computing power for training. Manually collecting labeled dataset requires huge amounts of human efforts. Thus, it is the next topic that how to explore unsupervised learning of CNNs.

Furthermore, one major barrier for applying CNN on a new task is that it requires considerable skill and experience to select suitable hyper-parameters such as the learning rate, kernel sizes of convolutional filters, the number of layers etc. These hyper-parameters have internal dependencies which make them particularly expensive for tuning. It is also a challenge of next study.

In this study, the most interesting is data exploration and background study. Each problem has their story. Exploring the data can help us to understand the problem in more depth. The biggest challenge as I mentioned before is model tuning. Only with optimized parameters can get optimize result. By sklearn-GridSearchCV, we can estimator the parameters and optimized by cross-validated grid-search over a parameter grid.

Environment

- **Programming language :** Python 3.6
- **Libraries :** Keras, Tensorflow, Scikit-learn, XGboost (for model set up), Pandas (for data review and pre-process), Matplotlib, Seaborn, Plotly (for data visualization)

References:

[1] Michael Denbina, Michael J Collins, and Ghada Atteia: On the detection and discrimination of ships and icebergs using simulated dual-polarized radarsat constellation data. Canadian Journal of Remote Sensing, (just-accepted):00–00, 2015.

[2] Carlos Bentes ; Anja Frost ; Domenico Velotto ; Bjoern Tings: Ship-iceberg discrimination with convolutional neural networks in high resolution SAR images. IEEE, 05 Sep 2016, ISBN: 978-3-8007-4228-8

[3] Carlos Bentes; Domenico Velotto; Susanne Lehner: Target classification in oceanographic SAR images with deep neural networks: Architecture and initial results. In IEEE International Geoscience and Remote Sensing Symposium (IGARSS) 2015, page 4, 2015.

[4] K. Punnam Chandar; M. Mahesh Chandra; M. Raman Kumar; B. Swarna Latha: Multi scale feature extraction and enhancement using SVD towards secure Face Recognition system. IEEE , 23 Sep 2011.

[5] Myeongsu Kang; Jong-Myon Kim: Singular value decomposition based feature extraction approaches for classifying faults of induction motors. MSSP, Dec 2013, Volume41, Issues 1-2, Pages 348-356

[6] Machine Learning, Ethem Alpaydin (2004), page 224

[7] https://www.kaggle.com/devm2024/keras-model-for-beginners-0-210-on-lb-eda-r-d

[8] http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

[9] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[10] http://xgboost.readthedocs.io/en/latest/parameter.html

[11] https://www.kaggle.com/phunter/xgboost-with-gridsearchcv

[12] https://www.kaggle.com/jayatou/xgbregressor-with-gridsearchcv

[13] https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

[14] https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/

[15] Tianqi Chen; Carlos Guestrin: XGBoost: A Scalable Tree Boosting System

[16] https://xgboost.readthedocs.io/en/latest/model.html