Lab 2 - Cloud Data, Stat 214, Spring 2025

March 22, 2025

1 Introduction

Cloud detection in polar regions is an important and challenging task in remote sensing technology. Ice-covered surfaces exhibit spectral features similar to clouds, making traditional cloud detection methods less effective. To address this challenge, Yu et al. [1] introduced a method in a previous study to differentiate between cloud and non-cloud surfaces using three key features: normalized difference angle index (NDAI), standard deviation (SD), and correlation (CORR).

In this study, we utilize the same dataset and features as Yu et al., which contain 164 images from Multi-angle Imaging SpectroRadiometer (MISR), three of which have expert labels identifying cloud areas and non-cloud areas. Our goal is to develop and evaluate predictive models that can effectively classify cloud and non-cloud pixels. This study employs a structured approach: we first perform exploratory data analysis to visualize the labeled data and understand the distribution of key features. Next, we apply feature engineering techniques, including autoencoder-based embedding, to enhance the feature set. Then, we implement and compare multiple classification models to assess their performance through cross-validation and various evaluation metrics. Finally, we do some post-hoc EDA, stability and sanity check to provide some insights into how our model perform.

Note: The folder data/ contains three subfolders: data_labeled, data_unlabeled, and data_test. The data_labeled folder includes three .npz files with expert-provided labels, along with their corresponding embedding .csv files. The data_unlabeled folder holds all remaining .npz files without labels. For sanity checks, two .npz files from the unlabeled set were selected and placed in the data_test folder, which also contains their embedding .csv files. This structure ensures a clear separation between labeled, unlabeled, and evaluation data.

2 EDA

Our original data structure is as follows in Table 1. All images are in `.npz` format and contain the columns from 0 to 9, but only the 3 labeled images have the 10th column.

Index	Feature Description
0	y coordinate
1	x coordinate
2	NDAI
3	SD
4	CORR
5	Radiance angle DF
6	Radiance angle CF
7	Radiance angle BF
8	Radiance angle AF
9	Radiance angle AN
10	expert label (+1: cloud, −1: not cloud, 0: unlabeled)

Table 1: Description of dataset features

2.1 Expert labels

We downloaded the three images with expert annotations, and after the data preprocessing function (introduced later in 2.4), we looked at the X, Y fields and the label field: X is an integer ranging from [70, 368] interval, Y is an integer ranging from [2, 383], and label is one of the values 1,0,-1. We draw the picture with X and Y as horizontal and vertical coordinates, respectively. In Figure 1, we use white to represent the presence of clouds (1), blue to represent the absence of clouds (-1), and gray to represent unlabeled data (0).

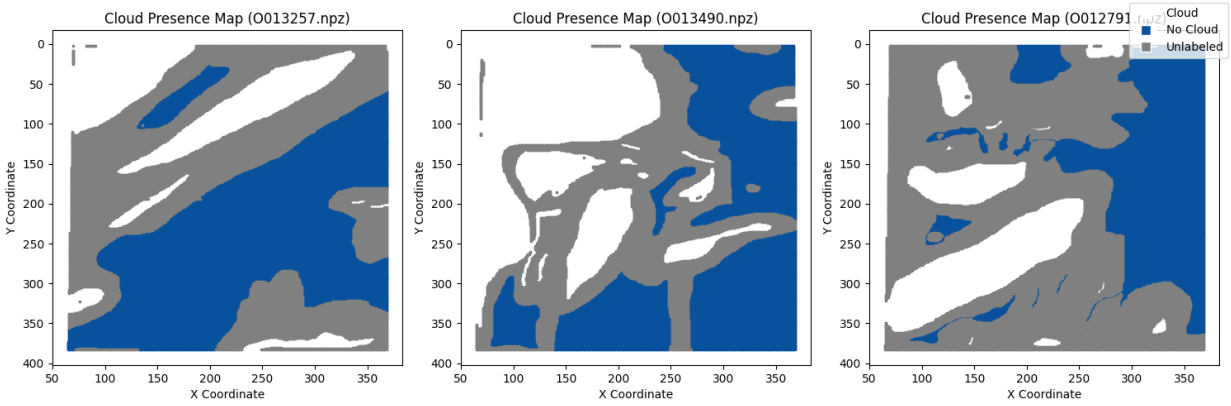


Figure 1: Presence and absence of clouds

In the three images, we can see that the clouds, non-clouds, and unlabeled areas each account for about one-third of the area, and that the cloud portion is characterized by large areas and small pieces of fragments coexisting.

2.2 Relationships & Differences based on the features

For the subsequent exploration of the relationships, we merged the data of the three images into a single dataframe, resulting in a total of 345005 rows. To explore the relationship between the angles, we calculated the correlation coefficients between each of the radiation angle variables and plotted heat maps to visualize them. As can be seen from Figure 2 and Table 2, `RCF`, `RBF`, `RAF`, and `RAN` are strongly positively correlated with each other, while these variables are all negatively correlated with `RDF`. We believe this is because angles of 60° and below are able to capture correct cloud information, and thus `RAN`, `RAF`, `RBF`, and `RCF` validate each other, indicating a high positive correlation. Whereas larger inclination angles, such as `RDF`, may not be able to capture information about the clouds because the further away from the vertical angle, the weaker the reflected radiation from the clouds becomes, and the interference increases, resulting in a negative correlation between the RDF and the other radiation angle variables.

Next, we removed all data labeled as cloudy and non-cloudy and performed a T-test on all radiances and features (Figure 3). The result is that there are **significant differences** between all the radiances and features of both groups assuming the same variance.

To visualize the difference between the two groups, we plotted box plots for every radiance angle and feature (Figure 3), allowing us to explicitly observe the differences in the data distributions between the two groups.

The results of the box plots align with those of the T-test, though some intuitive differences exist. For example, the patterns of `RAF`, `RBF`, `RCF`, and `RAN` are highly similar: in these cases, the mean for the no-cloud group is higher than that for the cloud group. In contrast, `RDF` exhibits the opposite trend, with a higher mean for the cloud group. Except for `CORR`, the no-cloud group generally has more outliers than the cloud group. For both `NDAI` and `SD`, the values in the no-cloud group, between the lower and upper quartiles, are tightly clustered, with a lower mean than the cloud group. Meanwhile, `CORR` is more dispersed in both groups, characterized by a wide interquartile range, no outliers, and a higher mean for the cloud group compared to the no-cloud group.

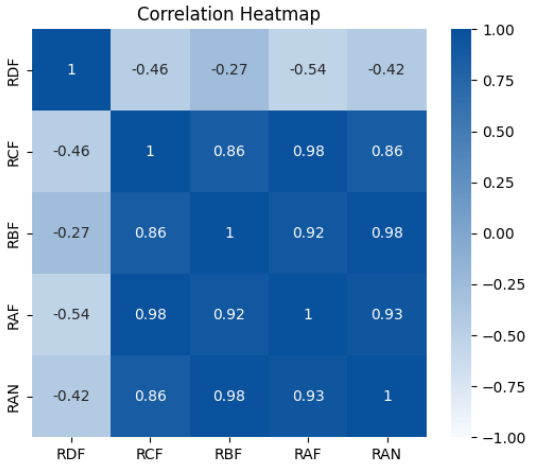


Figure 2: Correlation Heatmap

	Definition
RDF	Radiance Angle 70.5° Forward
RCF	Radiance Angle 60.0° Forward
RBF	Radiance Angle 45.6° Forward
RAF	Radiance Angle 26.1° Forward
RAN	Radiance Angle 0.0° Nadir

Table 2: Definition of Radiance Angles

	T-Statistic	P-Value	Conclusion
RDF	-28.294137	8.825297e-176	Significant
RCF	119.876521	0.000000e+00	Significant
RBF	217.495981	0.000000e+00	Significant
RAF	256.858983	0.000000e+00	Significant
RAN	255.994402	0.000000e+00	Significant
NDAI	-314.441308	0.000000e+00	Significant
SD	-259.943549	0.000000e+00	Significant
CORR	-25.492302	3.991317e-143	Significant

Table 3: T-test between the cloud group and non-cloud group

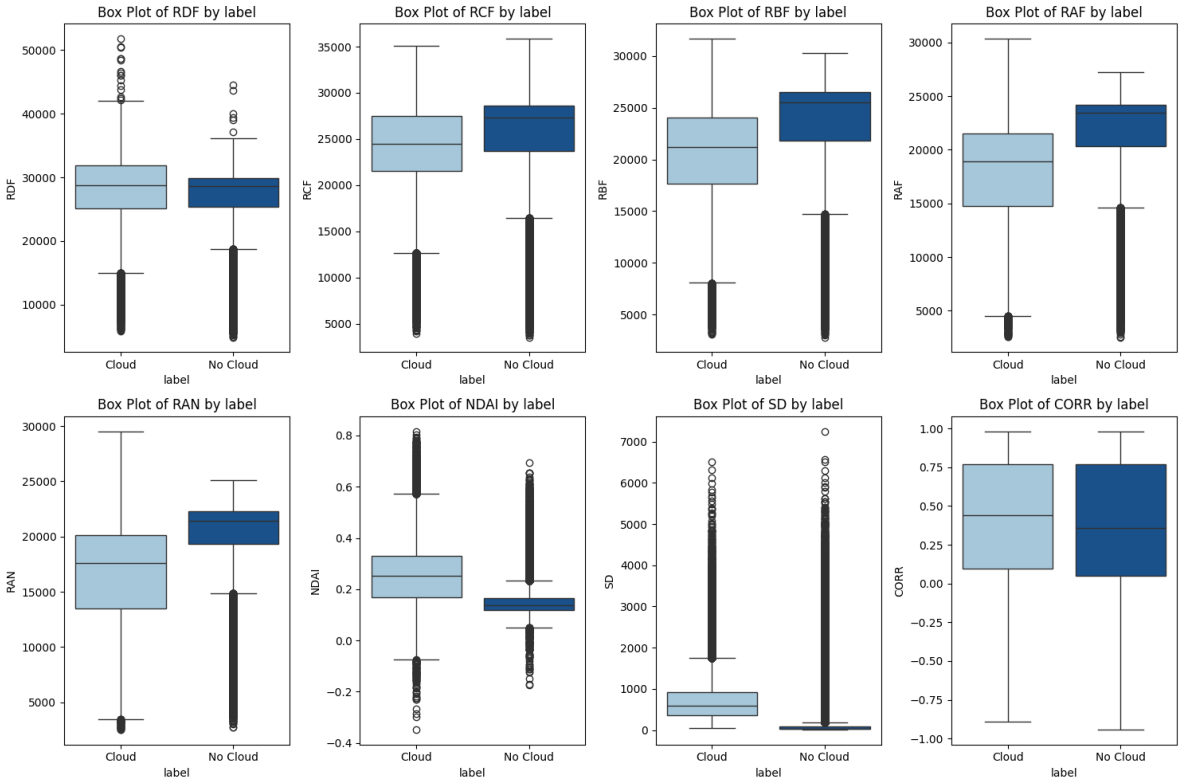


Figure 3: Distribution of difference variables compared between the two classes (cloud, no cloud)

2.3 Train & Test data split

We first examined the category distribution of all labeled data, where clouds accounted for approximately 39%, and the remaining 61% were labeled as non-clouds, as shown in Figure 4.

For the division of the training and test sets, we used the labeled data in the first two images (`0012791.npz`, `0013490.npz`) as the training set and the labeled data in the third image (`0013257.npz`) as the test set (Figure 5). When we need to cross-validate, we further separate the training set into two folds (Figure 6). Fold one is used to train with Image 1 and validate with Image 2, while Fold two is used to train with Image 2 and validate with Image 1. We then compute the average validation accuracy of the two folds.

We believe that our division is reasonable.

- First, when performing cross-validation, we ensure that the test set (`0013257.npz`) remains untouched.

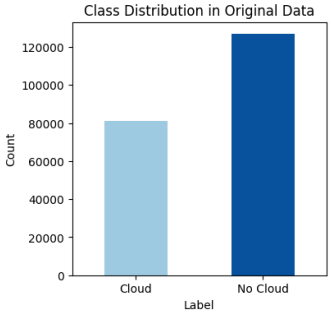


Figure 4: Original data

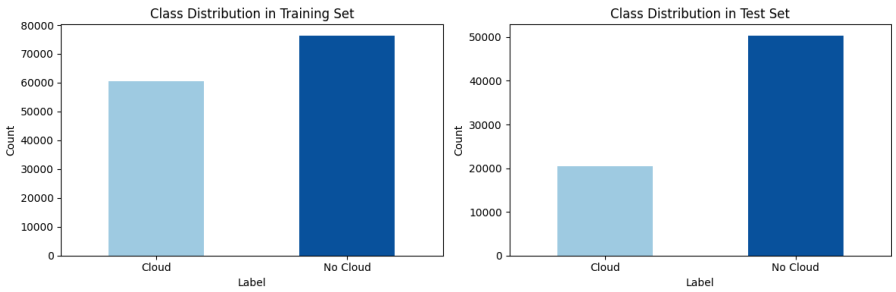


Figure 5: Training set & testing set

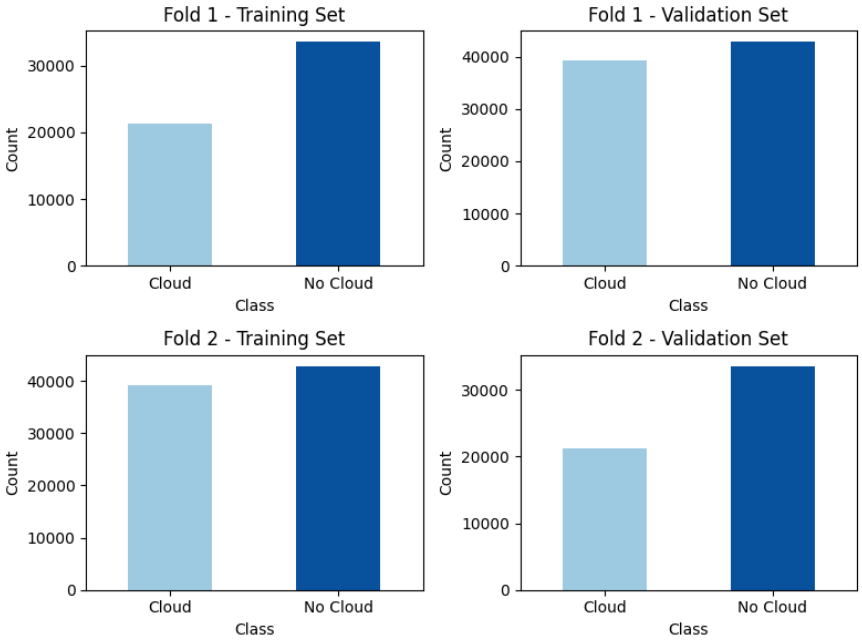


Figure 6: Class distribution in different folds of cross-validation

- Second, randomly shuffling the data and assigning neighboring data of a graph to both the training set and the test set would result in excessive test accuracy. Something similar can happen during cross-validation. Therefore, it is reasonable to partition the training set/validation set/test set in terms of images.
- Third, it can be seen that with such an allocation, the category distributions of the training and test sets are not similar. Since the similarity of the category distributions is not guaranteed when we apply our classification model to new cloud images, we believe that it is valid to split in this way.

We will select the model with the smallest average validation error during cross-validation. After selecting the model, we will retrain the model on the full training set before applying it to the test set for reporting of test errors. With this approach, we ensure that the training error is assessed objectively and the model selection is justified.

2.4 Data cleansing

In the data cleanup file, we wrote the directly reusable function *intodf* (which is in the functions.py file) that decompresses the image from the specified filename, reads the data into a dataframe, and automatically detects if it is labeled to rename the column names of the data.

In addition, we conducted further exploratory data analysis (EDA) to check for missing values (none were found) and examined the distribution of labels across the three labeled images. Notably, file **O012791.npz** contains the highest proportion of unlabeled pixels. Overall, the percentage of unlabeled pixels is non-negligible, and we have decided to exclude them from the validation and test sets since they do not contribute to assessing model performance.

For PCA analysis, we found that three principal components were sufficient to explain most of the variance in the data. However, the first two components did not strongly separate cloud and non-cloud pixels, as evidenced by significant overlap in their distributions in 2D PCA space. This suggests that linear feature combinations may not be sufficient for distinguishing cloud from non-cloud pixels, and alternative methods (e.g., non-linear dimensionality reduction or additional feature engineering) may be needed.

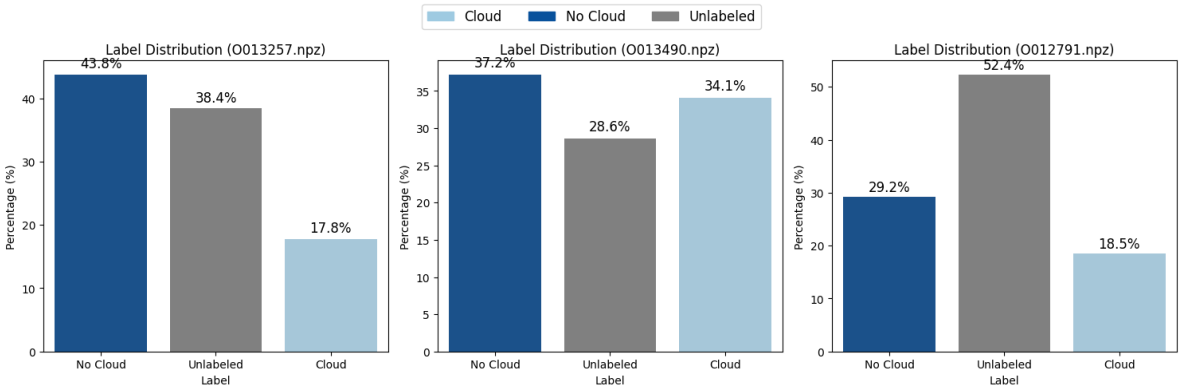


Figure 7: Label Distribution for Labeled images

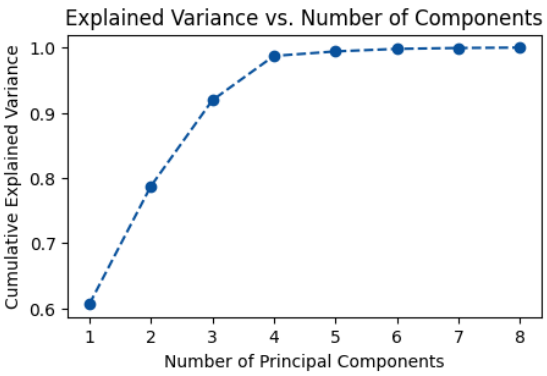


Figure 8: Explained Variance vs Number of Components

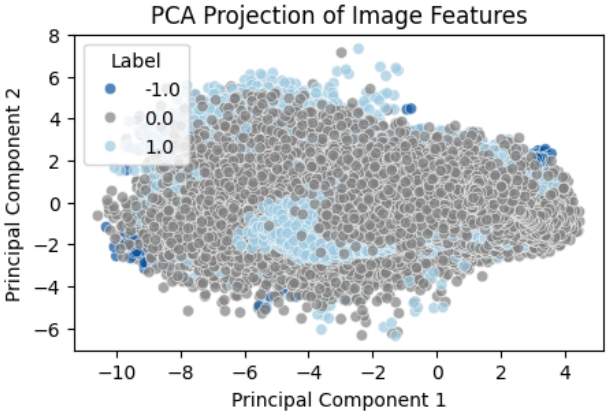


Figure 9: PCA Distribution

3 Feature Engineering

3.1 Informative features

We examined both a Random Forest and an XGBoost model trained on the original eight features, along with the correlation matrix, to identify the three most informative predictors. Figure 10 shows the feature importance of an XGBoost model. Here, we can see that `SD` was the top-ranked feature by a wide margin, with `CORR` following closely behind.

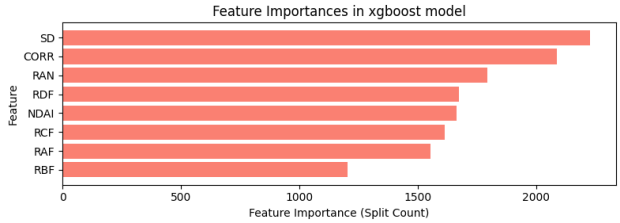


Figure 10: Feature importance of XGBoost model fit.

The feature importance plot for the random forest is shown in Figure 11. Again, `SD` appears to be the most important feature, followed by `NDAI`, and then `RDF`. In both models, `SD` stands out as a crucial variable.

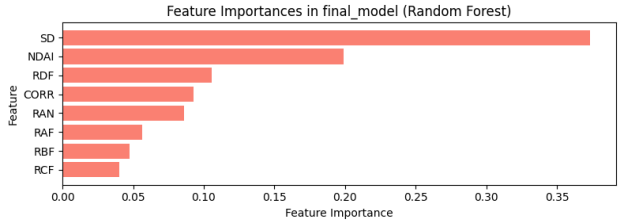


Figure 11: Feature importance of random forest model fit.

From the correlation plot in Figure 12, we see that `SD` has a higher absolute correlation with the label than most other features. `NDAI` also exhibits moderately strong correlations with the label, which is consistent in at least one of the models.

From these analyses, as well as the correlation analysis conducted in the EDA, we conclude that `SD`, `NDAI`, and `CORR` are the three most informative original features for distinguishing between cloud and non-cloud pixels. However, even though these are the most informative features, we emphasize that we will use other features in the following to train our models.

3.2 New Features

`NDAI`, `SD`, and `CORR` are powerful features developed by domain experts for cloud detection. However, they are computed on a per-pixel basis, ignoring the spatial context around each pixel. Since clouds and polar surfaces can appear very similar in the raw data, incorporating local spatial information can help the model improve at distinguishing subtle patterns in texture or brightness.

To incorporate spatial context, we added local mean and local standard deviation factors for `NDAI`, `SD`, and `CORR`. Specifically, for each pixel, we considered a 3×3 patch that is centered on a point in the image. We computed the mean and standard deviation for `NDAI`, `SD`, and `CORR` from this. We implemented this by pivoting each image into a 2D array, applying a 2D filter to the 3×3 neighborhood of each pixel, and then merging the results back into our data frame.

In the heatmap shown in Figure 13, we observe that the newly added local mean and standard deviation features exhibit moderate to strong correlations with their corresponding original features, indicating that they capture similar underlying information. However, the correlation is not perfect, suggesting that some of these features might capture additional spatial context not present in the raw pixel-wise values. Therefore,

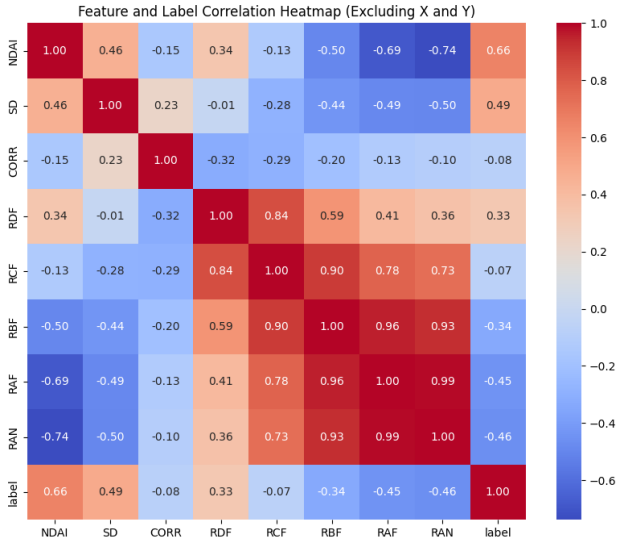


Figure 12: Correlation matrix of the original features.

if our model fitting does not yield satisfactory results in the following, we can consider incorporating some of these new features to improve model performance.

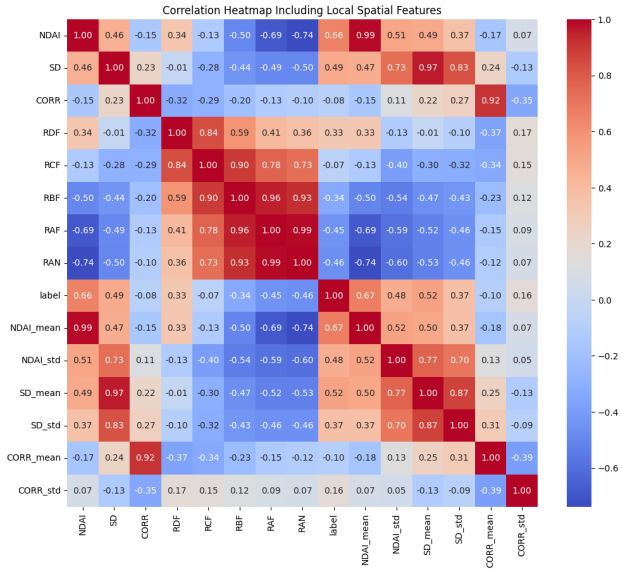


Figure 13: Correlation matrix with newly added features.

3.3 Transfer learning

Transfer learning is a machine learning technique in which a model pre-trained on a task is fine-tuned for a new related task. For this lab, we implement transfer learning by pretraining an autoencoder on a large set of unlabeled images and then tuning it on three images with expert labels. The objective is to apply unsupervised learning to generate new features that can effectively distinguish cloudy and non-cloudy pixels. The autoencoder is a type of neural network. It consists of an encoder that can compress input data into a low-dimensional latent space and a decoder that can transform it back to the original format.

To improve performance, we first try to modify the architecture of the neural network. We test multiple changes: add an extra linear layer, add more units at each layer, and adjust the activation function; different

classes are defined in the autoencoder.py file as seen in Table 4. We keep the default parameters and train models with different structures. Figure 14 shows that the best-performing models appear to be those with more nodes and deeper layers. Adjusting activation functions, such as LeakyReLU, ELU, and SELU, show similar performance trends, but none outperform the more complex architectures. We also tried to combine deeper layers and more units per layer together to see if that could further improve performance. The empirical result shows that it is actually worse than the baseline model, likely due to excessive complexity. We chose the one with the deeper layer as our final structure because it has the least validation loss.

Structure	**Description**
Structure 1	Baseline model using given structure
Structure 2	Add extra Linear layer
Structure 3	Double units in each layer
Structure 4	Use LeakyRELU as activation function
Structure 5	Use ELU as activation function
Structure 6	Use SELU as activation function
Structure 7	Combine structure 1 and 2, add two layers and double units per layer

Table 4: Neural Network Structures and Their Descriptions

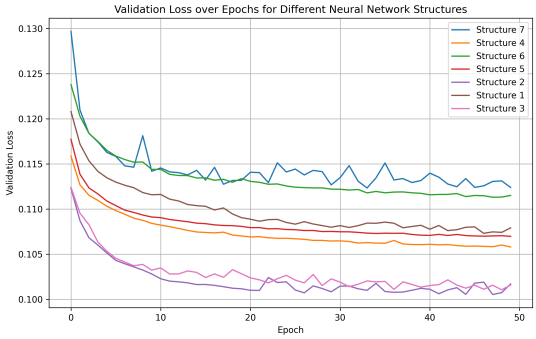


Figure 14: Validation loss over epochs for different neural network structures.

Name	**Learning Rate**	**Batch Size**	**Embedding Size**	**Max Epoch**
1	0.001	8192	8	50
2	0.0005	8192	8	50
3	0.0001	8192	8	50
4	0.001	4096	8	50
5	0.001	2048	8	50
6	0.001	8192	16	50
7	0.001	8192	32	50
8	0.001	8192	8	25
9	0.001	8192	8	100

Table 5: Comparison of Models with Different Parameters

Then, we proceed to parameter tuning, focusing on four parameters: `learning_rate`, `batch_size`, `embedding_size`, and `max_epochs`. We tested `learning_rate` at [0.0001, 0.005, 0.001], `batch_size` [8192, 2096, 2048], `embedding_size` [8, 16, 32] and `max_epochs` [25, 50, 100](Table 5) and generate corresponding configuration yaml files in `code/configs` directory. Figure 15 shows that increasing the embedding size can significantly reduce the validation error. A slower learning rate and a smaller batch size will increase the error. Finally, we choose `learning_rate = 0.001`, `batch_size = 8192`, `max_epochs = 50`, and `embedding_size = 32` as

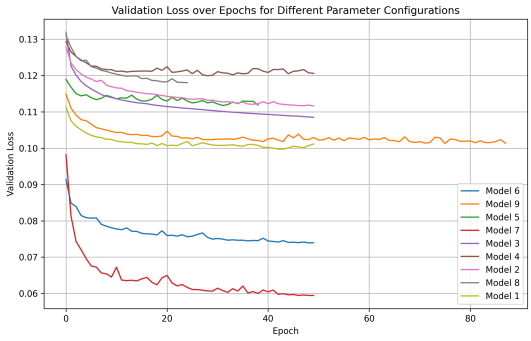


Figure 15: Validation loss over epochs for different parameter.

our best parameters since it has least validation loss and steepest decline rate. We use this model to get embeddings from 3 labeled images, which gives us three csv files with 32 new features.

4 Modeling

4.1 Model 1- LightGBM

We employed LightGBM (LGBM), a high-performance gradient boosting framework, to classify cloud and non-cloud pixels in satellite imagery, leveraging its efficiency for both the original 8 features (final_model_1) and the expanded 40 features (8 original + 32 autoencoded, final_model_2). To optimize hyperparameters, we conducted a grid search over `num_leaves` $\in \{15, 31, 63\}$ and `learning_rate` $\in \{0.01, 0.05, 0.1\}$, using two-fold cross-validation on the training datasets (`df1` and `df2` which corresponds to `0012791.npz`, `0013490.npz`). Each fold was trained on one dataset and validated on the other, with early stopping (patience=20) to mitigate overfitting, and the best parameters were selected based on the lowest average binary error. The models were retrained on the combined `df1` and `df2`, achieving test accuracies of 91.72% for final_model_1 and 92.42% for final_model_2 on `df3`, though final_model_2 exhibited a bias toward non-cloud predictions due to the class imbalance (71% non-cloud, 29% cloud).

4.2 Model 2- K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric, instance-based classification algorithm. It classifies a new data point by finding the k most similar points (neighbors) in the training data and assigning the most common class label among them. KNN relies on several key assumptions. First, we assume that neighboring data points are similarly labeled. This makes sense, as common sense suggests that clouds are continuously distributed, and we can solidify this with the box plots in Section 2.2 that the `SD` of both clouds and non-clouds are mainly distributed on very small scales.

Second, the features should be of a uniform scale so that different magnitudes do not result in different weights for the features when calculating the distance. To this end, we are using `StandardScaler` function in Python to perform a z-score normalization on all feature columns before applying the algorithm.

Third, the number of neighbors K should be chosen carefully to balance the bias-variance trade-off. A small K may overfit, while a large k may oversmooth decision boundaries. To determine the optimal value of K, we used cross-validation, searching within the range [3,10]. Additionally, we tested different weight settings, either uniform or distance-based.

For KNN with the initial features, the best parameters were `K=8` with uniform weights, achieving a test accuracy of 82.10%. When incorporating both initial and embedding features, the optimal parameters changed to `K=4` with distance-based weights, resulting in a slightly higher test accuracy of 82.47%. Adding the autoencoder-generated features resulted in an accuracy improvement of 0.37%. While recall and F1 scores showed improvement, accuracy slightly decreased. Given that our prediction dataset has an imbalanced category distribution (with non-cloud samples being the majority and cloud samples the minority), we

believe that the additional 32 features still contributed positively to the model's predictive performance.

4.3 Model 3- XGBoost

XGBoost is an efficient and scalable gradient-boosting framework that, similarly to LGBM, builds an ensemble of decision trees in a sequential way. However, unlike LGBM, which utilizes histogram-based algorithms to achieve high speed on very large datasets, XGBoost offers a more flexible realization framework and also provides options for more hyperparameters, such as `min_child_weight`, which allows for greater control over the model's complexity. In our implementation, we tuned hyperparameters, specifically `max_depth` and `learning_rate` (and an additional exploration of `min_child_weight` for the expanded feature set), using a two-fold cross-validation scheme similar to the one described for the LGBM model. After the hyperparameter tuning, we trained the XGBoost model on the full training set. Table 6 shows that the accuracy for the XGBoost model with the original features was approximately 91.35%, while the test accuracy improved to around 92.45%.

4.4 Cross Validation

We follow the cross-validation methodology outlined in Section 2.3 for model parameter selection. Specifically, for each class of models we have proposed, we train the model with different parameter settings on Image 1 (`O012791.npz`), validate it on Image 2 (`O013490.npz`), and compute the error. Then, we swap Image 1 and Image 2, repeat the process, and calculate the error again.

The two validation errors obtained are then averaged to determine the average validation error. The model with the lowest average validation error is selected as the representative for this class of models. Finally, it is trained on both Image 1 and Image 2, and its test error and other evaluation metrics are computed on Image 3 (`O013257.npz`) to assess overall model performance.

4.5 Model Assessing

We started by plotting the confusion matrix (Figure 16). (Because of space constraints, we did not put all the confusion matrices in the report.) We can see that XGBoost and LGBM are much more accurate than KNN, and that the second model with all features is more accurate for both XGBoost and LGBM. Both have extremely low false positive rates, half that of XGBoost using the basic features.

Table 6 compares various metrics, including accuracy, precision, recall, F1 score, AIC, and ROC-AUC. As observed, XGBoost and LGBM, with all features added, have near-optimal data on accuracy, precision, recall, and F1 score. As shown in Figure 17, we are very pleased to discover that even with a ROC-AUC of up to 98%, their model complexity is not high (indicated by AIC). We can consider both XGBoost and LGBM as balanced models capable of balancing both positive and negative label prediction under various metrics.

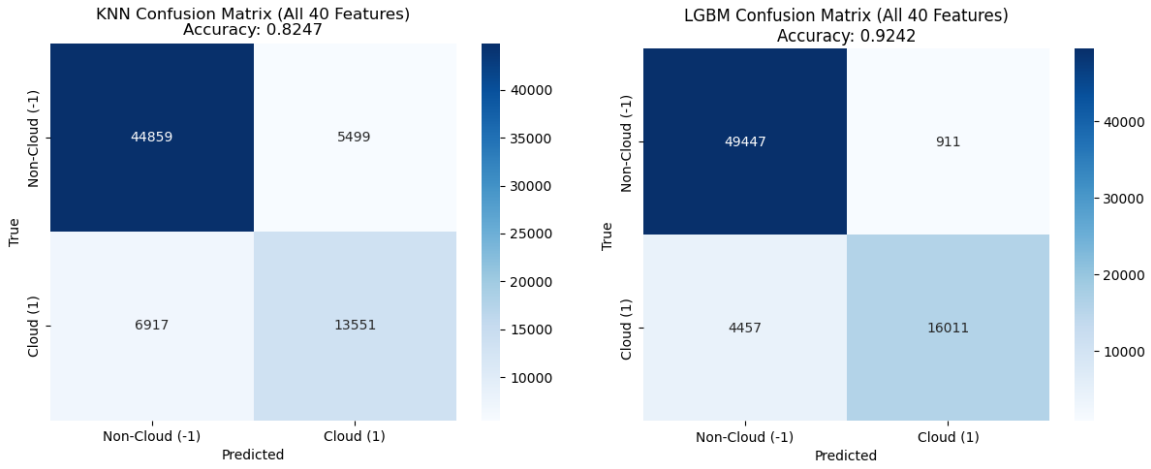
Model	Accuracy	Precision	Recall	F1 Score	AIC	ROC-AUC
LGBM with original features	0.9172	0.8999	0.8027	0.8485	**14048.86**	0.98
LGBM with new features	0.9242	**0.9462**	0.7822	0.8564	19438.33	0.98
KNN with original features	0.8210	0.7405	0.5860	0.6543	209435.30	0.75
KNN with new features	0.8247	0.7113	0.6621	0.6858	268500.46	0.78
XGBoost with original features	0.9135	0.8792	**0.8121**	0.8443	16386.47	0.98
XGBoost with new features	**0.9245**	0.9451	0.7842	**0.8572**	17022.83	**0.99**

Table 6: Comparison of Models with Different Parameters

4.6 Best Classifier: LGBM

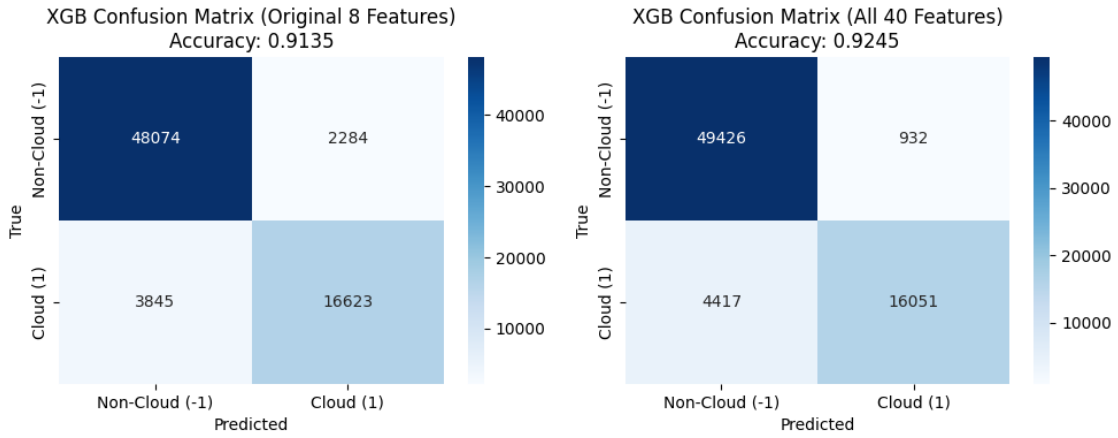
Therefore, we will check the two log-loss convergence plots for XGBoost and LGBM.

Figure 18 compares XGBoost and LGBM based on their convergence plots, and we observe key differences in training and validation log-loss behavior. The training log-loss for XGBoost decreases smoothly but does not reach zero, flattening at a slightly higher value than LGBM. This suggests that XGBoost applies stronger regularization, which can help prevent overfitting. In contrast, LGBM's training log-loss decreases much faster and gets closer to zero, indicating that it learns more aggressively.



(a) KNN Confusion Matrix

(b) LGBM Confusion Matrix



(c) XGBoost Confusion Matrix

Figure 16: Confusion Matrices for KNN, LGBM, and XGBoost Models

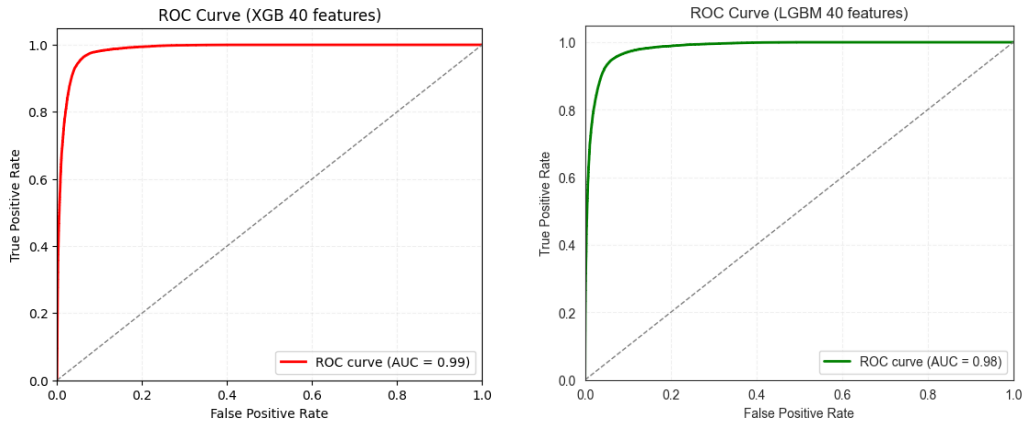


Figure 17: Comparison of ROC Curves for XGBoost and LGBM Models (40 Features)

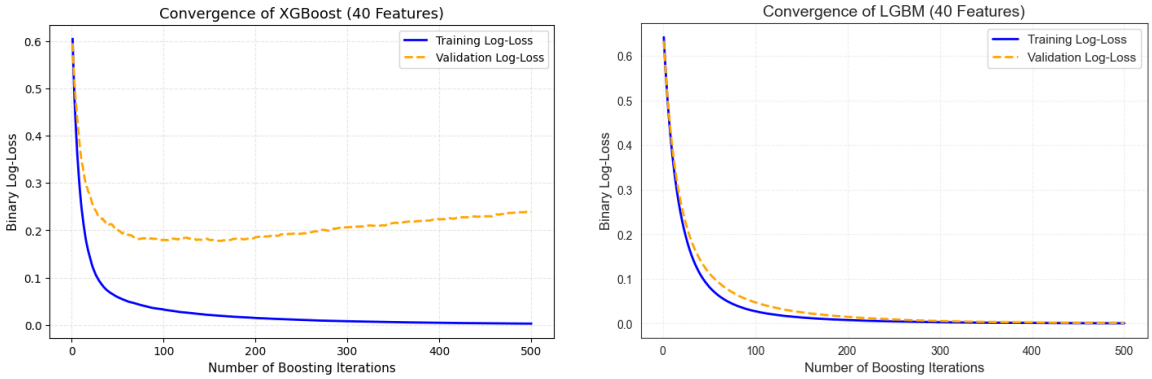


Figure 18: Comparison of Log-loss Convergence for XGBoost and LGBM Models (40 Features)

Regarding validation log-loss, XGBoost initially decreases rapidly but begins to rise slightly after around 200 iterations, suggesting signs of overfitting. On the other hand, LGBM maintains a more stable and consistently decreasing validation log-loss, suggesting better generalization. This indicates that LGBM is likely more efficient in capturing patterns without overfitting, making it a strong candidate when prioritizing generalization performance. Therefore, we chose LGBM with all the features as the best model.

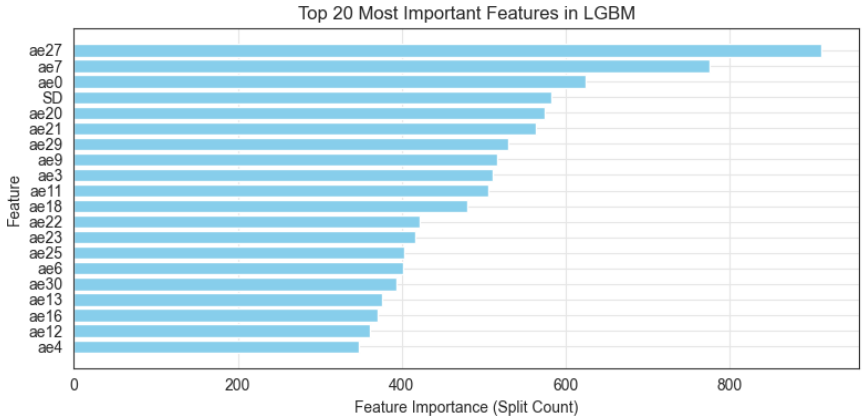


Figure 19: Top 20 most important features in the LGBM model with 40 features.

In LGBM, the split count measures how often a feature is used to split a node across all the decision trees in the model. According to Figure 19, features like `ae27`, `ae0`, and `SD` have high split counts, meaning they were frequently used to split nodes in the trees. Notably, the vast majority of important features are embedding generated features, which proves that our autoencoder is indeed able to recognize cloud patterns. It is worth mentioning that `SD` still occupies a high importance among the original features.

4.7 Post-hoc EDA

The scatter plots (Figure 20) compare true cloud labels (left) with predictions from the LightGBM model using all 40 features (right) on image `O013257.npz`. The model accurately identifies non-cloud pixels (gray) but struggles with cloud pixels (blue), misclassifying many as non-cloud (increased false negatives). This is likely due to the imbalanced dataset (71% non-cloud, 29% cloud), which biases the model toward non-cloud predictions.

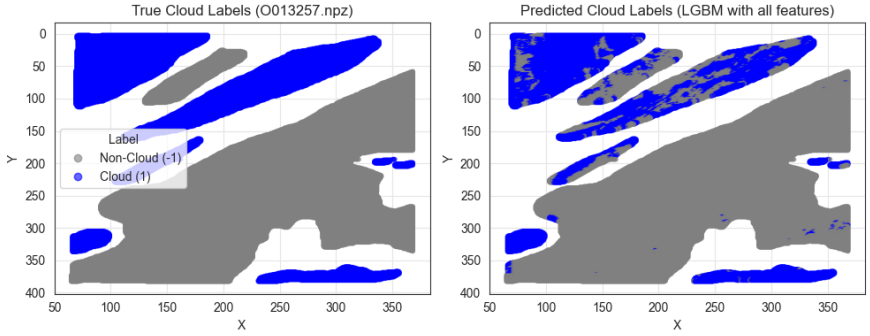


Figure 20: Post-hoc EDA for LGBM model on test data

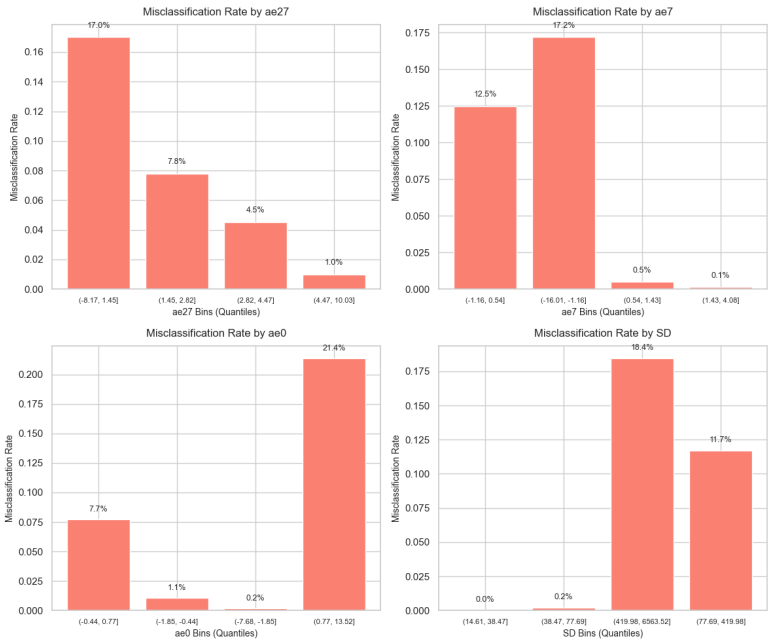


Figure 21: Misclassification rate for different quantiles for the 4 most important features of the model.

To assess individual features, we can examine different quantiles for the four most important features of the model. This is displayed in Figure 21. Interestingly, the misclassification rate for the most important feature `ae27` shows that it has higher accuracy for higher values of this particular feature. However, the `SD` feature

is the opposite. For higher values of the SD variable, we observe more misclassification errors compared to lower values.

4.8 Future Applications and stability analysis

To assess robustness of our prediction, we conducted two stability check tests. First, we added small Gaussian noise to each feature in the test dataset. And to make sure the noise does not drastically change the feature, the standard deviation of noise term is set to 1% of the feature standard error. Then we used the trained model to predict outcomes for both the original and perturbed test datasets. We evaluated robustness by comparing key metrics like accuracy, precision, recall, F1 score, AIC, and ROC, and we also compare confusion matrix for both cases. We found metrics remain nearly the same (Table 7), which means the model is stable when having small perturbations. The confusion matrices also show minimal differences between original and perturbed datasets (Figure 22).

The second stability check used bootstrap. We generated 10 bootstrap samples from the training dataset and retrained the model using the same parameters as the original one. Then we make prediction using each bootstrapped model. Finally, we computed and compared the mean performance metrics across all bootstrapped models with the original model metrics. The result is still very close to the original results, although the variation is slightly larger compared to the first method (Table 7). In conclusion, our model demonstrates high stability under both small test data perturbations and variations in training data. Therefore, we believe our model can work well on future dataset with expert labels.

Model	Accuracy	Precision	Recall	F1 Score	AIC	ROC-AUC
Original data	0.924209	0.946165	0.782245	0.856432	19438.33	0.983559
Add noise	0.924152	0.946680	0.781561	0.856233	19464.63	0.983498
Bootstrap	0.926918	0.942847	0.795324	0.862819	18979.71	0.982847

Table 7: Stability check comparison

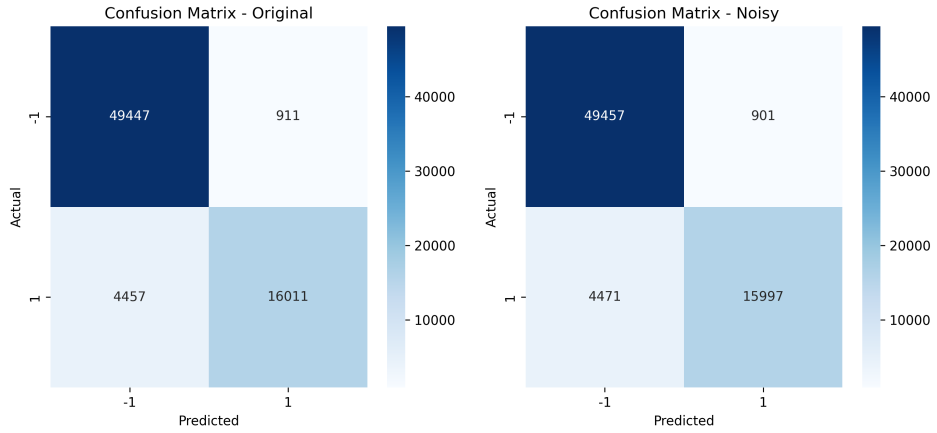


Figure 22: Confusion matrix with/without error

For a sanity check, we applied our model to predict on the unlabeled dataset 0002772.npz and generated a scatter plot to visualize the results. However, one challenge in evaluating our predictions is the lack of true labels. Instead, we analyze the scatter plots of SD, CORR, and NDAI to check if our predictions align with the distributions of these three features. We selected these features because they were manually generated by experts based on domain knowledge [1], and they also ranked high in feature importance among the original dataset features.

Upon examining the visualizations, we observe a cloudy region concentrated in the lower-left corner of the figure (Figure 23). Notably, this same area exhibits high SD values, indicating significant variability in pixel

intensity. Additionally, `NDAI` values in this region are higher compared to the rest of the image, suggesting distinct angular reflectance differences. The alignment of high `SD` and elevated `NDAI` values with the model's target predictions in the lower-left corner suggests the model has successfully identified an anomalous region, which appears to be a reasonable prediction.

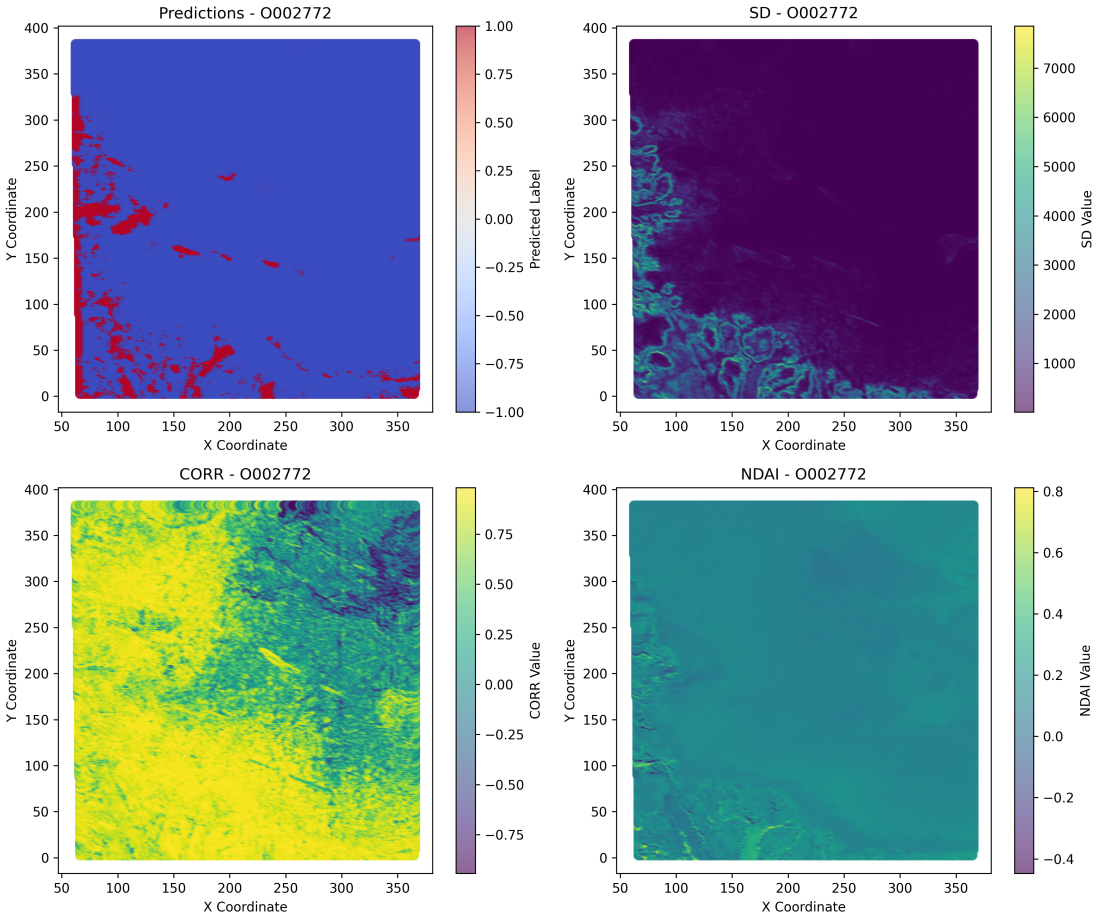


Figure 23: Prediction on `O002772.npz` along with feature distribution

5 Bibliography

References

[1] Tao Shi, Bin Yu, Eugene E Clothiaux, and Amy J Braverman. Daytime arctic cloud detection based on multi-angle satellite data with case studies. *Journal of the American Statistical Association*, 103(482):584–593, 2008.

A Academic honesty

A.1 Statement

We hereby make the following statements: We personally designed and conducted all data analysis processes presented in this report. We have written and produced all text and graphs included in this report. Additionally, we have incorporated and documented all procedures into our workflow, ensuring that the results are fully reproducible. We have properly attributed all references to others' work.

We strongly believe that integrity in academic research is of utmost importance. First and foremost, we

must take responsibility for the results of our research. Since science thrives on collaboration, our findings will serve as the foundation for future research. Dishonest or non-reproducible research can lead to a cascade of erroneous results. Furthermore, plagiarism contributes nothing new to the scientific community and is disrespectful to the original authors, as it misappropriates credit for their work. Therefore, we are committed to maintaining honesty, transparency, and reproducibility in all aspects of our research.

A.2 LLM Usage

Coding

The code portion of this report only uses chatGPT to help refine formatting and annotation writing and to help with data visualization. We ensure that data exploration, preprocessing, model selection, comparison, interpretation, and tuning are entirely manual by the team members.

Writing

The writing portion of this report only uses chatGPT to help with the layout of the images and the rephrasing of very few sentences. We make sure that the vast majority of the content is brainstormed, discussed, organized, and typed by us in person.