STAT 214 Lab 2

1 Introduction

Cloud detection in polar regions is challenging because both clouds and ice/snow-covered surfaces appear white and cold in satellite imagery, making traditional detection methods ineffective. This study uses data from NASA's Multi-angle Imaging SpectroRadiometer (MISR) sensor, which captures images at nine different viewing angles.

The dataset consists of 164 satellite images, with only three containing expert labels that identify pixels as either cloud or non-cloud. The report is divided in three parts and follows the structure as we perform exploratory data analysis to understand the data structure and identify patterns distinguishing clouds from non-clouds; next, we engineer effective and new features for cloud detection and implement transfer learning using autoencoders pre-trained on unlabeled images; finally, we develop and evaluate several classification models, selecting the most effective one for operational use in cloud detection.

In the end, we build a prediction model to distinguish cloud from non-cloud for each pixel using the available MISR images.

2 Exploratory Data Analysis

2.1 Visualization of Expert Labels

We begin our analysis by visualizing the expert labels for the three labeled images (O013257, O013490, and O012791) to gain an understanding of the spatial distribution of clouds and non-clouds in the dataset. Each image contains approximately 115,000 pixels (O013257 with 115,000 pixels, O013490 with 115,032 pixels, and O012791 with 114,973 pixels). Figure 1 shows the expert labels mapped according to their X and Y coordinates.

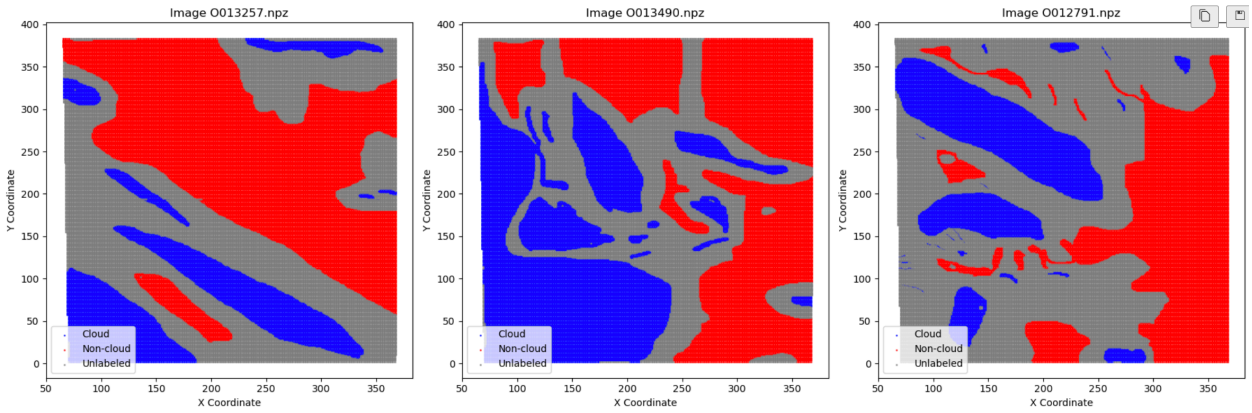


Figure 1: Expert Labels for the Three Labeled Images

The above spatial visualization reveals distinct patterns in cloud formation across the three images: Image O013257 shows significant non-cloud regions (red) in the upper portions, with cloud formations (blue) primarily in the lower sections of the image. There are also substantial unlabeled regions (grey) throughout the image; Image O013490 shows a more balanced distribution, with larger cloud formations in the lower half and non-cloud areas predominantly in the upper half; and Image O012791 shows significant cloud coverage (blue) in the center of the image forming band-like patterns, with non-cloud areas (red) along the edges.

These patterns suggest that clouds in polar regions form connected structures rather than appearing as random, scattered pixels across an image. The significant presence of unlabeled regions in all three images highlights the challenge of getting thorough expert labels in remote sensing applications.

2.2.1 Relationships Between Radiances at Different Angles

Next, we examine the relationships between radiances at different angles to understand how they might help differentiate clouds from non-clouds. Our dataset contains measurements from five different angle perspectives: DF (70.5° forward), CF (60.0° forward), BF (45.6° forward), AF (26.1° forward), and AN (nadir, 0°). From the three labeled images, we identified a total of 207,681 labeled pixels (70,826 from O013257, 82,083 from O013490, and 54,772 from O012791).

The scatter plots in Figure 2 indicates that cloud pixels (blue) generally have higher radiance values across all angles compared to non-cloud pixels (red), with cloud pixels showing greater spread and variability. Both classes displays strong linear relationships between measurements taken at different angles, but form distinguishable clusters. The diagonal plots (density distributions) show that cloud pixels have wider and more variable distributions compared to the tighter distributions of non-cloud pixels.

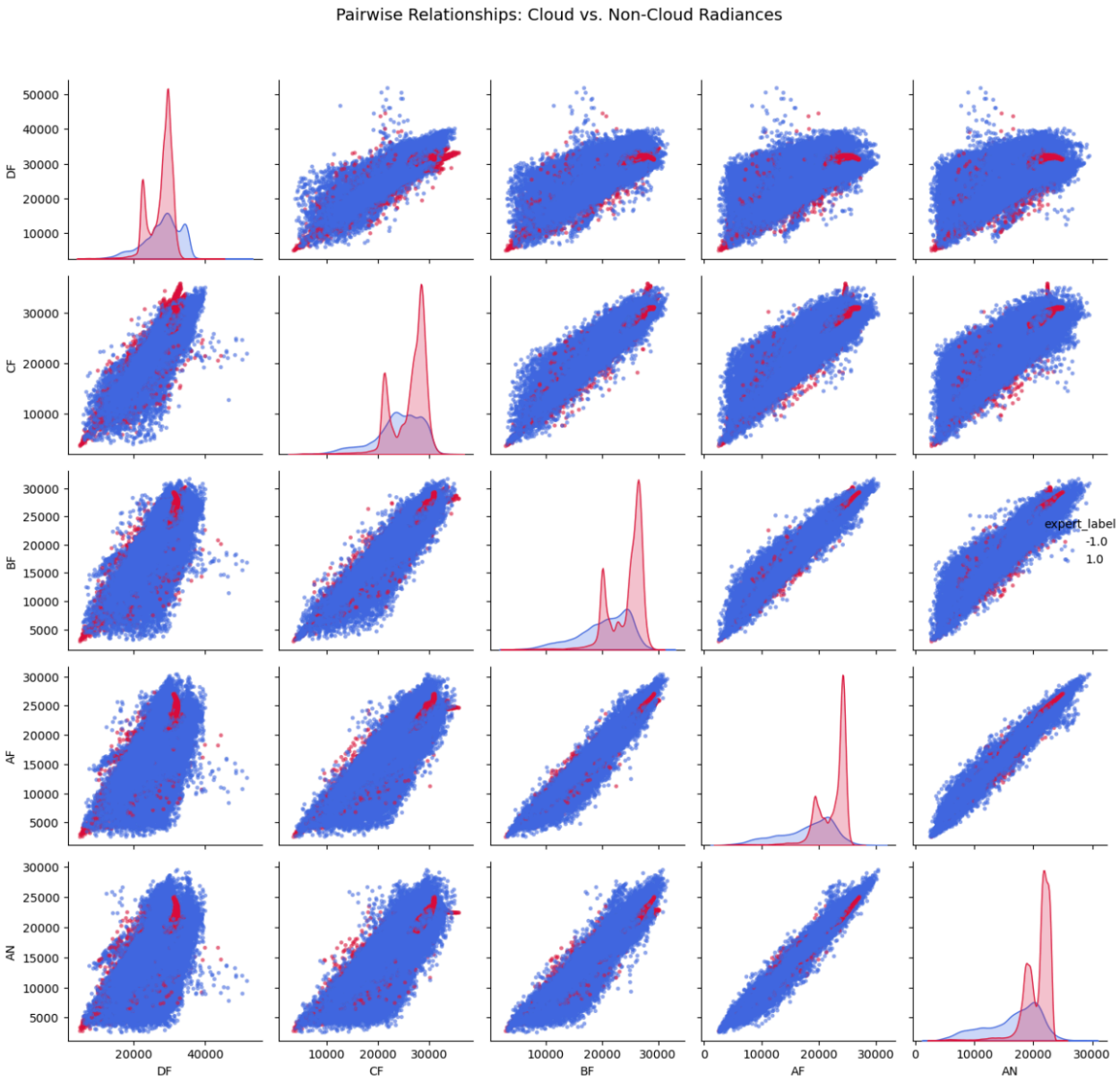
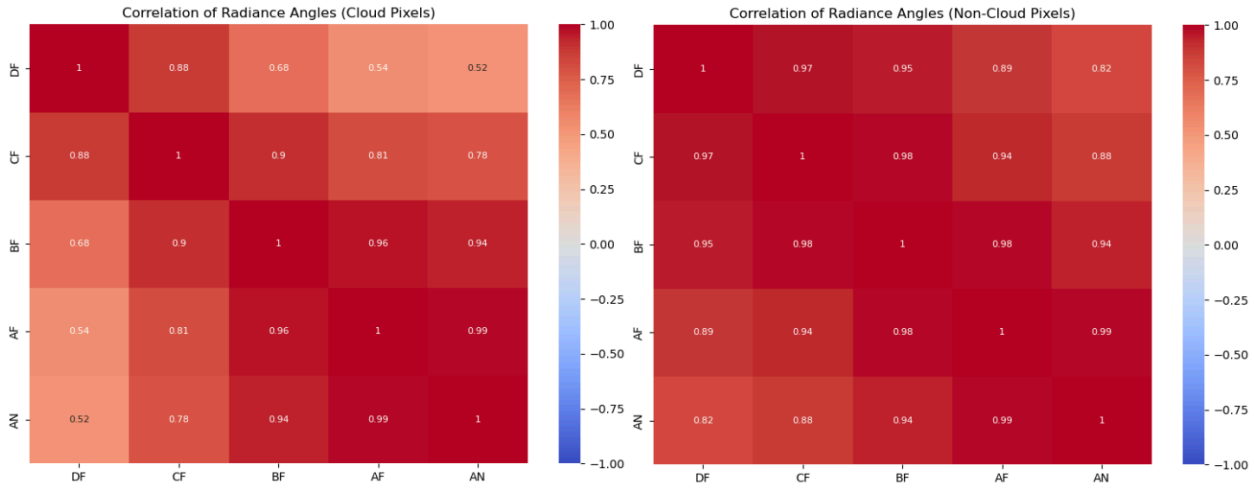


Figure 2: Pairwise Relationships Between Radiance Measurements (Cloud vs Non-Cloud)

To quantify these relationships, we computed correlation matrices for cloud and non-cloud pixels separately. The correlation matrices in Figure 3 shows that for cloud pixels, the correlations between angles range from 0.52 to 0.99, with adjacent angles (like AF-AN with 0.99) showing the highest correlations while for non-cloud pixels in figure 4, correlations are generally higher across all angle pairs, ranging from 0.82 to 0.99. The most significant difference is in the correlation between DF and AN, which is 0.52 for cloud pixels but 0.82 for non-cloud pixels.

This pattern directly supports the understanding described in Yu et al. (2008) that ice and snow surfaces scatter radiation more isotropically than clouds. When a surface scatters radiation isotropically (more evenly in all directions), the measurements from different viewing angles will be more consistent, resulting in higher correlations. In contrast, clouds exhibit more complex and anisotropic



(a) Figure 3: Correlation Matrix for Cloud Pixels

(a) Figure 4: Correlation Matrix for Non-Cloud Pixels

scattering behaviors, particularly when comparing extreme angles like DF (70.5° forward) and AN (nadir, 0°).

2.2.2 Analysis of Feature Characteristics

We further analyze the three features highlighted in the Yu et al. (2008) paper: NDAI (Normalized Difference Angular Index), SD (Standard Deviation), and CORR (Correlation). These features were specifically developed to help in polar cloud detection.

The statistical analysis reveals clear differences between cloud and non-cloud pixels:

Feature	Class	Mean	Std Dev	Min	Max
NDAI	Cloud	0.264589	0.126909	-0.348311	0.816864
	Non-cloud	0.142714	0.043273	-0.172772	0.693426
SD	Cloud	723.74166	529.91462	44.707161	6516.2803
	Non-cloud	163.76302	443.07198	13.329613	7251.0093
CORR	Cloud	0.413331	0.383378	-0.888728	0.980773
	Non-cloud	0.366549	0.422800	-0.942551	0.982818

Key observations:

- NDAI values for cloud pixels are significantly higher (mean: 0.26) than for non-cloud pixels (mean: 0.14), with cloud pixels also showing greater variability.
- SD is substantially higher for cloud pixels (mean: 723.74) compared to non-cloud pixels (mean: 163.76), indicating greater textural variability in cloud surfaces.
- CORR shows a smaller difference between the classes, with cloud pixels having a slightly higher mean (0.41 vs. 0.37).

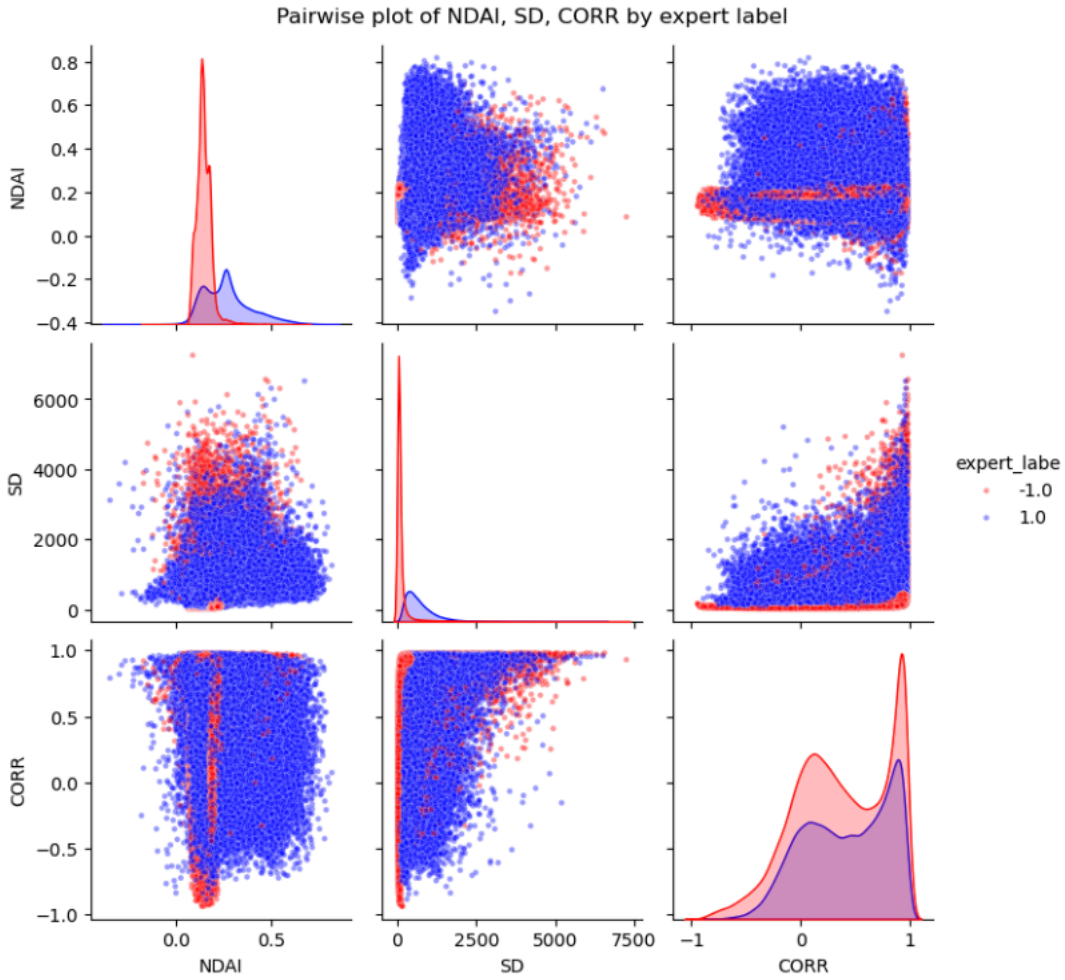


Figure 5: Pairwise Relationships Between NDAI, SD, and CORR by Expert Label

The pairwise plots in Figure 5 reveal that the combination of features, particularly NDAI and SD, provides enhanced classification power. In NDAI vs. SD plot, we observe that non-cloud surfaces cluster predominantly in the region of low NDAI and low SD values, while cloud pixels spread toward higher values in both dimensions. This creates distinct regions in the feature space that can be leveraged for classification. These differences align with Yu et al's approach of identifying surface characteristics rather than cloud characteristics which proves particularly valuable in polar regions where traditional cloud detection methods struggle due to their similar brightness and temperature characteristics of clouds and snow/ice surfaces.

2.3 Data Splitting Strategy

We decided to perform a holdout data split after splitting our image data into quadrants. The process begins by splitting each image into four quadrants based on x-y coordinates and then grouping them together based on quadrant. Next we aggregate data from three quadrants and use them for training while the remaining quadrant is used for testing.

Prior to model training and testing, we filter out data that does not contain expert labels (i.e. we will keep data with cloud/no cloud labels).

We decided to use a hold-out data split due to the simplicity and computational speed associated with it. Additionally, it prevents data leakage by ensuring that the test set is never seen during training which may help detect overfitting if the model performs well on training but poorly on testing. Lastly, using a hold-out method is good for hyperparameter tuning because it ensures that chosen hyperparameters don't contaminate the final test set.

The choice of using a hold-out data split was determined after several iterations of model training and performance evaluation.

2.4 Data Cleaning and Preprocessing

When dealing with real world data, these images may have imperfections that can affect analysis quality. Our cleaning approach focuses on:

- Outlier Detection: Outlier detection approach recognized that cloud and non-cloud pixels may have different distributions, so we applied the cleaning process separately for each class using a z-score threshold of 3. This resulted in the removal of 6,236 outliers (about 3% of the dataset), with a slightly higher percentage in non-cloud pixels (3.33%) compared to cloud pixels (2.49%). The SD feature showed the most significant impact from outlier removal, with maximum values dropping from 7251 to 2313, and non-cloud means decreasing by about 40%. The NDAI feature range narrowed from [-0.348, 0.817] to [-0.115, 0.645], though not a big change in mean values. The CORR feature remained relatively stable throughout the cleaning process. This measured approach preserved the natural variability in the dataset while eliminating only the most extreme values.
- Missing Value Handling: We examined the dataset for missing or invalid values that could impact our analysis. No missing values (NaN/null) were found in any of the columns. Additionally, we checked for implausible measurements, specifically looking for negative or zero radiance values that would indicate sensor errors, but none were detected. This suggests the expert-labeled dataset was already pre-processed to include only valid measurements.

Our cleaning approach created cleaner feature distributions while keeping the key differences between cloud and non-cloud pixels unchanged. The biggest change was in the SD feature for non-cloud pixels, which better matches what we would expect from smooth ice and snow surfaces.

3 Feature Engineering

3.1 Feature Importance Analysis

Before we deal with the modeling and classification of individual pixels, we want to take a closer look at the current features of our data set and, in particular, at the importance of existing features. Such an importance of existing features can be carried out using a CART model, for example. The CART model creates a decision tree and has the special feature that only two branches may be present. This makes the model relatively easy to interpret, which is why it can be used here for

the initial determination of feature importance. It was determined that a total of only 4 levels are permitted in order to facilitate the complexity and thus the interpretability of the CART model.

Figure xy shows the decision tree on the original data set. Here it is clear that the attributes SD and CORR, which are the expert features, have the highest feature importance, followed by the other features.

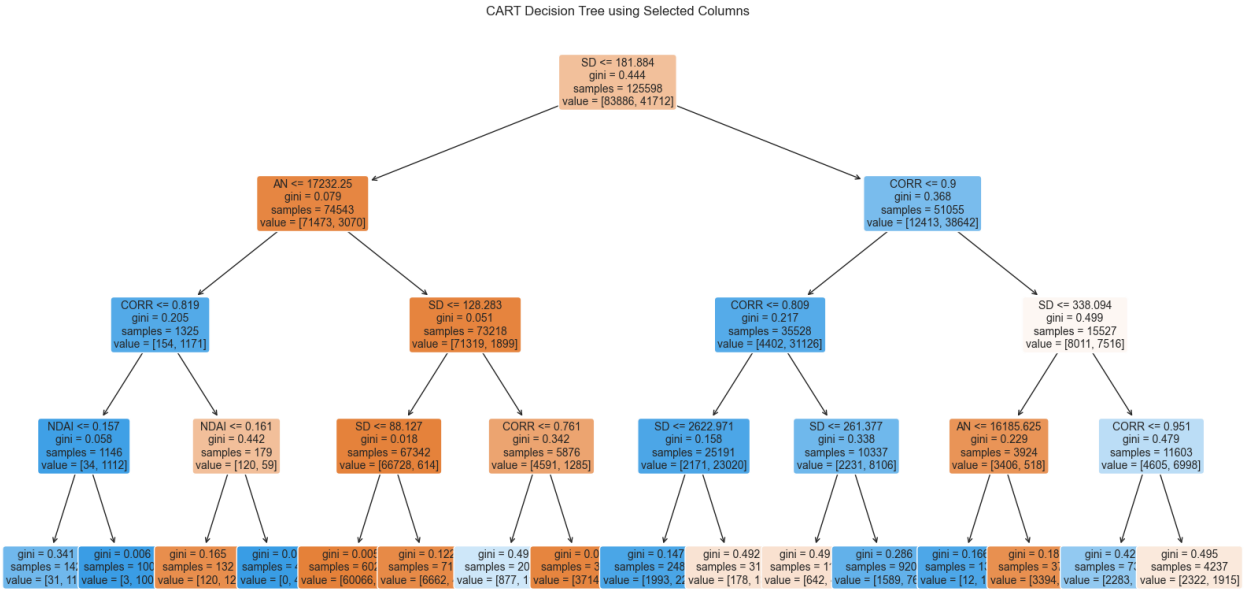


Figure 6: CART Model before Feature Engineering

We can also see that an accuracy of x% can already be achieved with this simple CART model, which is already an outstanding performance for such a “simple” model. Nonetheless, in the next chapters, we will continue to engineer more features and at the end we will set up a corresponding CART model to show the differences between the original dataset and the newly created dataset with the new features.

3.2 Feature Creation

In this chapter, we will take a closer look at which new features we can add to the data set for better prediction. The basic idea here is based on the findings of the EDA, where we can clearly see on the images that clouds form in larger clusters and these clusters then always form a wave/cloud front together. This means that the pixels in our surroundings are usually also clouds, which means that if we integrate this environmental information into our current pixel/data point, we obtain more in-depth information about the surroundings and can take this into account in the classification. The only disadvantage that arises is the choice of train and test split, since a single data point now already contains information about the environment and therefore no random split may be made, but entire regions/images must be assigned to either the train or the test data set.

Now to the exact implementation of the above principle. For each of the existing features such as SD, CORR, NDAI, DF, CF, BF, AF and AN, a variant is now calculated for the mean value, max

value and min value from the environment. After several attempts, the environment was calculated with a 3x3 window, a 5x5 window and a 9x9 window, whereby the mean, min and max values were calculated from the 80 surrounding points in the last area.

After this step, the data set has therefore grown significantly, as one NDAI feature has now become a total of 9 additional NDAI features with NDAI_3_Mean, NDAI_3_Min, NDAI_3_Max, NDAI_5_Mean, NDAI_5_Min, ..., NDAI_9_Max. This data set will therefore grow to over 80 features as a result of this step, which means that a large number of features are available across the environment, which must be reduced again in the modeling if necessary.

3.3 Transfer Learning with Autoencoders

We attempt to develop new features that efficiently aggregate information including surrounding pixels using autoencoders. There are a wide range of hyperparameters, including the structure of the autoencoder (such as the number of layers), learning rate, and number of embeddings, and tuning all of these using grid search is computationally too expensive, so we first determined the structure of the autoencoder and learning rate.

We set up three patterns for the structure of the autoencoder: a model with three fully-connected layers ("Fully Layer model"), a model with two fully-connected layers and one 2D convolutional layer ("Combined model"), and a model with one fully-connected layer and two convolutional layers ("CNN model"). We also set up three patterns for the learning rate: 0.001, 0.005, and 0.01, and calculated the validation loss (mean squared error) for the nine combinations. As a result, we found that Fully Layer model with 0.01 learning rate was the best.

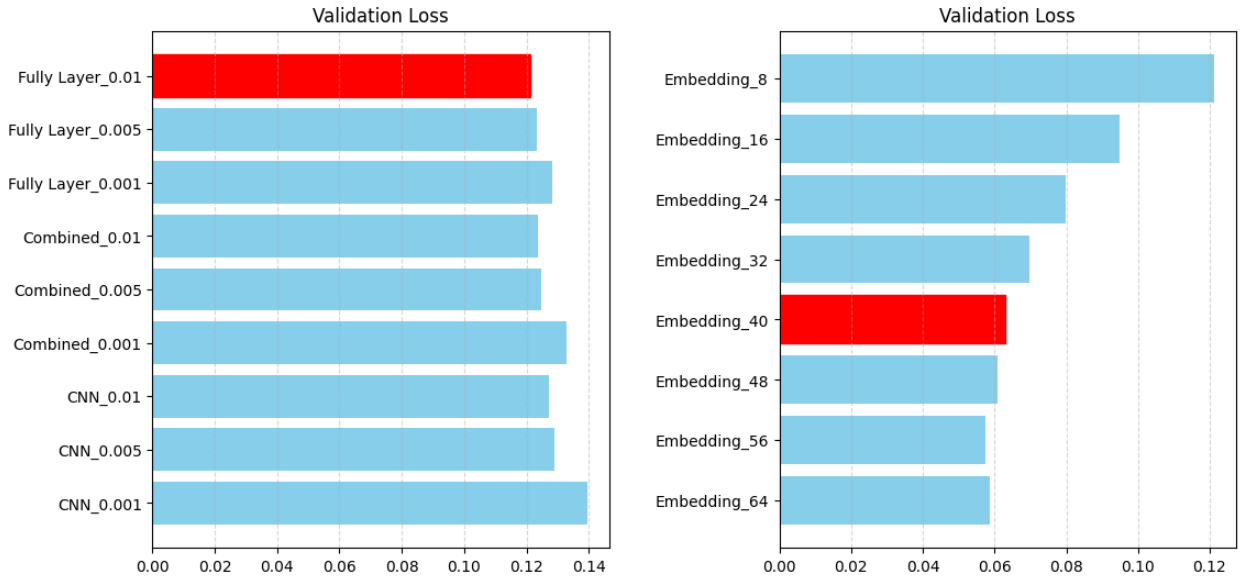


Figure 7: Comparison of Validation Loss

In addition, we tried nine different numbers of embeddings from 8 to 72, and found that increasing the number of embeddings to 40 greatly reduced the validation loss, but the subsequent decrease was limited. Considering the negative effects on modeling of increasing the number of less informative

features, we decided on 40 as the number of embeddings. As a result of these efforts, we were able to reduce the validation loss to 0.06, almost the half of the initial score.

To avoid the data leakage, the images used for training and those used for validation were completely separated (the same image was never used for both training and validation, which is different from the provided code. As autoencoders use information from surrounding pixels, if this method is not used, data leakage will occur to a certain extent). In addition, given that the modeling part uses some unlabeled images for prediction, we trained the autoencoder using 132 of the 161 unlabeled images, and obtained the embeddings of the labeled images.

From now on, we will refer to the features obtained through this process as "ae1" to "ae40".

3.4 Summary

This chapter now discusses the steps taken in feature engineering. In the first chapter, we saw that the original features with the CART model already enable relatively good performance and that the two features SD and CORR are particularly important. In chapters 3.2 and 3.3, we have now created further features using the surrounding pixels and an autoencoder. To conclude this chapter, we will now analyze the feature importance with the CART model again and observe the changes there.

The next figure shows the result, which is performed on the old and on the new features. It can be seen that the differentiation of the clusters is hardly determined by the "old" features, but is mainly based on the new features from 3.2 and 3.3. It can also be seen that the features from the surrounding pixels have the greatest importance and are located relatively high up in the tree. The features of the autoencoder are present in the tree, but at a lower position.

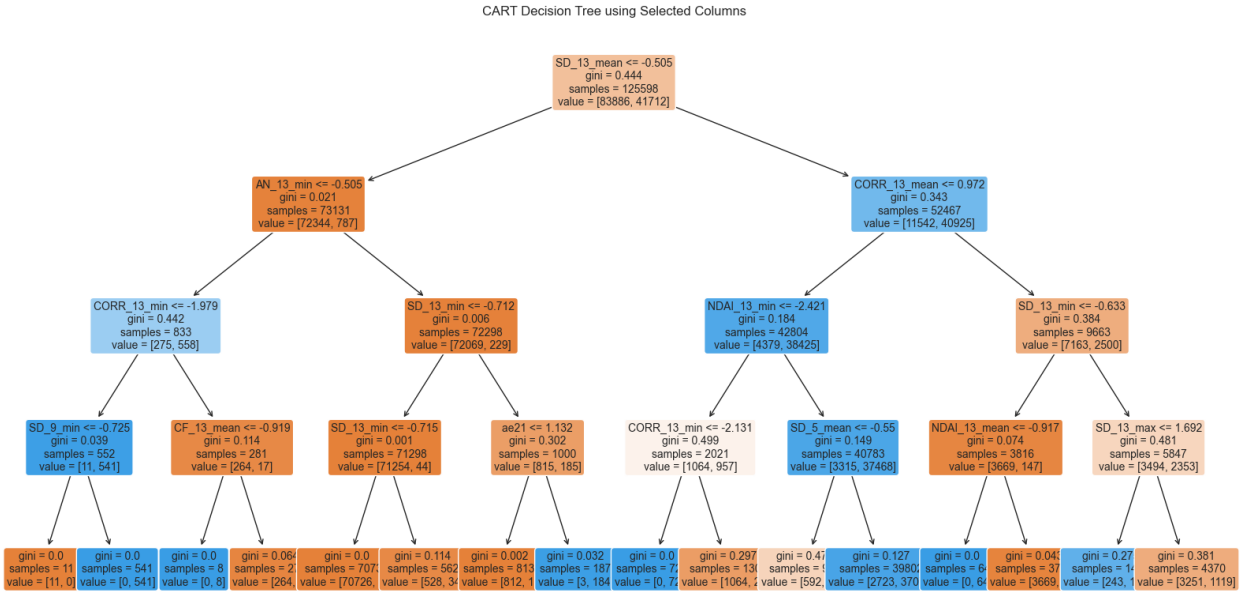


Figure 8: CART Model after Feature Engineering

Really interesting and a little surprising is, that the CART-Modell get a accuracy on the test dataset from 98% only based on that tree. Such a high accuracy show us, that the detection of clouds is possible with the combination only of that few data points and guarantee still a incredible performance on the test set.

Overall, it can be said after this analysis that the feature engineering from this chapter has once again greatly expanded and modified our data set. Fortunately, the analysis also shows that the steps from 3.2 and 3.3 have led to the addition of features with great importance, which should make classification in the next step much easier and produce significantly better results.

4 Predictive Modeling

In this chapter, we will take a closer look at the modeling and examine the corresponding model results. For all models, we assume that the prediction of non-cloud and cloud is equally important, which allows us to use the standard accuracy and loss functions in the models.

In addition, the training and testing split is performed as explained in the previous section (the two images ("O13257.pnz", "O12791.pnz") are divided into four parts and cross-validation is performed, with the other image ("O13490.npz") used as test data). For k-cross-validation, we have introduced a column called Quadrant, which can be used to perform a corresponding 4k cross-validation by always using one quadrant as validation and three quadrants as training. This also means that the test and validation data sets are only slightly dependent on each other due to the edge pixels of the quadrants, but are otherwise completely independent of each other and can therefore still achieve good results.

In the following, we decided to program a Random Forest, a K-Nearest Neighbor (KNN) classifier, and a LightGBM. We decided not to use a complex model such as a deep neural network (DNN), as the performance of this model was significantly worse than the above models in initial tests. In the subchapters, we discuss the functionality of each model, the strengths of the model, the reasons for the selection, and the performance on our dataset.

4.1 Model Development and Assestment

4.1.1: Random Forest

In this case, we choose a random forest algorithm as one of our classifiers because we already see such good results with the CART model in Feature Engineering, which signaled to us that a relatively easy differentiation is possible with a tree-based method.

A random forest is an ensemble learning method that builds multiple decision trees during training and merges their predictions to produce a more robust and accurate final result. Each tree is trained on a randomly sampled subset of the data (with replacement) and considers only a random subset of features when determining splits. This approach introduces diversity among the trees, reducing the likelihood of overfitting and increasing overall generalization. In classification tasks, the final prediction is made by aggregating the outputs of the individual trees. Random forests are particularly effective for complex datasets because they capture non-linear relationships and interactions between features while maintaining interpretability through analysis of feature importance.

To identify the best set of variables for the random forest, we perform a hyperparameter search using the 4k cross-validation described above, where we try different values for the parameters "n_estimators", "max_depth", "min_samples_split", "min_samples_leaf", and "bootstrap". In the end, we see that we still get the best results when we use a max_depth = 4 and n_estimators = 20, which means that the decision tree also performs best when we use only a really simple version of it.

With this parameterization, we get the following confusion matrix:

[Confusion Matrix]

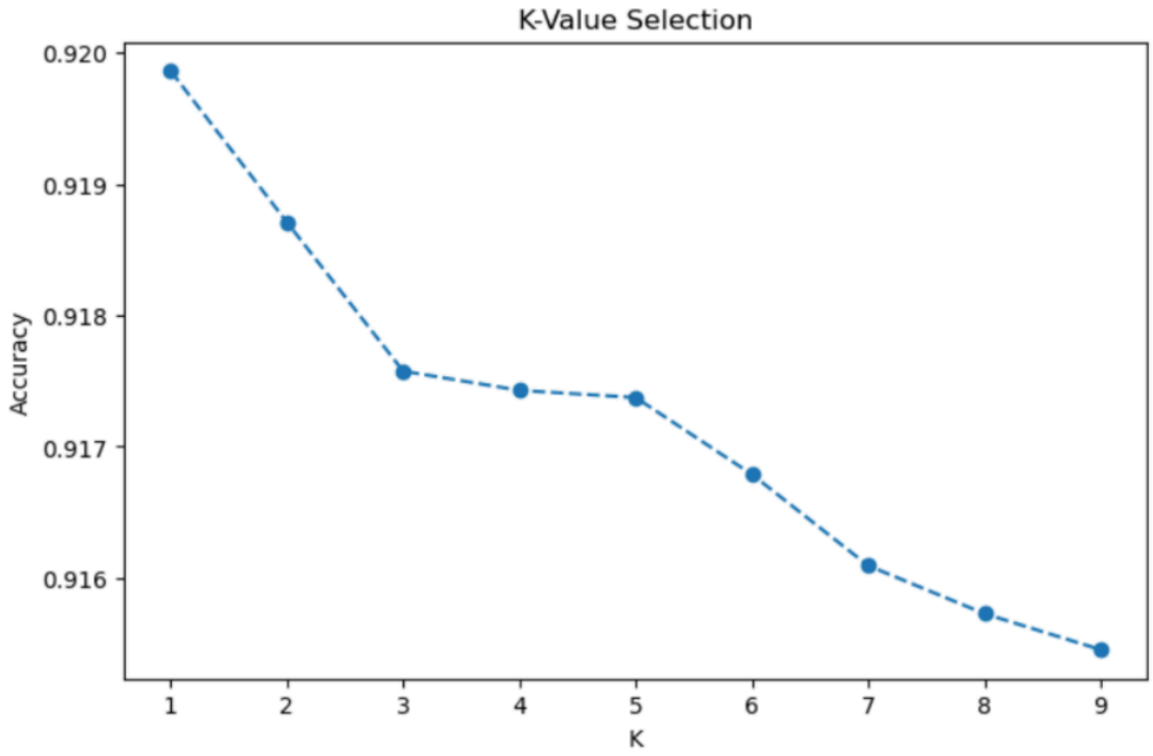
This shows that we have an accuracy of 98% on the test dataset, which is quite impressive and shows a good performance at all on the test dataset, which shows us that the model is neither overfitted nor underfitted. The interesting part is that this performance of the best decision tree is still slightly worse than the CART algorithm on the full dataset.

4.1.2: K Nearest Neighbors

k-Nearest Neighbors (kNN) is a non-parametric supervised machine learning algorithm commonly used for classification and regression tasks. The process begins by choosing the number of neighbors to consider (k). Then, for every unseen data point, we evaluate the Euclidean distance between every datapoint. For the first k datapoints closest to our selected unseen datapoint, we identify the most common class among the nearest k labelled datapoints. This class is then our predicted value for the unseen datapoint.

In our model building process, we predict the class label of every datapoint in our test set based on nearest neighbors in our training set.

For hyperparameter tuning of k, we evaluate the test accuracy for a range of values of k. This is shown in Figure X where test accuracies are shown ranging from X% to Y% for k values 1 to 9. The graph shows that as k increases testing accuracy decreases dramatically from ~92% to 91.75% for k = 1 to 3. The testing accuracy then plateaus from k = 3 to 5 where test accuracy is maintained at around 91.75%. As k increases further from k = 5 to 9, the testing accuracy drops gradually from 91.75% to 91.55%. We decided to select k = 3 as the optimal value to both maximize the accuracy while restricting overfitting to our training data.




```
Test Accuracy: 0.9174

Classification Report:
              precision    recall  f1-score   support

        -1.0       0.94      0.91      0.93     30945
         1.0       0.89      0.92      0.91     23699

    accuracy                           0.92     54644
   macro avg       0.91      0.92      0.92     54644
weighted avg       0.92      0.92      0.92     54644


Confusion Matrix:
 [[28253  2692]
 [ 1823 21876]]
```

4.1.3: Light Gradient Boosting Machine ("LightGBM")

LightGBM is a model that aims to improve the performance of decision trees by using gradient boosting. One of its features is that it uses both training and validation data during the training

process, and it repeatedly learns to reduce the error between the predicted and actual values of the validation data. The disadvantage of this approach is that the amount of calculation increases in proportion to the amount of data, but LightGBM speeds up this process using various methods. As with random forest, if the data is unbalanced (the number of cloudy cells and non-cloudy cells differs greatly), we need to do specific adjustments to improve accuracy, but in this analysis, no such problems were observed.

For hyperparameter tuning, we conducted cross-validation using four quadrants with the number of leaves as a candidate for [15, 31, 63] and the learning rate as a candidate for [0.05, 0.1, 0.15], and the number of leaves was selected as 31 and the learning rate as 0.15. In addition, for training the final model, we trained the model using the data in the fourth quadrant as the validation data. The results of the fitting to the test data are as follows. The accuracy of 98.18% is the highest of the three models (in the next section, we will discuss the comparison of the models, including this point).

(Accuracy: 98.18%)	Actual: Unclouded	Actual: Clouded
Prediction: Unclouded	41,579	1251
Prediction: Clouded	244	39,009

4.2 Best model, Diagnostics and Feature Importance

When comparing the performance of the three models on the test data, all of them performed extremely well, but LightGBM slightly outperformed the other two models.

	Random Forest	KNN	LightGBM
Accuracy	98.15%	94.45%	98.18%

However, choosing the best model based solely on the prediction performance for a single image can lead to neglecting the risk of overfitting. Therefore, we also check the results of the 4K cross-validation using the training data (when the four quadrants are highly independent, the results of cross-validation suggest the degree of generalization performance of the model for other images). As a result, we can also see that LightGBM consistently demonstrates higher performance than random forest and KNN. This suggests that LightGBM predictions are effective for identifying clouds using satellite images.

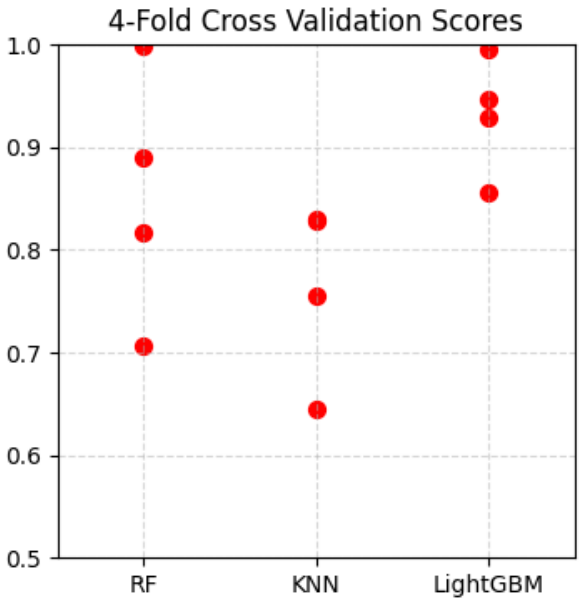


Figure 9: Comparison of the Cross Validation Scores

Next, let's check the feature importance and permutation importance to see what variables play a major role in the LightGBM prediction model. For both evaluation criteria, we can see that the variable "SD_13_mean" has a high degree of importance. This is consistent with the previous discussions that the distribution of SD varies largely for each cloud label. In addition, the fact that "SD_13_mean" was selected rather than "SD" strongly suggests that feature engineering using surrounding variables has led to improved the performance of the model.

If you check the other variables, you can see that the features engineered by the autoencoder starting with "ae" also contribute to a certain extent. In addition, it should be noted that none of the features we were given in advance had a high degree of importance. This suggests that the features we engineered clearly outperform existing features by making effective use of information from surrounding pixels.

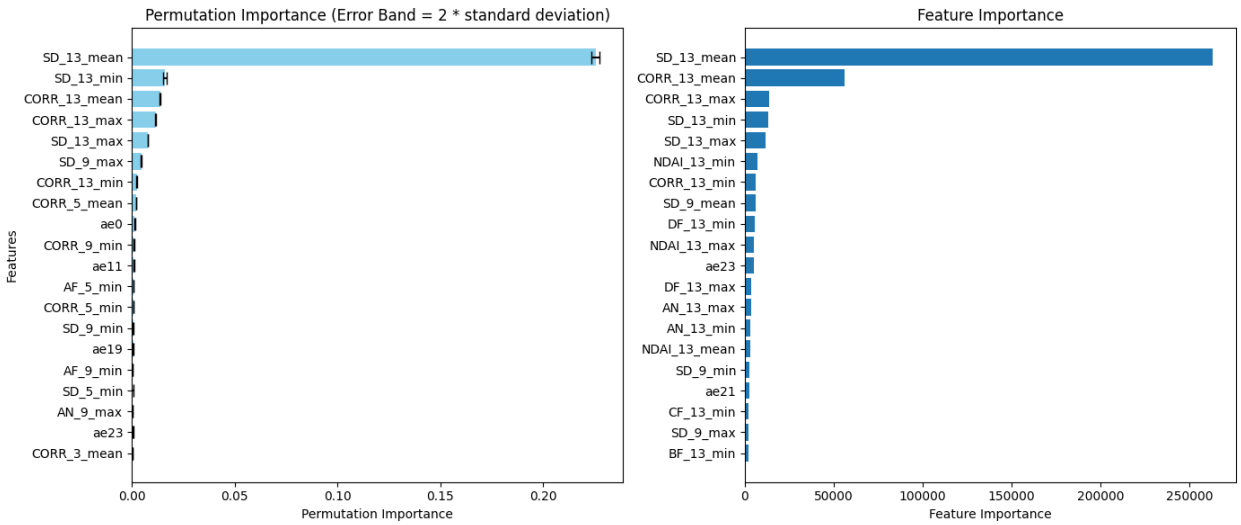


Figure 10: Feature importances of the lightGBM

4.3 Post-hoc EDA and Error Analysis

As post-hoc EDA, let's compare the results of the modeling using lightGBM with the actual expert labels. As the high accuracy shows, there is almost no difference between the two labels, but we can see that the predictions and actual results differ in (i) the edges and (ii) the center part of the images.

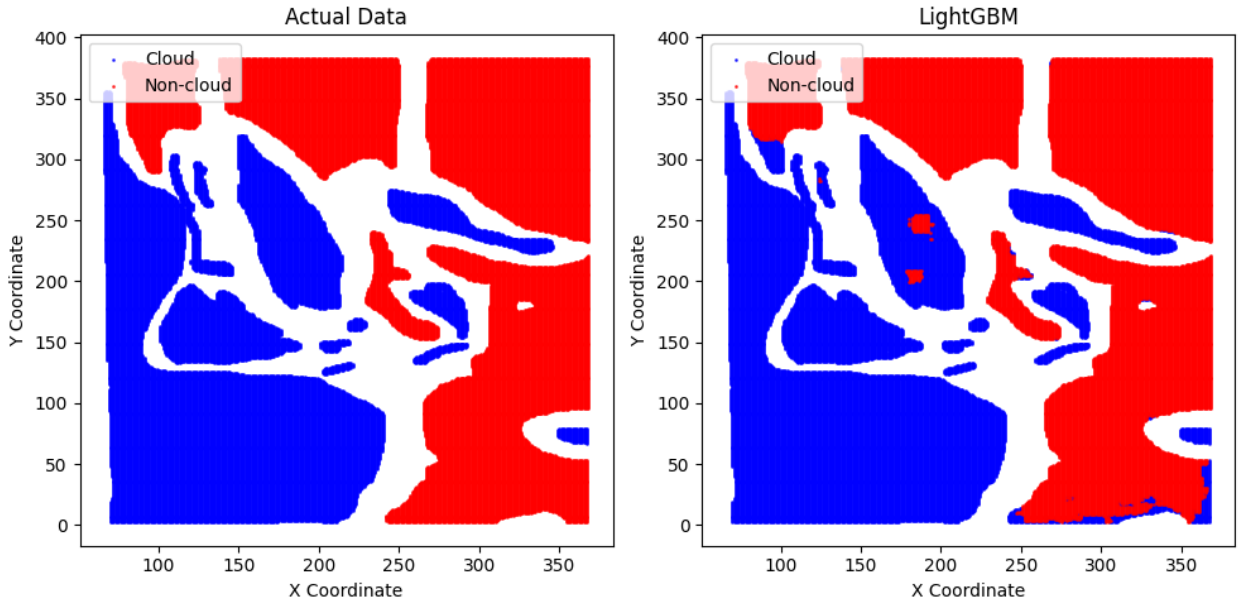


Figure 11: Comparison between the expert label and the prediction

The prediction errors at the edges of the image can be explained by the information constraint that it is not possible to obtain all the information from the surrounding pixels. In fact, when patches

are created using autoencoders, etc., if there are missing values in part of the patch at the edge of the image, the image is flipped and the missing values are filled in, and the impact of errors caused by this method cannot be ignored.

The prediction error in the center of the image can be discussed in relation to the main variable "SD_mean_13". Looking at the distribution of this feature value for the test data, the center part is colored red, which is similar to the area without clouds. For the remaining areas, "SD_mean_13" is similar to the actual expert labels, so it can be said that this feature has high performance as a variable, but it does not have the performance to predict all the expert labels.

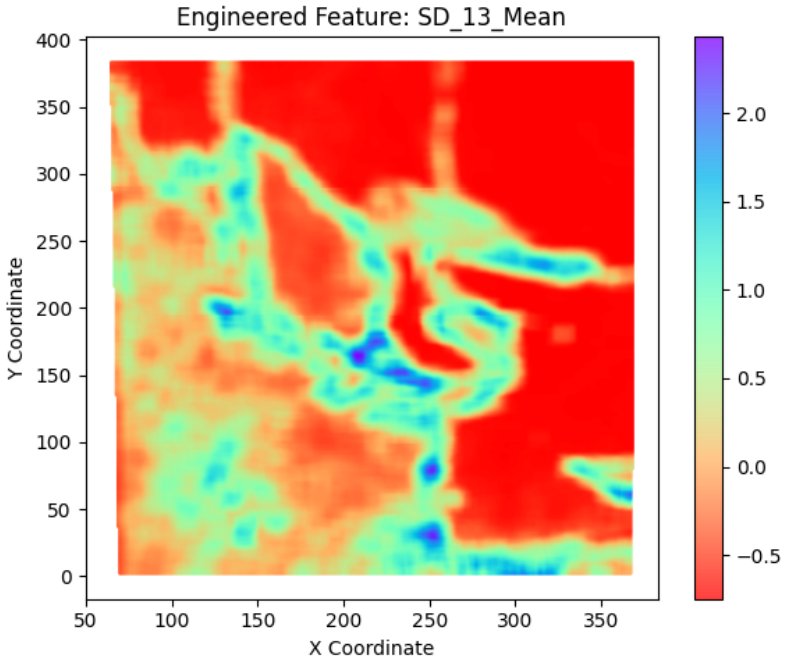


Figure 12: the Distribution of the Engineered Feature "SD_mean_13"

4.4 Model Generalization and Stability

In order to evaluate the effectiveness of a predictive model for future data, we can approach it from the perspective of how stable the model's conclusions are in the face of changes in the data, and whether the results of the survey are independent on the assumptions we used in the analysis.

As a first approach, we added noise to the data in two ways.

First, let us add noise to the variables used in the prediction (e.g., engineered features). We add random noise generated from a normal distribution to 10% of each data point (the variance of the random noise is equal to the sample variance of the feature values). As a result, we can see that there is almost no effect on the accuracy, although there is a slight increase in misjudgments in the middle part of the data.

Next, we consider adding noise in the same format to the observation data and feature values we had at the time we started the analysis. After adding noise to the data, we engineered features in the same process, including autoencoder features (as the autoencoder has been trained using unlabeled

data, there is no need to relearn the model), and then made predictions using LightGBM. Looking at the results, we can see that, unlike the previous results, the predictions have not maintained their accuracy at all. However, this does not only point to the low stability of this particular prediction model. When random noise is added to the data at a rate of 10%, errors propagate during the process of generating features while incorporating information from surrounding data. Therefore, this result suggests that the approach of utilizing the surrounding pixels has a structural weakness in terms of stability in that the effects of data observation errors and other factors propagate over a wide range.

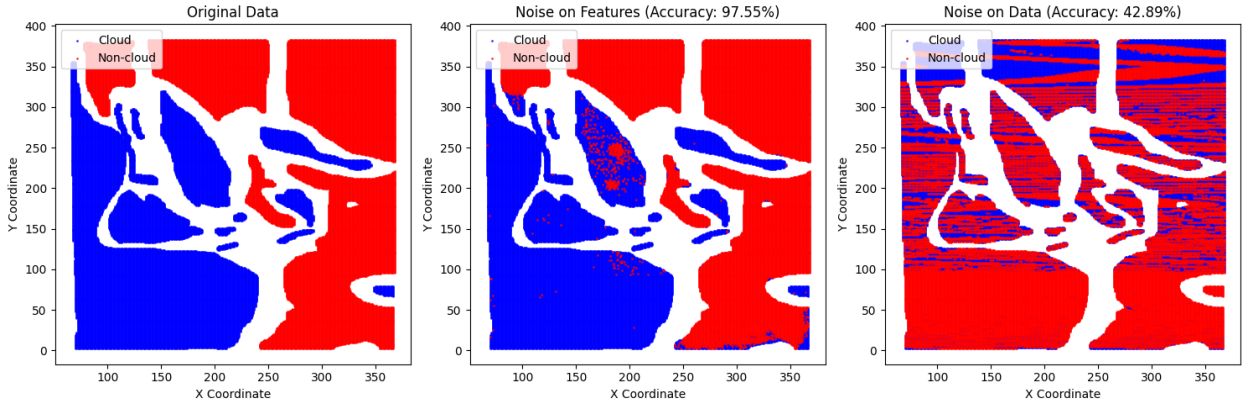


Figure 13: Stability Check: Add Noise on Features and Data

Finally, we will examine the major assumption we used in our predictions that we used "O13490.npz" as the test data, out of the three images with expert labels. After tuning the parameters and making predictions in exactly the same way as before, we can see that the accuracy obtained from the two images ("O13257.pnz", "O12791.pnz") is slightly lower than the original result. However, the fact that the correct answer rate is over 87% in all cases where the experiment was conducted using either image as test data can be said to support the fact that there is a certain degree of stability in the prediction model using LightGBM.

Image_ID (test data)	O013490.npz (default)	O013257.npz	O012791.npz
Accuracy	98.18%	91.27%	87.23%

4.5 Sanity Check

Finally, as a sanity check, let's check the results of the predictions we made for the image without a label, after obtaining the feature values. Note that the image used in this case ("O120204.npz") was not used for training the autoencoder.

Since it is difficult to discuss the validity of the model based on the results of LightGBM alone, as there are no expert labels on this image, let us evaluate the validity by showing the results of all three prediction models. The results of the three prediction models are similar in terms of the general direction, and they all conclude that there are no clouds in the upper left and there are clouds in the lower part. However, it is also important to note that the three models' predictions

differ slightly in their conclusions for the central part of the image (LightGBM can be said to be a result that is somewhere between Random Forest and KNN, and it can also be interpreted as a moderate conclusion).

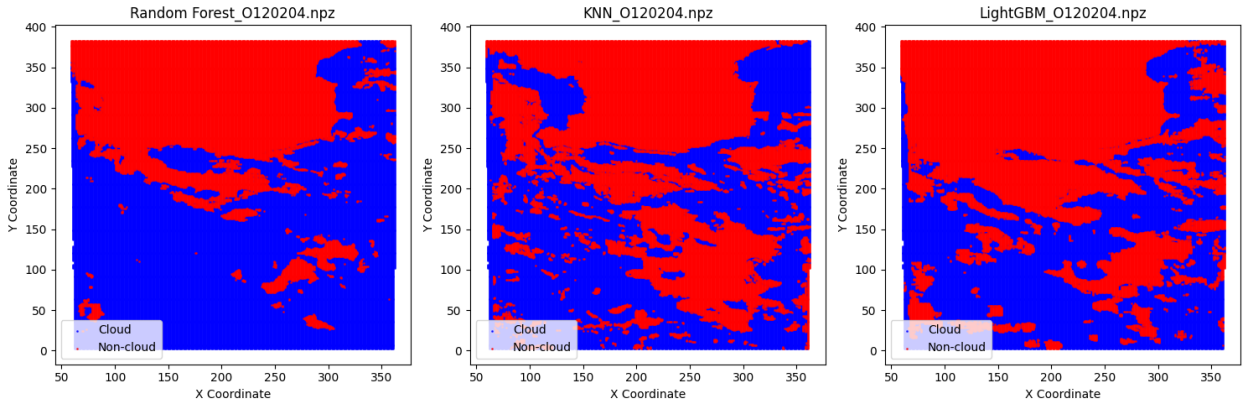


Figure 14: Sanity Check: Prediction Results for unlabeled data

As a conclusion of the sanity check, while it is positive that the three prediction models are likely to be able to capture the general trend, it also shows that the high performance obtained with the test data may not necessarily be obtained for all images. There are multiple reasons for this, but one of the major factors is that there are only three images with expert labels (and only two images for training data), so the risk of overfitting is potentially high (although it should be noted that we have improved the generalization performance using various methods, such as cross-validation with four quadrants).

5 Conclusion

7 Collaborators

I certify that I have only collaborated with my group members.

6 Academic honesty statement

To: Bin Yu

I declare that the work presented in Lab 2 is entirely my own and my group members. We ourself designed and performed all the data analysis, methods and procedures presented in this report. We ourself wrote all the texts, wrote codes for data exploration, feature engineering, modelling, and produced all the figures in this report. We have included and documented all the procedures in the workflow and the results can be fully reproduced. Wherever we included the work from others, we cited the sources. We used LLM specifically for improved grammar for report writing style.

By submitting this lab report and all github material, I certify that we have complied with the academic integrity standards as set in lab 2 instructions.