# Project 2 - XGBooster Shot

Johnny Antoun (0679537) Jose Pliego (2716768)

November 2021

## Contents

# 1 Data Collection and Exploration

## 1.1 Paper summary

The impact of increasing amounts of atmospheric carbon dioxide on Earth's climate is an important issue today. Prevalence of carbon dioxide is a major topic due to its association with increasing surface air temperatures, with the strongest dependencies in the Arctic region. In the polar regions, cloud coverage is important in modulating the sensitivity to increasing surface air temperatures, making cloud detection useful. Unfortunately, existing cloud detection algorithms, such as the MISR level 2 top-of atmosphere algorithm (L2TC), do not perform well in these regions due to the similarity between electromagnetic radiation emitted from clouds and snow/ice-covered surfaces.

This study describes NASA scientists and statisticians attempt to devise a cloud detection algorithm that works well in polar regions. Instead of searching for cloud pixels algorithmically, they opt to model the surface because it doesn't change materially from different views. They rely on measurements from the Multiangle Imaging SpectroRadiometer (MISR) that differs from traditional multispectral sensors that take measurements in a single view. The MISC cameras cover a 360-km-width swath of Earth's surface that extend across daylight side of the Earth from Arctic down to Antarctica in approximately 45 minutes with a total 233 distinct, but overlapping, such swaths. MISR completes accumulation of data from all 233 paths in around 15 days with each path subdivided into 180 blocks.

The data collected in this study consists of 6 data units from consecutive 10 MISR orbits of path 26 (rich in surface features). Three data units are excluded because the surfaces were open water and around 71.5% of valid pixels in the data are labeled by experts. The data collected by MISR is massive which makes standard classification difficult to implement. In addition, clustering is not ideal either because data units (three consecutive blocks) could be entirely cloud-covered or cloud-free.

The proposed algorithm, enhanced linear correlation matching (ELCM), is based on thresholding three features: correlation of MISR images of the same scene from a different angle (CORR), standard deviation of MISR nadir camera pixel values across a scene ($SD_{an}$) and normalized difference angular index (NDAI) which relates to changes in a scene with changes in view direction. The CORR and $SD_{an}$ cutoff values are set for fixed values during operational processing whereas the NDAI threshold is either kept the same or updated at a new data unit based on a data-adaptive algorithm. Pixels are then labelled as cloudy in two scenarios: $SD_{an} < threshold_{SD}$ or CORR $> threshold_{CORR}$ and NDAI $< threshold_{NDAI}$. At a high level, this labeling relies on the fact that clouds are rarely both extremely smooth and relatively weakly forward scattering. Next, Fisher's QDA trained from the ELCM labels is used to provide an estimate of confidence of cloudiness which is especially useful in partially cloudy locations.

To assess the different algorithms, results are compared against expert labelled pixels. In addition, a simple-minded offline SVM classifier is trained on randomly sampled expert labels. After checking the results, the scientists and statisticians confirmed ELCM performed significantly better than LT2C algorithms. The offline SVM performed worse than the ELCM but was in similar range as LT2C. While the ELCM-QDA did not improve the agreement rate relative to ELCM, it provides additional color represented by the probability labels. Some ELCM data units have low agreement $\sim 70\%$ (low CORR in non-cloudy regions due to poor terrain data registration).

In conclusion, the study shows that the three physical features (CORR, $SD_{an}$, NDAI) contain enough information to separate clouds from ice-/snow-covered surfaces. The ELCM algorithm combines classification and clustering in a way that makes it suitable for real-time, operational MISR data processing and is more accurate than existing MISR operations algorithms. Careful combination and application-specific modification of existing statistical methods, provided solution to a difficult problem.

## 1.2 Data Summary

From this section and throughout our work, we "trimmed" the images so that they are perfectly rectangular. To do this, we filtered each image to have $x$-coordinate values between 70 and 368. By doing this cleaning step, we lost 2902 pixels in total (0.84%), with each of the three images loosing a similar proportion of pixels.

We decided to do this trimming because some features like Gabor filters require a rectangular matrix as an input (C. H. Chen (2006), Haghighat (2015), Mouselimis (2021)).

As a first step in the exploratory data analysis, let's take a look at the distribution of the expert labels in each image and in aggregate. The distribution is shown as a percentage frequency table (table 1). We can see that the distribution varies greatly from one image to another, with image 1 having a similar amount of cloud and non-cloud pixels, image 2 having a majority of non-cloud pixels, and image 3 having most of its pixels unlabeled.

| Label | Img1 | Img2 | Img3 | Total |
|---|---|---|---|---|
| No cloud | 37.54% | 43.98% | 29.35% | 36.96% |
| Unlabeled | 28.68% | 38.24% | 52.17% | 39.7% |
| Cloud | 33.77% | 17.78% | 18.48% | 23.34% |

Table 1: Label distribution in the data.

We can plot labels in the $(x, y)$ plane to visualize them. In figure 1 we can see that cloudy regions and surface regions are separated by unlabeled regions, where the expert could not determine if pixels correspond to clouds or ice/snow.

(a) Labels for image 1. (b) Labels for image 2. (c) Labels for image 3.

Figure 1: Image labels plotted in the $(x, y)$ plane. White pixels correspond to clouds, gray pixels to land surface, and black pixels are unlabeled.

Figure 1 shows strong spatial dependence present in the images. Cloudy pixels are generally surrounded by cloudy pixels, and the same pattern repeats for surface/non-cloudy pixels. This presents a challenge for pixel classification because we cannot treat the different pixels as independent and identically distributed. We must account for the spatial dependence in our classification models so they can generalize well to future unobserved images.

## 1.3 Exploratory Data Analysis

To get a sense of how the predictors are related between them and with the labels, we include a correlation plot of the relevant variables in figure 2. The label column is encoded as 1 if the pixel is a cloud and -1 if it is not, so a positive correlation between label and a predictor can be roughly interpreted as an increase in the predictor being associated with an increase in "cloudiness." For this figure, we remove the unlabeled pixels to retain the numerical interpretation of the labels and the correlation with the predictors.

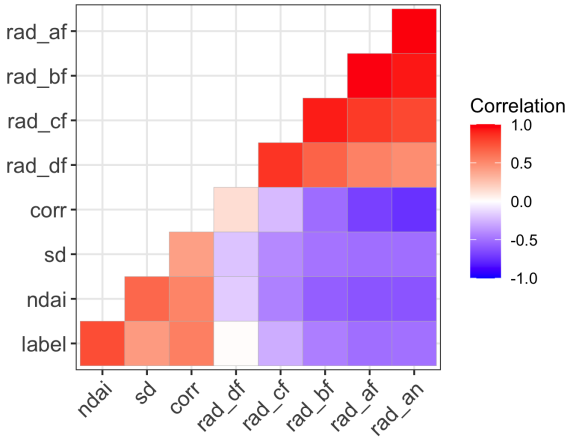



Figure 2: Visual representation of Pearson's correlation matrix.

Figure 2 shows a positive correlation between label and the author-defined variables `ndai`, `sd`, and `corr`, and also shows a negative correlation between label and most radiance measurements. It is also interesting to note that the author-defined variables are negatively correlated with the radiance measurements while positively correlated among themselves. Similarly, radiance measurements are positively correlated. This was expected since these variables are all measuring light reflection at different angles.

To take a closer look at some of these relationships, figure 3 shows the distribution of the author-defined variables and `rad_an` for the two pixel classes. The figure includes density estimates and the median for each class. All variables were centered and scaled. For the variable `sd`, we use a logarithmic transformation because the original variable is heavily right-skewed.

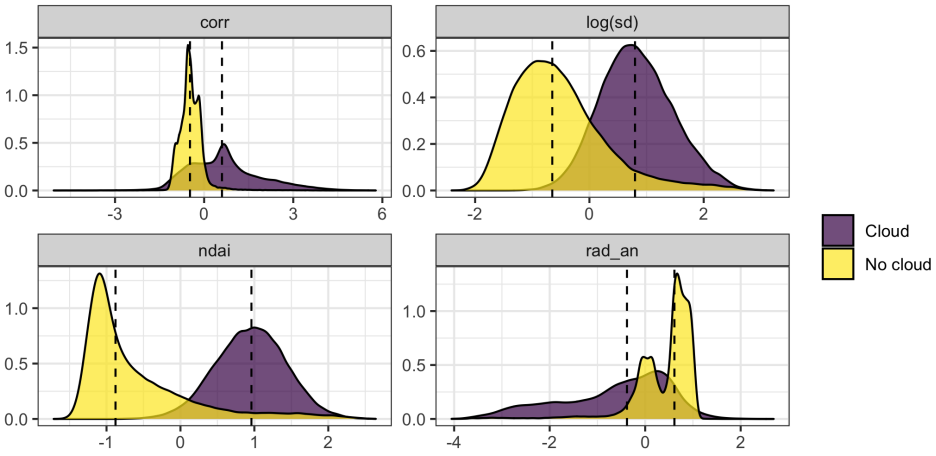



Figure 3: Distribution of author-defined variables and nadir radiance (rad_an) across pixel labels (median values highlighted).

In figure 3 we can see that the distributions are visibly different for both classes. The differences and the distance between the medians are greater in the plots of `log(sd)` and `ndai`. It is interesting to note that Shi et al. (2008) use fixed cutoff thresholds for `sd` and `corr` but not for `ndai` since the "appropriate thresholds vary from one data unit to another." In our case, it seems like `ndai` is the most discriminating feature. We formalize this notion of "best features" on section 2.

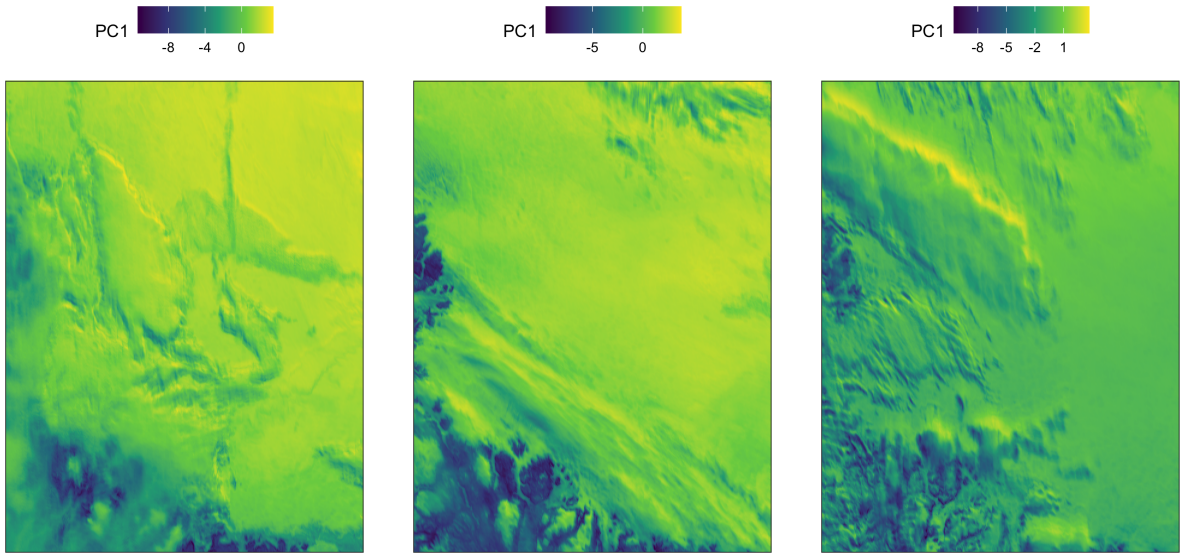
We include two scatterplots between the predictors that convey interesting properties in figure 4. The relationship between `ndai` and `sd` is shown in panel 4a. We see that the two variables are positively correlated (as was shown in figure 2) but also see that the spread of `sd` increases as `ndai` increases. Similarly, panel 4b shows the negative correlation between `corr` and `rad_an`, while also showing that the variance in `rad_an` increases as `corr` decreases. We chose to include these two scatterplots because other correlated variables show a more straightforward linear relationship without the heteroscedasticity property.


(a) Scatterplot between `ndai` and `sd`.

(b) Scatterplot between `corr` and `rad_an`.

Figure 4: Scatterplots for two pairs of highly correlated features.

To conclude the exploratory data analysis, we use Principal Component Analysis to reconstruct the three images using information from all the predictors. Since the principal components do not use label information, we obtain the scores and loadings using all the pixels. In figure 5 we can see the first principal component scores plotted in space. Note that the first principal component shows spatial dependence between neighboring pixels. This detail is interesting because the $(x, y)$ coordinates were not used to calculate the principal components, so it supports the authors' claim that some features convey implicit spatial structure when defined using local patches of pixels.




(a) PC1 representation for image 1. (b) PC1 representation for image 2. (c) PC1 representation for image 3.

Figure 5: First principal component scores plotted in the $(x, y)$ plane.

The first two principal components account for 80.7% of the variance in the dataset. Using the principal components as predictors instead of the original features may be useful in the modeling stage. As we saw in this analysis, some of the original predictors are highly correlated and using a set of orthogonal predictors (principal components) may yield better model results.

Finally, to support the claim that the principal components contain some of the spatial structure present in the data, we calculate the median and interquartile range (IQR) of the first two principal components by label. We choose IQR as a measure of spread and the median as representing the center because the distributions are not symmetric or unimodal.

| Label | PC | Median | IQR |
|---|---|---|---|
| No cloud | PC1 | 1.69 | 1.94 |
| No cloud | PC2 | -0.50 | 0.88 |
| Cloud | PC1 | -0.86 | 3.06 |
| Cloud | PC2 | 0.98 | 2.12 |

Table 2: Summary statistics for principal components.

In table 2 we see that the first two principal components have more spread in cloudy pixels, and the medians are different for both types of labels. This suggests that principal components may be useful in classification models. In section 3 we formalize these claims and assess model performance.

# 2 Preparation

## 2.1 Data Split

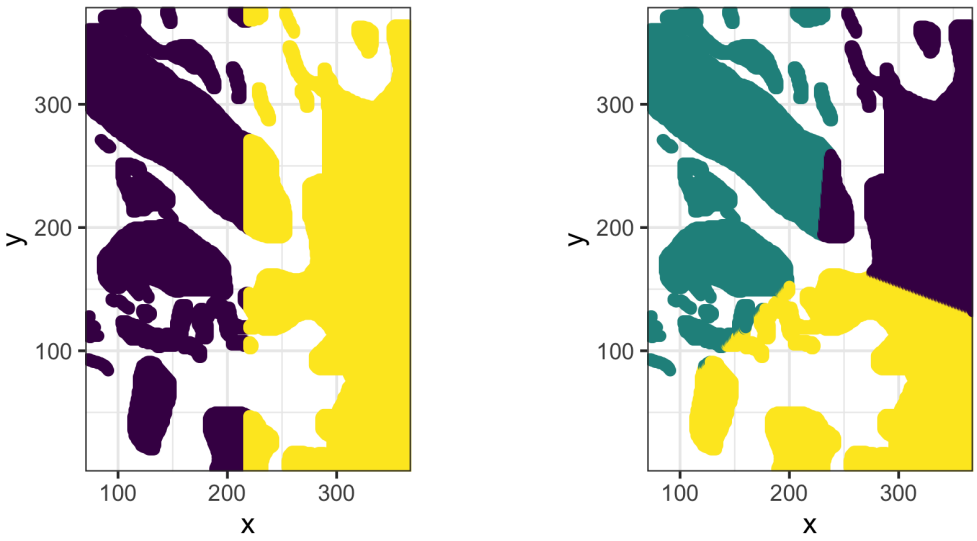
In this section, we propose two methods to split the data into three sets (train, validation, and test). The two methods are block splitting (Valavi et al. (2019)) and clustering (Brenning (2012)). Conventional random splitting is not ideal for spatial data since pixels close to each other tend to be more similar. This means that we could be filtering information from the training to the test set with neighboring pixels, and model assessments in the test set would be overly optimistic.

The idea behind block splitting is dividing each image into similar sized blocks. The blocks should be big enough so that the spatial correlation between blocks is relatively low while simultaneously capturing the spatial correlation structure of the pixels within each block. We also have to be careful and make sure that our test set does not contain an overwhelming majority of cloud or non-cloud labels, since a test set with any of these characteristics would not be a good proxy for how a model will perform with unseen data.

We chose to split each image into two blocks and take one random block as test set, another random block as validation set, and the four remaining blocks as training data. This corresponds to a 66.6%-16.6%-16.6% training-validation-test split. After trying different block sizes, we settled for two blocks in each image so we can capture a diversity of pixel characteristics in each set, while also making sure that there are enough pixels to train classification models. When we use cross-validation to fit the models in section 3, then the train-test split becomes a 83.3%-16.6% split. The blocks were defined including the unlabeled pixels, even though these pixels are not useful for classification. We decided to include these pixels on the block design because creating blocks that account for missing pixel patches is complicated and our blocks are big enough so that none of the sets has an overwhelming majority of unlabeled pixels.

For the clustering split, we use k-means clustering on the $(x, y)$ coordinates for each image (see Brenning (2012)). We identify two main advantages of using k-means over block splitting. The first is that we can remove the unlabeled pixels so that the clustering algorithm only uses coordinate information of the pixels that will actually be used for modeling. The second advantage is that we can specify any number of clusters, while the block splits restrict us to using a number of blocks specified as rows$\times$columns.

Following the same reasoning as in the block splits, we choose to split each image in 3 clusters, taking two of the nine clusters as test set, one as validation set, and the remaining six as training set. This corresponds to a 66.6%-11.1%-22.2% training-validation-test split. We also center and scale the coordinates before clustering so there is not more variation in one of the dimensions.




(a) Block splits for image 3.

(b) K-means splits for image 3.

Figure 6: Block and clustering splits for image 3.

Figure 6 shows a visual representation of the clustering and block splits for image 3. The unlabeled points were removed after doing the block splits. We can see how k-means is not restricted to a square grid structure and can take more flexible shapes. It is important to note that there is still some information leakage between points that are close to the group borders, but it is considerably better than using purely random splitting (see Valavi et al. (2019), Brenning (2012), C. H. Chen (2006)).

## 2.2 Baseline model

We report the accuracy of a trivial classifier that predicts every pixel as non-cloudy on the test and validation sets, using both splitting methods mentioned before. This trivial classifier serves as a baseline comparison for the classification models used in section 3. The accuracies are shown in table 3.

| Splitting Method | Set | Accuracy |
|---|---|---|
| Block | Validation | 78.8% |
| Block | Test | 20.2% |
| K-means | Validation | 81.8% |
| K-means | Test | 86.3% |

Table 3: Accuracy of a trivial classifier.

We can see that the trivial classifier performance is greatly dependent on the set used to evaluate. Trivial classifiers perform well when the data is unbalanced, in this context when a set is composed of mostly non-cloudy pixels. In table 3 we see that the validation sets for both splits, as well as the test set obtained with k-means, consist of mostly non-cloudy pixels so the classifier has relatively high accuracy. However, the accuracy is still far from 100% so the hope is that a well designed classification model will outperform the trivial classifier. We also see that our splitting methods work well since we manage to capture different pixel behaviors in each set.

## 2.3 Feature selection

To formalize the selection of "best features" mentioned in section 1, we fit single-predictor models using each of the predictors and assess this models' performance. We fit a logistic regression on each training set (block

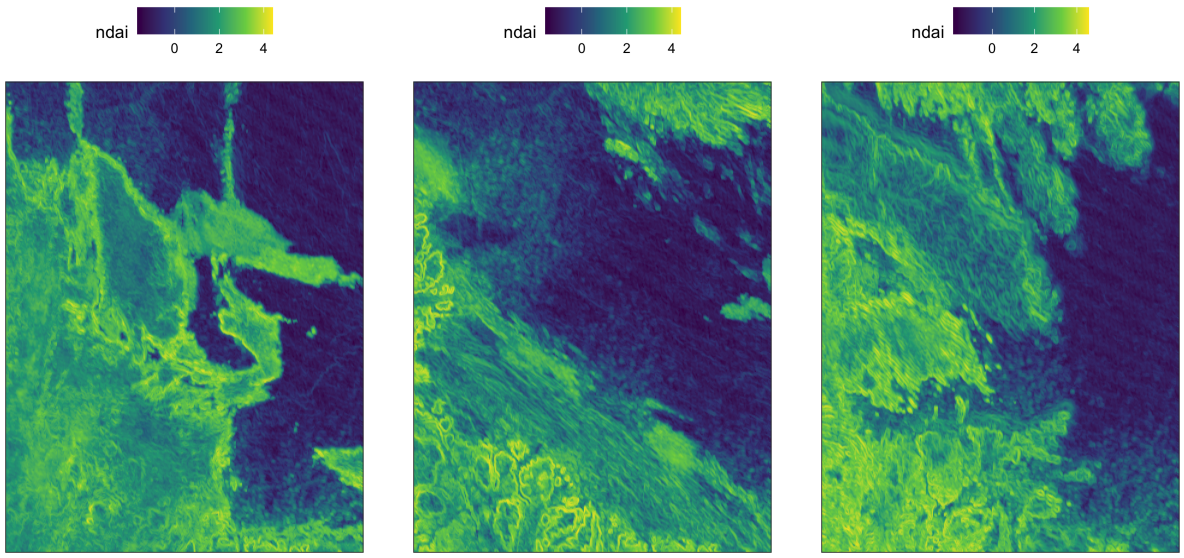
and cluster splits), calculate the area under the curve (AUC) of the ROC Curve in the validation sets, and average this values to rank the predictors. We choose AUC as a selection criterion because this metric takes into account all possible probability threshold cutoffs, and we choose logistic regression as an initial model because this model returns a coefficient for the predictor and we can relate the probability of a pixel being cloudy or not with a higher or lower value of each predictor.

The results are shown in table 4. We can see that `ndai` is the clear winner, followed by `sd`, both of which are author-defined predictors. Going back to figure 3, we expected `ndai` to perform well because the distributions between the two classes were well separated.

| var | AUC Block | AUC K-means | Avg AUC |
|---|---|---|---|
| ndai | 1.00 | 0.99 | 0.99 |
| sd | 0.94 | 0.91 | 0.92 |
| rad_af | 0.82 | 0.93 | 0.87 |
| rad_bf | 0.87 | 0.83 | 0.85 |
| rad_an | 0.71 | 0.88 | 0.80 |
| corr | 0.31 | 0.99 | 0.65 |
| rad_cf | 0.87 | 0.20 | 0.54 |
| rad_df | 0.86 | 0.05 | 0.45 |

Table 4: Single-predictor model performance.

As a visual assessment, we show the values of `ndai` plotted in space for the three original images. In figure 7 we see what figure 3 suggested, that higher values of `ndai` are associated with cloudy pixels. This is further supported by the logistic regression model having a positive coefficient $\hat{\beta}_{ndai} = 2.008$.




(a) Image 1. (b) Image 2. (c) Image 3.

Figure 7: `ndai` values plotted in the $(x, y)$ plane.

The visual cues in figures 3 and 7, along with the quantification of AUC in table 4, suggest that `ndai` is the most useful feature for predicting pixel labels. The AUC results suggest `sd` and `rad_af` as the second and third most important features, but this can change based on how the predictors behave when used simultaneously in a classification model.

### 2.4 Cross Validation

To ease the implementation of different classification algorithms using the splitting methods discussed before (block splits and k-means), we used the R package `tidymodels` (Kuhn and Wickham (2020)). First of all, we created two splitting functions, one for each method, that return a split object as created by the package `rsample` (Silge et al. (2021)). This split object can then be used with the `tidymodels` constructors.

As a brief overview, our function takes as inputs a training set, split method, classification algorithm, a set of metrics, and additional parameters related to these objects, and returns the metrics obtained in each cross validation fold.

## 3 Modeling

For the modeling, we use the block splits created in section 2 as test and training sets. For the training set, we combine both the original training and validation set. Because we have to split the training set for cross validation, we rearrange the training set by shifting the $x$ coordinates for the different images. This way, whether we use block splits or k-means, blocks and clusters will mainly contain pixels from the same image.

In terms of binary classifiers, we decided to fit logistic regression, LDA, QDA, SVM, XGBoost and Naive Bayes. For each classifier, we do 9-fold cross validation in the two separate scenarios, data split using block or k-means. We choose to use 9 folds as it falls in between the standard 5 and 10. We found it to be rational to go smaller than the initial split (testing/training/validation) of 6 as we are using all 9 of the segments for testing. We did not want to go too small on the blocks as we would get blocks with only cloud or no clouds at all.

| | Logistic Regression | LDA | QDA | Naive Bayes | SVM | XGBoost |
|---|---|---|---|---|---|---|
| | 0.68 | 0.68 | 0.65 | 0.74 | 0.71 | 0.74 |
| | 0.77 | 0.77 | 0.84 | 0.65 | 0.65 | 0.73 |
| | 0.71 | 0.68 | 0.79 | 0.54 | 0.65 | 0.71 |
| | 0.87 | 0.87 | 0.82 | 0.88 | 0.93 | 0.91 |
| | 0.89 | 0.90 | 0.96 | 0.91 | 0.94 | 0.93 |
| | 0.86 | 0.87 | 0.84 | 0.98 | 0.98 | 0.97 |
| | 0.94 | 0.93 | 0.99 | 0.99 | 0.91 | 0.90 |
| | 0.76 | 0.78 | 0.80 | 0.87 | 0.92 | 0.93 |
| | 0.83 | 0.84 | 0.86 | 0.91 | 0.97 | 0.97 |
| average | 0.81 | 0.81 | 0.84 | 0.83 | 0.85 | 0.87 |

Table 5: Classification CV Errors for Block Splits.

| | Logistic Regression | LDA | QDA | Naive Bayes | SVM | XGBoost |
|---|---|---|---|---|---|---|
| | 0.61 | 0.88 | 0.99 | 0.87 | 0.80 | 0.99 |
| | 0.86 | 0.89 | 0.60 | 0.70 | 0.88 | 0.83 |
| | 0.89 | 0.72 | 0.80 | 0.90 | 0.67 | 1.00 |
| | 0.77 | 0.63 | 0.61 | 0.99 | 0.99 | 0.96 |
| | 0.95 | 0.74 | 0.95 | 0.80 | 0.81 | 0.77 |
| | 0.35 | 0.99 | 0.98 | 0.75 | 0.99 | 0.91 |
| | 0.98 | 0.99 | 0.87 | 0.96 | 0.72 | 0.71 |
| | 0.93 | 0.46 | 0.84 | 0.96 | 0.96 | 0.96 |
| | 0.75 | 0.91 | 1.00 | 0.68 | 0.59 | .75 |
| average | 0.79 | 0.80 | 0.85 | 0.85 | 0.82 | 0.88 |

Table 6: Classification CV Errors for K-means Splits.

**Assumptions** :

Logistic Regression:

LDA: Estimates a multivariate distribution for the predictors separately for the data in each class (Gaussian with common covariance matrix). Bayes' theorem is used to compute the probability of each class, given the predictor values.

QDA: Estimates a multivariate distribution for the predictors separately for the data in each class (Gaussian with separate covariance matrices). Bayes' theorem is used to compute the probability of each class, given the predictor values.

SVM: Cost and Sigma parameters are tuned using grid.

XGBoost: Tree_depth, min_n, loss_reduction, sample size mtry, learn rate are all tuned using grid.

Naive Bayes: We assume independence of predictors.

For selecting the ROC threshold, generally we have to consider the type of problem at hand and whether it is more costly to commit false positives or false negatives. That is, the problem entails cost function optimization (specifically minimization). After obtimizing cost, we are able to obtain a threshold and to derive the respective specificity and sensitivity. In this case, we will assume that the cost of a false negative (detecting no cloud when there is a cloud) and a false positive (detecting cloud when there is no cloud) as equally costly due to the fact that both errors, overestimating and underestimating clouds, lead to incorrectly predicting increases in temperature. We therefore find the point on the ROC that has the minimum Euclidean from the top left corner in the ROC curve (i.e. sensitivity = 1, 1-specificity = 0) and the associated threshold at that point.

# 4 Diagnostics

Building from the models used in section 3, we now do a deep dive into the XGBoost algorithm. We choose XGBoost because it performed well in the previous section, it has many tunable hyperparameters that are interesting to explore, and it has performed well in many online prediction competitions (T. Chen and Guestrin (2016)). We work only with the testing and training data sets obtaiined with the block splits because, as we saw in section 3, the splitting method does not make a huge difference in model performance. First, let's do an overview of XGBoost and boosting in general. The overview presented here is mainly adapted from the original paper by T. Chen and Guestrin (2016).

The idea behind tree boosting algorithms is to combine simple classification trees and create a final classifier taking into account all these weak classifiers. Boosting is known to prevent overfitting and produce overall better classification accuracy (Freund and Schapire (1999)). Different boosting algorithms differ in the way they incorporate the weak classifiers. For example, the first boosting algorithm called AdaBoost (short for Adaptive Boosting) by Freund and Schapire (1995), combines the classifiers and adaptively reweighs the data so misclassified points get more weight.

XGBoost minimizes a regularized loss function, where the regularization term penalizes both the number of leaves in the tree and the vector of leaf weights. At each step, the algorithm uses a greedy approach to select the tree structure that yields a larger loss reduction. Furthermore, XGBoost incorporates a shrinkage term that scales the weights and reduces the influence of each individual tree. Just like random forests, XGBoost samples features at each split. According to T. Chen and Guestrin (2016), the shrinkage term and feature sampling help prevent overfitting.

Implementation of XGBoost is attainable in R using the `xgboost` package by T. Chen et al. (2021). We now go over some of the tunable parameters in `xgboost`. Keep in mind that the argument names are not the same that are used when fitting XGBoost through the `tidymodels` interface.

- Number of trees used in the ensemble (`nrounds`).
- Number of randomly sampled features at each split (`colsample_bynode`).
- Minimum number of observations for a node to be split further (`min_child_weight`).
- Maximum depth of trees (`max_depth`).

- Shrinkage term (`eta`).
- Reduction in the loss function required to split further (`gamma`).
- Proportion of the data that is exposed to the fitting routine at each iteration (`subsample`).
- Iterations without improvement before stopping (`early_stop`).

To choose the optimal parameters, we fix the number of trees at 1000, we specify no early stopping and use a grid search to optimize the rest. Our first attempt was with no early stopping and 1000 trees, we kept these two parameters because the cross-validation estimates showed no signs of overfitting.

The final values for the parameters are shown in table 7. It is important to note that optimizing many parameters in a grid search results in sparse combinations tested when taking into account the multidimensional parameter space. To better select the grid, we use Latin Hypercube Sampling (Loh (1996)) available through the `grid_latin_hypercube()` function in the `dials` R package (Kuhn and Frick (2021)).

| Parameter | Value |
|---|---|
| `colsample_bynode` | 2 |
| `min_child_weight` | 8 |
| `max_depth` | 9 |
| `eta` | 0.0408 |
| `gamma` | 0.0191 |
| `subsample` | 0.944 |

Table 7: Optimal parameters for XGBoost.

We look at some diagnostics for the XGBoost fit. After settling on parameter values, we fit the model an all the training data set to obtain diagnostics. From an optimization point of view, we chose the negative log-likelihood as a loss function because it takes into account not just the class predictions, but also the probabilities for each class. Figure 8 shows the change in the training loss as the number of trees increases. We also include the loss obtained in the testing data.

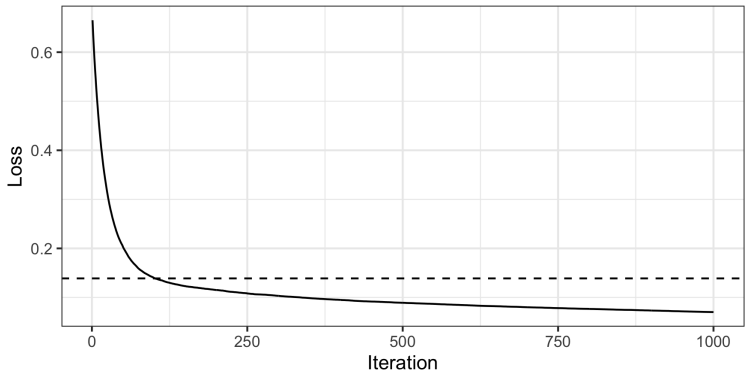



Figure 8: Log-likelihood loss for XGBoost.

We can look at the feature importance in the model. As an importance measure we use gain, which is the improvement in accuracy brought by a feature to the branches it is on. Figure 9 shows the variable importance plot. We can see that the author-defined variables take the top three spots. As we anticipated in section 2, `ndai` is the most important feature. It is interesting to note that `corr` is second in importance, when the individual predictor method ranked it as the third least important variable. This supports the authors' feature engineering and highlights the relevance of domain knowledge in complicated classification tasks.

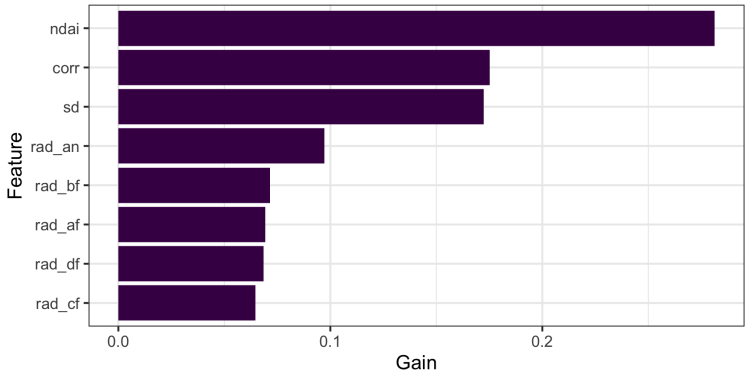
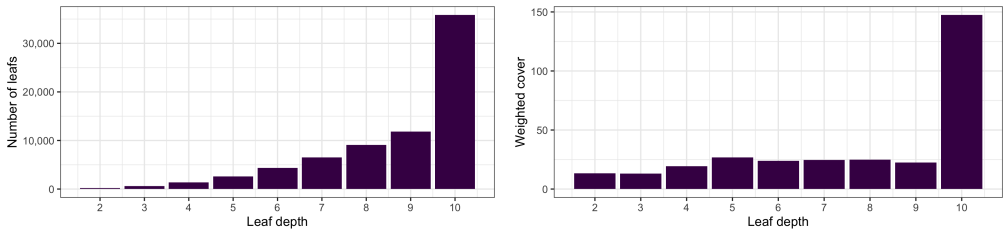



Figure 9: Feature importance plot for XGBoost.

To visualize the model complexity, we include two plots related to tree depth in figure 10. Figure 10a shows the distribution of the number of leafs according to depth level. We can see that the number of leafs grows exponentially with the depth. Figure 10b shows the distribution of the average weighted number of observations ending up in leafs at certain depths. We can see that the number grows greatly at the deepest level.




(a) Number of leafs at each depth level. (b) Weighted observations at each depth level

Figure 10: Visualization of XGBoost model complexity.

We can also plot the decision boundary using principal component analysis. To achieve this, first we calculate the principal components on the training data, create a dense grid in the plane spanned by the first two principal components, and project this grid back into the original space to obtain label predictions. The first two principal components explain 84.7% of the variance in the training data. This low-dimensional approximation of the decision boundary is shown in figure 11 and the probability threshold used is 0.5. We can see that the cloud prediction region is diagonal across the plane. It is important to keep in mind that this low-dimensional representation is not telling the whole story, the true decision boundary may look very different in the original feature space.

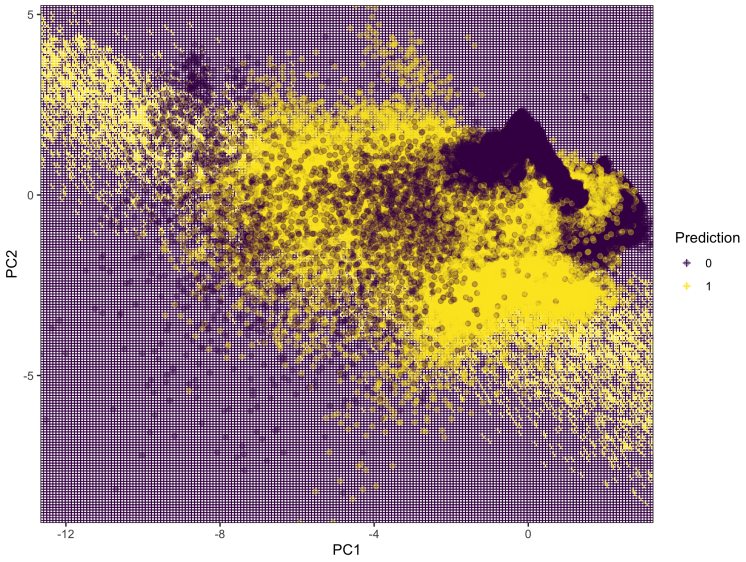



Figure 11: Decision boundary in the (PC1, PC2) plane.

Let's take a look now at how the model performs on the test set. With the test set we can see how our model generalizes to previously unseen data, verifying that we are not overfitting the training data and obtaining performance estimates for when the model is put in production. This is crucial because our model may seem too complex at first, having a large number of trees and large tree depth.

Fixing the threshold at 0.5, the confusion matrix in the test set is shown in table 8. The true labels are shown in the rows and the predicted labels in the columns. The overall accuracy is 95.7%, sensitivity is 96.0%, and specificity is 94.3%. This means that the model is making more mistakes when classifying cloudy pixels.

| | No Cloud | Cloud |
|---|---|---|
| No Cloud | 8,173 | 493 |
| No Cloud | 1,338 | 32,811 |

Table 8: Test set confusion matrix.

Figure 12 shows the cloud probability in the x-y plane. Comparing it to the true labels in section 1, we see that some pixels in the top left region have high cloud probability, when the true labels correspond to surface pixels. The second region that stands out is in the lower middle region, where we see a patch of low cloud probabilities. This figure is interesting because the probability map seems to reflect some of the textures that we expect to see in cloud satellite images, while the label maps are a lot smoother.

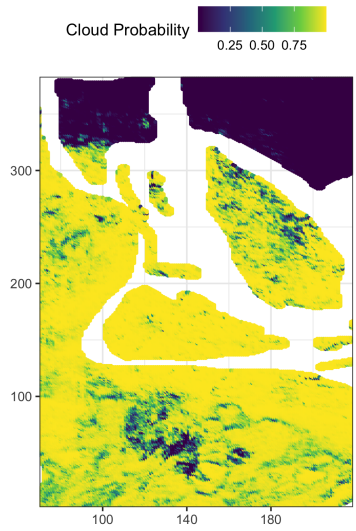
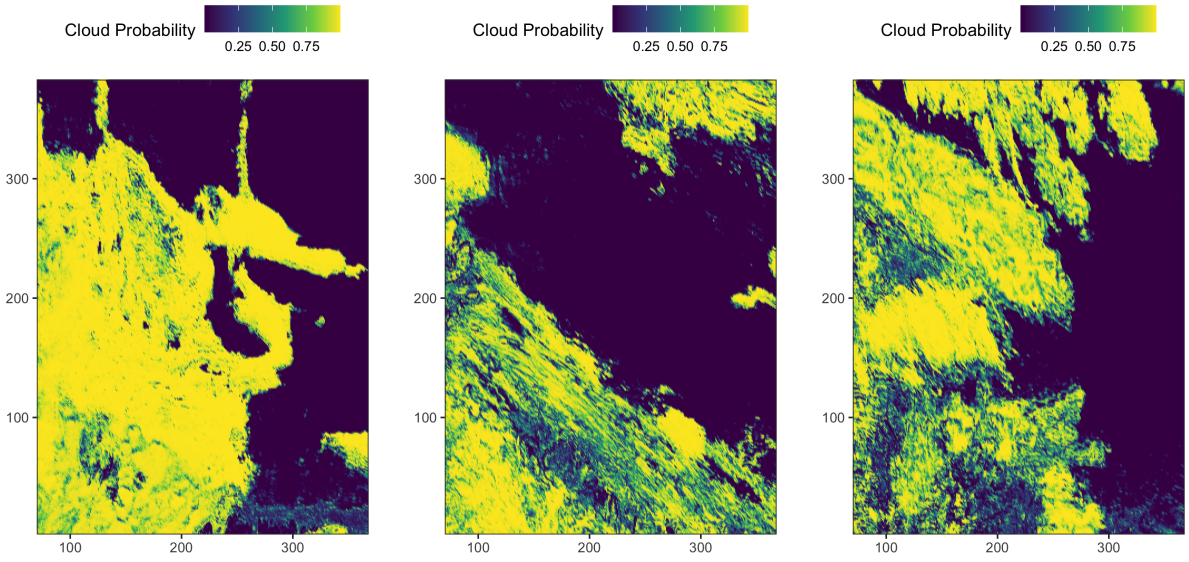



Figure 12: Test cloud probabilities in the $(x, y)$ plane.

Finally, we can use our model to get predictions for the original unlabeled pixels. The cloud probability maps for each image are included in figure 13. We would expect our model to have similar classification accuracy as in the test set, close to 95%.




(a) Image 1. (b) Image 2. (c) Image 3.

Figure 13: Cloud probability maps for each image.

# 5 Concluding Remarks

Overall, pixel classification for satellite images in polar regions is not an easy task. The similarities in surface and cloud pixels make traditional methods fail. In this project, we were able to create a very good classifier using boosted tree model, achieving high classification accuracy.

The good model performance relies heavily on the feature engineering we had access to in the training data. The variables defined by Shi et al. (2008) encapsulate most of the spatial structure in the data and allow us to not use explicit spatial models for classification.

We identify two main areas of opportunity for our classification model. First of all, we could consider reducing model complexity by reducing the number of trees or the maximimum tree depth in XGBoost. In this case, the high model complexity did not result in overfitting problems, but for other applications this may not be the case. Overall, we could see that XGBoost has effective ways to control complexity, avoid overfitting, and produce excelent classification models.

The second area of opportunity we identify is that we can take advantage of the spatial structure in the data to produce even better predictions. One way we could do this is by applying smoothing filters via k-nearest neighbors or other spatial methods to get more homogeneous predictions for nearby pixels. In our application we think that this extra step is not essential because clouds are not necessarily smooth objects, specially when seen at a high spatial resolution, but other applications or image resolutions may benefit more from the smoothing step.

# Bibliography


Brenning, Alexander. 2012. "Spatial Cross-Validation and Bootstrap for the Assessment of Prediction Rules in Remote Sensing: The r Package Sperrorest." In *2012 IEEE International Geoscience and Remote Sensing Symposium*, 5372–75. https://doi.org/10.1109/IGARSS.2012.6352393.

Chen, C. H. 2006. *Signal and Image Processing for Remote Sensing*. 1st ed. CRC Press.

Chen, Tianqi, and Carlos Guestrin. 2016. "XGBoost." *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August. https://doi.org/10.1145/2939672.2939785.

Chen, Tianqi, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, et al. 2021. *Xgboost: Extreme Gradient Boosting*. https://CRAN.R-project.org/package=xgboost.

Freund, Yoav, and Robert E. Schapire. 1995. "A Desicion-Theoretic Generalization of on-Line Learning and an Application to Boosting." In *Computational Learning Theory*, edited by Paul Vitányi, 23–37. Berlin, Heidelberg: Springer Berlin Heidelberg.

———. 1999. "A Short Introduction to Boosting." In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1401–6. Morgan Kaufmann.

Haghighat, Mohammad. 2015. *gabor: Gabor Feature Extraction*. https://github.com/mhaghighat/gabor.

Kuhn, Max, and Hannah Frick. 2021. *Dials: Tools for Creating Tuning Parameter Values*. https://CRAN.R-project.org/package=dials.

Kuhn, Max, and Hadley Wickham. 2020. *Tidymodels: A Collection of Packages for Modeling and Machine Learning Using Tidyverse Principles*. https://www.tidymodels.org.

Loh, Wei-Liem. 1996. "On Latin hypercube sampling." *The Annals of Statistics* 24 (5): 2058–80. https://doi.org/10.1214/aos/1069362310.

Mouselimis, Lampros. 2021. *OpenImageR: An Image Processing Toolkit*. https://CRAN.R-project.org/package=OpenImageR.

Shi, Tao, Bin Yu, Eugene E Clothiaux, and Amy J Braverman. 2008. "Daytime Arctic Cloud Detection Based on Multi-Angle Satellite Data with Case Studies." *Journal of the American Statistical Association* 103 (482): 584–93. https://doi.org/10.1198/016214507000001283.

Silge, Julia, Fanny Chow, Max Kuhn, and Hadley Wickham. 2021. *Rsample: General Resampling Infrastructure*. https://CRAN.R-project.org/package=rsample.

Valavi, Roozbeh, Jane Elith, José J. Lahoz-Monfort, and Gurutzeta Guillera-Arroita. 2019. "blockCV: An r Package for Generating Spatially or Environmentally Separated Folds for k-Fold Cross-Validation of Species Distribution Models." *Methods in Ecology and Evolution* 10 (2): 225–32.