Cloud Classification Project

The goal of this project was to explore and model cloud detection in the polar regions based on radiance recorded by the MISR sensor abroad the NASA satellite Terra. We attempted to build a classification model to distinguish the presence of cloud from the absence of clouds in 3 NASA images using several important the features.

These features included the y coordinates of the pixel, x coordinates of the pixel, expert label (+1 = cloud, -1 = not cloud, 0 unlabeled), NDAI (normalized difference angular index that characterizes the changes in a scene with changes in the MISR view direction), SD (the standard deviation of MISR nadir camera pixel values across a scene), CORR (the correlation of MISR images of the same scene from different MISR viewing directions), Radiance angle DF (MISR sensor camera angle at 70.5 degrees), Radiance angle CF (MISR sensor camera angle at 60 degrees), Radiance angle BF (MISR sensor camera angle at 45.6 degrees), Radiance angle AF (MISR sensor camera angle at 26.1 degrees), and Radiance angle AN (MISR sensor camera angle at 0 degrees).

Prerequisites

One must first install RStudio, load a new R notebook, and install the following R packages in a new R code chunk before running any commands:

```{r, message=FALSE}
library(ggplot2)
library(ggthemes)
library(GGally)
library(gridExtra)
library(ModelMetrics)
library(caret)
```

```{r, message=FALSE}
library(dplyr)
library(MASS)
library(randomForest)
library(ROCR)
library(mvnmle)
library("car")
library(reshape2)
```

For example, one can install the ggplot2 package by a simple call in the RStudio console window:

install.packages(‘ggplot2’)

Code

```{r}
img_1 = read.table('image_data/image1.txt')
names(img_1) = c("y", "x", "label", "NDAI", "SD", "CORR", "rad_angle_DF", "rad_angle_CF",
"rad_angle_BF", "rad_angle_AF", "rad_angle_AN")
```

Load in the image data, ensure correct columns are present, and rename the columns with the names above.

Run all code chunks below the data loading and expect a wait time of 1-2 hours.

How to Use

Our code consisted of Data Loading, Data Preparation, EDA, Modeling, and Diagnostics. Most of the function calls we used are base functions or from libraries mentioned above, whose documentation can be found online.

We created a generic cross-validation function called CVGeneric, which takes a generic classifier (as a string), training features (as a vector of strings), training label (a string), number of folds K (integer), a loss function (function call), and a training set data frame as inputs and outputs the K-fold CV loss on the training set.

```
```{r, cache = TRUE}
# CV accuracy, temporal split
CVgeneric(classifier = "glm" , trainFeats = c("CORR","SD","NDAI","rad_angle_AN"), trainLabels = "label",
foldsK = 5, lossFunc = mse, training = train_temp_new)
```
```

```
    fold1     fold2     fold3     fold4     fold5
0.9031767 0.9057220 0.9020412 0.9026478 0.9047250
```

Note that using CVgeneric with classifiers like KNN and SVM might take more than 2 hours to run one function call.

To reproduce the report, simply run all code chunks in the order presented in the R notebook.

Authors

Yiming Shi and Jessica Cherny

Acknowledgments

- Raaz Dwivedi, Yuansi Chen, Bin Yu
- StackOverflow
- Medium Blogs