Project Report

Initial Problem

The initial problem we were given was to match gathered weather data with gathered webcam pictures to produce known weather and try to create a program that predicts weather based on pictures. The prediction step was supposed to be done using machine learning and classifiers.

Data

The gathered data and pictures are based on hourly recording of webcam pictures and weather attributes. The weather data was 17 months of weather data from the Canadian government's weather website and the webcam pictures were gathered from Kat Kam's website (with permission) for the same time period. The weather data was separated by month, and to get them all in one dataframe, we concatenated them.

Initial Cleaning

The cleaning step is always one of the biggest and most time consuming steps in the data science and analytics world. It was the same for this project as well. The first thing that we did for the cleaning of the weather data was making sure that we chose rows that did not have "NA" for the "weather" column. Then we dropped columns that were unnecessary, such as wind direction. The next thing we did was addressing the issue with the human-generated labels for weather data. The labels given were not generalized, but extremely specific. To change this we created 6 categories: Clear, Cloudy, Rain, Ice, Snow, Thunderstorms. After that, we reduced the 39 initial categories into 6 but with some weather conditions having multiple labels (ex: "Rain,Snow,Cloudy"). In the next part we modified the webcam images in the initial folder with initial modifying and shortening of filename to get it into the YYYYMMDDHHMM format to match the format we wanted. Then we changed the weather data's date/time format to YYYYMMDDHHMM as well, so both dates would match when joined.

Dataframe Creation

The weather data was separated by month, and to get them all in one dataframe, we concatenated them. For each image, we used scipy.misc.imread to give each picture as input and get an array of RGB triplets for each pixel to get the pixels of each image in a row. Then we added a date

attribute to each row, so that we could join the 2 dataframes later on. The final step was to join the two dataframes and this was done using the date columns available in both dataframes. Then we wrote the final dataframe to a csv file to use further in the analysis part.

Secondary cleaning

To get the RGB pixel triplets in the right shape, we needed to convert them somehow. One of the options was to expand each triplet into three columns. This option did not make much sense as the running time would be a lot and also the number of columns we got would get out of hand. What we decided was to take the RGB triplet and take them out of the weird brackets they were in ([[R,G,B]), convert them to integers and get a singular value for each pixel. This singular value must be a unique number (like a hash index). We had two options for this: RGB color integer value and grayscale. We chose RGB color integer value with the formula:

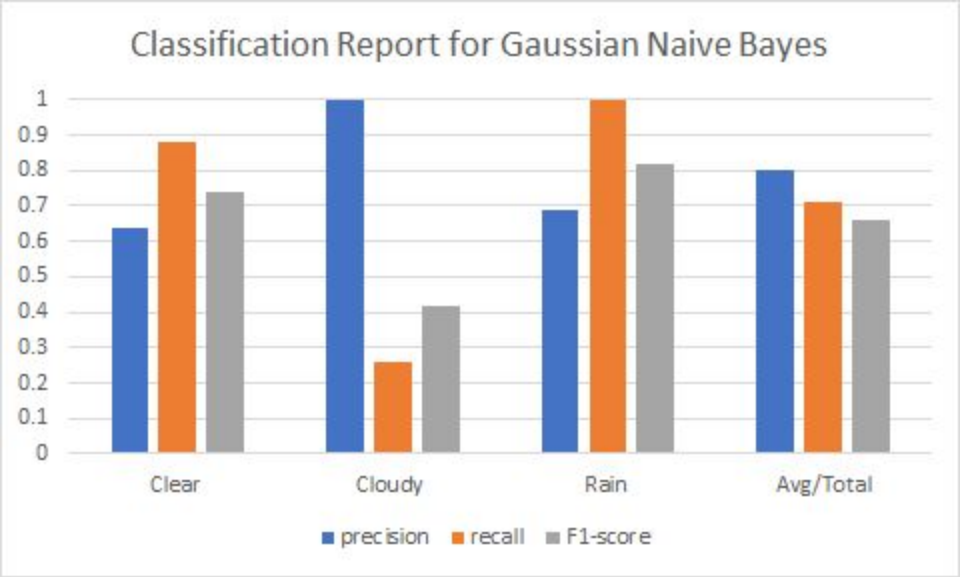
$$(R) * 256^2 + (G) * 256^1 + (B) * 256^0$$

This means that the highest intensity will be put on Red, Green and Blue will almost have no effect on the unique number.

After reading the CSV, we dropped 4 unnecessary columns (mostly date columns), applied the RGB triplet conversion function and separated the X and y values

Machine Learning Tools

Initially we did not address the multi label classification problem and just applied machine learning classifications tools. The initial tools we chose were Gaussian Naive Bayes Classifier, K nearest neighbours classifier and Support Vector Classifiers. Out of the three,the were Gaussian Naive Bayes Classifier showed the highest accuracy score (a little below 75%) and K nearest neighbours classifier and Support Vector Classifiers performed about the same (between 60% and 50%). After a lot of research we did not find specific machine learning tools for images or RGB values (except for tensorflow which uses deep learning) to improve the accuracy score and make the model more efficient. Then we looked at the multi label classification problem at hand and decided to use classification tools that dealt with multiple labels y values ("Rain,Snow,Cloudy,Clear"). After doing a lot of research about these classifiers we decided to go with Multi label K nearest neighbours classifier. However, we had to convert the single labels to multiple labels and after researching we chose Multi Label Binarizer, which would give binary values to each of the 6 possible categories for each row. With MLkNN we got an average accuracy score of 55%, therefore we decided to ignore the Multi Label classification problem altogether and return to Gaussian Naive Bayes Classifier.



In this chart we can see the precision, recall and f1 scores. Precision is if we predicted Clear or Cloudy or Rain, how often was it Clear or Cloudy or Rainy. Recall is what percentage of actual Clear weather or actual Cloudy weather or actual Rainy weather did we find correctly. The F1 score can be interpreted as a weighted harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.

Analysis and what we learned

One of the patterns we saw from the machine learning classification tool's accuracy score was how low it was compared to the accuracy scores that the in-class lectures models got (over 90%). The inaccuracy for predictions can have many reasons such as:

1. Improper relabeling of weather labels (even though more than 4 hours was spent trying to research and figure out a labelling system)
2. Improper classifiers (maybe neural networks and deep learning can be used next time)
3. Improper image and pixel extraction (we only got a 192 columns which was the width of each image, instead we should have gotten 256*192= 49,152 columns for each picture)
4. Improper RGB triplet to single unique value conversion (we used $(R) * 256^2 + (G) * 256^1 + (B) * 256^0$)
5. Not enough training/test data available (due to the running time for data.py file if we had more than 500 initial images in the Kat Kam folder)
6. The accuracy score for the Gaussian Naive Bayes Classifier usually ranged from 70% to 50%. This probably means that we do not have enough training/test data and intuitively this makes sense, since we only had at most every set of initial images halved due to the

filtering process of weather rows and images (for example for about 450 images we got down to about 210 pictures after filtering, and running of that file took about 15 minutes, and these number of pictures are not enough)

7. After going through the images we saw that some images were taken at night or early morning which made the screen all black and therefore affecting the training data and test data
8. The images had different locations (YVR and English Bay)
9. The weather data was not taken exactly on the hour
10. After looking at different pictures we realised the difference between a cloudy weather's picture and a rainy weather's picture is not that much different and could decrease the accuracy score (the color of the sky would be the same sometimes)

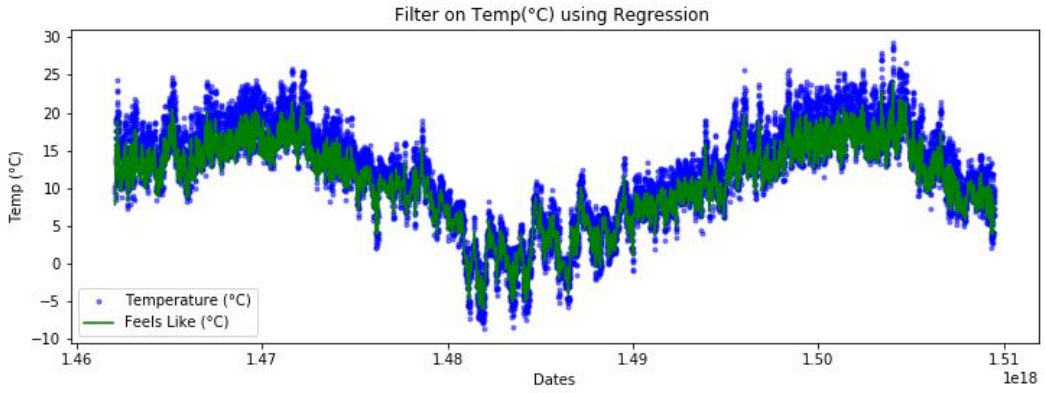
Based on the above reasons it made sense not to get a great accuracy score, but a LOT of time was spent researching different ways to solve these (above) problems and almost all solutions had little or no effect on the results.

Other paths pursued (but failed)

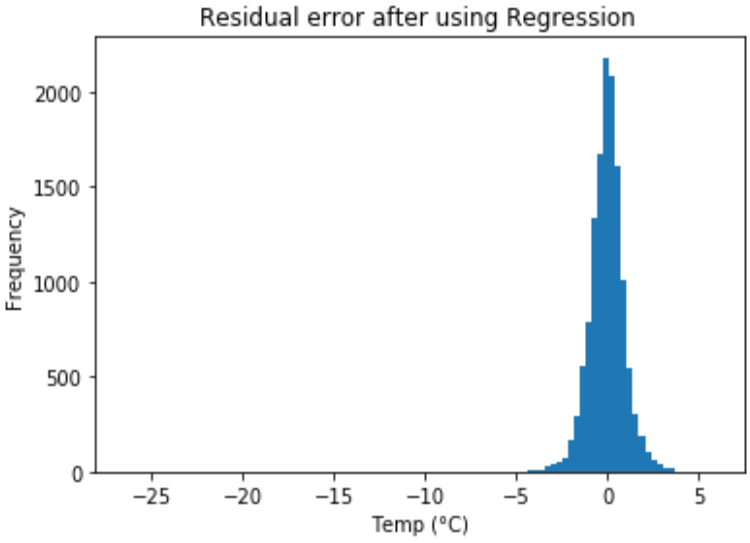
One of the other ideas we looked at was to see if we could use temperature, dewpoint and relative humidity to predict one from the other two (This was supposed to be done using machine learning). After a bit of research and analysis on the given data we found out that there is a correlation and a formula for these three weather attributes, which can be used to calculate one attribute from the other. Also there would be no real life use or purpose for predicting the dewpoint from temperature and relative humidity. For these two reasons we decided to scrap this idea and figure out new ideas. Something else we thought about doing was using the visibility column and matching it with its appropriate picture and trying to predict the visibility of a given picture. By looking at specific pictures that had extremely low visibility and extremely low visibility we found out that there was not a significant enough difference between the two pictures to categorize them, therefore we ditched this idea as well (the pictures we looked at had visibilities of 4 km and 32 km). Another idea we looked at was to figure out the time of the day using a picture alone. Since the daytime/nighttime ratio changes day by day and month by month (ex: fully dark by 5 pm in the winter and fully dark in the summer by 10 pm) this idea would not work unless the user inputs the month and we take into account the month the picture is from(this would not be an automatic process since we need user input: "what month is it?").

A unique idea

After all of these ideas that failed we came up (on our own, with no research) with something unique and useful: What It Feels Like Temperature or Feels Like (°C). This idea is unique (hopefully no other group used this idea), and is used all the time by climatology networks and stations. The basic idea is to use machine learning and a kalman filter to do a smoothing of the temperature to get a feels like temperature, similar to the ones weather stations use. The first thing we did was that we cleaned up the data by getting rid of useless columns (kept Temperature (°C), Dew Point (°C), Relative Humidity (%), Wind Speed (km/h)) and then converting date using datetime. Then we used machine learning to feed the training data, do kalman smoothing on the data and output the new Feels Like temperatures.



The first plot shows the temperature and feels like, and we can see in general the feels like is usually colder than the actual temperature and this is usually because of the wind affecting it (which was considered in our smoothing).



The second plot shows the difference between the temperature and feels like.

Challenges faced

- Running time. The running time of the weather prediction program with 450 initial pictures as input was about 15 mins.
- Human generated weather labels. The labels were too specific and needed to be reorganized using a system devised by us (after about 5 hours of research and data organization).
- Multilabel classification problem. In some cases there were multiple labels for each weather recording label, therefore an appropriate classifier + label splitter was used (after 3 hours of research)
- Pixel arrays not in the right format. The pixels needed the weird, asymmetrical brackets removed and then converted to integers. (took about 2 hours)
- Converting RGB values into single, meaningful, unique number. Had to research how to convert RGB values into a meaningful and unique number using a formula and whether to expand the triplets or combine into one (after 5 hours of research)
- Choosing the appropriate classifying tool. After a quite bit of research we gathered the possibly useful classifying tools (after 7 hours of research)
- Number of tests done. The number of times the program was run, and the time consumed (about 7 hours of testing)
- Amount of time spent trying to understand documentations. Some functions had little or no documentation and little or no examples, so using them was done through guessing, experimentation and testing
- Amount of time spent researching different methods of image processing (about 6 hours of research)
- Fixing specific, encrypted bugs

Limitations

- Hardware limitations (for running)
- training/test data set size
- Utility and tool restrictions (could not use neural networks, deep learning and big data tools like spark)
- Time limitations (final exams, other assignments) (even though we put in MUCH MUCH more than 3.5 times the time and effort put in for weekly exercises but we were not able to get the results we wanted)

What we would do next time:

- Use spark or some other big data tool to get better running time
- Optimize program to decrease running time
- Filter unnecessary data and remove unnecessary operations even more and in the first step
- Have a more module based, function based and interchangeable system in place so the code is easier to read and maintain

Project Experience Summary

James Zhang:

- Managed to filter images and get useful pictures needed for weather prediction
- Merged and filtered csv files to get appropriate columns
- Converted dates for both images and weather data to appropriate format
- Optimized data.py to get a better running time
- Worked on feelslike.py to create a feels like temperature
- Merged csv files to create a training set and test set
- Filtered and cleaned CSV files to get to right format
- Converted dates in feelslike.py to get it into appropriate format
- Trained a model to figure out how much effect each weather attribute has on feels like
- Created a kalman filter to smooth temperature and get feels like temperature
- Created a plot to see how much temperature and feels like differ
- Applied course concepts to create feelslike.py
- Created visualizations for the feels like analysis which helped with understanding of it
- Wrote the initial problem, data, initial cleaning, dataframe creation, and a unique idea sections of the project report

Seyed Amirreza Derakhshani:

- Created sophisticated labels and categorized weather labels
- Modified getpixels function to meet our needs
- Modified the data.py file to filter for specific needs
- Visualized weather predictions to explain analysis
- Created analysis.py to predict weather based on webcam images
- Optimized data.py and analysis.py to increase performance
- Created function to convert RGB values to specific, unique color values

- Modified and separated X and y for training and testing
- Researched on different machine learning classifiers to train model
- Researched on different multi label classifiers to solve multi label classification problem
- Applied different classifiers to get the best accuracy score
- Applied Multi Label Binarizer to get y in right format
- Modified data.py and analysis.py to improve accuracy score
- Studied image and weather data records to learn specific characteristics of data
- Wrote the secondary cleaning, machine learning tools, analysis, and other paths pursued sections of the project report