SETI Final Report: Mulan Xia

Project Machine Learning on GBT Data
Date January 17, 2017

Contributors Griffin Foster
Yunfan Zhang
Steve Croft
Andrew Xu
Pragaash Ponnusamy

Overview Many of the algorithms that are currently run on Green Bank Telescope data search for simple tones, pulses, or drifting narrow-band signals. But these signals can be difficult to distinguish from human-generated radio frequency interference (RFI), and in addition the pipeline may miss more complex signals that could be of interest. The task is to explore and develop unsupervised/semi-supervised machine learning algorithms at scale to identify, extract, and classify signals of interest into various populations, and analyze how these signals cluster under varying feature sets.

Can signals of interest (SOI) for SETI (those that are rare and located only in one place, or a few places, on the sky, or are unique in their statistical properties, shape, etc.) be distinguished from various classes of RFI?

Approach Using various ML techniques, I explored the following approaches in tackling the issue of distinguishing between RFI and SOIs:

a. **Unsupervised Learning: cluster ON/OFF pairs by morphological similarity**
   Pairs in which the ON and OFF show a strong morphological similarity (drifting is fine) can be assumed to be RFI. SOIs are expected to be signals repeatedly present in ON files, but not their OFF counterparts – this would suggest that this signal is not terrestrial based and worth further human investigation.
b. **Supervised Learning: classify ON and OFF files individually and equally, probabilistically as different classes of RFI**
   First examine a diverse group of RFI and create a database of different RFI type signals present in the data. Then implement and train a class of classifiers which will assign a probability that a given input is of a type of RFI. This should allow us to filter down a large database, eliminating signals which are confidently classified as RFI.
c. **Expansion of Dataset: implement a GAN to increase our database of labelled data**
   GANs (generative adversarial networks) are a relatively new type of deep neural net, using 2 networks in parallel (commonly referred to as the generator and discriminator). Given an input dataset, a GAN expands the dataset by creating structurally similar outputs to the input. The generator receives as input the input dataset and then outputs what it believes is a similar image to the input. The discriminator receives as input both the original dataset as well as the generator's

output. With each image, it tries to guess whether the input is part of the original or new dataset, and passes this feedback back to the generator. A successful GAN will create outputs that are indiscernible from the original dataset, thus holding the potential to very quickly expand the dataset.

d. **Supervised Learning: develop a CNN to classify ON/OFF pairs based on similarity between ON and OFF files**
CNNs (convolutional neural nets) have proven to be very successful in image recognition. It would be worth exploring whether a CNN could take in as input a ON/OFF pair and classify whether it was of interest based on the structural similarity within the pair. However, CNNs require millions of data points, and thus before attempting a CNN, a solid attempt to acquire more labelled data must be made. Human labelled data would be optimal, and our labelling GUI may be able to aid in this. Along with the GAN expanded dataset, we explored other ways of increasing input data (shifting images, flipping images).

Techniques

Histogram of Oriented Gradients (HOG)
Principal Component Analysis (PCA)
t-distributed Stochastic Neighbor Embedding (t-SNE)
Positive Naive Bayes Classifier
Logistic Regression
Support Vector Machine (SVM)
Linear Discriminant Analysis (LDA)
Quadratic Discriminant Analysis (QDA)

Procedure

1. GBT data (ON and OFF pairs) was loaded in as FITS files from a public Google Cloud database (run the command gsutil -m cp -r gs://fits-dataset/*/*/*/*/*/*/*/*/*/*.fits ./[destination folder] in terminal). This results in a dataset of 47242 pairs (94 484 files).
2. ON and OFF files were featurized by three variations of HOG:
   a. HOG — convert each image to grayscale; run HOG
   b. Log normalized HOG — convert each image to grayscale; log normalize the image; run HOG
   c. Binarized HOG — compute threshold of 80% of noise; binarize image based on threshold; run HOG

   See Jupyter notebook Loading in Data.ipynb.
3. See **Task 1: Clustering**.
4. See **Task 2: Traditional Machine Learning**.
5. See **Task 3: Generative Adversarial Network**.
6. See **Task 4: Convolutional Neural Network**.

SETI Final Report: Mulan Xia

Task 1: Clustering

Goal

Cluster ON/OFF pairs by morphological similarity

Files

Loading in Data.ipynb
PCA and t-SNE.ipynb

Data

Previously, Pragaash Ponnusamy of the SETI Lab had worked on this project and created a library to aid in calculating morphological similarity between ON/OFF pairs. This is a public Python git repository and can be accessed at the following link: https://github.com/UCBerkeleySETI/fits-corr

Feel free to read the documentation for a more in-depth explanation of the capabilities and purpose of this library, but to summarize, ON and OFF pairs first have their Pearson correlation coefficient (pcc), structural similarity (ssim), regional mutual information (prmi), and normalized mutual information (nmi) calculated. Then, the OFF image is repeatedly shifted and compared to the ON image, until the shift amount at which the ON and OFF images are most structurally similar is calculated. Then pcc, ssim, prmi, and nmi are calculated again. This is all abstracted away in the function _score_multi().

Featurization

And so for each of the 47242 pairs, we calculated the correlation coefficients. This is written to a file (dictionary) called pairs.npy. This is our first set of features of our files.

Then HOG (not normalized, not binarized) was run on each individual file. We ran it with 72 orientations, but condensed the final HOG product into 21 buckets. See details of how this was done in the Jupyter notebook Loading in Data.ipynb. This information was written to an array stored in hog_image_64by16_72_wb_ed_p.npy. For our second set of features, we took the absolute value of the difference of the HOG values of each ON and OFF pair.

Our third set of features involved information extracted from the header of the ON/OFF pairs. Various combinations were explored, but this information included: frequency of observation, right ascension, declination and mean julian date. This third set of features (when we ran PCA) seemed to make results worse, so it ended up being eliminated.

Procedure

After collecting all features, PCA was run on the feature set to ensure that at least 2 features were responsible for most of the variation (not just 1). Then, using t-SNE we plotted and clustered the data. A challenge encountered was that t-SNE (when run on my local machine), was unable to complete analysis of all 47242 points and Bokeh (our plotting software) would crash, or severely lag the computer when attempting to visualize the data. Thus, we shuffled data and ran t-SNE on groups of 2000 at a time. Depending on the hyperparameters, success of clustering morphologically similar vs. dissimilar pairs varied. The figure below shows one what we would consider our best success:

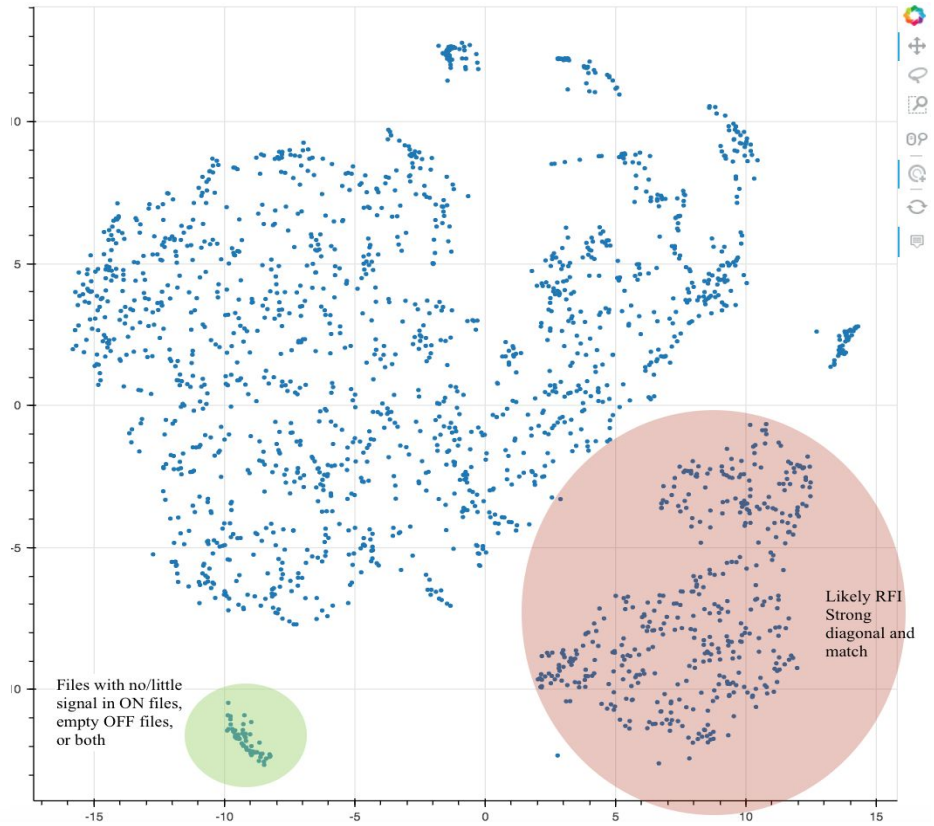

fig 1. PCA clustering of 2000 points. Does not include header featureset or correlation information.

After plotting these points, the two highlighted regions are the only significant regions of good clustering we could find. And while this shows that there does exist morphological similarity between ON/OFF pairs that may be able to be exploited, it was not promising enough to continue pursuing.

This may be due to having an inadequate or weak feature set. Note that when we included correlation information in our t-SNE visualization, the following plot was produced:

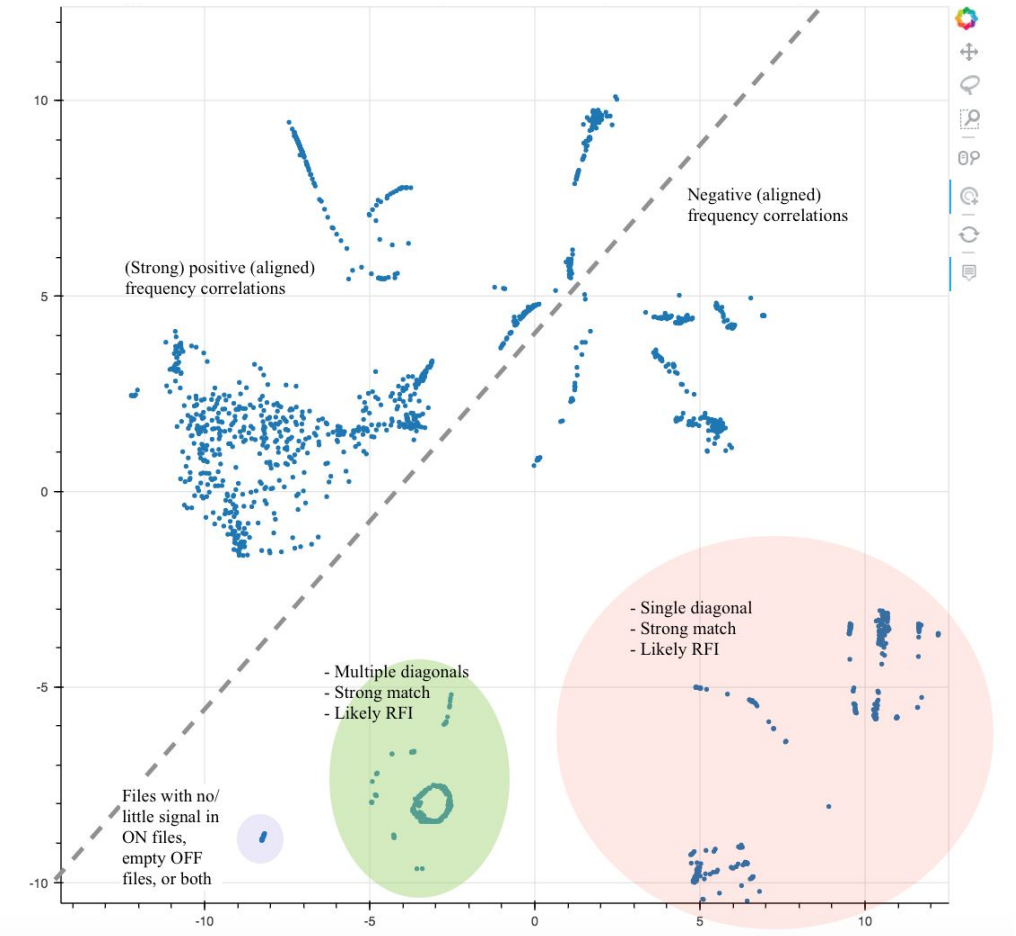

fig 2. PCA clustering of 2000 points. Does not include header feature set, includes correlation information.

While these seem to show more distinct clustering, data also seems to be overfit (there is no distinct visual difference between small, closely spaced clusters).

SETI Final Report: Mulan Xia

Task 2: Traditional Machine Learning

Goal Classify files probabilistically as different classes of RFI

Files
Loading in Data.ipynb
Probabilities CSV.ipynb
Single GUI Guide.ipynb
Positive Naive Bayes Classifier_GPS.ipynb
SVM.ipynb

Data

This approach aimed to train classifiers on different classes of RFI, and thus labelled data was required. To acquire this, Andrew Xu (another SETI intern) wrote a GUI for such labelling (see fits_pair_GUI.py and fits_single_GUI.py). fits_pair_gui is an interface in which an observer rates similarity between ON/OFF pairs; fits_single_GUI.py is an interface in which an observer categorizes files by morphological structure. For this approach, data from fits_single_GUI.py. Documentation on how to classify can be found in the Jupyter notebook Single GUI Guide.ipynb. We were able to acquire 17000 labelled data points for testing.

Originally, the classifier had 15 categories; however, for our analysis, we only used categories in which we had acquired +100 data points and omitted the category 'signal of interest' and 'combination' (it quickly became evident that we were unable to categorize signals of interest, because these signals shared no similarity with each other; their only commonality was that they differed from all other known categories. Additionally combination signals simply had too much variation). This left us with 9 categories. The Jupyter notebook Probabilities CSV.ipynb simply shows how labelled data was retrieved from the files on my computer, as well as some statistics on the labels themselves. Labelled data can be found in the dataframe labelled_binary_log_hog_dataframe.pkl. The information is found in the 'label' column and numbers correspond as following:

0	Narrow diagonal (1)
1	Narrow diagonal (1+)
2	Narrow horizontal
3	Narrow vertical (1)
4	Narrow vertical (even)
5	Narrow vertical (inf)
6	Wide diagonal (1)
7	Wide diagonal (1+)
8	Wide horizontal
9	Wide vertical (1)
10	Wide vertical (even)
11	Wide vertical (inf)
12	No signal

13 Combination
14 Signal of interest

Featurization for all classifiers involved using all three variations of HOG. An intensity difference measurement was also extracted and used in the positive naive Bayes classifier, though it seemed to do little.

Bayes

In Jupyter notebook Positive Naive Bayes Classifier_GPS.ipynb, I attempted to train a Positive Naive Bayes Classifier on a very specific class of ON/OFF pairs: ON signal = narrow or wide horizontal signal (GPS); OFF signal = nothing. To gather these specific ON/OFF labelled pairs, I found signals by the groupings from Approach 1. It was expected that I had erroneously classified some signals (as I took large clusters) and that I had missed a significant amount (I did not check every point in a plot).

I initially had hoped this semi-supervised technique would be successful as it only required positive examples ('anomalies'), to categorize a large group of unlabelled data. And it did relatively well: on my validation set which originally consisted of 137 interesting signals and 9311 non-interesting signals, it classified 140 interesting signals and 9308 non-interesting signals. There were 3 signals that differed, and these 3 had originally been mislabelled by me. On my test set which originally consisted of 122 interesting signals and 9327 non-interesting signals, it classified 128 interesting signals and 9321 non-interesting signals. There were 6 signals that differed; 5 had been incorrectly labelled by me.

Unfortunately, I did not pursue this approach as this degree of accuracy took a significant amount of time of playing with the probability hyperparameter. A positive naive Bayes classifier assumes that there is a certain and easily determined probability of the anomaly appearing in unlabelled data. With RFI, depending on the frequency, RA and DEC, MJD of the observing session, these probabilities can all vary widely. And thus this classification method is not very generalizable.

One vs. rest

The next approach I took involved using one vs. rest classifiers (see Jupyter notebook SVM.ipynb). In this, I used 80% of my approximate 17000 labelled data points to train and 20% to test. Remember, some data points were filtered out. Each classifier could only classify an input as 'of x category' or 'not of x category'. That is to say, it could not categorize inputs into one of several categories. I used SVM and logistic regression. Both performed reasonably well (0.97 and 0.98 overall accuracy, respectively), depending on the category (see fig 3 below). Some categories did better simply due to having more training data. See the notebook for precision, recall and area under curve statistics.

My initial idea was to chain the one vs. rest classifiers (first filter out all narrow diagonal signals, then horizontal, etc.) and have the remaining signals be potential signals of interest. However, as I went down the chain, the training dataset got smaller and smaller, and

accordingly, accuracy rates severely dropped (down to 0.50). Thus, I then attempted multiclass classification.

Multiclass

This code can also be found in Jupyter notebook SVM.ipynb. Again I used 80% of my approximate 17000 labelled data points to train classifiers, and 20% to test. These classifiers took inputs and classified them into one of 9 categories. I used SVM, logistic regression, LDA and QDA. Overall, they did worse than both one vs. rest classifiers (accuracy rates of 0.92, 0.31, 0.85 and 0.84) respectively. However, they are advantageous in that they are significantly more accurate (excluding logistic regression) in classifying the overall group (when compared to chaining one class classifiers).

See the Jupyter notebook for a confusion matrix visualization of each of the classifiers. It can be observed that most incorrectly classified samples were classified into morphologically similar categories (i.e. confusion between wide diagonal signal (1) and wide diagonal signal (1+)). Thus, in the future, it may be worth trying to condense visually similar categories.

Note that multiclass classifiers could not filter down the database to just a remaining group of SOIs – all input signals had to be classified in a bin. Though I did not have the time to attempt it, code could be altered such that input signals would only be classified if the classifier exceeded a certain threshold in terms of confidence in categorization. This would be possible with logistic regression, LDA, and QDA (they all output probabilities), but not SVM (outputs a binary 0/1 classification).

fig 3. Average accuracies of classifiers.

fig 4. Legend to fig 3.

Task 3: Generative Adversarial Network

Goal — Implement GAN to expand dataset of labelled files.

Files — Based largely off the O'Reilly tutorial[1] on GANs.

Summary — With an original labelled dataset of 17000 points categorized into 1 of 15 categories based on morphological structure, I attempted to use train a GAN to produce similar ON/OFF images. Note that every ON image was treated equally to an OFF image (each was fed in individually).

Results — The input to the GAN consisted of images like the following:

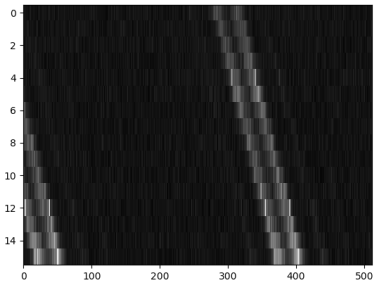

fig 5. Input to GAN, unnormalized, grayscale.

All images were loaded in grayscale. We experimented with normalizing and cleaning the images in different ways; binarization seemed to produce the best results. Additionally, we obtained significantly better results when feeding in only a single class of images (as originally labelled) to the GAN. Using the GAN (with minor edits to hyperparameters), feeding in a single-class (diagonal below) binary input produced outputs like the following:

[1] https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners

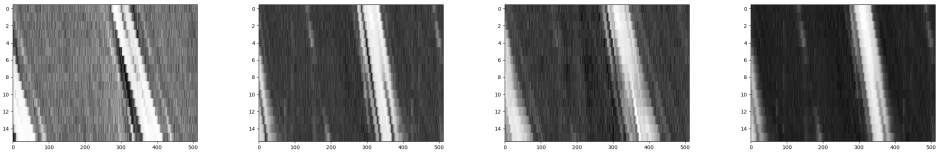

fig 6. Sample outputs of GAN, single class input of narrow double diagonal signals.

This particular iteration showed the following losses:

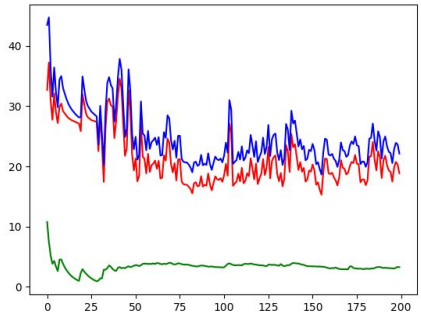

fig 7. Losses of GAN, red = generator loss, green = discriminator loss, blue = total loss.

Conversely, feeding in all classes at once produced outputs like the following:

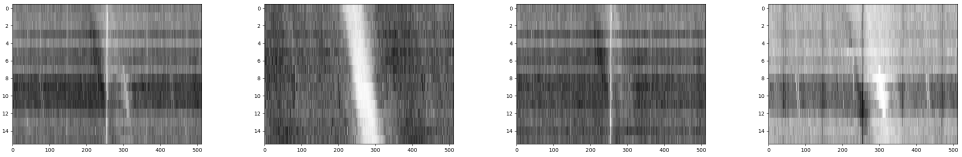

fig 8. Sample outputs of GAN, input included all classes of labelled data points.

Conclusion

Although we were initially excited about the potential of using a GAN to quickly expand our labelled dataset (thus avoiding the hassle and time involved in manually labelling), we realized that the task was much more difficult than initially thought. The GAN managed to produce structurally similar images, but even to the human eye, images of the original dataset and those artificially generated were distinctly different.

Furthermore, the GAN produced images generally had a very noisy background, which could be mistaken for a multitude of weaker, interesting signals. Simply put, the variance and inconsistency of generated images left too many variables uncertain for our task of searching for a (possibly very weak) interesting signal.

Further Work

Thus, to expand our dataset, we reverted to traditional methods of shifting and rotating signal images. For all further work with the CNN, images were shifted randomly between -25 (left) and 25 bits (right). Additionally, some were flipped horizontally and vertically (ON and OFF files of the same pair would both be flipped in the same manner). This resulted in a labelled dataset of ~170 000.

SETI Final Report: Mulan Xia

Task 4: Convolutional Neural Network

Goal

Train a CNN on expanded dataset for classification of interesting ON/OFF pairs.

Files

binary_classifier.py
CNN.ipynb

Data

Input data were pairs of ON/OFF files, each labelled with their class as loaded in from labelled_files.pkl. Then in the training_data() function of binary_classifier.py, pairs were either collectively labelled as the class of the ON file if ON and OFF files were of the same class; or 10 plus the class of the ON file if ON and OFF files were of different classes (signalling that this pair may be interesting: there is either something present in the OFF that was not present in the ON, or vice versa). Additionally, ON/OFF pairs were normalized using median removal and IQR scaling.

Note that some of the original categories were removed due to an inadequate number of examples per this category. We ultimately kept 10 of the original categories, and since each category had 10 added to it if the ON and OFF did not match, our input files fell into one of 20 categories, and our output classifier ON/OFF pairs into one of 20 categories.

To see the full structure of our binary classifier, refer to binary_classifier.py. During trials, various hyperparameters/parameters were tuned (batch size, normalization techniques, pooling layers, learning rate, learning decay, dropout, method of categorization).

A batch size of 16 seemed to be optimal, along with IQR normalization (as opposed to binarizing the image). Learning rate, so long as it was small enough, did not seem to have a large effect on results; nor did the type of learning decay. An input and output of 20 classes seemed to be optimal, though more work could be done in exploring this.

Conclusion

By the end of our work, our classifier achieved ~0.89 accuracy into one of the 20 categories. Some classes of signals (generally those with 1000+ previously labelled examples) were very well classified by our classifier and had a high confidence level. This suggests the importance of more labelled examples. Refer to CNN.ipynb for a full description of results.