Blob-match: Machine learning for cross-identification of radio surveys

James Gardner, supervisors: Cheng Soon Ong, Matthew Alger

Semester 2 2019

Abstract

Achieving success in radio-radio survey cross-identification would be determining the real, physical objects that we're looking up at. The simplest measure of whether two sources (or 'blobs') are a match for an actual object is their separation on the sky. Using this separation, we create a common catalogue of matches from the radio surveys TGSS (TIFR GMRT Sky Survey Alternative Data Release 1) and NVSS (NRAO VLA Sky Survey). We train a logistic regression binary classifier on this catalogue, then use its predictions to naively partition a patch of the sky into objects, by transitively grouping any chain of predicted matches. We find that this naive partitioning fails to convincingly identify objects in the sky, but do not find fault with the classifier.

Figure 1: Giant Metrewave Radio Telecope and Very Large Array (TGSS) 150 MHz / resolution 25″ vs. (NVSS) 1.4 GHz / resolution 45″

1 Introduction

We consider matches between sources in TGSS (TIFR GMRT Sky Survey Alternative Data Release 1) and NVSS (NRAO VLA Sky Survey). A source in each of the surveys is a gaussian (blob) fitted over the raw data. To clarify, there are three layers at play: the real physical object that exists out in the universe, the raw photometric data detected by each of the Giant Metrewave Radio Telescope (TGSS) and Very Large Array (NVSS), and the sources as gaussians that are fit to the plane of raw data. Our task is to go from pairs of these sources (one in each survey) to partition the sky back into the original physical sources as best we can.

TGSS catalogues the entire radio sky north of -53° DEC in 0.6 million sources at 147.5 MHz with resolution $25''$. From TGSS we extract for each source: the source name in IAU form TGSSADR JHHMMSS.S+DDMMSS, the RA and DEC position on the sky (in degrees), the lengths of the major and minor axes of the fitted gaussian (in arcseconds), the integrated (total) flux density (in milli-janskys: mJy) and peak flux (in mJy/beam), and the position angle (orientation on the sky) of the major axis (in degrees).

NVSS catalogues the entire radio sky north of -40° DEC in 2 million sources at 1.4 GHz with resolution $45''$. From NVSS we extract for each source: the source name in IAU form NVSS JHHMMSS+DDMMSS, the RA and DEC position on the sky (in degrees), the lengths of the major and minor axes (in arcseconds), the peak flux (in mJy/beam, uncorrected), the position angle (in degrees), as well as the integrated and peak residual flux (in mJy and mJy/beam), the integrated polarised flux density (in mJy) with its Stokes Q and U linear polarisation brightnesses (in mJy) as of the source's centre.

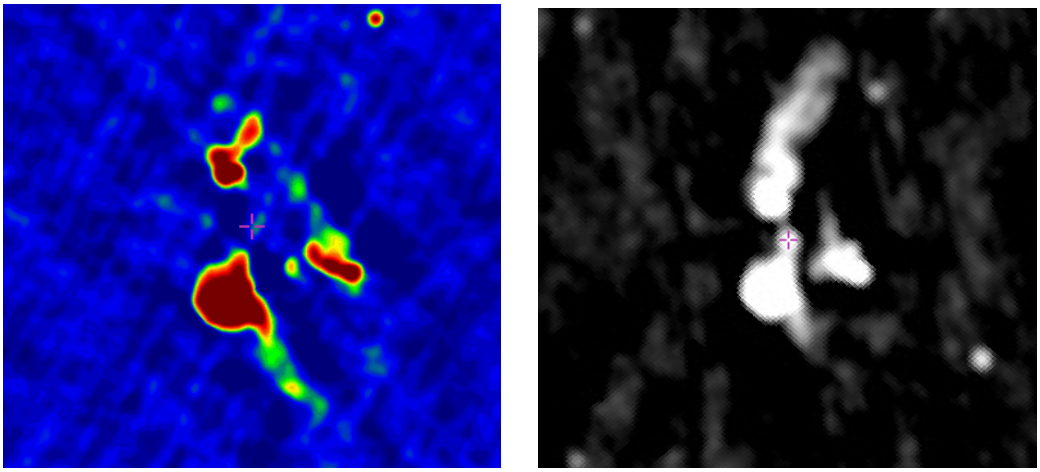

Figure 2: Sources (blobs) of 3c40 in TGSS and NVSS

2 Positional matching

Positional matching is determining matches purely based off the relative positions (RA and DEC) of each source on the sky. Assuming isotropy, this amounts to just considering the angular separation (geodesic distance on the celestial sphere) between their centres. For compact sources (tight, that subtend a small angle) this is a great filter as we'd scarcely expect two compact blobs on the sky degrees apart to have come from the same physical object. However, in radio, this is not a universal assumption as we have sources like the radio galaxy Centaurus A that cover about 8° on the sky (to the Moon's 0.5°), or more commonly sources like 3c40 in figure 2 that cover arcminutes. We take $2'$ as our separation limit when matching the entire sky, note that the expected number of matches grows as the square of this limit as the area of the disk of possible matches grows. We reject all matches beyond this limit, so we're at least missing direct matches for the degree sized objects on the sky. Whether partitioning can recover them remains to be seen.

2.1 Positional matching results

Initially, we consider matches of TGSS sources to NVSS sources over the entire sky that both surveys cover (north of $-40°$ DEC). In part, because we have a reference to the same being done recently in Tiwari [2019], albeit with some differences: Tiwari [2019] uses a closer $30''$ separation limit and drops all non-unique matches. When we train a logistic regression classifier, we do so on a $20° \times 20°$ patch of common southern sky and with a separation limit of $10'$.

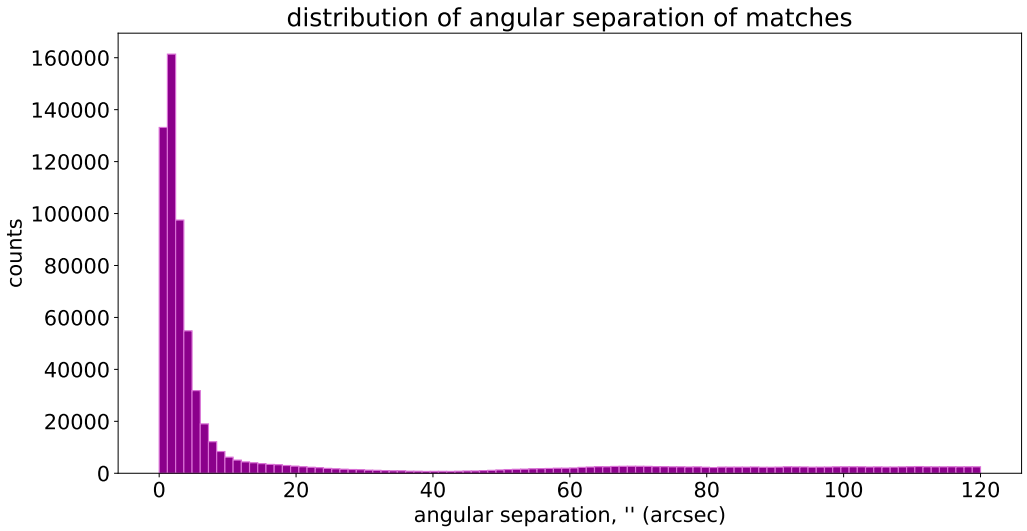

Figure 3: Distribution of angular separation of positional matches TGSS-NVSS over entire common sky (north of $-40°$ DEC) for a separation limit of $2'$, note the bi-modality within this range into populations either side of $40''$.

See figure 3 for the distribution of separations for matches (TGSS-NVSS) over the entire common sky. In total, we find over 720000 matches within $2'$ in a bi-modal distribution split either side of $40''$, with the vast majority of matches within $10''$. This bi-modality is hypothesised as being the result of two effects, geometric more than cosmological: actual matches between objects (Intema et al. [2017] estimates 94.4% of TGSS sources have a NVSS counterpart within $45''$) and then the start of a long quadratic as the search window widens and collects more and more matches. That is, if we extended the search out to $180°$ we'd match the entire sky and we expect the curve to get there to be quadratic in the search radius. We'll use this cut at $40''$ to threshold (activate) the label of the match as it's the clearest distinction that positional matching returns. To be clear though, we have no reason to suspect all these matches within $40''$ to be between sources from the same physical object, but we do chose to train that that positional matching label.

2.2 Catalogue of matches

We construct a common catalogue of the positional matches of TGSS to NVSS over a $20° \times 20°$ square patch of sky between $-35°$RA, $149°$DEC and $-15°$RA, $169°$DEC (i.e. centred on $-25°$RA, $159°$DEC), with a separation limit of $10'$. The choice of patch is arbitrary, testing shows no difference if the patch is moved (as long as it stays totally within the common sky),

$-35°$RA, $149°$DEC happens to be Canberra's latitude and longitude.

The main advantage of making a common catalogue is that we can add derived features from a combination of both sources. See section 1 for details of each of the raw and fitted features from the source catalogues that we save into the common catalogue. The first of the derived features we add is separation. Later, we'll assume isotropy and remove the RA and DEC position values. The disadvantage of making this common catalogue from positional matching is precisely that we must chose some separation limit and rule out degree sized objects, or create far too cumbersome a catalogue filled mostly with matches between unrelated compact sources far away from each other.

2.3 Spectral index

The second derived feature that we can add is the spectral index, α. The observed spectral index for a TGSS-NVSS match assumes the peak flux follows a power law of frequency, takes the ratio of sources, and re-arranges for the best guess at the exponent (α) were the two sources the same object:

$$\begin{aligned} S_\nu &= \nu^\alpha \\ S_{\text{TGSS}}/S_{\text{NVSS}} &= (\nu_{\text{TGSS}}/\nu_{\text{NVSS}})^{\alpha_{\text{obs}}} \\ \alpha_{\text{obs}} &= \frac{\log(S_{\text{TGSS}}/S_{\text{NVSS}})}{\log(\nu_{\text{TGSS}}/\nu_{\text{NVSS}})} \end{aligned} \tag{1}$$

See figure 4 for the distribution of the observed spectral index of matches over the entire common sky, and a comparison of the same plot from Tiwari [2019]. Note here a mistake (on my part) in that peak flux did not get corrected for beam size for either survey when calculating each match's spectral index. The main distribution appears roughly identical between plots, except for the fatter tails in ours with a right tail preference for higher peak fluxes. Unlike Tiwari [2019], we did not drop non-unique matches and had a larger separation limit and those must therefore allow matches with more diverse alpha values. This makes sense, the more non-unique matches we allow, the more different we'd expect the sources to be and so the more diverse alpha values we'd allow. That the right tail preferences higher peak fluxes from TGSS follows from higher fluxes allowing stronger ratios within the spectral index formula, see equation 1.

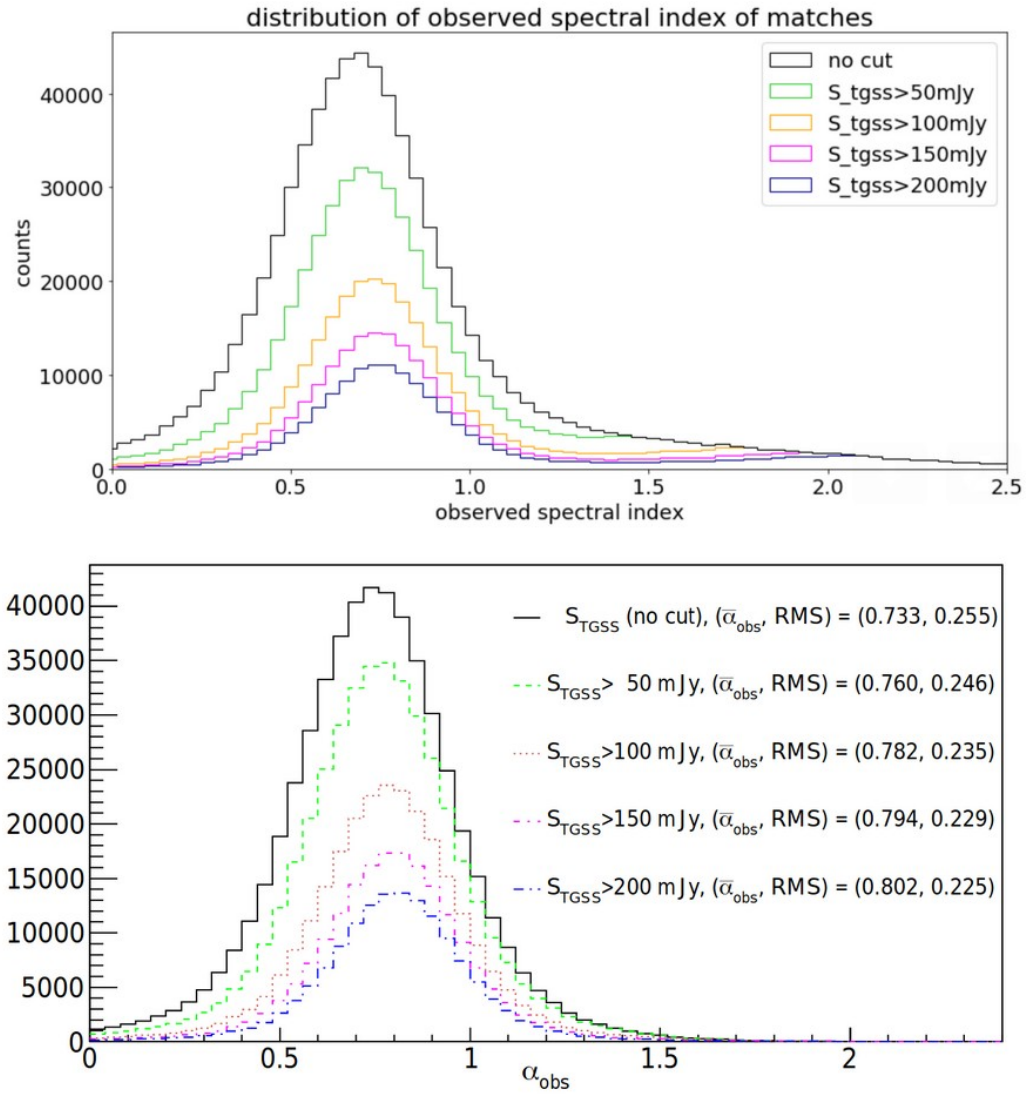

Figure 4: Spectral index α of positional matches over sky comparison to Tiwari [2019], caution: image used without permission. Note the fatter tails of our matching and the right tail preference for higher peak fluxes.

The last derived feature was the logarithm of the various flux features as in section 1, we added this only later in an input step, it isn't saved into the created catalogues. This is a common technique for treating the large range that fluxes can fall over.

3 Logistic regression

Having recreated existing positional matching results, we now return to the central problem of cross-identification to find physical objects. With our common catalogue of TGSS to NVSS positional matches, we choose to approach this problem as a binary classification task. Given a positional match, tell whether the sources come from the same physical object. Moreover, we can train an automated classifier to perform this task.

It's worth noting that this is not the only way to approach cross-identification, it doesn't even need to be binary classification. We could instead aim to train a predictor to take in any TGSS source and return the best match in NVSS, or take in a list of sources and return the likelihood of them all coming from the same object, or any number more of variations on the theme of cross-identification. We chose the binary classification given a pair of sources as it can work directly off of the positional matching catalogue already made and since it's is generally easier.

3.1 A napkin-sized introduction to machine learning

Supervised machine learning considers a function, $f : X \to Y$

- f is the function, called the classifier/predictor
- X is the input real-valued feature vectors from some space $\mathbb{R}^D$, where each component is called a feature
- Y is the output predictions to be compared against labels (of a similar type), predictions/labels are not necessarily real-valued scalars and decisions may need to be forced to transform predictions to the form of the labels

And we are interested in how the function operates on data: known inputs and their corresponding labels, i.e. dataset $= \{(\vec{x_1}, \hat{y_1}), \ldots, (\vec{x_N}, \hat{y_N})\}$. Where a distinction between the initial form of the predictions and form of the labels. Here, $\hat{y_i}$ is the known label associated with a feature vector $\vec{x_i}$, and we might call the prediction $f(\vec{x_i}) = y_i$. Alternatively, we can collapse the decision processes into the function and assume that prediction and labels are directly comparable.

Three related algorithms, all called machine learning, concern this function:

(1) training: optimising the function against training data

(2) testing/prediction: evaluating the function on other, testing data

(3) hyperparameter tuning/model selection: changing the other (non-feature) arguments and the form of the function and the training (e.g. changing the regulariser λ, see definition of cost below)

Training is optimising some objective for the function. With labels, this objective is often to minimise a cost function. Here, we consider cost functions of the form: the average of some loss function over the training data plus a regularising term with argument λ (a hyperparameter). Loss functions are a metric between a prediction compared to a label, loss $= l(\hat{y}_i, f(\vec{x_i}))$. See section 3.3 for more about loss. An example cost function would be avg loss $+ \lambda \left\|\vec{\theta}\right\|^2$, which would penalise models that strongly weighted lots of features. These three terms are very similar but operate on different scales, loss is at a particular data point, cost is over the whole training dataset, and the objective is more generalised goal of the training process (which here is to minimise the cost function).

Logistic regression is when the function is of the form:

$$\begin{aligned} f(\vec{x}) &= \sigma(\vec{\theta} \cdot \vec{x} + \theta_0) \\ \sigma(x) &= \frac{1}{1 + e^{-x}} \end{aligned} \tag{2}$$

Where σ is the logistic sigmoid, $\vec{\theta}$ the weights, and θ_0 the bias. Compare to linear regression with $g(\vec{x}) = \vec{\theta} \cdot \vec{x} + \theta_0$. One can think that the function is really $f(x; \vec{\theta}, \theta_0)$ and that training is adjusting these weight/bias arguments. Because logistic regression evaluates the feature vector all at once, we put it in a general class of so called linear classifiers, despite the fact that with the sigmoid it can learn non-linear features.

3.2 Feature vectors and labels

In the language of the previous section, we set up a logistic regression binary classifier to take in feature vectors of the common catalogue and return the likelihood of a match being real. To cut down the size of the problem, from here on we consider the common catalogue of the 20° patch with separation limit 10′, rather than the entire shared sky.

Formally, translating raw data (z), in whatever form, through a feature map (ϕ) into real-valued feature vectors ($\vec{x}$) is the work of a 'domain expert'.

In our case, all of the data in the common catalogue is already numerical (except the source names). For our feature map, we drop the RA and DEC positions and add in logarithm values of the flux and flux densities but otherwise don't alter the catalogue. In a sense, the real domain expertise came much earlier in the construction of TGSS and NVSS from truly raw photometric/geometric data. Also note that logistic regression generally assumes that all the features have been normalised, which our haven't, and binary classification generally assumes that the labelled population sizes are roughly equal, which our are (at 63% negative to 37% positive).

Finding labels and justifying them is the critical obstacle to supervised machine learning. We use the results from positional matching, in particular figure 3, to label all matches with separation less than $40''$ as a positive match, and all those with separation greater as a negative match. This is by the arguments in section 2.1, in particular the observation of the distinct population of separations less than $40''$. We write an explicit score function for this labelling, which sets separation 0 to score 1 and then linearly decreases to 0 at and beyond the threshold of $40''$. See figure 5 for a plot of these scores over the patch.

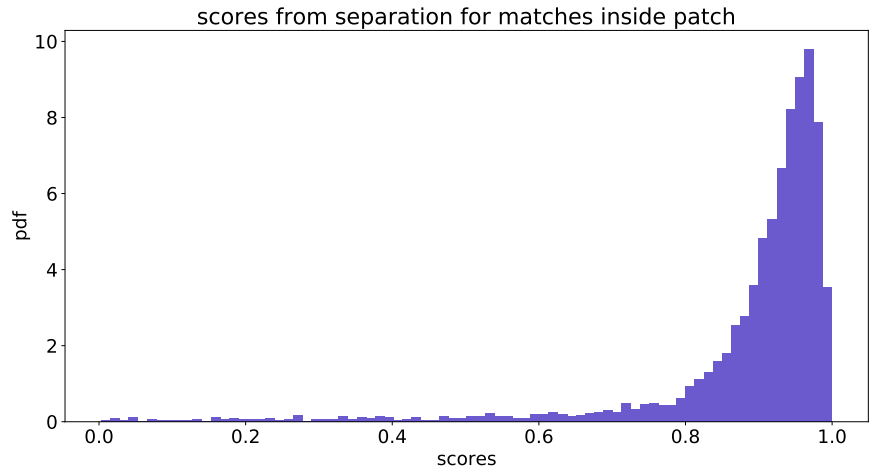

Figure 5: Separation based score with threshold at $40''$, distribution for all matches in patch. Compare to figure 3.

During research, we also created some other explicit score functions for general matches than the separation scorer. Variations included: checking if a match was in the catalogue, using the spectral index histogram as a pdf, and

weighted combinations thereof against the separation score. These hand-made score functions did not produce nice results and were left behind for logistic regression.

3.3 Loss

We use the binary cross-entropy (BCE) of labels against predictions as our loss term. Information theoretically, this represents the expected information gain ($-\log_2(\text{prob})$) over the labels from observing the correct prediction, or the expected coding length (in bits) of the labels given optimised lengths for the predictions. Note that the use of expected value instead of average here is improper. This is closely related to the Kullback-Leibler Divergence between the label and prediction distributions, $D_{KL}(\text{labels} \,||\, \text{predictions})$. Being binary means that the labels only take on values $y_n = 0$ or 1, and let f_n be the corresponding prediction to a particular label, with n observations going from 1 to the sample size N.

$$H_f(y) = \mathbb{E}_y[-\log_2(f)] = -\sum_i (y_i \log_2(f_i)) \overset{*}{=} -(y \log_2(f) + (1-y)\log_2(1-f))$$

$$\therefore \text{ average loss} = \mathbb{E}_n[H_{f_n}(y_n)] = -\frac{1}{N}\sum_{n=1}^{N}(y_n \log_2(f_n) + (1-y_n)\log_2(1-f_n)) \tag{3}$$

Note the trick in the last equality of the first line (*) that uses the binary label to explicitly write out both cases. And that cross entropy can also be notated (confusing it with joint entropy) as $H(y, f)$. Using this we can derive the average loss, which here equals the objective (average loss $+ \lambda \|w\|^2$) as we've set a null regulariser ($\lambda = 0$). Terms are confusing and often vague, but here the loss function is BCE, the cost function is average loss plus regulariser, which is also the objective function here. Since our objective function is just a cost function we'll drop the term 'cost' and call it the objective, but note that in general the objective does not have to be a loss-based cost function. This form of the average loss is the negative of the log-likelihood, so minimising this average loss is the same as maximising the likelihood of successfully observing the labels.

During training, the algorithm is told to minimise the objective (the above average loss) by gradient-decent. Again, here the regulariser (the penalty to over-fitted, complex models) is set null so average loss is cost is objective. In pytorch [Paszke et al., 2017] this is performed by backpropagation through automatic differentiation. Backpropagation is essentially the chain rule of differentiation except instead of applied to the entire function symbolically, applied along each sub-calculation throughout the function. This means that evaluating the function and evaluating its derivative have the same complexity since they follow the same set of sub-calculations. Once you have the gradient, you take a tiny step in the opposite direction and repeat.

3.4 Logistic regression results

$$z \xrightarrow{\phi} \Big| \; x \xrightarrow{score} \mathbb{R} \xrightarrow{\sigma} [0,1] \xrightarrow{activation} \{0,1\} \qquad (4)$$

It is worth (informally) remarking here that actually running logistic regression, after feature vectors and labels were all set up, went particularly fast leading to the remark: 'and I thought machine learning was difficult'! In this lies what I identify as the TV Chef Problem, where if all the ingredients are prepared off-screen, the problem looks easy. However, this hides that sourcing and preparing those ingredients (translating data to feature vectors and deciding on labels), as well as determining if you've even got the right recipe (choosing models and tuning hyper-parameters), to actually serving the meal (interpreting results) is where the real difficulty and ambiguity lies. Watching the TV Chef is effortless, but answering: 'is this meal right?' can be really hard.

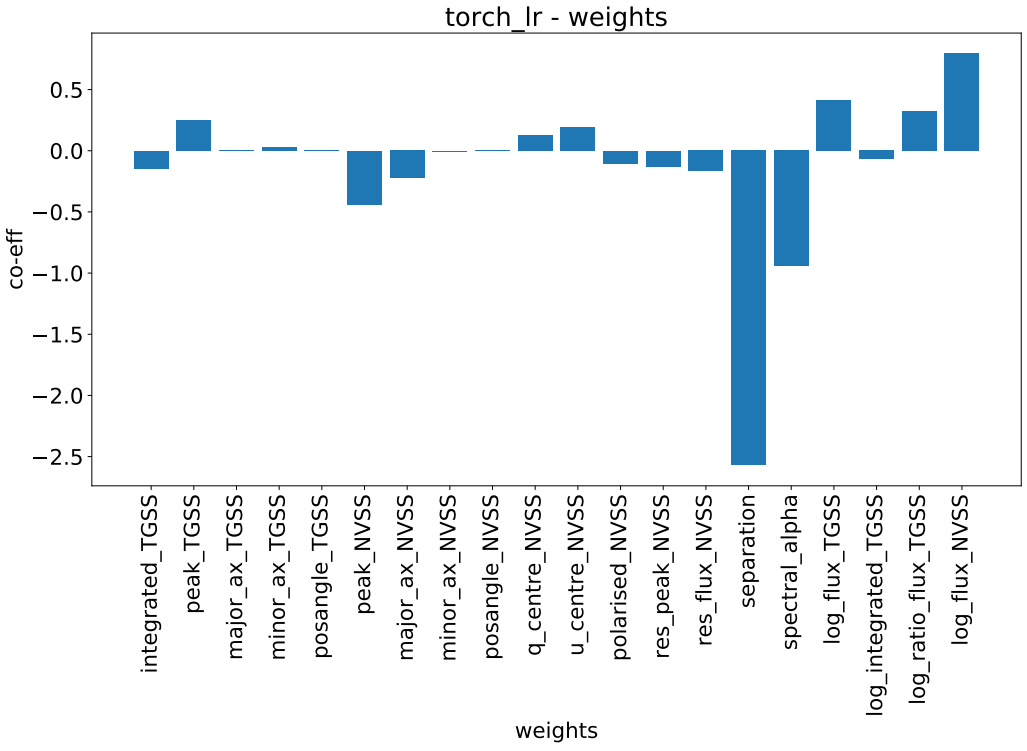

Figure 6: Weights in logistic regression model

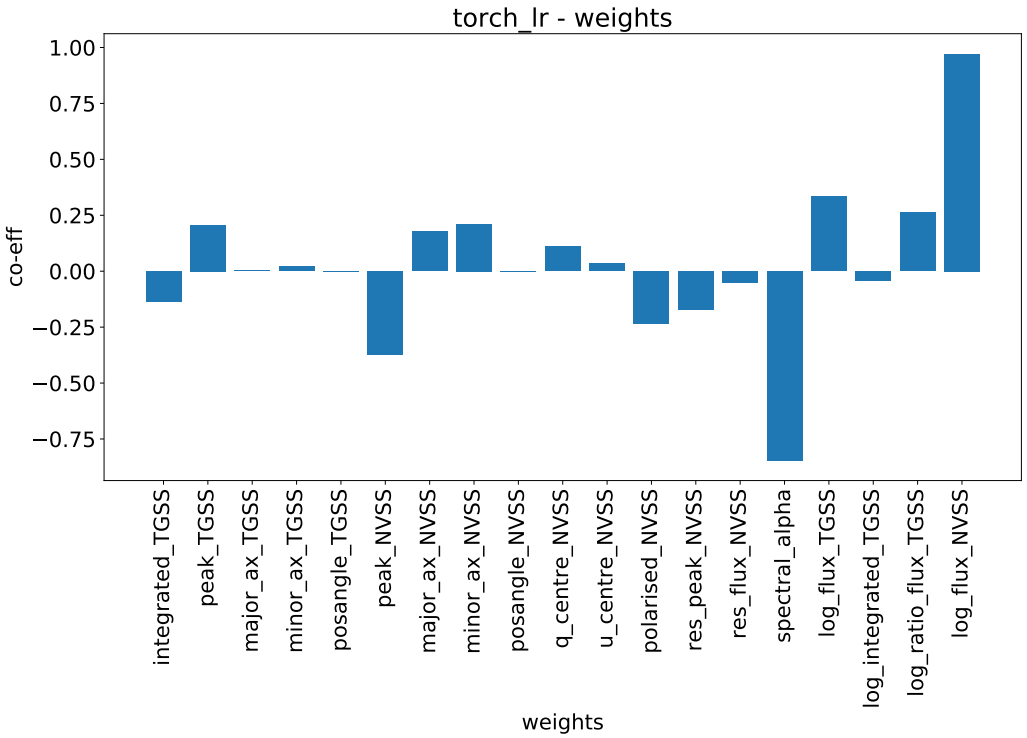

Figure 7: Weights in logistic regression model with separation removed

		accuracy	precision	recall
pytorch: all features	over patch catalogue	75.7	42.3	90.2
	over manual labels	80.0	80.0	100
pytorch: no separation	over patch catalogue	70.3	36.9	88.2
	over manual labels	80.0	80.0	100
scikit-learn	over patch catalogue	97.4	90.2	96.3
	over manual labels	82.5	100	78.1

Table 1: Accuracy percentages for logistic regressions: with all features or with all minus separation, and over the patch with positional matching labels or over manual labels. Note that recall is heavily preferenced against precision, so the classifier says a lot of matches are true, including most of the labelled true ones, but far more labelled false ones. See appendix 6.3 for scikit-learn information.

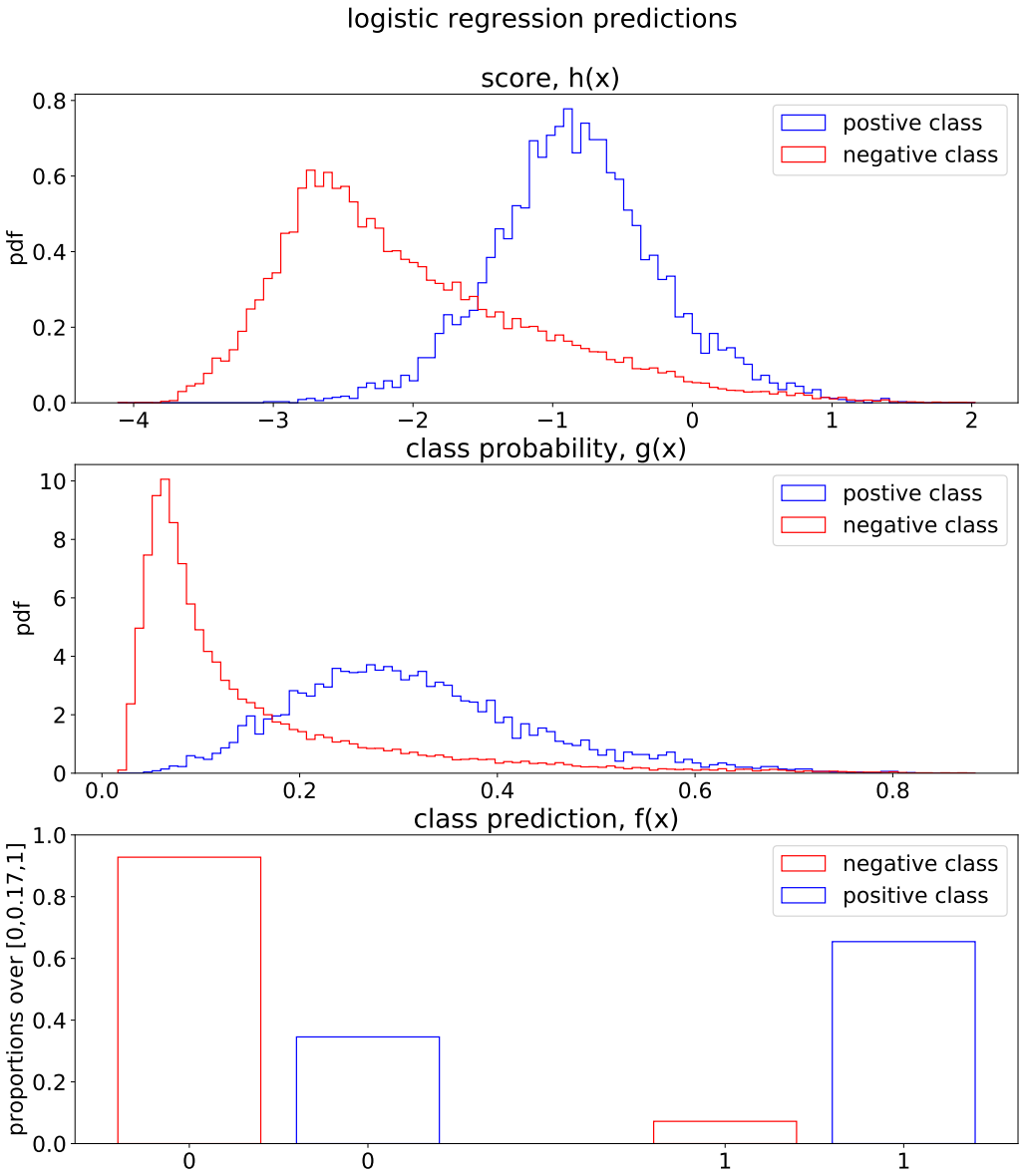

Figure 8: Three related types of predictors for the binary classifier: the raw score over the whole real axis, the class probability within $[0, 1]$, and the predicted class (0 or 1), here splitting where the two labelled populations meet at 0.17

3.5 Naive partitioning using classifier

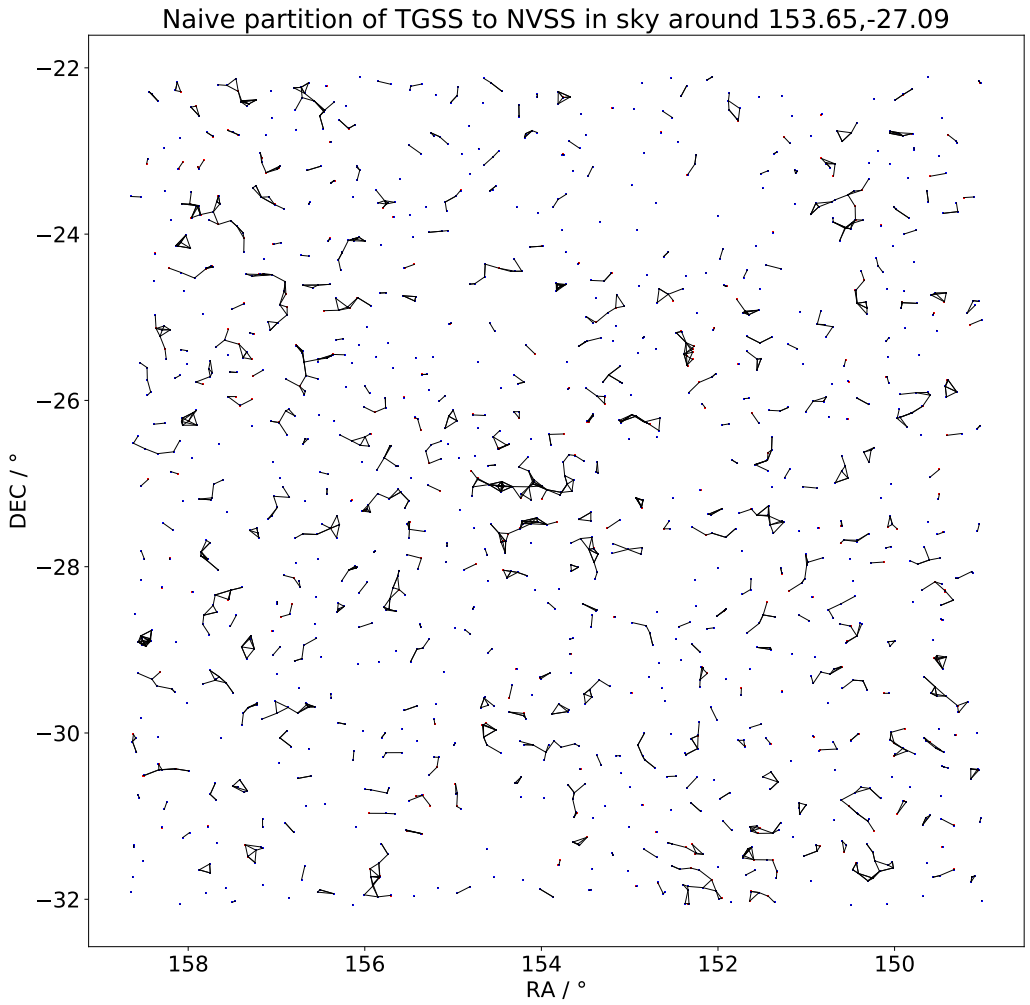

Figure 9: Naive partition of the sky using classifier: naive partition aggregates matches transitively, i.e. if T1 matches N1 matches T2 then (T1,N1,T2) forms an object on the sky and the two T-N (TGSS to NVSS) matches are filled in as line segments on the sky. Note how this leads to large (compared to 40') 'constellations' on the sky, almost all of which are thought to be unphysical. It also still leads to 1-to-1 matches, which just appear as dots, when nothing else is nearby.

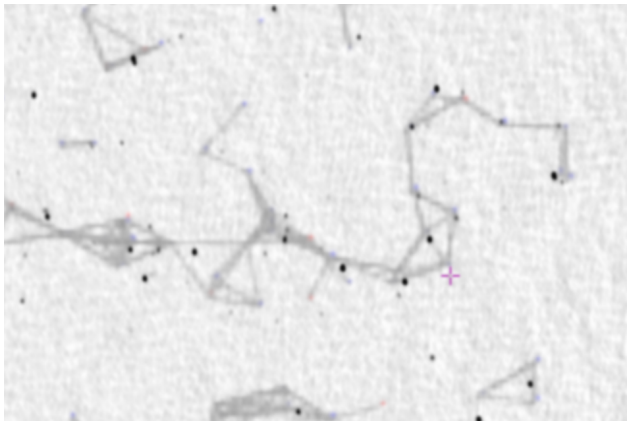

Figure 10: Overlay of detail from figure 9 over TGSS cut-out, not exactly aligned but one can clearly see that the naive transitive partition has clearly linked distinct compact sources together into a many-component, non-existent physical object

4 Discussion

5 Conclusions

Our goal was to cross-identify TGSS and NVSS as to be able to identify the real, physical objects in the sky. To this end, we constructed a common catalogue of positional matches of TGSS to NVSS sources within $2'$ over the shared sky and successfully reproduced the initial results of Tiwari [2019].

We then restated the problem as a binary classification task and trained a logistic regression classifier against positional matching labels. This classifier successfully predicted positional matching and manual labels, both with and without the separation feature.

Finally, we used the classifier to perform a naive transitive partition of the sky into predicted physical objects. This partition did not appear remotely physical. We find the fault to lie with naive transitive partitioning, positional matching, and not with the classifier.

References

T. Boch and P. Fernique. Aladin Lite: Embed your Sky in the Browser. In N. Manset and P. Forshay, editors, *Astronomical Data Analysis Software and Systems XXIII*, volume 485 of *Astronomical Society of the Pacific Conference Series*, page 277, May 2014.

J. J. Condon, W. D. Cotton, E. W. Greisen, Q. F. Yin, R. A. Perley, G. B. Taylor, and J. J. Broderick. The NRAO VLA Sky Survey. *Astronomical Journal*, 115:1693–1716, May 1998. doi: 10.1086/300337.

M. P. Deisenroth, A. A. Faisal, and C S. Ong. Mathematics for machine learning, 2019. URL `https://mml-book.github.io/book/mml-book.pdf`.

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

H. T. Intema, P. Jagannathan, K. P. Mooley, and D. A. Frail. The GMRT 150 MHz all-sky radio survey. First alternative data release TGSS ADR1. *Astronomy and Astrophysics*, 598:A78, February 2017. doi: 10.1051/0004-6361/201628536.

Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch, 2017.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Fernando Pérez and Brian E Granger. Ipython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3), 2007.

Prabhakar Tiwari. Radio spectral index from NVSS and TGSS. *Research in Astronomy and Astrophysics*, 19(7):096, Jul 2019. doi: 10.1088/1674-4527/19/7/96.

Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

6 Appendices

6.1 Artefact README.txt excerpt

Current build found at: `https://github.com/MatthewJA/blobmatch`
TGSS and NVSS radio object surveys available from:
`https://github.com/MatthewJA/blobmatch/releases/download/v0.1/TGSSADR1_7sigma_catalog.tsv.gz`
`https://github.com/MatthewJA/blobmatch/releases/download/v0.1/CATALOG.FIT.gz`

```
Directory structure:
blobmatch/
    source/
        (all .ipynb notebooks, manual_labels.csv)
    report/
        pics/
            (all plots as .pdf saved by above
            notebooks, also cut-out comparison)
        main.tex
        report.pdf
    project/
        (non-plot outputs of notebooks except
        sky_matches.csv and sky_catalogue.csv,
        also defunct scipts and plots)
    README.txt
    LICENSE
    .gitignore
```

blobmatch/source/ .ipynb notebooks:

- feature_vectors.ipynb
  (constructs feature vectors from source catalogues in a patch, labels based off of positional matching)
- torch_logistic_regression.ipynb
  (performs logistic regression using pytorch, partitions sky into physical objects)
- sklearn_logistic_regression.ipynb
  (performs logistic regression and random forest using sklearn)

- score_feature_vectors.ipynb
  (scores the match feature vectors in patch against various metrics, finds each individual source's best match)

- sky_positional_matching.ipynb
  (constructs catalogue of primitive feature vectors over entire sky in catalogues, performs positional matching)

6.2 Loss curve

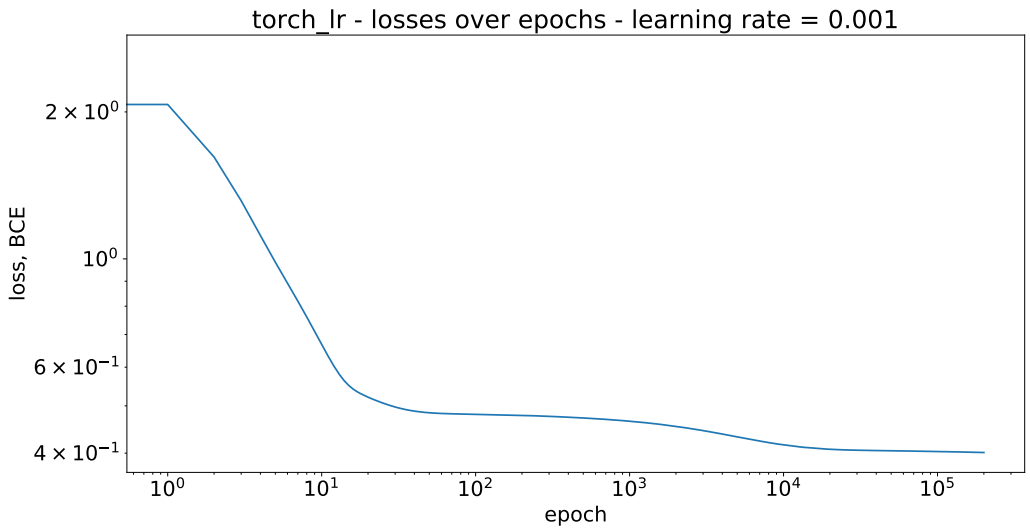

Figure 11: Loss curve for pytorch logistic regression without separation, note the stabilisation after 1e5 epochs of training, we can be hopeful that we've reached some local minimum

6.3 Scikit-learn

We also ran logistic regression through scikit-learn [Pedregosa et al., 2011], this proved both faster and more accurate than the pytorch model. See table 1 for accuracy.

Figure 12: Logistic regression weights (importances) from scikit-learn model

6.4 TGSS-NVSS view on partition

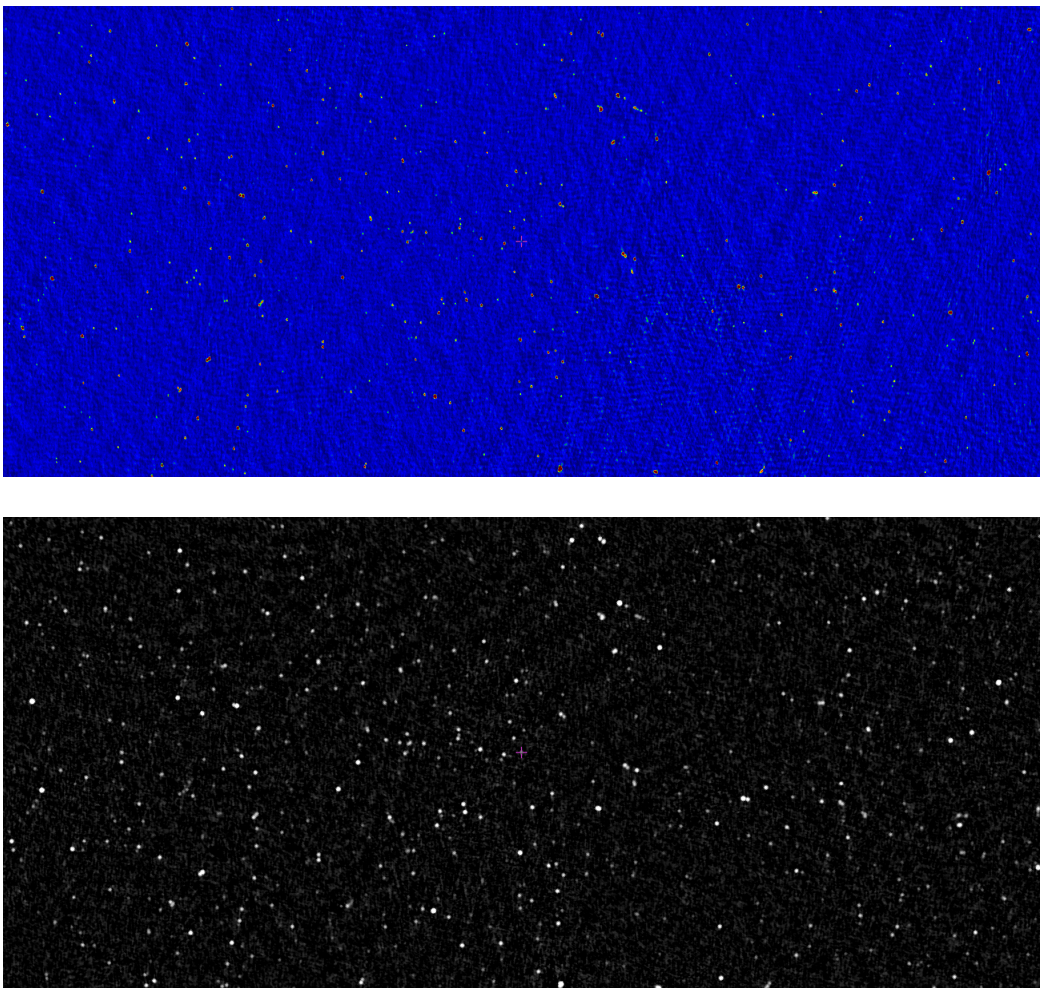

Figure 13: Cut-outs of TGSS and NVSS over the partitioned patch of sky shown in figure 9, taken from AladinLite [Boch and Fernique, 2014] in 5 degree window around J101436.8-270532