Classification of pulsar candidates with a neural network and a random forest

Elisabeth Strøm

December 19, 2019

Abstract

We use a random forest and feedforward neural network, for the binary classification of imbalanced potential pulsar candidates in the HRTU2 data set Lyon et al. (2016); Lyon (2016). The true pulsar signals only accounts for 10 % of the samples. To counter this we apply weights and SMOTE resampling, and run three analyses for each method, for a total of six analyses. Our metric of choice, had a near constant value of $F_1 = 0.88$ for both methods. Using weights and resampling did not improve the F_1 score, merely the recall (0.93). Compared to Lyon et al. (2016), we report a higher F_1 and precision score, but a lower recall. The F_1 score proves resistant to parameter tuning, meanwhile the recall and precision fluctuates between 0.80 and 0.95 during tuning. Future studies of HTRU2 may be better served by tuning those instead. Using the HRTU2 data set with other features might also improve the F_1 score, as Wang et al. (2019) reports $F_1 = 0.91$ with other features. Due to speed considerations during tuning and learning, the random forest algorithm is the preferred method for this data set.

Contents

1 Introduction

Since pulsars were first discovered by Jocelyn Bell in 1967, they are being found at an increasingly rapid rate. However, pulsars are difficult to detect as their signals are smeared by charged particles in the interstellar medium (ISM) on their path toward Earth. The majority of potential pulsar detections, called 'candidates', arise due to radio frequency interference (RFI), and noise. The aim of this report is to use different machine learning algorithms on the pulsar candidates in the HTRU2 data set (Lyon, 2016), to classify the real pulsar signals, from the false detections.

Pulsars have several potential uses in astronomy. Due to their predictable radio emission, we can measure subtle changes in their period, caused by, for instance, gravitational waves (Taylor et al., 2019), or a strong gravitational field around a supermassive black hole (Angélil et al., 2010). The data collected form pulsars can be compared to models of the evolution (Faucher-Giguere and Kaspi, 2006) and birthrate of pulsars (Keane and Kramer, 2008) and supernovae. They can also be used as probes of the interstellar medium, as their radiation is dispersed by the ISM along the line-of-sight towards Earth (Ferrière, 2001).

In the early 2000's, identifying pulsar signals was done manually by experts. This is a time consuming process, and getting increasingly infeasible as observational tools have become better, and large-scale surveys ensure candidates are detected at an increasing speed. For instance, the Square Kilometer Array (SKA), expected to go into commission in the mid 2020's, is expected to find at least 20000 pulsars (Smits et al., 2008), and have 200 million candidates (Wang et al., 2019). Machine learning algorithms have become a common solution to this candidate problem.

Several authors have used various methods to separate real pulsars from spurious ones (Lyon et al., 2016; Eatough et al., 2010; Bates et al., 2012; Morello et al., 2014; Wang et al., 2019). We have chosen to employ an artificial feedforward neural network, and a random forest, and will be comparing our findings with that of Lyon et al. (2016) (neural network), and Wang et al. (2019) (random forest) who have worked with the same data set.

The data set we examine in this report, HTRU2, is derived from the High Time Resolution Universe Survey (HTRU) using the Parkes 64-m radio telescope (Keith et al., 2010). We look closer at HTRU2 in section 2. The theory behind our algorithms, and the methods we employ are described in Section 3. How we implement our methods numerically is shown in Section 4. The result of our analyses can be found in Section 5, our discussion of our findings in Section 6, and finally our conclusions in Section 7.

2 Pulsar candidates data set

A pulsar is a rare type of rotating neutron star that is highly magnetized, causing it to emit radiation from its magnetic poles, detectable as radio waves here on Earth. The axis of the magnetic poles is not necessarily parallel to the rotation axis. This means that the emitted beam of radiation is only detectable whenever the magnetic axis points toward Earth. This happens regularly, with a period between a few milliseconds, and a few seconds.

The data set that is considered in this project, HTRU2 (Lyon et al., 2016; Lyon, 2016) is a subset of pulsar candidates detected in the HTRU survey. It contains 16259 false detections, and 1639 real pulsars, for a total of 17898 candidates. These samples have all been checked by human annotators. There are no missing values.

The data set has 8 numerical features plus 1 binary response variable. They are as follows:

1. Mean of the integrated pulse profile.
2. Standard deviation of the integrated pulse profile.
3. Excess kurtosis of the integrated pulse profile.
4. Skewness of the integrated pulse profile.
5. Mean of the DM-SNR curve.
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.
8. Skewness of the DM-SNR curve.
9. Class of pulsar candidates, 1 for true pulsars, 0 for false pulsars.

A pulsar signal is detected as a series pulses. These tend to be weak, and hard to discern from noise. To increase the signal-to-noise (S/N) ratio, several pulses are summed up, or rather, integrated with respect to the rotational period. The integrated profile usually has the shape of a single Gaussian peak (sometimes they are doubly peaked), and is unique to each pulsar. They are also averaged over frequencies and time.

Skewness is a measure of how asymmetrical the profile is about its mean. A distribution with positive skew tilts to the left of the mean, a negative skew means it tilts to the right of the mean. The skewness of a normal distribution is 0.

Kurtosis is a measure of how fat or thin the tails of the data is. A fat tail means that there are many events, or data points, outside the normal range: outliers. A high kurtosis, indicates a fat tail. Excess kurtosis is defined as kurtosis minus 3, where 3 is the kurtosis of the normal distribution.

A pulsar signal is smeared in time in a characteristic fashion by the free electrons in the ISM. This reduces the S/N, and creates a frequency-dependent delay in the arrival the time of signal. The dispersion measure (DM), is the free electron density in between the pulsar and Earth, integrated along the line of sight. It is a quantitative measure of this effect, and the DM value is chosen so that it maximized the S/N. Signals originating on Earth are not dispersed by the ISM, so applying unnecessary corrections will lower their S/N. A candidate with a peak S/N at DM=0 on the DM-SNR curve, is therefore likely terrestrial in origin. A peak in S/N at DM$\geq$ 0, is likely to be a pulsar.

These features were chosen by Lyon et al. (2016) as they found that they maximize the difference between true signal and noise candidates, while also keeping the number of predictive features small. The exact mathematical definition of these features, can be found in their article.

3 Method

From our data set, the 8 predictor variables, $\boldsymbol{x}_i$, make up the columns of the design matrix, $\boldsymbol{X}$, while the response variable $\boldsymbol{y}$ contains the binary classification of the pulsar samples (1 for a true

pulsar and 0 for a false pulsar). As in previous classification problems, our cost/loss function is the cross-entropy loss,

$$\mathcal{C}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=0}^{n-1} \left(y_i \log \sigma(\hat{y}_i) + (1 - y_i) \log[1 - \sigma(\hat{y}_i)]\right). \tag{1}$$

Here, y is the actual binary response, $\hat{y}$ is the predicted response, and $\sigma(x)$ is the sigmoid function,

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2}$$

We will employ two different methods in machine learning to solve our problem: A feedforward neural network, and a random forest. Their algorithms are explained in sections 3.2 and 3.3, respectively.

3.1 Metrics

As stated in Section 2, this is an imbalanced data set, where the true pulsar signals only make up approximately 10 % of the total datapoints. Accuracy is therefore not the best measurement we can use.

What we want is to classify as many true pulsar signals as such, as possible, while also minimizing the false detections. This implies that we want a high recall, but also a high precision, two metrics that can not be maximized at the same time, as there is usually a trade-off between the two. We therefore choose instead to try to maximize the F_1 score, or the *harmonic mean* of precision and recall. It is given by

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{3}$$

where

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{4}$$

and

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{5}$$

where TP is the true positive samples (correctly predicted 1's), FP the false positives (wrongly predicted 1's), and FN the false negatives (wrongly predicted 0's). Here a positive sample means a sample labeled as 1, while a negative sample is a sample labeled as 0.

We will also display the accuracy, which is the correctly labeled positive and negative samples,

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{6}$$

3.2 Feedforward Neural Network

The structure of a feedforward neural network, was explained in the report of project 2, and will not be repeated here. Instead of creating our own neural network, we use the **Keras** python library together with **Tensorflow**.

Keras allow us to quickly create a neural network with multiple layers and neurons, and with several different activation functions, optimization functions, and weight initializations. We perform

a grid search with three-fold cross-validation to find the optimal hyperparameters. We use **scikit-learn**'s `GridSearchCV` for this. Other hyperparameters to tune, is the number of epochs, the size of the batches we split our samples into, the learning rate, and different regularization methods.

Regularization methods are used to prevent the network from overfitting. **Keras** offer two types: A dropout rate, and L_1 or L_2 regularization.

L_2 regularization means adding a penalty term to the loss function. The penalty term is the hyperparameter λ multiplied with the squared magnitude of the regression coefficients β. This is done in Ridge regression, and is explained in the report of project 1.

Using dropout means that randomly selected neurons are ignored during a single training iteration, meaning their weights are not updated. This gives the effect of using a totally different neural network, which will introduce noise and reduce any learned dependencies between neurons that are not significant. The network is instead forced to learn features that are relevant for several random subsets of neurons. Dropout is often an important regularization technique for deep neural networks.

We will try out both of these techniques when we tune the hyperparameters.

3.3 Random Forest

The other algorithm we will use, is the random forest. But first we must describe the decision trees a random forest consists of.

A decision tree finds the features that contain the most information on the response variable, and split the data set two-ways along the values of these features. The most informative features are the ones that reproduce the response the best. This process of finding the most informative features continue until some stopping criteria. The final stop is then the leaf nodes or leaves, where we find the final predictions of the tree. We can say that a decision tree consists of a root node where the first split of samples occurs following some test of the features, the interior nodes where the test is performed again and then a split is made, and the final leaf nodes. Between the nodes are the branches, which corresponds to possible values of the features.

A measure for how informative a predictor is, is the Gini index, or Gini impurity. The Gini index of a node is the probability that a randomly chosen sample in the node would be incorrectly labeled by the sample distribution in that node. It is given by the equation

$$I_g(n) = 1 - \sum_{i=1}^{J} p_i^2, \tag{7}$$

where n is the node, and J is the number of classes in that node, which is 2 in our case of binary classification. p_i is the fraction of a samples belonging to class i. A node is split in such a way that it minimizes the Gini index.

We will stop the splitting of nodes when a node has less than some minimum number of samples. The final prediction of a leaf, is equal to the majority occurring class of the samples in that leaf.

The algorithm of a decision tree is relatively straightforward, but quickly leads to overfitting. To counter this, the tree can be pruned back as a function of a pruning parameter α, so that we only consider a smaller subtree. Another way to avoid overfitting, is by using a random forest.

A random forest is an ensemble of decision trees.

Trees are build on a random subset of the data. The samples are chosen via bootstrapping with replacement, meaning the same sample can appear multiple times. When the time comes to predict the classes of the test data, the predictions of all trees are averaged over. Averaging the predictions made by trees trained on subsets of bootstrapped data, is called *bagging*.

Another key element to the random forest algorithm, is that when splitting a node, only a random subset of the predictors are considered. This is to ensure that the different trees remain uncorrelated.

So while each tree have a very high variance, the random sampling of data and predictors means that in average, the forest have a low variance but without increasing the bias.

3.3.1 Hyperparameters

There are a number of hyperparameters to tune in a random forest, but the forest is much faster to compute than our FFNN.

The parameters that we will tune is the number of trees in our forest, which should be somewhere around 500 to 1000. We consider the number of levels we allow a tree to have, where increasing this number means a tree better learns the data. The number of features to be considered at each split, the minimum number of samples a node must have to allow a split, and finally, the minimum number of samples in a leaf node, are also up for tuning

3.3.2 Out-of-bag error

When using a random forest, the training accuracy or F_1 score will by the design of the algorithm lie very close to 1. A way of verifying that our model is not overfitting, is by using the out-of-bag (OOB) error estimate. OOB error is the mean prediction error of a training sample, x_i, when using only the trees that did not have x_i in their bootstrapped sample. We will instead compare the OOB F_1 score against the F_1 score of the predicted samples from the test set.

4 Implementation

We import our data set into a **pandas** dataframe. All the explanatory variables are numerical, only the response y is a binary categorical variable with values 0 or 1.

As previously stated, we use **Keras** together with **Tensorflow** to create and train our feedforward neural network, and **scikit-learn** to create our random forest.

We scale the numerical training data with **scikit-learn**'s `StandardScaler`, and then apply that transformation onto the test set. We save 20 % of our samples for testing.

We use both `RandomSearchCV` and `GridSearchCV` to look for the optimal pairing of hyperparameters while using three-fold cross-validation. This is faster than in project 2 where we had problems parallelising our code. We first run a wide search with `RandomSearchCV`, and then a narrower search with `GridSearchCV`.

4.1 Dealing with class imbalance

Our pulsar-candidates data set is imbalanced, where the true pulsar candidates make up only 10 % of the total number of samples. For this reason, we will oversample the true pulsar samples using the **SMOTE** method, which is included in the **imbalanced-learn** library. SMOTE creates synthetic samples of the minority class.

In project 2, SMOTE was used incorrectly. The entire data set was resampled before we had split the data into training and test sets. What we do now instead, is that we only resample the training data, and not the testing or validation data. **imbalanced-learn**'s `Pipeline` and `make_pipeline`, is passed to the grid search to avoid resampling the validation data as well.

Another way of handling class imbalance, is to apply weights to the two classes. Whereas resamlpling alters the data set, with weights we modify the cost function to make it consider the minority class more valuable. The weights we employ are calculated as inversely proportional to class frequency. So the weights w_i for class i, is

$$w_i = \frac{N_{\text{tot}}}{J \cdot N_i}, \tag{8}$$

where N_{tot} are the total number of samples, J is the total number of classes, and N_i are the number of samples in class i. This corresponds to **scikit-learn**'s `class_weight='balanced'` parameter in `RandomForestClassifier` and `compute_class_weight`.

5 Results

For our neural network without resampling, we find that using 400 epochs with early stopping at 256 epochs, and a batch size of 64. Our bias and weights are initialized from the normal distribution with $\mu = 0$, and $\sigma_{\text{STD}} = 0.05$. We use one hidden layer with 10 neurons and a dropout rate of 0.2. Adding a second hidden layer and changing the batch size has little effect. The optimizer, learning rate, and number of neurons seems to matter the most in terms of our metric scores. When trying out all the different optimizers that **Keras** offers, the Nesterov-Adam (Nadam) optimizer (Dozat, 2016) gives the best performance with its default settings, which the documentation recommends we do not change. Using SMOTE resampling and applying weights does not improve on the F_1 score when doing new grid searches, being at best 0.88 for resampling, or 0.87 for weights. We therefore continue to use the same parameters as stated. Recall becomes slightly higher (0.88), but precision declines from 0.94 to 0.88 and 0.86 for resampling and weights, respectively.

For our random forest, through our gridsearch and manual tweaking, we find that having 500 trees with maximum 15 levels, a maximum of 2 features to consider at each node, and a minimum of 10 samples in each node, and 4 samples in each leaf, give us the highest F_1 score of 0.88, for both the OOB and test sets. If we apply weights, we achieve the same value. When resampling with SMOTE, getting the OOB F_1 score proved difficult, but we nonetheless get similar test metric scores as when applying weights or without weights.

All results can be found in table 1, along with the results reported by Lyon et al. (2016) for a multilayered perceptor neural network (MLP), and Wang et al. (2019) for a random forest. The

baseline accuracy of this data set, is 0.90.

Table 1: The F_1 score, recall, precision, and accuracy for the HTRU 2 data set from the feedfoward neural network, and from the random forest. We show the scores for the original unchanged data set, and the data set where we oversample the true pulsars (-SMOTE) and where we apply weights to the classes (-W). The results of Lyon et al. (2016) is also shown

Data set	F_1 train	F_1 test	Recall	Precision	Accuracy
			FFNN		
HTRU2	0.89	0.88	0.83	0.94	0.97
HTRU2-W	0.88	0.87	0.88	0.86	0.98
HTRU2-SMOTE	0.88	0.88	0.88	0.88	0.98
Lyon et al. (2016)	-	0.75	0.91	0.65	0.95
			Random forest		
HTRU2	0.88	0.88	0.84	0.93	0.98
HTRU2-W	0.88	0.88	0.88	0.87	0.98
HTRU2-SMOTE	-	0.88	0.87	0.89	0.98
Wang et al. (2019)	-	0.91	0.87	0.96	-

6 Discussion

We find that for both a neural network and a random forest, the computed F_1 score is the same, at 0.88. Including oversampling or class weights to counter the imbalance in the data set, does not improve our results. What seems to vary slightly is the recall and precision metrics, precision being able to reach the score of 0.93 in the original data set without weights, for both the random forest and the NN.

What we also find during our parameter fitting, is that a great many different sets of parameter values gives approximately the same metric results on the training and test sets. Examples are using two hidden layers with different neuron combinations between 5 and 20, or having a max depth in the random forest between 10 and 40, but this is also dependent on the number of trees we have, where a 100 trees with maximum 40 levels give the approximately same F_1 score as 500 trees with maximum 15 levels. Again, what varies the most in these combinations is the recall and precision scores, so it might have been an idea to try to optimize one of those instead of the F_1 score.

One odd thing that happens during training in the neural network, is that when applying weights or resampling, the validation curve of loss against epochs is lower than the training curve. As the validation set contains fewer pulsar samples than training when oversampling, it should not be easier to predict than the training data. When using the unaltered data, the effect is less pronounced, and we have confirmed that it stems from using dropout regularization. This is because only the training set is affected by dropout, but not the validation data. In the other two cases, removing the dropout changes nothing. The opposite effect is seen in the accuracy curves. This is shown in Figure 1. However, as can be seen in Table 1, it does not negatively affect the predictive power of our model.

Compared to Lyon et al. (2016) we have a higher F_1 and precision score, but a lower recall value. As Lyon et al. (2016) aimed at improving recall, and our previous observations on parameter

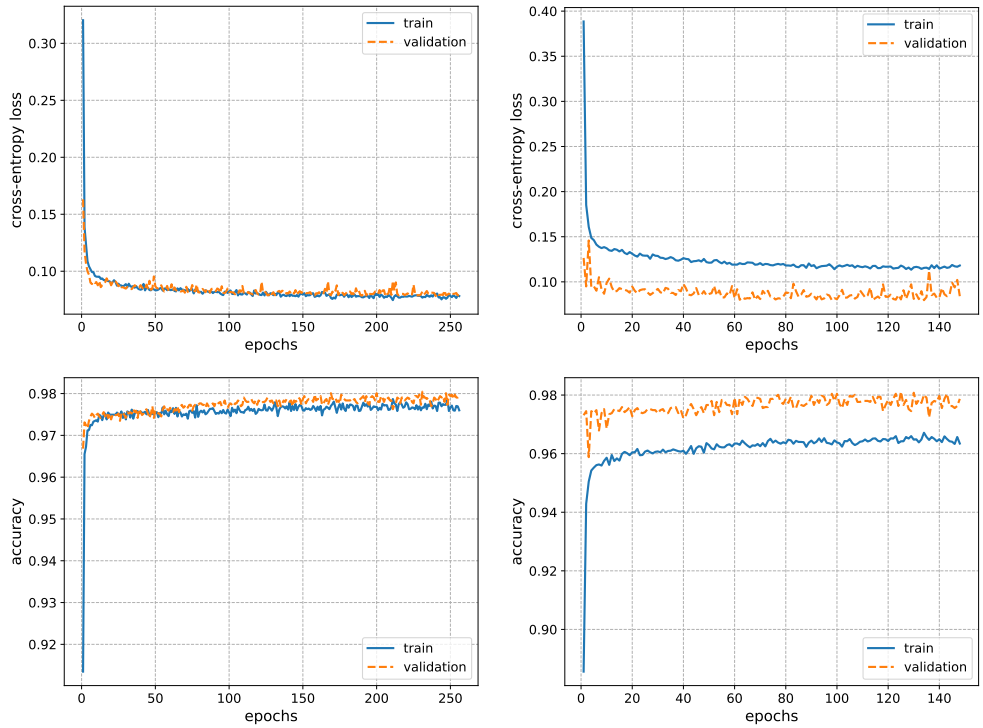

Figure 1: The cross-entropy loss (top frames) and accuracy (bottom frames) versus epochs in our neural network for training and validation data. The original data set without weights or resampling is to the left, and with resampling to the right.

tuning, this is perhaps not surprising.

Later studies have used variations of the HTRU2 data set that contains more datapoints and predictors (30 in total), and where they performed their own feature importance analysis, ending up with other predictors than Lyon et al. (2016)'s eight (Wang et al., 2019). The random forest algorithm with class weights applied used by Wang et al. (2019), reports a higher precision than ours (0.96), but a slightly lower recall (0.87) when we also consider our weighted analysis. They do not specify which metric they aimed at improving, but it appears that it might be precision. Their choice of other predictors with potentially greater predictive power, might also be the reason their F_1 score is higher than ours (0.91).

7 Conclusion

We have used a random forest and feedforward neural network, for the binary classification of potential pulsar candidates in the HRTU2 data set (Lyon et al., 2016; Lyon, 2016), as either true pulsars, or false signals arising due to noise or radio frequency interference.

The performance of the random forest and neural network was approximately the same, although the random forest was faster. The pulsar candidates data was imbalanced with true pulsar signals

only accounting for 10 % of the samples. We ran three analyses with each algorithm: One on the original data set, one where the pulsars were oversampled with SMOTE, and one where we applied weights to the classes. Irregardless of these alterations, both the random forest and neural network predicted $F_1 = 0.88$ on the unseen test data, which was the metric we aimed to maximize. Similarly, the accuracy was constantly 0.98, with a baseline accuracy of 0.90. The best precision was achieved on the unaltered data set, precision $= 0.93-0.94$ for the NN and random forest respectively. The best recall was achieved on the weighted or SMOTE sampled data set, with recall $= 0.88$. Using weights and resampling did not improve the F_1 score, merely the recall. Due solely to speed considerations during tuning and learning, the random forest algorithm is the preferred method for this data set.

Compared to Lyon et al. (2016), we had a higher F_1 and precision score, but a lower recall. The F_1 score proved resistant to hyperparameter tuning, meanwhile the recall and precision fluctuated between 0.85 and 0.95. Future studies of this data set may be better served by tuning one of those metrices instead. Looking into using other features of the HRTU2 data set can also be done, as it might also improve the F_1 score and the precision (Wang et al., 2019).

8 Appendix

All python code can be found at https://github.com/sefthus/FYS-STK4155/tree/master/project3/code

References

Raymond Angélil, Prasenjit Saha, and David Merritt. Toward relativistic orbit fitting of galactic center stars and pulsars. *The Astrophysical Journal*, 720(2):1303–1310, Aug 2010. ISSN 1538-4357. doi: 10.1088/0004-637x/720/2/1303. URL http://dx.doi.org/10.1088/0004-637X/720/2/1303.

S D Bates, M Bailes, B R Barsdell, N D R Bhat, M Burgay, S Burke-Spolaor, D J Champion, P Coster, N D'Amico, A Jameson, and et al. The high time resolution universe pulsar survey — vi. an artificial neural network and timing of 75 pulsars. *Monthly Notices of the Royal Astronomical Society*, 427(2):1052–1065, Dec 2012. ISSN 1365-2966. doi: 10.1111/j.1365-2966.2012.22042.x. URL http://dx.doi.org/10.1111/j.1365-2966.2012.22042.x.

Timothy Dozat. Incorporating Nesterov Momentum into Adam. February 2016. URL https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ.

R. P. Eatough, N. Molkenthin, M. Kramer, A. Noutsos, M. J. Keith, B. W. Stappers, and A. G. Lyne. Selection of radio pulsar candidates using artificial neural networks. *Monthly Notices of the Royal Astronomical Society*, 407(4):2443–2450, Jul 2010. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2010.17082.x. URL http://dx.doi.org/10.1111/j.1365-2966.2010.17082.x.

Claude-Andre Faucher-Giguere and Victoria M. Kaspi. Birth and evolution of isolated radio pulsars. *The Astrophysical Journal*, 643(1):332–355, May 2006. ISSN 1538-4357. doi: 10.1086/501516. URL http://dx.doi.org/10.1086/501516.

Katia M. Ferrière. The interstellar environment of our galaxy. *Reviews of Modern Physics*, 73(4):1031–1066, Dec 2001. ISSN 1539-0756. doi: 10.1103/revmodphys.73.1031. URL http://dx.doi.org/10.1103/RevModPhys.73.1031.

E. F. Keane and M. Kramer. On the birthrates of galactic neutron stars. *Monthly Notices of the Royal Astronomical Society*, 391(4):2009–2016, Dec 2008. ISSN 1365-2966. doi: 10.1111/j.1365-2966.2008.14045.x. URL http://dx.doi.org/10.1111/j.1365-2966.2008.14045.x.

M. J. Keith, A. Jameson, W. Van Straten, M. Bailes, S. Johnston, M. Kramer, A. Possenti, S. D. Bates, N. D. R. Bhat, M. Burgay, and et al. The high time resolution universe pulsar survey - i. system configuration and initial discoveries. *Monthly Notices of the Royal Astronomical Society*, 409(2):619–627, Sep 2010. ISSN 0035-8711. doi: 10.1111/j.1365-2966.2010.17325.x. URL http://dx.doi.org/10.1111/j.1365-2966.2010.17325.x.

R. J. Lyon. HTRU2. 2016. doi: 10.6084/m9.figshare.3080389.v1.

R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles. Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1):1104–1123, Apr 2016. ISSN 1365-2966. doi: 10.1093/mnras/stw656. URL http://dx.doi.org/10.1093/mnras/stw656.

V. Morello, E. D. Barr, M. Bailes, C. M. Flynn, E. F. Keane, and W. van Straten. Spinn: a straightforward machine learning solution to the pulsar candidate selection problem. *Monthly Notices of the Royal Astronomical Society*, 443(2):1651–1662, Jul 2014. ISSN 1365-2966. doi: 10.1093/mnras/stu1188. URL http://dx.doi.org/10.1093/mnras/stu1188.

R. Smits, M. Kramer, B. Stappers, D. R. Lorimer, J. Cordes, and A. Faulkner. Pulsar searches and timing with the square kilometre array. *Astronomy and Astrophysics*, 493(3):1161–1170, Nov 2008. ISSN 1432-0746. doi: 10.1051/0004-6361:200810383. URL http://dx.doi.org/10.1051/0004-6361:200810383.

Stephen R. Taylor, Sarah Burke-Spolaor, Paul T. Baker, Maria Charisi, Kristina Islo, Luke Z. Kelley, Dustin R. Madison, Joseph Simon, and Sarah Vigeland. Supermassive black-hole demographics and environments with pulsar timing arrays, 2019.

Y. Wang, Z. Pan, J. Zheng, L. Qian, and M. Li. A hybrid ensemble method for pulsar candidate classification. *Astrophysics and Space Science*, 364(8), Aug 2019. ISSN 1572-946X. doi: 10.1007/s10509-019-3602-4. URL http://dx.doi.org/10.1007/s10509-019-3602-4.