Pulsar Candidates Classification

EE 660 Project Type: Individual

Yixuan Zhou, zhouyixu@usc.edu

12/1/2018

Abstract

Pulsars, a rare type of Neutron star, is difficult to detect. The detection often requires extensive computation and the participation of astronomical experts. Recently, machine learning systems have been used in pulsar automatic detection problems. HTRU2 is a data set with 17,898 samples of pulsar candidates. In this paper, I applied several machine learning models to HTRU2, namely logistic regression, decision trees, random forest, SVM and KNN, to build a classifier that filter the pulsar candidates. Out of all classifiers, KNN performs best with f1 score 0.8976 and ROC AUC score 0.9498.

1 Introduction

1.1 Problem Statement and Goals

Pulsars are a rare type of Neutron star that produce radio emission detectable here on Earth. They are of considerable scientific interest as probes of space-time, the inter-stellar medium, and states of matter.

Each pulsar produces a slightly different emission pattern, which varies slightly with each. Thus, a potential signal detection known as a 'candidate', is averaged over many rotations of the pulsar, as determined by the length of an observation. In the absence of additional info, each candidate could potentially describe a real pulsar. However, in practice almost all detections are caused by radio frequency interference (RFI) and noise, making legitimate signals hard to find.

The problem here is basically a binary-classification problem. The goal here is to conduct a comparative study of different supervised learning methods to find the best model for filtering the pulsar candidates.

The difficulty of this problem includes two points. The first one is that it is complicated to derive a physical model for this problem. To interpret the model is far beyond my knowledge of physics and astronomy. Another difficulty is that the data samples are unbalanced. The ratio of positive samples to negative is about 10:1. This issue would be discussed in section 2.2.

1.2 Literature Review

Some researchers have already applied machine learning techniques to establish an autonomous identification system to detect real pulsars from noise (Eatough et al., 2010; Bates et al., 2012; Lyon et al., 2013; Lyon et al., 2014; Morello et al., 2014; Lyon et al., 2016).

Eatough et al. (2010) used Artificial Neural Network (ANN) to process 16 million pulsar candidates obtained by reprocessing data from the Parkes multi-beam survey. Then Bates et al. (2012) also used an ANN in the data-processing pipeline for the High Time Resolution Universe (HTRU) mid-latitude survey. They were able to reject 99% of the noise candidates and detect 85% of the pulsars through a blind analysis.

Lyon et al. (2013) studied the performance of various stream classifiers, such as very fast decision trees on pulsar data from the HTRU survey. They demonstrated the sensitivity of pulsar data to unbalanced learning problems and how imbalances severely reduce pulsar recall. Since then, Lyon et al. (2014) proposed a new classification algorithm for unbalanced data stream using Hellinger distance measurements. They can demonstrate that the algorithm can effectively increase the minority recall rate of unbalanced data.

Morello et al. (2014) used neural networks in a pulsar ranking pipeline called Straightforward Pulsar Identification using Neural Networks (SPINN). The pipeline recognizes all pulsars in the HTRU-S survey with a false positive rate of 0.64% and also helps reduce the number of scan candidates by up to four orders of magnitude.

Lyon et al. (2016) proposed a new approach to online filtering of pulsars from candidate noise using a tree-based machine learning classifier called the Gaussian Hellinger very fast decision tree. The algorithm is capable of processing millions of candidates in a matter of seconds with recall rate of 98%, and when applied to data from HTRU-1 and LOTAAS surveys, the recall rate is above 90%, and the false positive rate is less than 0.5%.

1.3 Prior and Related Work - None

1.4 Overview of Approach

In this project, I use five different machine learning models (logistic regression, decision trees, random forest, SVM and KNN) to this pulsar classification problem. Some resampling methods (random under-sampling, random over-sampling and Synthetic Minority Over-Sampling Technique) are applied to handle the unbalanced problem of this dataset. Each model is tested based on the resampled data set using the best resampling method. The performance metrics used are recall score, precision score, f1 score and Area under the Receiver Operating Characteristics (ROC AUC). Finally, the resulting confusion matrix for each model is also plotted.

2 Implementation

2.1 Data Set

HTRU2 is a data set which describes a sample of pulsar candidates. The data were collected during an analysis of High Time Resolution Universe (HTRU) Medium Latitude data by Thornton D. (2013), using a search pipeline that searched DMs between 0 to 2000 cm−3pc.

HTRU2 dataset has 17,898 instances in total, its feature space consists of 8 features and each instance is labeled as negative (spurious examples caused by RFI/noise) or positive (real pulsar examples), which means all instances can be categorized in either of these 2 classes.

The first four are simple statistics obtained from the integrated pulse profile (folded profile). This is an array of continuous variables that describe a longitude-resolved version of the signal that has been averaged in both time and frequency. The remaining four variables are similarly obtained from the DM-SNR curve.

Table 1 Feature Description

Attribute	Range	Type
Mean of the integrated profile	5.8125~192.6171	Real
Standard deviation of the integrated profile	24.7720~98.7789	Real
Excess kurtosis of the integrated profile	-1.8760~8.0695	Real
Skewness of the integrated profile	-1.7918~68.1016	Real
Mean of the DM-SNR curve	0.2132~223.3921	Real
Standard deviation of the DM-SNR curve	7.3704~110.6422	Real
Excess kurtosis of the DM-SNR curve	-3.1392~34.5398	Real
Skewness of the DM-SNR curve	-1.9769~1191.0008	Real

2.2 Preprocessing

The data in this data set is clean and complete. There is no missing data, and number of features is small. Feature engineering is not useful here since there is not much room for artificial design features.

Therefore, preprocessing mainly involves two issues, one is the over-fitting problem caused by the difference of the number of positive and negative samples (ratio about 10:1), and the other is the huge distribution range difference of 8 features (see table 1).

2.2.1 Normalization

The numerical distribution of features is extremely unbalanced. The variation between the eight features is too large. The large values can reach 1191 and the small is only 0.2. Such distributions are extremely unfriendly for input and training. When there is no normalization, training is difficult to converge. So normalization is necessary for this data set. Here I used the standardization method:

$$x' = \frac{x - \bar{x}}{\sigma}$$

where x is the original feature vector, $\bar{x}$ is the mean of that feature vector, and σ is its standard deviation.

In practice, I used sklearn.preprocessing.StandardScaler function to standardize all eight features. This standardization takes place after the data set is split.

2.2.2 Resampling

Since there are not many pulsars found in reality, the positive sample data volume is only 1639, while the negative sample has 16259, which is almost ten times the positive samples. Direct training with raw data can easily lead to over-fitting of negative samples. A variety of methods have been tried to solve this problem, including under-sampling, over-sampling, etc., which can modify the number of positive and negative samples in this dataset.

Random Under-Sampling

Random Under-sampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out. It can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge. It can discard potentially useful information which could be important for building rule classifiers. The sample chosen by random under sampling may be a biased sample. And it will not be an accurate representative of the population. Thereby, resulting in inaccurate results with the actual test data set.

Random Over-Sampling

Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample. Unlike under sampling this method leads to no information loss. Outperforms under sampling. It increases the likelihood of overfitting since it replicates the minority class events.

Synthetic Minority Over-Sampling Technique (SMOTE)

This technique is followed to avoid overfitting which occurs when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example and then new synthetic similar instances are created. These synthetic instances are then added to the original dataset. The new dataset is used as a sample to train the classification models. Mitigates the problem of overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances. No loss of useful information. While generating synthetic examples SMOTE does not take into consideration neighboring examples

from other classes. This can result in increase in overlapping of classes and can introduce additional noise. SMOTE is not very effective for high dimensional data.

In practice, the SMOTE is implemented using imblearn.over_sampling.SMOTE.

The resampling step is performed on the training set after splitting the training set and test set. Otherwise, the problem of data leakage will occur during the over-sampling, which will lead to the illusion that the model is very effective in evaluation.

This step is also performed after standardization. Because the SMOTE I used here is based on KNN. If not standardized, KNN is easily affected by the changing feature columns.

2.3 Dataset Methodology

The whole data set is split into training and test set with ratio 7:3, so there are 12,529 data points for training and 5,369 data points for test.

5-fold cross validation is used in two places:

1) model tuning
2) model evaluation (resampling test and final model test)

The data set is split using sklearn.model_selection.ShuffleSplit. And the 5-fold cross validation is implemented using sklearn.model_selection.cross_val_score.

The overall data flow can be described as follows:

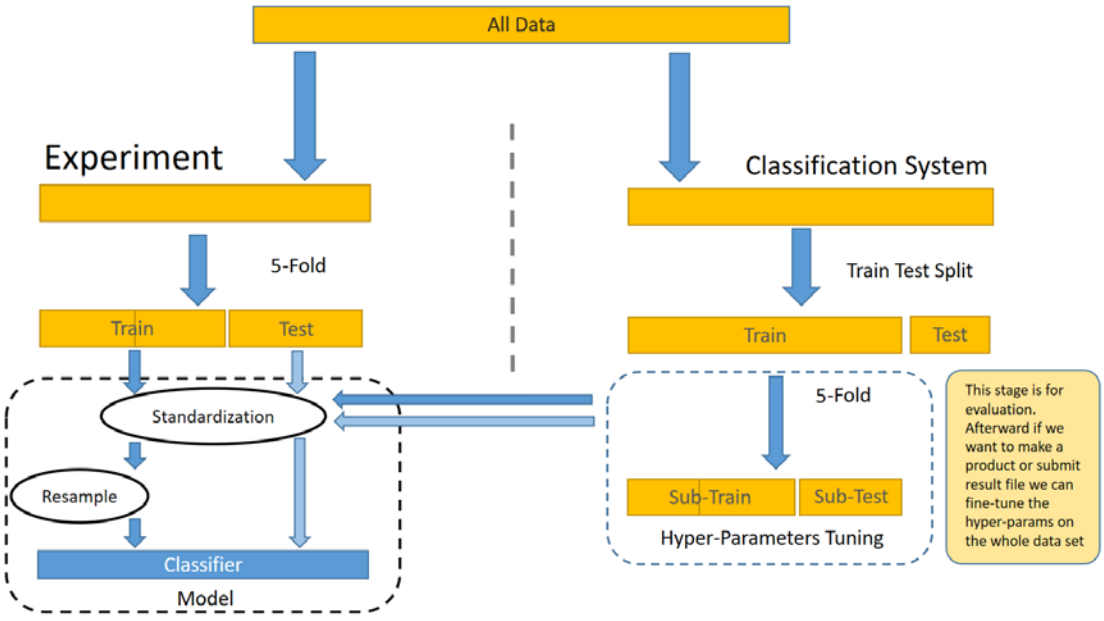

Figure 1 Data Flow

2.4 Training Process

2.4.1 Decision Tree

Decision Tree model is a classifier with tree-shape structure. In decision tree, each node, or leaf, represent class labels while the branches represent conjunctions of features leading to class labels.

The model of Decision Tree is to recursively partition the input space into multiple sub-regions and assign corresponding weights to each resulting sub-region:

$$\hat{f}(X) = \sum_{m=1}^{M} c_m \mathbb{I}\{(X1, X2) \in R_m\}$$

where $\mathbb{I}$ denotes the indicator function, and c_m is the corresponding weight in region R_m.

Decision trees have good interpretability. It can handle both linear and non-linear boundaries and include automatic feature selections. And it works fast. So I use Decision Tree for the baseline.

The model is implemented using sklearn.tree.DecisionTreeClassifier. The hyper-parameters are *max_depth* (the maximum depth of the tree), *min_samples_split* (the minimum number of samples required to split an internal node) and *min_samples_leaf* (the minimum number of samples required to be at a leaf node).

2.4.2 Random Forest

Random Forest begins with a standard machine learning technique Decision Tree which, in ensemble terms, corresponds to the weak learner. Random Forest combines trees with the notion of an ensemble. Thus, in ensemble terms, the trees are weak learners and the forest is a strong learner. In addition, Random Forest repeatedly draws multiple subsets from the entire data set with replacement. In machine learning literature, this method is called Bagging. In this way, the variance of the final model can be reduced, resulting in a consistent estimator.

The predicted outputs are defined as following:

$$\mathbb{p}(\hat{y} = c|x, D^*) = \frac{1}{B}\sum_{b=1}^{B} \mathbb{P}_c^b(x)$$

where B denotes the number of trees in the forest and $\mathbb{P}_c^b(x)$ denotes the probability of label c in the bth tree, and D^* is the corresponding data subset.

Random forest is simply a collection of decision trees, and I expect it to have better performance than a single Decision Tree in this problem.

The model is implemented using sklearn.ensemble.RandomForestClassifier. The hyper-parameters are *n_estimators* (the number of trees in the forest.), *max_depth*

(the maximum depth of the tree), *min_samples_split* (the minimum number of samples required to split an internal node) and *min_samples_leaf* (the minimum number of samples required to be at a leaf node).

2.4.3 Logistic Regression

Logistic regression predicts the probability of an outcome that can only have two values. The prediction is based on the use of one or several predictors (numerical and categorical).

$$s = \sum_{i=0}^{d} w_i x_i$$

with logistic function θ:

$$\theta(s) = \frac{e^s}{1 + e^s}$$

Logistic regression is a linear model, which has good anti-noise performance. The number of features in this problem is not large, and thus logistic regression can be applied to this problem.

The model is implemented using sklearn.linear_model.LogisticRegression. The hyper-parameters are *penalty* (l1 or l2, used to specify the norm used in the penalization) and *C* (inverse of regularization strength).

2.4.4 Support Vector Machine

Support Vector Machine (SVM) optimizes over support vectors and associated hyperplanes. This method is often called large margin classifier.

Given training data $x_i \in \mathbb{R}^d$ and class labels $y \in 0,1$, the classification problem can be formulated as an optimization problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \zeta_i$$

$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \qquad \zeta_i \geq 0$$

Its dual is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

$$\text{subject to } y^T \alpha = 0, 0 \leq \alpha_i \leq C$$

where e is the vector of all ones. $C \geq 0$ is the upper bound, and $Q_{ij} = y_i y_j \cdot K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T \phi(x_i)$ is the kernel.

According to problem formulations above, the hyper-parameter C gives the amount of regularization. Also, the choice of kernel can greatly affect the final result.

The total number of samples in this problem is only 10,000, and the number of features is small. More importantly, it is a binary classification problem. All of these imply that this problem is very suitable for using SVM to solve.

The model is implemented using sklearn.svm.SVC. The hyper-parameters are C (penalty parameter of the error term), *kernel* and $gamma$ (kernel coefficient).

2.4.5 K Nearest Neighbors

K-Nearest-Neighbors (KNN) belongs to the family of supervised learning, also refers to one of non-parametric algorithms. KNN avoids modeling the underlying data distribution by not making any assumptions on the dataset while it minimizes the pairwise distances between data points in feature space. The distance metric is often Euclidean distance (L2 distance).

$$d(x, x') = \|x - x'\|_2$$

Typically, KNN has a good performance when the samples are dense and the feature space is small (no curse of dimensionality). Since the number of features of this data set is small, KNN is worth a try here.

The model is implemented using sklearn.neighbors.KNeighborsClassifier. The only hyper-parameter is the value of k, which have the model find the top k closest images according to the distance metric (l2).

2.5 Model Selection and Comparison of Results

2.5.1 Visualization

After normalizing the data, I reduce the feature dimension to 2D through PCA using sklearn.decomposition.PCA, and then visualize the distribution of the data through the scatter plot:

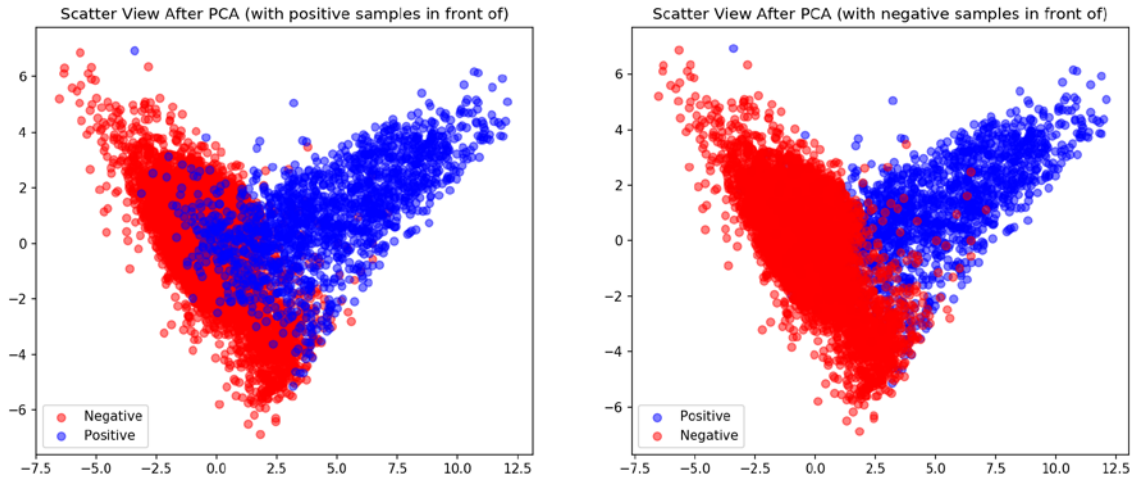

Figure 2 Scatter plot of data samples

From Figure 2, one can tell that a large part of a positive sample in a two-dimensional space can be easily distinguished from a negative sample, so it is easy to train a good classifier on this data in a simple way.

And positive samples do have many outliers mixed in negative samples, which cannot be distinguished in 2-dimensional space and require higher dimensional information. Thus, it may be difficult to train a good classifier.

2.5.2 Resampling

It is mentioned above that I applied several methods to deal with the unbalanced issue, the comparison result is as follows, all F1 scores are derived by averaging scores from 5-fold cross validation. I also tried with ROC AUC scores, but their difference among different resampling methods is unnoticeable. Thus, I choose F1 score as the metric here.

The ratio of random over-sampling is 1/5, ratio of random under-sampling is 1/5, and ratio of SMOTE is 1/6. I manually tested ratio from range 0 to 1, and these number gave the best performance.

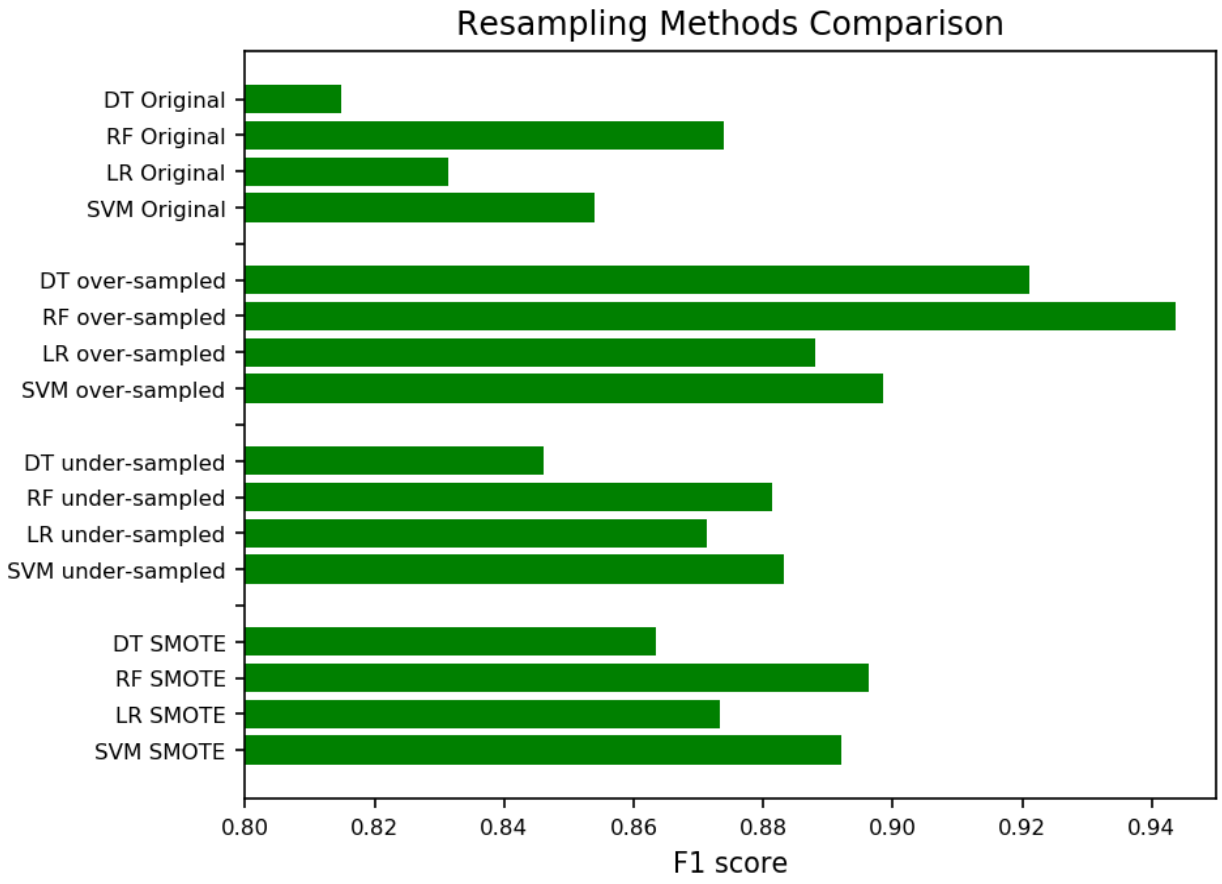

Figure 3 Resampling Methods Comparison

Resampling methods work for tree-based algorithms, namely Decision Tree and Random Forest. It makes sense because Decision Tree is sensitive to class imbalance, and Random Forest is built upon decision trees.

Logistic regression model is a probabilistic model which means it does not care much about unbalanced problem. However, it still shows improvement on F1 scores when using resampled data set.

SVMs work fine on sparse and unbalanced data. Class-weighted SVM is designed to deal with unbalanced data by assigning higher misclassification penalties to training instances of the minority class.

KNN depends entirely on the spatial distribution of the sample, regardless of whether the sample distribution is balanced. So KNN does not participate in this comparison.

From Figure 3, it is clear that using random over-sampling is the best resampling choice for this problem. Therefore, I would use random over-sampled data set with ratio 1/5 for all remaining experiments.

2.5.3 Tuning

As mentioned above, in order to ensure the objective of the final evaluation effect, I split a test set from the training set to do hyper-parameter tuning. The original test set remains untouched. The ratio of the test set here is 0.3. And the training set is resampled using random over-sampling methods mentioned above with ratio 1:5.

In the specific implementation, I use the greedy method to optimize the hyper-parameters, and each time through cross validation to get the model's score. Specifically, I define in advance what the hyper-parameters are, and their scope, and then the method picks the best hyper-parameters one by one to pick the best classifier. The function I used here is sklearn.model_selection.GridSearchCV.

The hyper-parameter candidates ranges are set as follows:

Table 2 Hyper-parameter Candidates for each model

Decision Tree	*max_depth* = 1,2,3...32 *min_samples_split* = 2,3,4...32 *min_samples_leaf* = 2,3,4...16
Random Forest	*min_samples_leaf* = 2,3,4...16 *max_depth* = 2,3,4...16 *min_samples_split* = 2,3,4...32 *n_estimators* = 2, 4, 8, 16, 32, 64, 100, 200
Logistic Regression	*C* = 0.001, 0.01, 0.1, 1, 10, 100, 1000 *penalty* = 'l1' or 'l2', *tol* = 0,001, 0.0001, 0,00001
SVM	*C* = 0.001, 0.10, 0.1, 10, 25, 50, 100 *kernel* = 'linear', 'rbf' or 'sigmoid' *gamma* = 0.01, 0.001, 0.0001, 0.00001 (for rbf and sigmoid kernel)
KNN	*n_neighbors* = 3,5,7...51 (24 odd integers in total)

The result of best hyper-parameters is as follows:

Table 3 Best Hyper-Parameters

Decision Tree	*max_depth* = 22, *min_samples_split* = 2, *min_samples_leaf* = 2
Random Forest	*min_samples_leaf* = 2, *max_depth* = 16, *min_samples_split* = 2, *n_estimators* = 200
Logistic Regression	*C* = 0.1, *penalty* = 'l1', *tol* = 0.0001
SVM	*C* = 0.1, *kernel* = 'linear', *gamma* = 0.001
KNN	*n_neighbors* = 5

3 Final Results and Interpretation

3.1 Performance

The final scores for each model are as follows, table 4 shows the scores on training set, and table 5 shows the scores on test set:

Table 4 Scores on training set

	Decision Tree	Random Forest	Logistic Regression	SVM	KNN
Recall	.9986	.9929	.8488	.8523	.8938
Precision	.9861	.9899	.9626	.9627	.9567
ROC AUC	.9979	.9954	.9211	.9228	.9428
F1	.9923	.9914	.9021	.9042	.9242

Table 5 Scores on test set

	Decision Tree	Random Forest	Logistic Regression	SVM	KNN
Recall	.8455	**.8638**	.8536	.8516	.8500
Precision	.8237	.8966	.9150	.9148	**.9520**
ROC AUC	.9136	.9268	.9228	.9218	**.9498**
F1	.8345	.8799	.8832	.8821	**.8976**

It can be seen from that table 4 that KNN performs best, followed by logistic regression, random forest and SVM. Their performances are very close to each other. Not surprisingly, the performance of the decision tree is the worst.

3.2 Confusion Matrix

A confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known. Calculating a confusion matrix gives an intuitive result of what the classification model is getting right and what types of errors it is making:

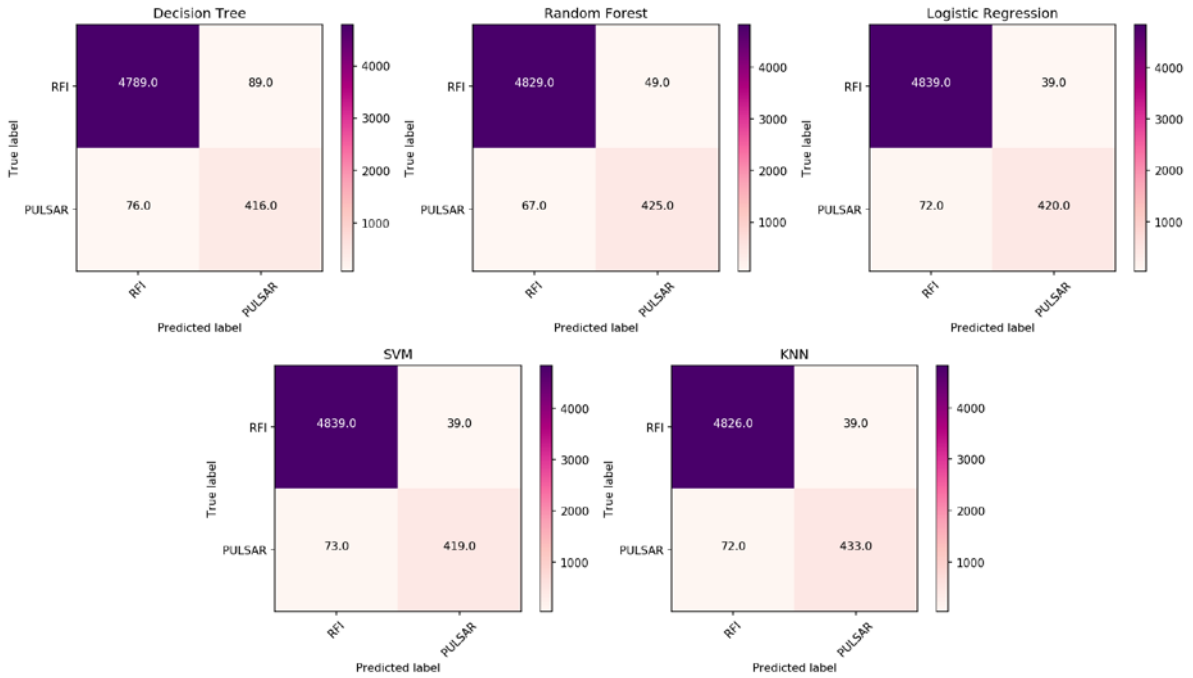

Figure 4 Confusion Matrix

3.3 Interpretation

Decision Tree & Random Forest

Theoretically, Random Forest should perform better than its base learner - Decision Tree. This is obviously correct when looking at the final result: Random Forest outscored Decision Tree in every metric. However, it worth noting that both the scores of Decision Tree and Random Forest on the training set is much higher than the score on the test set, and all scores on the training set are close to 1. This means that both models are highly over-fitted. The reason for that may due to the hyper-parameters I set for them are not good enough or perhaps tree-based model is not suitable for this problem.

Logistic Regression

The logistic regression performed pretty well. This may be due to the fact that this problem essentially can be linearly separable. It worth noting that the precision score of logistic regression on test set dropped about 5% compared to its score on training set, which means that it prone to falsely identify real pulsars as noise. However, the recall scores on both sets are almost the same. Overall, considering the computation speed and implementation difficulty, logistic regression model is a good choice for this problem.

Support Vector Machine

SVM ought to perform well in binary classification problem, and the performance is not bad actually, although not as good as KNN. However, same issue on logistic

regression also occurred on SVM. The precision score on test set is lower than which on training set. Since the kernel I used for SVM is linear, chances are that linear model is susceptible to this issue. This may explain why KNN performed better in this problem. Also, the running time of SVM is noticeably longer than which of other models.

K Nearest Neighbors

KNN performs surprisingly well in this problem that it beats all other models. There are two reasons to explain this phenomenon. First, the cluster conditions for positive and negative samples are explicit (they are not mixed together), and the data distribution is suitable for KNN. Second, probably the hyper-parameters set for other models still require further tuning. Unlike linear model as logistic regression and SVM, KNN has the same precision scores on both training set and test set. However, the recall score on test set dropped about 4%. In summary, as a non-parametric model, KNN works both simple and well in this problem.

4 Summary and Conclusions

In this project, I aimed at building a classifier based on HTRU2 data set. Firstly, I tested several resampling methods (random under-sampling, random over-sampling and Synthetic Minority Over-Sampling Technique) to handle with unbalanced data issue. It turned out that random under-sampling works well with this dataset. Then a variety of machine learning methods (logistic regression, decision trees, random forest, SVM and KNN) are experimented on this resampled data set. According to the final result, the simple KNN model outperforms all other models. However, due to lack of time, I was unable to try semi-supervised models like clustering and some more advanced models, such as neural networks or ensemble models. This is interesting to do in the future.

References

Bates, S.D., Bailes, M., Barsdell, B.R., Bhat, N.D.R., Burgay, M., BurkeSpolaor, S., Champion, D.J., Coster, P., D'Amico, N., Jameson, A., Johnston, S., Keith, M.J., Kramer, M., Levin, L., Lyne, A., Milia, S., Ng, C., Nietner, C., Possenti, A., Stappers, B., Thornton, D., van Straten, W., 2012. The High Time Resolution Universe Pulsar Survey - VI. An artificial neural network and timing of 75 pulsars. MNRAS 427, 1052–1065. 1209.0793.

Eatough, R.P., Molkenthin, N., Kramer, M., Noutsos, A., Keith, M.J., Stappers, B.W., Lyne, A.G., 2010. Selection of radio pulsar candidates using artificial neural networks. MNRAS 407, 2443–2450. 1005.5068.

Lyon, R., Brooke, J., Knowles, J., Stappers, B., 2013. A study on classification in imbalanced and partially-labelled data streams, in: Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on, IEEE. pp.1506–1511.

Lyon, R.J., Brooke, J.M., Knowles, J.D., Stappers, B.W., 2014. Hellinger Distance Trees for Imbalanced Streams. ArXiv e-prints 1405.2278.

Lyon, R.J., Stappers, B.W., Cooper, S., Brooke, J.M., Knowles, J.D., 2016. Fifty Years of Pulsar Candidate Selection: From simple filters to a new principled real-time classification approach. MNRAS 459. 1603.05166.

Morello, V., Barr, E.D., Bailes, M., Flynn, C.M., Keane, E.F., van Straten, W., 2014. SPINN: a straightforward machine learning solution to the pulsar candidate selection problem. MNRAS 443, 1651–1662. 1406.3627.