Statistical Methods 1 Group Project

Ben Anson Emma Ceccherini Sam Bowyer

January 16, 2023

Abstract

Imbalanced class priors can cause problems in many classification tasks, especially when dealing with positive and unlabelled (PU) data. In this paper, we examine the use of cost-sensitive classifiers to classify PU data as well as the effect of different priors and different proportions of labeled to unlabeled data. For our analysis, we applied support vector machines (SVMs), random forest classifiers, and Gaussian Process (GP) Classification. The dataset we analyze is taken from the HiggsChallenge [3], containing records of simulated particle decays from the Large Hadron Collider (LHC) at CERN. We construct subsets of the simulated dataset in order to analyze classification performance at varying levels of class prior imbalance—we also do the same to control the proportion of unlabelled data. We observe that the latter has a higher negative impact than the former on the classifier performance. Following work done by du Plessis et al. [4] which showed that PU classification can be performed via regular cost-sensitive classification, we see improved accuracy in our SVM and GP models—but not our random forest models—by adding prior-dependent class weights.

1 Introduction

The performance of a classifier is greatly dependent on the quality of the dataset on which it is trained, meaning that techniques must be employed to ameliorate the problems caused by imperfect data. One very common issue occurs when there is an imbalance of class priors, leading to a dataset in which there are many more training examples of one class than of another, making it hard for a model to accurately learn the characteristics of the latter class. Furthermore, if one class is by far the most common, a classifier can achieve high accuracy by simply always predicting this class, despite the fact that in reality very little has been learned.

In the case of binary classification, imbalanced priors can be even harder to deal with when the dataset contains only positive and unlabelled examples—a certain unknown proportion of the unlabelled data should be classified as positive, so it is difficult to estimate class priors. Du Plessis et al. [4] showed that assuming the class prior as known then PU classification can be performed simply as cost-sensitive classification, with the costs depending on the prior (see Section 3 for this derivation). With higher proportions of positive examples left unlabelled, the classification task becomes more difficult hence we shall examine the behaviour of our models at varying levels of this proportion, as well as with various priors.

Our chosen dataset would naturally have imbalanced priors, with signal (negative) examples corresponding to the very rare decay of a Higgs boson and background (positive) examples corresponding to similar, but much more common decays, however, the actual data comes from simulations of these decays, allowing the dataset to contain similar numbers of signal and background examples. This allows us to more easily make subsets of the dataset with different class priors and different rates of mislabelled positive data. Using this dataset in particular also allows us to discuss one final issue that can be a detriment to training: missing data.

Each training example contains information on the particles produced in a decay, however, not all decays produce the same number of particles. This leads to many training examples with missing features because the particle corresponding to those features was not produced (or at least not detected). In section 3.4.1 we briefly discuss potential techniques for dealing with missing data.

In general, we are less concerned with the specific classification accuracy of our models, and more interested in examining how their performance is affected by decreasing the quality of the dataset. This is done by changing both the class priors and the proportion of unlabelled positive samples, whilst the missing data remains an issue for all models.

An R package containing relevant functionality alongside documentation and testing of these features can be found at https://github.com/sambowyer/HiggsML.

2 Physics Background

Whilst an understanding of the physics behind the Higgs Challenge dataset is not strictly necessary for the analysis performed in this paper, here we present a brief explanation to provide some intuition towards the classification problem we are solving.

The ATLAS detector within the LHC detects and records the collision of high-speed protons—between roughly 10 and 35 such collisions every 50 nanoseconds. These collisions produce various other particles, most of which decay quickly into lighter particles which may in turn decay themselves. The type (electron, muon, photon, etc.), energy, and 3D direction of the surviving particles at the end of this decay-chain are measured by the ATLAS detector. From this data, we attempt to work our way back up the chain and infer the heaviest parent particles originally produced in the proton-proton collision—in our case we want to know when this parent was a Higgs boson.

The final surviving particles of interest to us are electrons, muons (both being charged leptons that can be detected by ATLAS directly), and hadronic taus, which are pseudo particles (bunches of smaller particles) produced in the decay of taus (these are the other type of charged lepton—they decay too quickly to be detected by ATLAS directly). The detector also records the presence of two other pseudo particles: *jets*, which originate from high-energy quarks or gluons; and the *missing transverse energy*, from which we can infer the presence of neutrinos produced in tau-decay (which escape direct detection by ATLAS). Each entry in the dataset is a record of a collision that produced two charged leptons—a tau (although we only see the resulting hadronic tau) and either an electron *or* a muon—as well as potentially (though not necessarily) some jets and the missing transverse energy. Collisions that did not result in exactly one hadronic tau and either one electron

or one muon were not included in this dataset, although they do occur commonly. For each of these particles (and pseudo particles) we are given the transverse momentum (the momentum projected onto the x-y plane where the colliding protons were traveling along the z axis) and two angles (the azimuthal angle ϕ in the x-y plane and the polar angle θ in the y-z plane[1]) so that we can fully describe the particle's energy and 3D position. That is, with the exception of the missing transverse energy, for which the polar angle cannot be computed since some particles/energy may escape detection by moving too far along the z axis. We also note that whilst each collision may produce many jets or none at all, the dataset only records information from the two jets with the highest transverse momentum (if each exists), along with the scalar sum of the transverse momentum of all jets. Along with the primitive features discussed so far (transverse momentum, azimuth, and polar angle, etc.), the dataset also contains various derived features, representing useful quantities that can be computed using the raw features, such as the estimated mass of the Higgs boson candidate.

The signal samples in this problem come from collisions in which a Higgs boson was produced and decayed into two taus, one of which further decayed into an electron or muon and two neutrinos, whilst the other decayed into a hadronic tau and one neutrino. (This process may also produce some jets and missing energy.) The background samples come from other decay processes which similarly result in at least one electron or muon and one hadronic tau. There are three such sources in the dataset: the decay of a Z boson into two taus; the decay of a pair of top quarks; and the decay of a W boson (although in this final case, the simultaneous detection of a hadronic tau and an electron or muon will only happen due to imperfections in the particle identification procedure). These three processes share many similarities to the double tau Higgs decay, and they also occur much more frequently, hence why a discussion of imbalanced priors is relevant here.

3 Methodology

The goal of binary classification is to take an input vector $\mathbf{x} \in \mathbb{R}^d$ and to assign it to one of the two discrete classes c_1, c_{-1}. The input space is thereby divided into decision regions whose boundary is called decision boundary and it is simply the 0-level set of a function $f(\mathbf{x}) = 0$. The form and estimation of this function depend on the chosen classifier.

A cost-sensitive classifier selects $f(\mathbf{x})$ to minimize the weighted expected misclassification rate. Let $y_i \in \{-1, 1\}$ for $i = 1, ..., n$ and let c_1 and c_{-1} be the per-class costs. The optimal decision function minimizes the risk:

$$R(f) : \pi c_1 P_1(f(\mathbf{x}) = -1) + (1 - \pi) c_{-1} P_{-1}(f(\mathbf{x}) = 1)$$

where $\pi = P(y = 1)$ is the class prior.

PU classification is the classification of data that consists of labeled data drawn from the positive class c_1 and unlabeled data which could be positive or negative. The authors [4] claim that PU

[1] In reality, we actually receive the *psuedorapidity* $\eta = -\ln \tan(\theta/2)$ rather than the polar angle, but these give us equivalent information.

classification can be addressed as cost-sensitive classification. They show that the risk can be written as

$$R(f) = 2\pi P_1(f(\mathbf{x}) = -1) + P_x(f(\mathbf{x}) = 1) - \pi$$

Finally, we can re-parameterize the risk as

$$R(f) = c_1 \eta P_1(f(\mathbf{x}) = -1) + c_x(1 - \eta) P_x(f(\mathbf{x}) = 1) - \pi$$

ahere $c_1 = \frac{2\pi}{\eta}$ $c_x = \frac{1}{1-\eta}$ and $\eta = \frac{n}{n+n'}$ where n and n' are the numbers of positive and unlabeled samples, respectively. Therefore PU classification can be treated as cost-sensitive classification with any classification model that allows assigning weights to classes. We applied support vector machines (SVMs), Random Forest (RF) classifiers, and Gaussian Process Classification (GP/GPC).

3.1 Support vector machines

Support vector machines (SVM) look for the decision boundary that minimizes the error on unseen datasets rather than training data. The optimal decision boundary has the smallest generalization error or equivalently the maximum margin. The latter is defined to be the smallest distance between the decision boundary and any of the samples. Assume that the decision boundary is characterized by $f(\mathbf{x}, \mathbf{w}) = 0$, then the perpendicular distance of a point $\mathbf{x}$ from the hyperplane defined by $f(\mathbf{x}, \mathbf{w}) = 0$ is given by $\frac{|f(\mathbf{x},\mathbf{w})|}{||\mathbf{w}'||}$. Therefore the distance between the hyperplane $f(\mathbf{x}, \mathbf{w}) = 0$ and $f(\mathbf{x}, \mathbf{w}) = 1$ is $\frac{1}{||\mathbf{w}'||}$, which is the thickness of the margin. Now the optimization problem is:

$$\text{argmin} \, ||\mathbf{w}'||^2 + \sum_{i=1}^{n} \epsilon_i$$

$$\text{subject to } \forall i, y_i f(\mathbf{x}_i,) + \epsilon_i \geq 1, \epsilon_i \geq 0$$

where ϵ_i for $i = 1, .., n$ are the slack variables.

This is a convex constrained optimization problem and the resulting classifier is called Soft-Margin Classifier. The dual problem is:

$$\max_{\lambda} -\frac{\tilde{\lambda}^T X^T X \tilde{\lambda}}{4} + \sum_{i=1}^{n} \lambda_i$$

$$\text{subject to } 0 \leq \lambda_i \leq 1 \sum_{i=1}^{n} \lambda_i y_i = 0$$

where $\tilde{\lambda} := [\lambda_1 y_1, ..., \lambda_n y_n]$ and $X := [x_1, .., x_n] \in \mathbb{R}^{d \times n}$

This allows the problem to be reformulated using kernels.

The SVM optimization problem can easily be reformulated to include per-class costs [1]:

$$\text{argmin}\, \frac{1}{2}||\mathbf{w}'||^2 + C[c_1 \sum_{i|y_i=1} \epsilon_i + c_{-1} \sum_{i|y_i=-1} \epsilon_i]$$

$$\text{subject to } \forall i, y_i f(\mathbf{x}_i, \mathbf{x}) + \epsilon_i \geq 1, \epsilon_i \geq 0$$

In our analysis we fit cost-sensitive SVM with the library LIBSVM [2], for computational complexity we used just the first 10000 observations to fit the models. We chose the RBF kernel for its flexibility. For each combination of π and π_{PU} the hyperparameters of the kernel and C are optimized with cross-validation on a subset of the data of size 1000. All the models are additionally fitted with weighted and unweighted SVM.

3.2 Random Forest

Random forests [7] are ensemble models consisting of many decision trees, each of which makes a prediction of an input's class. The prediction of the entire random forest is then simply the majority class predicted by all trees in the forest. Since decision trees easily overfit to training data, being models with low bias and high variance, by combining the predictions of many slightly different trees we attempt to reduce the variance with hopefully only a small increase in bias and a large increase in accuracy.

Decision trees work by asking a series of binary questions about the features of a training set, with each question chosen to maximally differentiate between training examples of different classes given the responses to the previous questions. By 'maximally differentiate' we mean that the resulting groups of training examples that have answered every question identically should be as 'pure' as possible—ideally containing only examples of a single class. There are various metrics by which we define the 'impurity' of these groups, two popular examples being the Shannon entropy and Gini impurity of the groups. By defining the proportion of class k examples in group g as p_{kg}, the Gini impurity of the group is given by $\sum_k p_{kg}(1 - p_{kg})$ whereas the Shannon entropy is calculated as $-\sum_k p_{kg} \log p_{kg}$.

Single decision trees are very interpretable models since they can be represented by binary trees with each node representing a question. The root node represents the first question, e.g. for some $1 \leq j \leq d$, is $\mathbf{x}^{(j)} > 0$? This is asked to all training examples $\{\mathbf{x}_i\}_{i=1}^n$, resulting in two new groups of training examples: $\{\mathbf{x}^{(j)} > 0\} := \{\mathbf{x}_i : \mathbf{x}_i^{(j)} > 0\}$ and $\{\mathbf{x}^{(j)} \leq 0\} := \{\mathbf{x}_i : \mathbf{x}_i^{(j)} \leq 0\}$, which will be 'purer' than $\{\mathbf{x}_i\}_{i=1}^n$. In the second layer of the tree, one question is chosen to further split up $\{\mathbf{x}^{(j)} > 0\}$ and a separate question is chosen to do the same for $\{\mathbf{x}^{(j)} \leq 0\}$. That is, unless one of the groups meets some pre-defined termination criteria, in which case it becomes a leaf node—no further questions will be asked of this group. Common termination criteria include checking whether the tree has reached some maximum depth, whether the node contains at most/at least a certain number of examples, and or whether any questions exist that could split up the group and decrease the impurity by at least some fixed amount. Training is finished once the tree has been fully developed and no more questions are to be asked of the data. Testing then works by sending a test example down the binary tree according to the answers it gives to each question

until it reaches a leaf node, at which point the tree will predict the majority class of the training examples in that leaf node. (It is important to note that the groups in the leaf nodes will often not be entirely pure, this is on purpose as it reduces overfitting.)

Clearly, the decision tree described above is deterministic given a dataset to train on, so in order to stop our forest from simply containing many copies of the same tree we must inject some randomness. The two methods by which we do this are bootstrap aggregating (*bagging*) and the random subspace method (also called *feature bagging*). With bootstrap aggregating, we train each tree of the forest on a subset of the training set, sampled at random with replacement. In the context of decision trees, the random subspace method works by only considering a random sample of features when deciding upon the question to ask a group of examples. If there are a total of d features, it is common to consider $log_2(d)$ or $\sqrt{d}$ features at each step, however, the choice is data-dependent, as are the termination criteria—these should be chosen through cross-validation before evaluation on the test set.

In our analysis, we use Scikit-learn's [10] random forest implementation with 1000 trees per forest, each of which trained on 10,000 examples without bootstrap aggregation and using Gini impurity to decide on splits based on a random subset of $\log_2 d$ features. We also found that requiring leaf nodes to contain at least 10 training examples led to improved performance.

Whilst the Scikit-learn implementation does have a `class_weights` parameter, this works by only allowing splits that leave each node with a non-negative sum of weights, hence there is no natural way to incorporate the per-class costs into our random forest models. However, we believe that random forests still provide interesting results important to this paper's analysis, especially since the inner working of decision trees will be able to deal with the missing data (set to -999) in a more categorical way than the SVMs and GPs—splitting a node by asking the question "is $\mathbf{x}^{(\text{missing feature})} > -999$?" will really be asking "is $\mathbf{x}^{(\text{missing feature})}$ missing?"

3.3 Gaussian Process Classifier

Gaussian process classification is the non-linear generalization of logistic regression [12]. We assume

$$p(y = +1|\mathbf{x}, \mathbf{w}) := \sigma(f(\mathbf{x}, \mathbf{w}))$$

where σ is the sigmoid function. In logistic regression we further assume that the function is linear: $f(\mathbf{x}, \mathbf{w}) =< \mathbf{x}, \mathbf{w} >$; while, in GP classification we put a prior over the so-called latent function $f(\mathbf{x})$:

$$f(\mathbf{x}) = \mathcal{GP}(\mathbf{0}, k)$$

where k is the kernel or covariance function. Define $\pi(\mathbf{x}) := p(y = +1|\mathbf{x}) := \sigma(f(\mathbf{x}))$, now as $f(\mathbf{x})$ is stochastic also $\pi(\mathbf{x})$ is. Note that we actually are interested in the values of $\pi(\mathbf{x})$ not $f(\mathbf{x})$. For binary classification, the latent Gaussian process prior is combined with a Bernoulli likelihood $y \sim \text{Bernoulli}(f(\mathbf{x}))$. To predict the label of a new data point $\mathbf{x}*$ we need to first compute $f*$ and

then $\pi*$. The marginal distribution of $f*$, the value of $f(\mathbf{x}*)$, is

$$p(f_*|X, \mathbf{y}, \mathbf{x}_*) = \int p(f_*|X, \mathbf{y}, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f}|X, \mathbf{y}) \mathrm{d}\mathbf{f}$$

where $\mathbf{f}$ is the latent function evaluated at all input points [12]. Finally, we can obtain a probabilistic prediction as follows.

$$\pi_* = p(y_* = +1|X, \mathbf{y}, \mathbf{x}_*) = \int \sigma(f_*) p(f_*|X, \mathbf{y}, \mathbf{x}_*) df_*$$

Unfortunately, these integrals don't have a closed-form solution and they need to be analytically approximated.

Our data set has $O(10^5)$ data points making it computationally unfeasible to directly apply GP classification. This is because GP is very computationally intensive as they involve matrix inversions which are roughly $O(n^3)$. A solution is to use Stochastic Variational GP (SVGP) [6]. Their main advantage is the ability to optimize in mini-batches, in a similar way to training a neural network. The main idea of SVGP is to augment the likelihood of $\mathbf{f}$ by a set of m pairs of inducing points and inducing variables. then using variational inference, we can approximate the marginal likelihood with a tractable lower bound on the marginal likelihood. The latter is used as an objective function to optimize the kernel hyperparameters and the inducing points. This approach results in computational complexity of $O(nm^2)$ rather than $O(n^3)$ for the full GP.

Finally, we easily extended GP classification to include per-class weights by adding them to the Bernoulli log-likelihood as follows:

$$c_1(y \log p) + c_x(1 - y)(1 - \log p).$$

For our analysis, we fit a Gaussian Process Classifier (GPC) with RBF-ARD kernel, $k(\mathbf{x}, \mathbf{x}') := \sigma^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \Theta^{-2}(\mathbf{x} - \mathbf{x}')\right)$, where Θ is the length scale parameter. We link the GP to our data via a Bernoulli likelihood $y \sim \text{Bernoulli}(f(\mathbf{x}))$. For SVGP we used 32 inducing points and a batch size of 256, as this was found to deliver the best accuracy in our preliminary experiments. For each combination of π and π_{PU}, we train a separate SVGP for 10 epochs. SVGP hyperparameters are optimized during training, so it is not necessary to perform cross-validation. Since we applied the automatic relevance determination (ARD) [9] kernel which has a different lengthscale parameter for each dimension, our model performs implicit feature selection.

For numerical reasons, it was necessary to left and right censor the dataset. Specifically, without the censoring, we found that the positive semi-definiteness constraints of the computed Gram matrices were violated during training. For simplicity, we censored every feature using $x \to \min\{10, \max\{-10, x\}\}$. We do not expect the censoring to negatively impact the classifier (except in the case of outliers, see Section 3.4.2), since censoring was performed after standardizing each feature with mean 0 and s.d. 1.

Our analysis uses the implementation of GPs provided by GPyTorch [5].

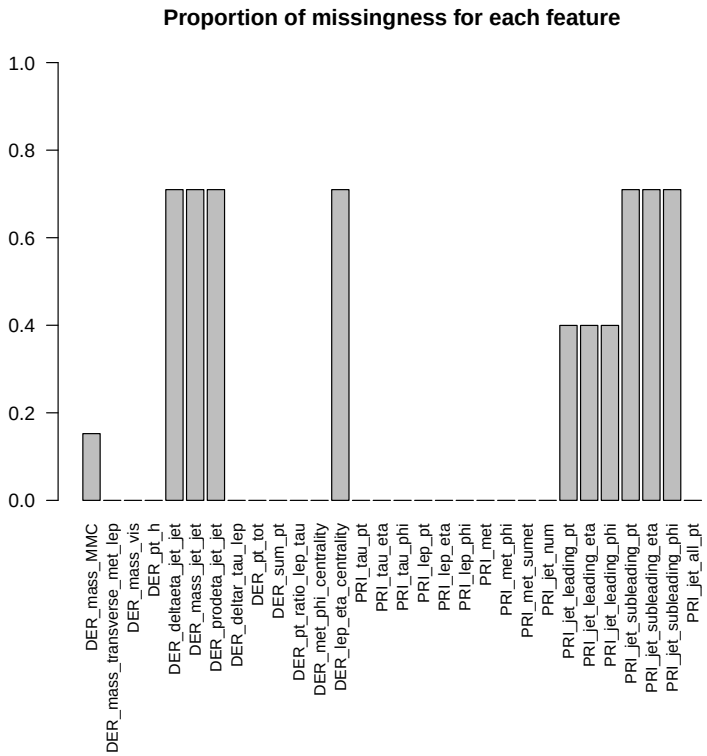

Figure 1: Proportion of missingness, for each feature.

3.4 Data set generation and feature engineering

In our analysis, we employed 50 data sets generated from the fully labeled Higgs Challenge dataset. We varied two quantities: the class prior, π, and the proportion of positive samples correctly labeled, π_{PU}.

3.4.1 Missing Data

Around 18% of data is missing in the training set, but Figure 1 shows that most features have no missingness. The features with the most missingness relate to `jet` (e.g. `PRI_jet_leading_pt`). The missing entries in `jet` features can be explained by noting that some features only exist when the number of jets is `1`, `2`, or `3` (or more). This also explains why the proportions of missingness are exactly the same for some of the features. The other features that have missing entries are `DER_mass_MMC` and `DER_lep_eta_centrality`. The latter feature is derived from other `jet` features, but `DER_mass_MMC` is missing whenever the collision event has an unexpected topology.

Using single or multiple imputations to resolve the missing data makes little sense since most

often the features in the cases where they are missing have no meaning. However, we do need to resolve the missing data in order to fit a machine-learning model. We opted to maintain the practice in the original dataset of setting missing values to -999 since this is extremely out of the range of values for the non-missing data, which would hopefully mean that the classifiers would be able to learn the significance of the missing data whilst still using their standard implementations. To help the classifier, we added extra features that encode missingness of features (for this dataset they have an interpretable meaning):

- `jets_0`,
- `jets_1`,
- `jets_2`,
- `jets_geq3`,
- `unexpected_topology` (missingness indicator for `DER_mass_MMC`),

though strictly speaking `jets_0` is redundant since $\mathbb{R}^n \ni \texttt{jets\_0} = \mathbf{1} - \texttt{jets\_1} - \texttt{jets\_2} - \texttt{jets\_geq3}$.

3.4.2 Importance of outliers

Figure 2 shows that there is a high correlation between many features. The high correlations are perhaps due to the fact that many of the features are related via physical laws. We examine a particular pair of features more closely, namely `DER_sum_pt` and `DER_pt_h`. Figure 3 shows a plot of these two features, with the red points corresponding to 'signal' data points. There is an outlier with large MTEV so we hypothesize that this may be a strong indicator for 'signal'. Similar outliers which appear as the signal can be found by comparing other combinations of features (e.g. `DER_pt_h` vs. `DER_deltar_tau_lep`). Because of these potentially useful hints for the classifiers, we avoid removing outliers where possible (though unfortunately, we have to left and right censor the data when fitting the Gaussian Process Classifier — see Section 3.3).

3.4.3 Extra HiggsML features

We found that with only the raw features provided in the ATLAS dataset, our classifiers did not perform well. We implemented some hand-crafted features, inspired by a successful Kaggle solution, HiggsML [13]. We found that the addition of these new features increased the accuracy of our models.

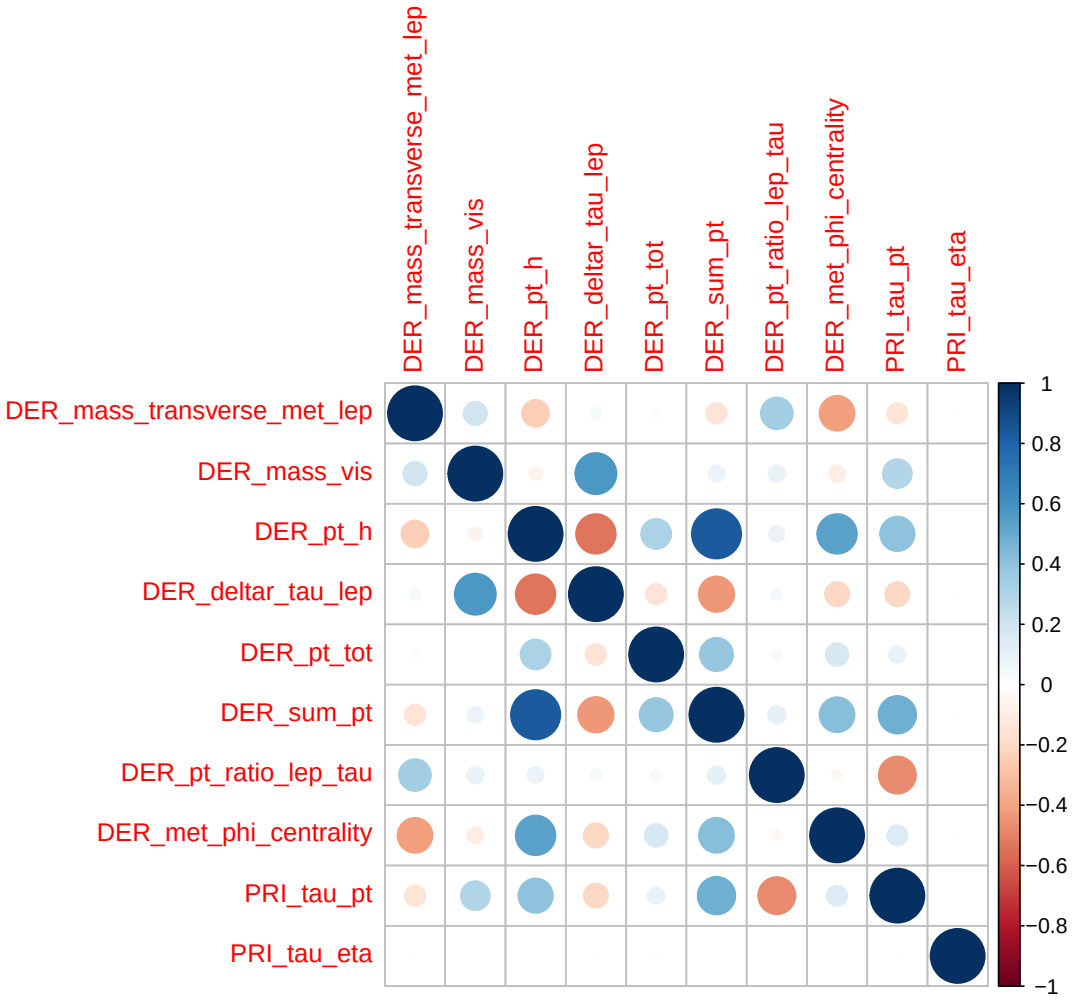

Figure 2: Correlations between a subset of features.

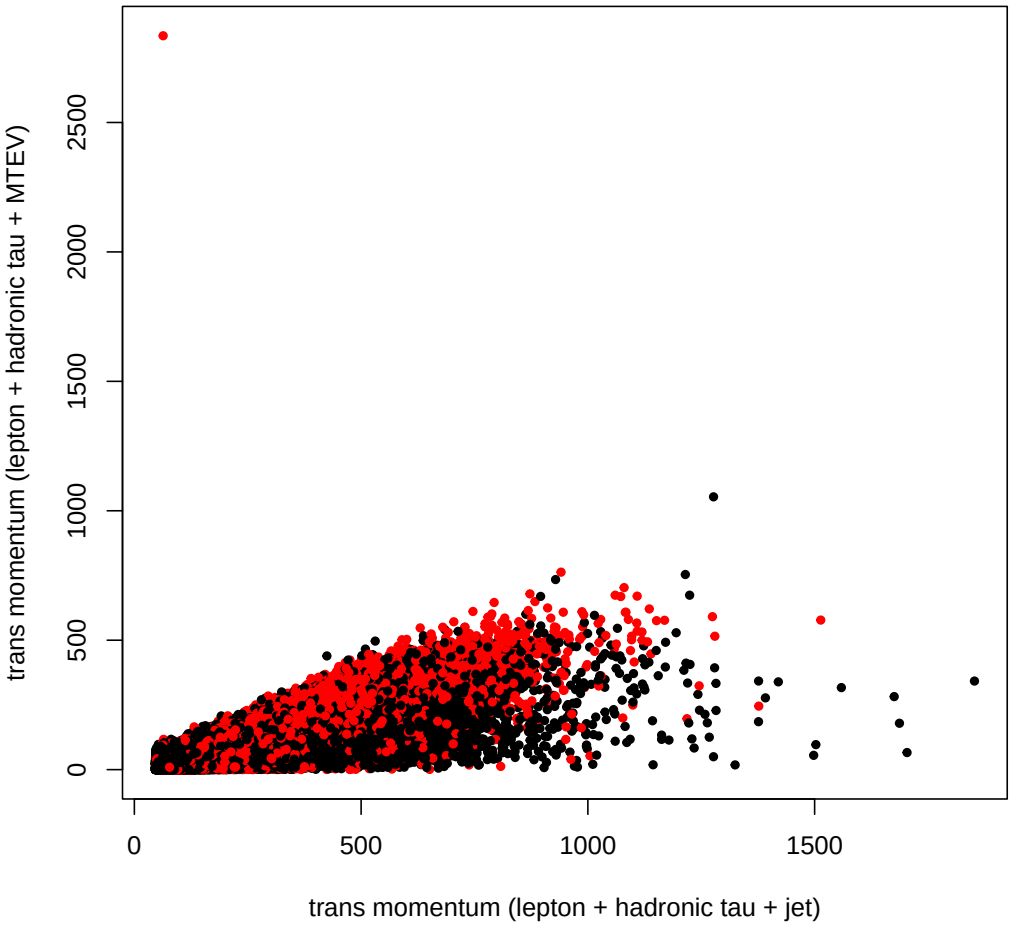

Figure 3: `DER_pt_h` vs. `DER_sum_pt`, with data points corresponding to 'signal' coloured in red.

4 Results

4.1 Excess validation accuracy

We fit 3 different models: GPC, RF, and SVM, each of them on 50 data sets as described in Section 3.4. To allow for easier comparison between models with different π, we consider the *excess validation accuracy*; that is, the accuracy improvement of the fitted classifier compared to a classifier that guesses everything as background. Note that there are other metrics that aid comparison, such as binary cross-entropy, but we prefer excess validation accuracy for its simplicity.

Figure 4.1 shows excess validation accuracy for our classifiers. As expected, the classifiers perform best when the data is perfectly labeled and the dataset is balanced. While the classification problem becomes more difficult as both π_{PU} and π decrease. All the classifiers show similar behaviour to the decrease of π_{PU} and π; however, the validation accuracy of GP classification has a jagged profile. This might be because GP models are not scalable to new data points; the inherent random component of GP optimization can lead to quite different results for every minor change to the input data set. In general, all models improve their performances as π_{PU} and π increase, reaching an excess validation accuracy of 0.35 on the balanced data set for $\pi_{\text{PU}} > 0.8$ with the Random forest classifier. This behavior is entirely explained by the fact that models struggle to learn a good decision function as the amount of positive and/or labeled decreases.

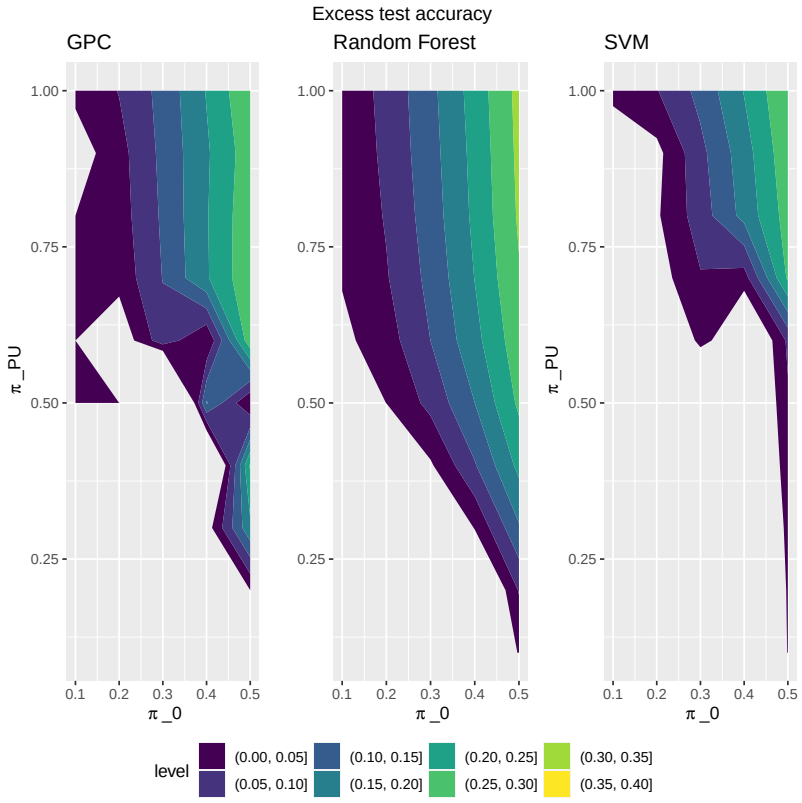

Figure 4: Excess validation accuracy of different classifiers.

4.2 Effect of π_{PU} on classifier performance

From Figure 4.1 we can sense that π_{PU} has a more negative dramatic effect than π. To further investigate this, we look at the relative change in accuracy of the Random Forest classifier as we vary π_{PU}. We choose to only perform this analysis on the Random Forest classifier as this is our final model (see Section 4.4 for details).

Let the validation accuracy of the Random Forest classifier for a fixed π be $\mathrm{va}(\pi_{\mathrm{PU}})$. π_{PU} ranges over the grid $\{0.1, \ldots, 0.9, 1\}$; as we move over the grid, we define the relative change in validation accuracy as $\Delta(\pi_{\mathrm{PU}}) = (\mathrm{va}(\pi_{\mathrm{PU}}) - \mathrm{va}(\pi_{\mathrm{PU}} - 0.1))/\mathrm{va}(\pi_{\mathrm{PU}} - 0.1)$. We can think of $\Delta(\pi_{\mathrm{PU}})$ as a metric for how much the classifier performance would degrade by decreasing π_{PU}.

Figure 5 shows the relative change in validation accuracy, for 5 different values of π, as we decrease π_{PU}. We see that, as expected, decreasing π_{PU} degrades validation accuracy. After reaching the threshold of 0.6, increasing π_{PU} has almost no effect on accuracy for all the values of π. While for $\pi_{\mathrm{PU}} \in [0.3, 0.6]$ we see the biggest degradation in performance. We can also see that π has a significant effect on performance only for $\pi_{\mathrm{PU}} \lessapprox 0.6$. This corroborates our hypothesis that a low value of π_{PU} is more critical than a low value of π.

The original Hadron collider data has low π, despite it being artificially increased by the *Kaggle weights*. Therefore, we expect this would pose a challenge in the analysis of the original data, unless $\pi_{\mathrm{PU}} \lessapprox 0.6$, for a classifier similar to ours. However, a more thorough investigation with smaller π would need to be conducted to determine the effect more precisely.

Interestingly, performance actually *increases* by moving from $\pi_{\mathrm{PU}} = 0.2$ to $\pi_{\mathrm{PU}} = 0.1$; however, we can ignore this since for π_{PU} small, our classifier fails to capture any information about the true labels.

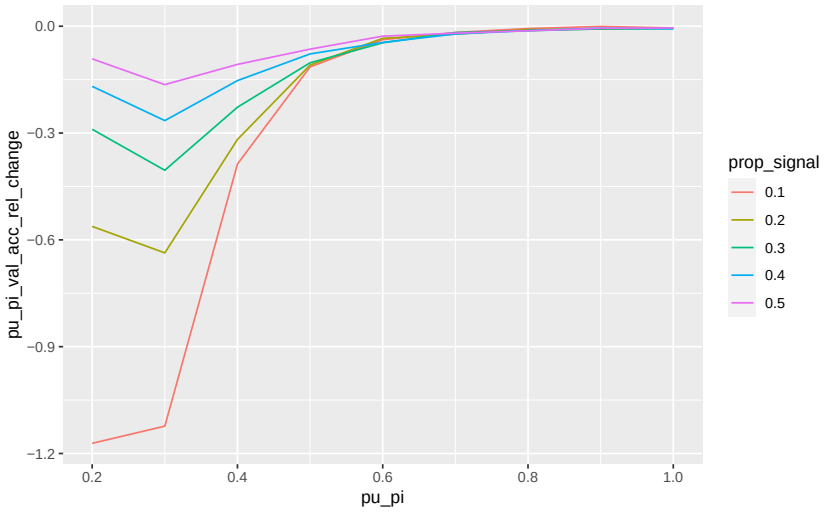

Figure 5: Effect of π_{PU} on RF performance for different π.

4.3 Effect of PU weights on the fitted model

The authors of [4] claim that adding class weights to any classifier will aid its performance, we expect this effect to be more evident as π_{PU} decreases. We analyze this effect on the balanced data set which is the easier setup, to avoid losses in accuracy due to other challenges other than PU classification. In Figure 6 we show the improvement in accuracy when moving from an unweighted classifier to a weighted one. Unsurprisingly, when π_{PU} is large, the difference is close to zero for all classifiers.

The three classifiers' accuracy has different behavior as we decrease π_{PU}. For SVM, we see little improvement, a possible explanation is that SVM might not be flexible enough for the problem at hand, so it is not able to learn the pattern with or without the weights. Then, in GPC for $\pi_{\text{PU}} < 0.3$ the weighted and unweighted accuracy is similar, this is because for such low π_{PU} the model doesn't have enough labeled examples to learn from so the accuracy is very low anyway. For $\pi_{\text{PU}} > 0.8$ we again have similar accuracy as expected, while for $0.3 \leq \pi_{\text{PU}} \leq 0.8$ the accuracy significantly improves by adding the weights. Finally, there is no improvement in accuracy in adding the weights for the random forest classifier, this is expected as illustrated in Section 3.2.

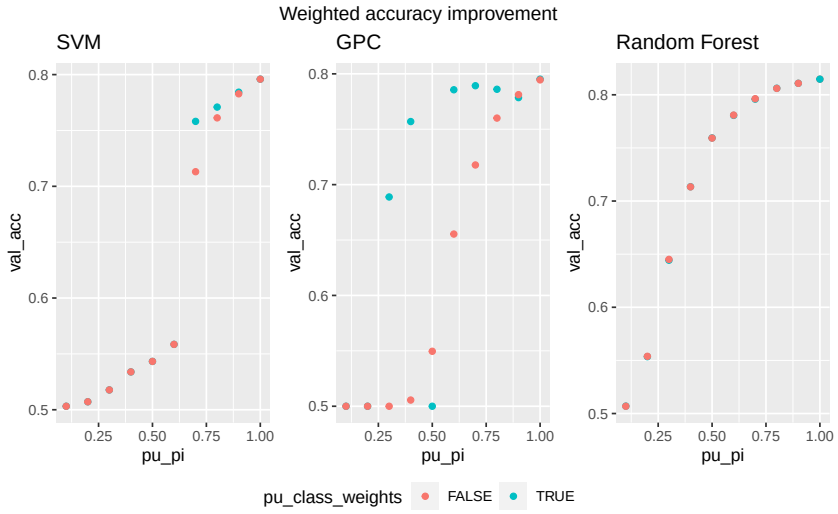

Figure 6: Improvement in GPC, SVM, and Random Forest validation accuracy when using class weights vs. not using class weights.

4.4 Model comparison

Finally, we compare the three classifiers using excess validation accuracy, for four values of π_{PU}. For $\pi_{\text{PU}} = 0.1$ all the classification performs similarly, they are all worse than a classifier that predicts all background. This is easily explained as 90% of the signal is unlabeled; this setup is too challenging even for very flexible models like GP and Random forests. Although it doesn't benefit from the class weights Random forest performs the best for $\pi_{\text{PU}} \geq 0.4$

SVM and GP performs similarly for $\pi_{\text{PU}} = 1$, while GP has higher excess validation accuracy than SVM for $\pi_{\text{PU}} = 0.9$ and for $\pi_{\text{PU}} = 0.4$ lower if $\pi < 0.4$ and higher than SVM otherwise. A possible explanation for this behaviours is that GP benefits more than SVM from a larger amount of labeled data.

Overall, we choose as the final model the Random forest for its superior behaviour regardless of the values of π_{PU} and π. Therefore we fit Random Forest classifier as described in Section 3.2 on the balanced test data set. Table 1 shows the excess test accuracy of this final fitted model on each of the 50 datasets.

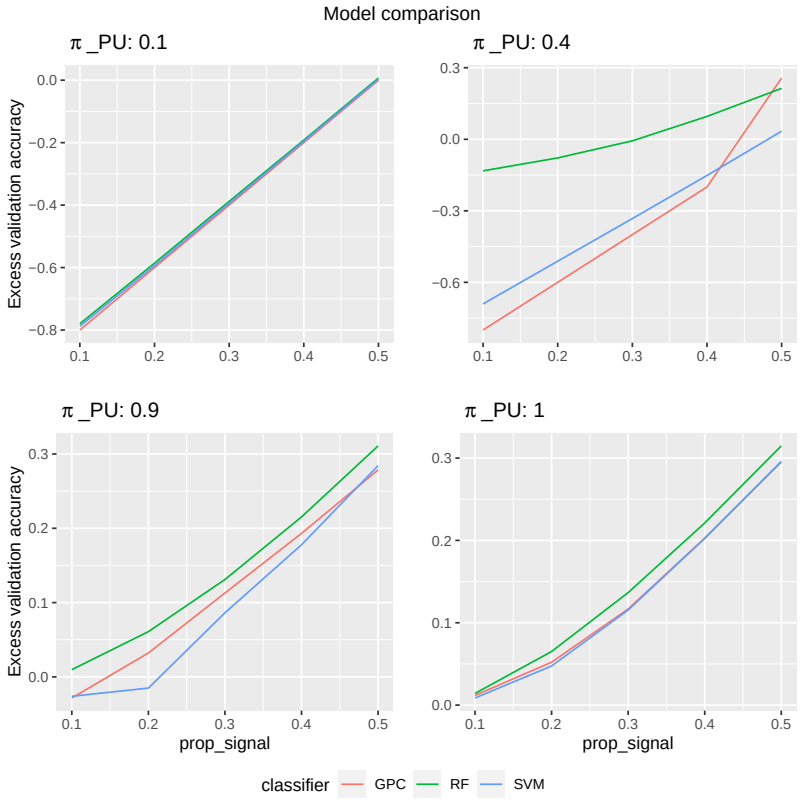

Figure 7: Comparing validation accuracy of GP, SVM and Random Forest for .

π	π_{PU}									
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.10	-0.782	-0.625	-0.324	-0.143	-0.046	- 0.012	-0.001	0.006	0.006	0.014672
0.20	-0.588	- 0.480	- 0.255	-0.085	0.002	0.032	0.049	0.052	0.062	0.066
0.30	- 0.39	- 0.289	- 0.162	- 0.008	0.071	0.102	0.119	0.128	0.135	0.139
0.40	-0.188	-0.130	0.010	0.089	0.154	0.185	0.203	0.214	0.218	0.223
0.50	0.006	0.054	0.156	0.210	0.263	0.285	0.299	0.309	0.314	0.317

Table 1: Excess test accuracy of Random Forest classifier

5 Conclusion

In this paper, we examined the effect of imbalanced priors and unlabelled data on the performance of three different classification algorithms. We found that whilst the cost-sensitive approach suggested by du Plessis et al. [4] led to significantly improved performance for the GP classifier, the benefit

seen in our SVMs and random forests was marginal at best. A simple extension to this analysis would be to apply the novel non-negative risk estimator and large-scale PU learning algorithm [8] proposed by the authors of [4]. They claim that this algorithm improves their previous methods applied in this analysis. Moreover, throughout the paper we assumed the class prior as known which is unlikely in real data, therefore a possible extension is to analyze the losses in performance when the class prior is estimated, for example by methods proposed in [11].

We analyzed the general effect of jointly changing the class priors and the proportion of unlabelled positive data on a dataset with a significant amount of missing data. Each of the three algorithms responded to these changes in a slightly different way due to the details of their implementations, however, the overall effect was largely similar, illustrating the difficulty of learning from imbalanced datasets with mislabelled training examples.

References

[1] Francis R. Bach, David Heckerman, and Eric Horvitz. Considering cost asymmetry in learning classifiers. *Journal of Machine Learning Research*, 7(63):1713–1741, 2006.

[2] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3), may 2011.

[3] ATLAS collaboration. Dataset from the atlas higgs boson machine learning challenge 2014, Jan 1970.

[4] Marthinus C du Plessis, Gang Niu, and Masashi Sugiyama. Analysis of learning from positive and unlabeled data. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[5] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with GPU acceleration. *CoRR*, abs/1809.11165, 2018.

[6] James Hensman, Alexander Matthews, and Zoubin Ghahramani. Scalable variational gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR, 2015.

[7] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995.

[8] Ryuichi Kiryo, Gang Niu, Marthinus C. du Plessis, and Masashi Sugiyama. Positive-unlabeled learning with non-negative risk estimator, 2017.

[9] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Marthinus Christoffel DU PLESSIS and Masashi SUGIYAMA. Class prior estimation from positive and unlabeled data. *IEICE Transactions on Information and Systems*, E97.D(5):1358–1362, 2014.

[12] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006.

[13] Tim Salimans. HiggsML. https://github.com/TimSalimans/HiggsML/, 2014.