Columbia University

Fall 2018 Research (COMS 6901 E)

Importance of Hyperparameters and their best values

Author:
Aditya Jadhav
[aaj2146@columbia.edu]

Supervisors:
Prof. Andreas Mueller
Dr. Jan Nicolaas Van Rijn

Fall 2018

December 21, 2018

Contents

1 Abstract

In this project, I have conducted a study of hyperparameter importance and their best values as done in [1], and investigated the results presented in the same. To this end, this work primarily focuses on the Support Vector Machine Algorithm and its hyperparameters. In particular, I look at the performance of the SVM algorithm as we optimize the gamma and C parameters for best performance. Prior distributions of hyperparameters are constructed from values that provide the best performance over a diverse range of classification tasks. Using the OpenML task performance data, I have also constructed surrogate functions that predict the performance of SVM for a given set of hyperparameter values. To tailor the hyperparameter values for particular 'types' of tasks I have also investigated similarity or clusters in the datasets based on their metafeatures like percentage of class imbalance, composition in terms of numerical and categorical features etc. Each of these studies is outlined in detail in the following sections.

2 Comparison of performance based KDEs and uniform sampling

In this work we primarily work with OpenML datasets. For the uninitiated, OpenML is an online machine learning platform for sharing and organizing data, machine learning algorithms and experiments. It is designed to create a frictionless, networked ecosystem, that you can readily integrate into your existing processes/code/environments, allowing people all over the world to collaborate and build directly on each other's latest ideas, data and results, irrespective of the tools and infrastructure they happen to use.

Specifically, there are several datasets that correspond to different Machine Learning 'tasks'. Users can download the data, perform the required operations to complete these tasks and report their findings. It is a way of crowd-sourcing machine learning experiments and documenting their results. Thus, for a set of 42 supervised classification tasks from the OpenML database, we have performance data (cross validation accuracy values) for 2000 different hyperparameter configurations per task for the Support Vector Classifier (SVC) algorithm. This data for each task can help us understand which of the hyperparameters influence performance the most and what (generally speaking) are the best values for these hyperparameters.

The objective of this exercise is to automate the tedious and time consuming process of hyperparameter tuning for classification tasks by first ranking the individual hyperparameters for each algorithm by contribution to algorithm performance followed by providing a prior distribution for each hyperparameter that is optimized for performance of the algorithm.

For the data at hand, we we will be looking at classification accuracy as the performance metric. Hwever, one might argue that accuracy is not the measure of performance for classification tasks, especially for multi-class imbalanced datasets. That however, is out of the scope of this work.

2.1 Methodology

The Support Vector Classifier (SVC) hyperparameters as provided in it's implementation in scikit-learn are as follows:

Hyperparameter	**Significance**
C	Penalty parameter C of the error term. (Floating point number)
kernel	Specifies the kernel type to be used in the algorithm. It is one of 'poly', 'rbf', 'sigmoid'
gamma	Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. (Floating point number)
coef0	Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.
shrinking	Whether to use the shrinking heuristic. (Boolean value)

Of these, C and gamma are determined to be the most influential ones when it comes to performance of SVC as per [1]. To evaluate the best values for these parameters we will filter out the upper quartile (top 25% by performance) of the performance data and fit a Kernel Density Estimator (KDE) on each of gamma and C. i.e. There will be a univariate KDE corresponding to each of Gamma and C. The KDE should in most cases correspond to values that give good accuracy. To verify this claim we will compare the performance of a Random Search using the KDE with that of a Uniform distribution. For the Random Search we will use the RandomSearchCV [3] function in the scikit-learn library. Likewise for the KDE we use an custom class that is derived from the scipy Gaussian KDE module [5].

Following the methodology in [1], KDEs are fit on two different slices of the data based on kernel type i.e. we have 2 KDEs for the gaussian and rbf kernels respectively. We will now look at the shape of the distributions produced by the two KDEs in the next subsection.

2.2 Shapes of the Distributions

We will first look at the shapes of the fitted KDEs and tune the KDE parameters to ensure they make sense and that they look like the original data that they are fitted on.

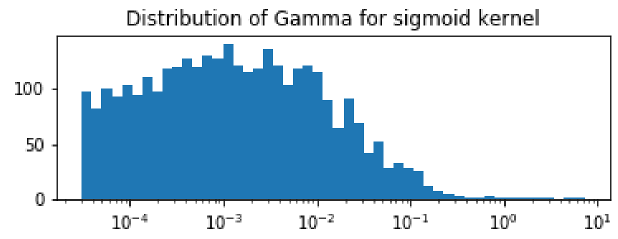

Figure 1

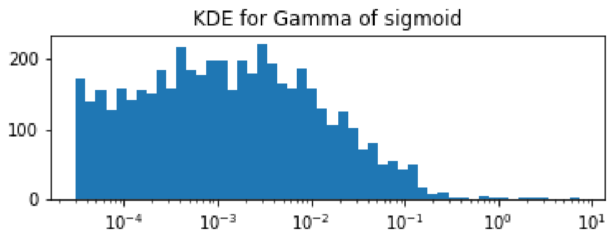

Figure 2

Figures 1 and 2 are the log histograms of the KDEs and raw data for the gamma parameter when the kernel is sigmoid. The X-axis is chosen to be log scale 10 to show the spread of the gamma values. For the KDE histogram we basically sample about 3000 values from the fitted distribution. We can see that it is a skewed distribution that prefers lower values of gamma. From a machine learning perspective that makes sense. As a side-note, it is important to mention that to ensure that the fitted KDEs resemble the inherent data, tuning the bandwidth parameter is extremely important. Using the default 'auto' or 'scott' values for bandwidth can lead to grossly different shapes. In this case a badnwidth of about 1e-07 works best.

Likewise, the distributions for the rbf kernel are shown in Figures 3 and 4:

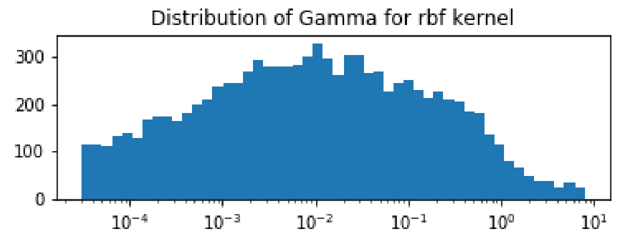

Figure 3

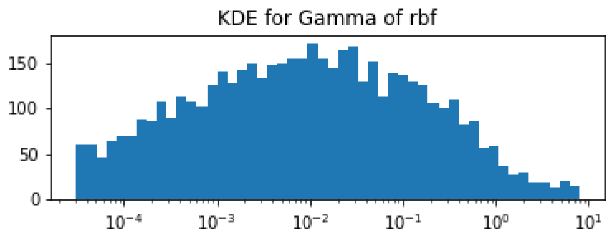

Figure 4

Notice that the log distribution for rbf is fairly symmetric and centred at 1e-02 while that for the sigmoid kernel is skewed with a peak at 1e-03.

We will wrap up the distribution plots with those for the other important parameter in SVC, i.e. the complexity parameter C:

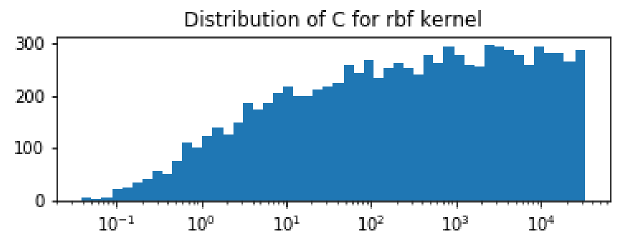

Figure 5

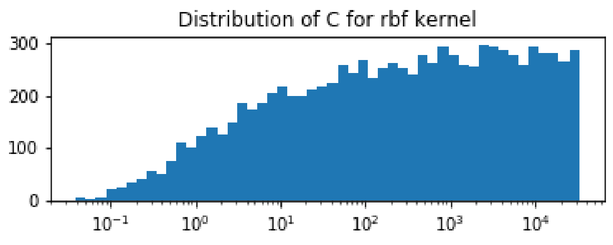

Figure 6

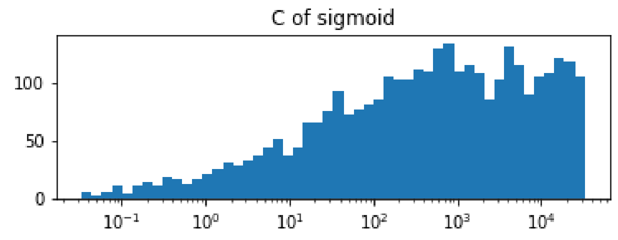

Figure 7

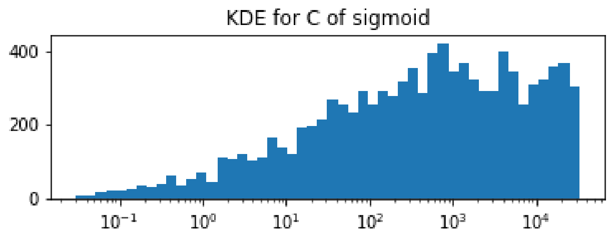

Figure 8

2.3 Convergence time and difference of accuracy for priors based on KDEs and Uniform distributions

We now come to the crux of this analysis: Measuring the performance of the KDEs in terms of best accuracy and convergence time where convergence time is the number of iterations taken by the Random Search to come up with the best accuracy model.

Ideally, we would like the KDEs to find the best parameter settings in the fastest time when compared to a uniform search across the parameters. To see if this is true, we run a random search for 20 iterations first using the KDEs for C and gamma followed by log uniform distributions over C and gamma. For reference, Figure 9 shows a representative example of a log uniform distribution.

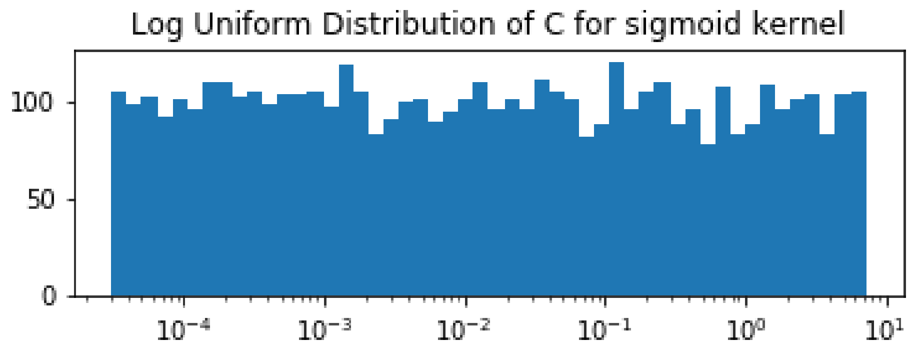

Figure 9

To compare difference in performance we look at the difference between best accuracy obtained by using a KDE and that by a log uniform distribution. Likewise, for looking at convergence time we find the difference between indices of the best accuracy obtained from KDE and log uniform distributions. Our hypothesis about the KDEs suggest that on average the index difference should be negative while the difference in accuracy should be positive.

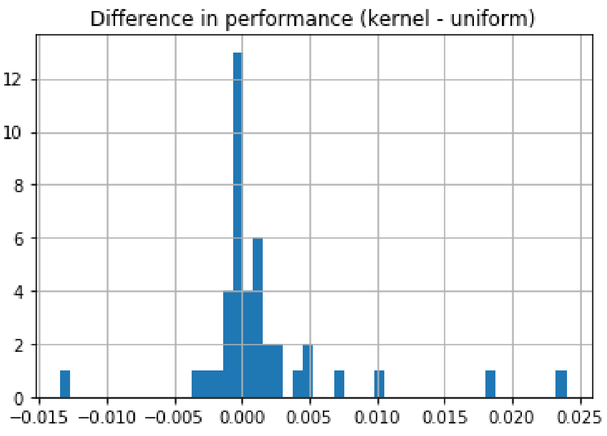

Figure 10

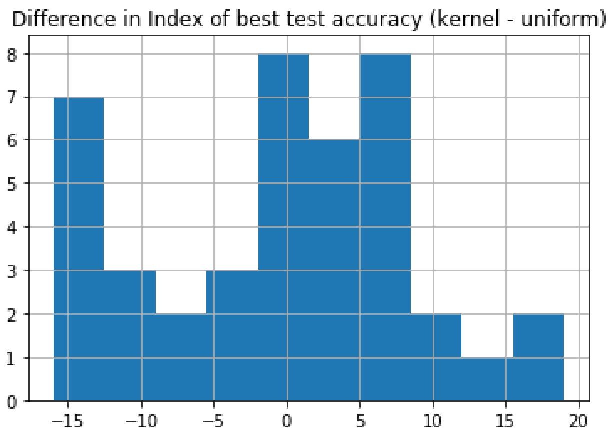

Figure 11

We observe that on average the KDEs provide the best score no faster than the log uniform ones. Also, there is no statistically significant difference in performance either. For the 42 observed tasks, the average accuracy difference is 0.1566%. While this is not surprising since we are running a search across 2 variables C and gamma over 20 iterations,

what is surprising though is the convergence time to get the best score. We observe that only 18 of the 42 tasks have the KDE getting the best performing index faster than the loguniform. On average though the mean is slightly negative. This could mean that for certain datasets, the KDE is able to find the best parameter values much faster than a uniform search. This could stem perhaps from a similarity in datasets which will be explored in the next section. Therefore, we observe that the KDEs generally speaking perform no better (in terms of accuracy or convergence time) than their uniform counterparts atleast for the case of the two most important parameters of the SVC algorithm. These findings are tabulated below:

Number of tasks with faster convergence	18 (out of 42 observed tasks)
Number of tasks with better performance	22 (out of 42 observed tasks)
Average difference in best performance index (KDE - uniform)	-0.52
Average difference in accuracy	0.1566%

3 Clustering - Finding possible clusters of similar 'type' of datasets

In this section, we look for clusters of similar 'types' of datasets (if such clusters exist at all). To this end, we have data corresponding to metafeatures of 100 datasets in the OpenML database. Like the previous section, these datasets correspond to supervised classification tasks.

3.1 Exploring the metafeatures

This metafeatures dataset contains 19 features for each of the 100 datasets, these range from percentage of minority classes, percentage of binary features, percentage of numerical features etc. All the values are scaled to be between 0 and 1.

In Figure 12 we see the univariate distributions of the 19 features. Apart from the class percentages, the features have very low variance. This will be reflected in the PCA analysis. Also, features like percentage numeric features and percentage symbolic features are complementary to each other i.e. negative correlation. Thus, we will pick one of these features for further analysis.

Furthermore, the low variance of the features indicates, a power transformation would be ideal for zooming into the patterns across each feature. This will also help us discern cthe existence of any clusters. For this purpose Figure 13 shows the power transformed distributions of the meta features.

Majority Class size and Number of Instances seem to be positively correlated from the bivariate scatterplots of the features (In the interest of space and visibility of the large plot, it is not included in the report but can be viewed here.)

As seen in Figure 13, the power transformation spreads out the distributions of the metafeatures quite well. We will leverage this transformation to better separate out clusters.

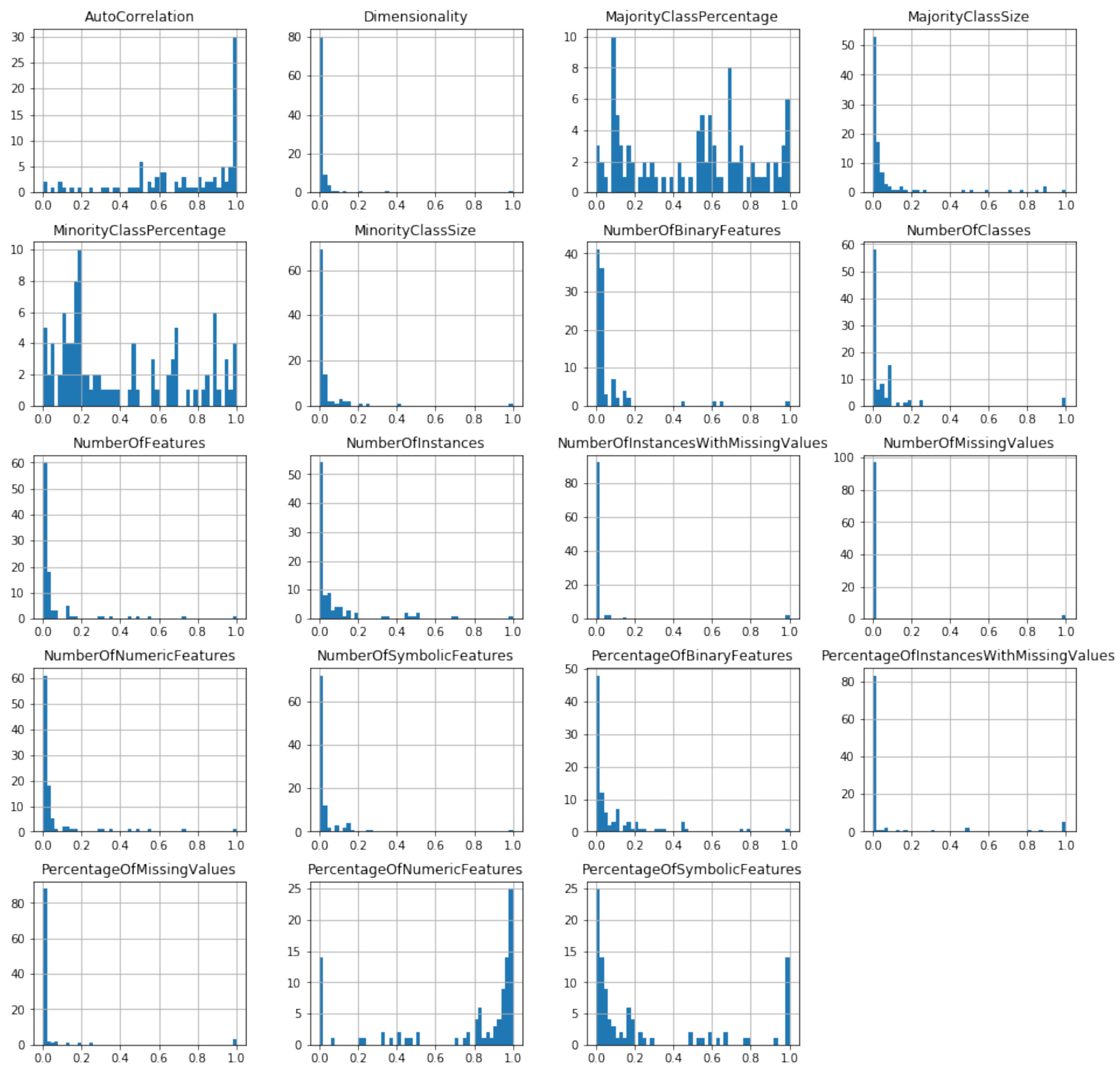

Figure 12: Univariate distributions of the raw metafeatures

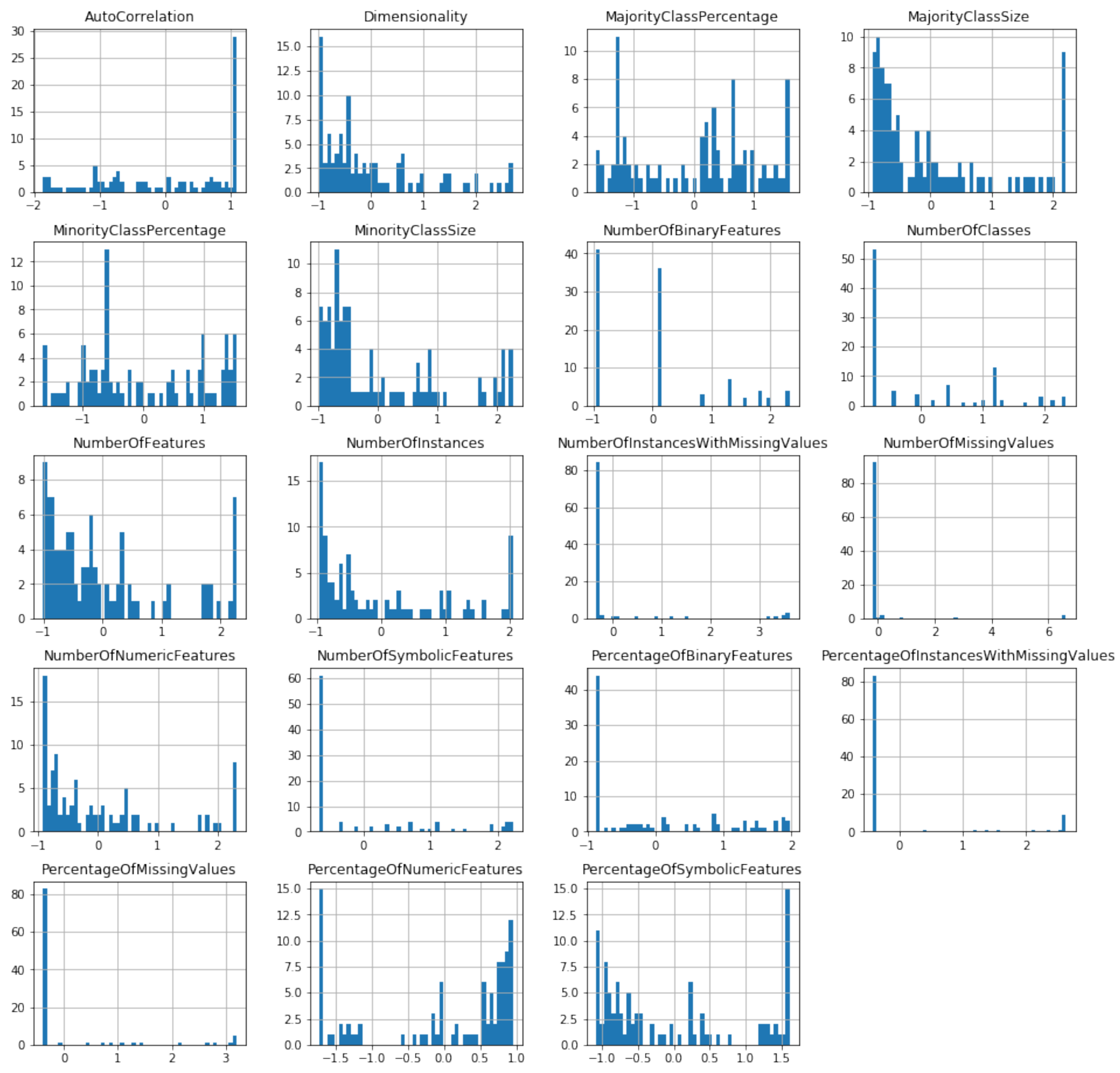

Figure 13: Power Transformed Univariate meta feature histograms

3.2 Clustering Analysis - K-means

In the following subsections we will try different unsupervised clustering techniques to find clusters of similar types of datasets. We will look at Unsupervised clustering implementations of K-means, PCA, t-SNE and DBSCAN on both the raw data as well as power transformed features.

The first approach is K-means clustering. Here we will try to look for the simplest case of 2 clusters and see if there are any visibly separate clusters.

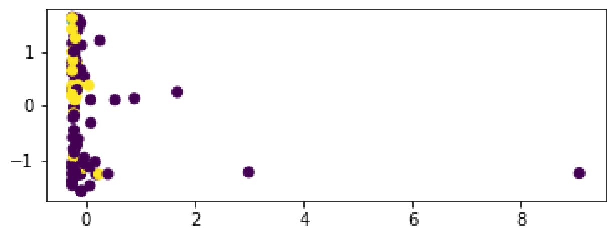

Figure 14: K-Means raw data

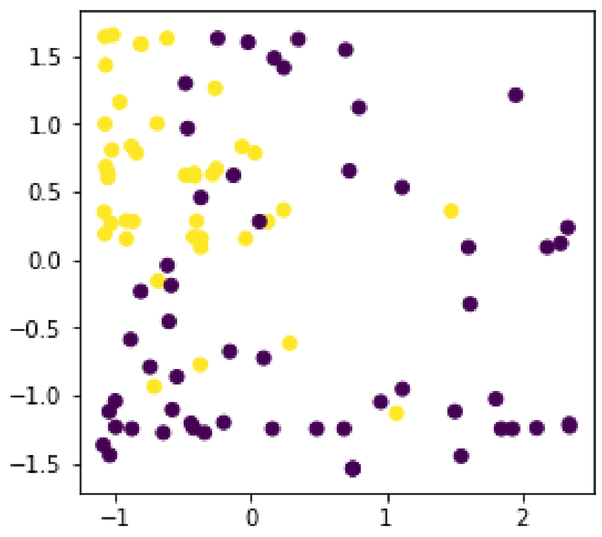

Figure 15: K-means on power transformed data

3.3 Clustering Analysis - Elliptic Curve

For the Elliptic curve prediction, we specify an impurity of 50%. This number is based loosely on the convergence numbers seen earlier wit the KDEs vs Uniform analysis. Intuitively, it makes sense to assume a 50-50 separation since for about half the tasks we observed that the convergence was significantly faster. However, no conclusive clusters can be seen as shown in Figures 16 and 17.

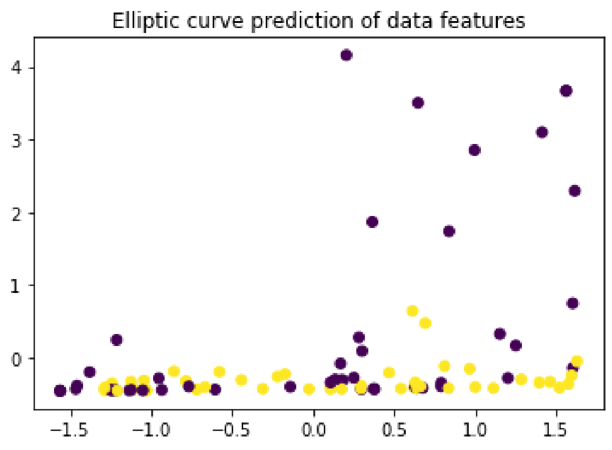

Figure 16: Elliptic raw data

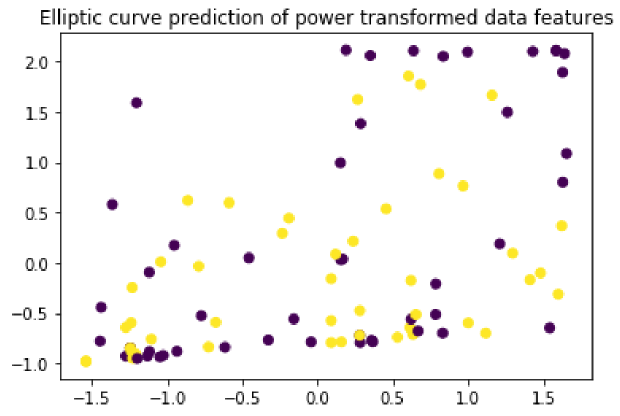

Figure 17: Elliptic curve on power transformed data

3.4 Clustering Analysis - PCA

Next up we look at the PCA plots. We had seen earlier in the univariate feature distribution plots that 2 features in the raw data had the maximum variance. The colors in the plot correspond to the predictions made in the Elleiptic curve clustering. The outlier in both plots is actually 2 tasks overlapped. They are tasks number 3946 and 3948. Both have highly imbalanced data with majority class percentage of 98% and the highest missing values in the meta features dataset.

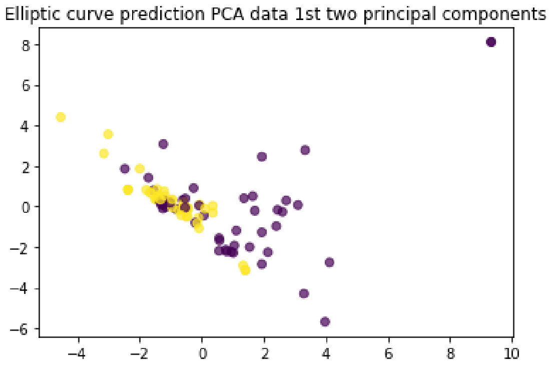

Figure 18: PCA on raw data

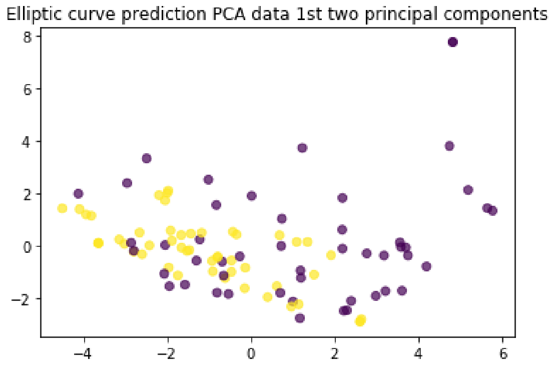

Figure 19: PCA on power transformed data

3.5 Clustering Analysis - t-SNE

Next we look at the more esoteric t-SNE clustering. As we can see for perplexity values of 20, 25 and 30 we can see 3 clusters (qualitatively). Upon further investigation, we find that the three clusters correspond to mostly the same data points! The plots for these perplexity settings is coupled with K-Means clustering to see the three distinct clusters qualitatively in Figures 20, 21 and 22. To see how much the three plots correspond to the same data points, we look at the averaged F1 score. For each combination of 2, we find that the F1 score is quite high in all 3 combinations with scores of 0.91, 0.98 and 0.91.

Also, looking further into what the clusters correspond to in terms of the meta features, we find that a cluster corresponds to data with a high percentage of numerical features (92% on average), another cluster with high class imbalance and high categorical feature content (onlyy 24% numeric features) while the final class loosely corresponds to Balanced class problems. Overall, the clusters do seem to make sense on a high level.

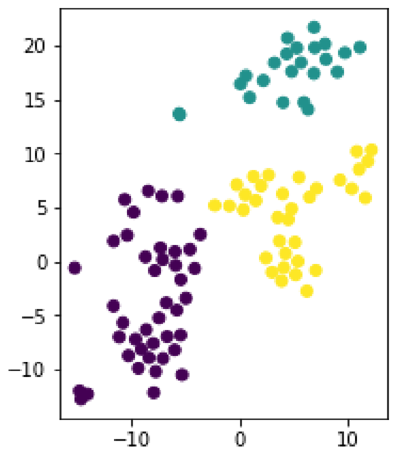

Figure 20: Perplexity of 20

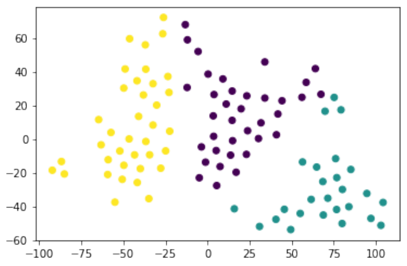

Figure 21: Perplexity of 25

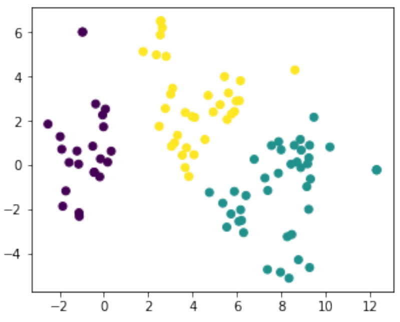

Figure 22: Perplexity of 30

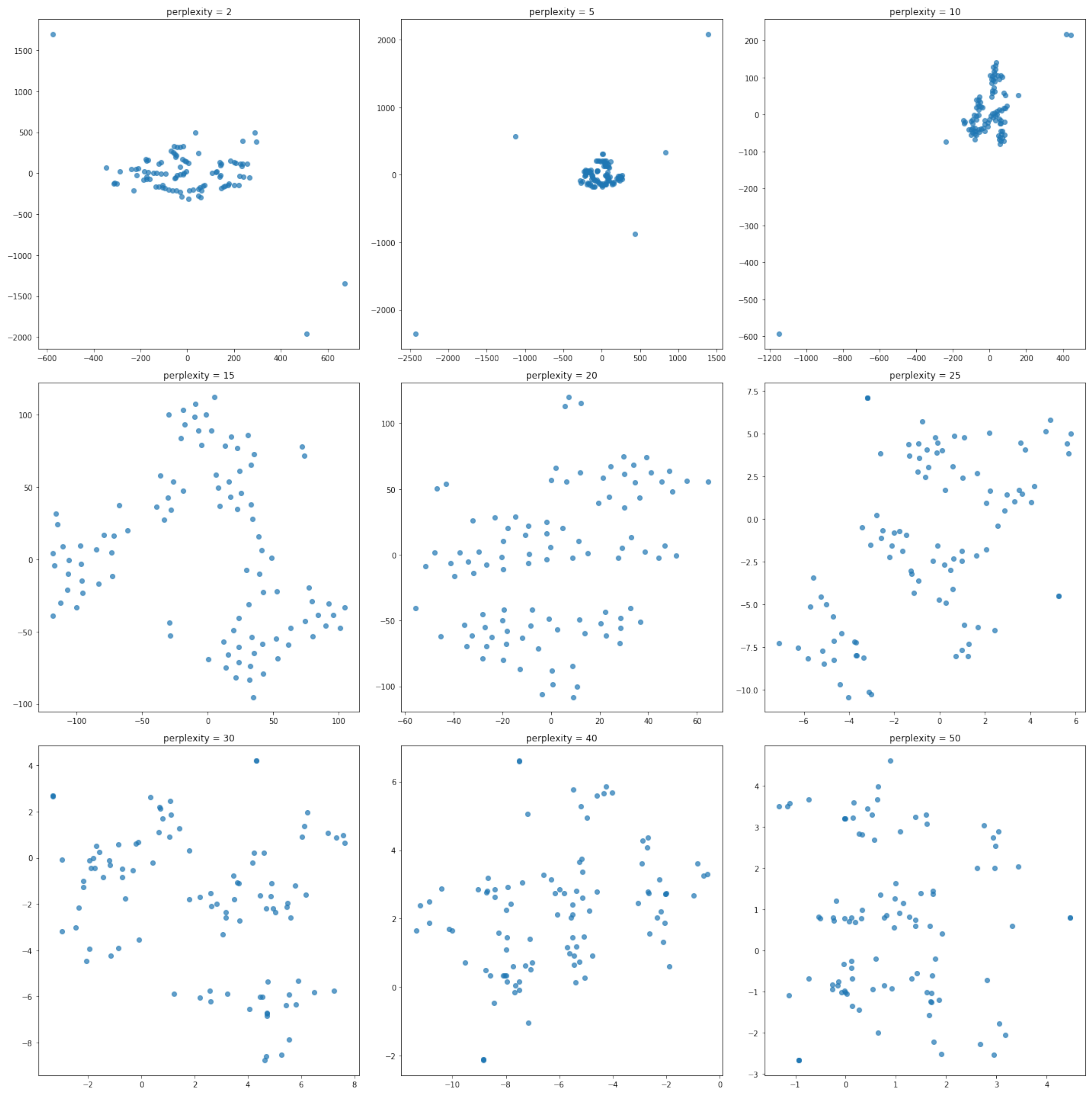

Figure 23: t-SNE on raw data with multiple perplexity parameters

Likewise, the t-sne for the power transformed data is seen in the following Figure 24. No conclusive and consistent clusters are observed as in the case of the t-SNE for raw data.

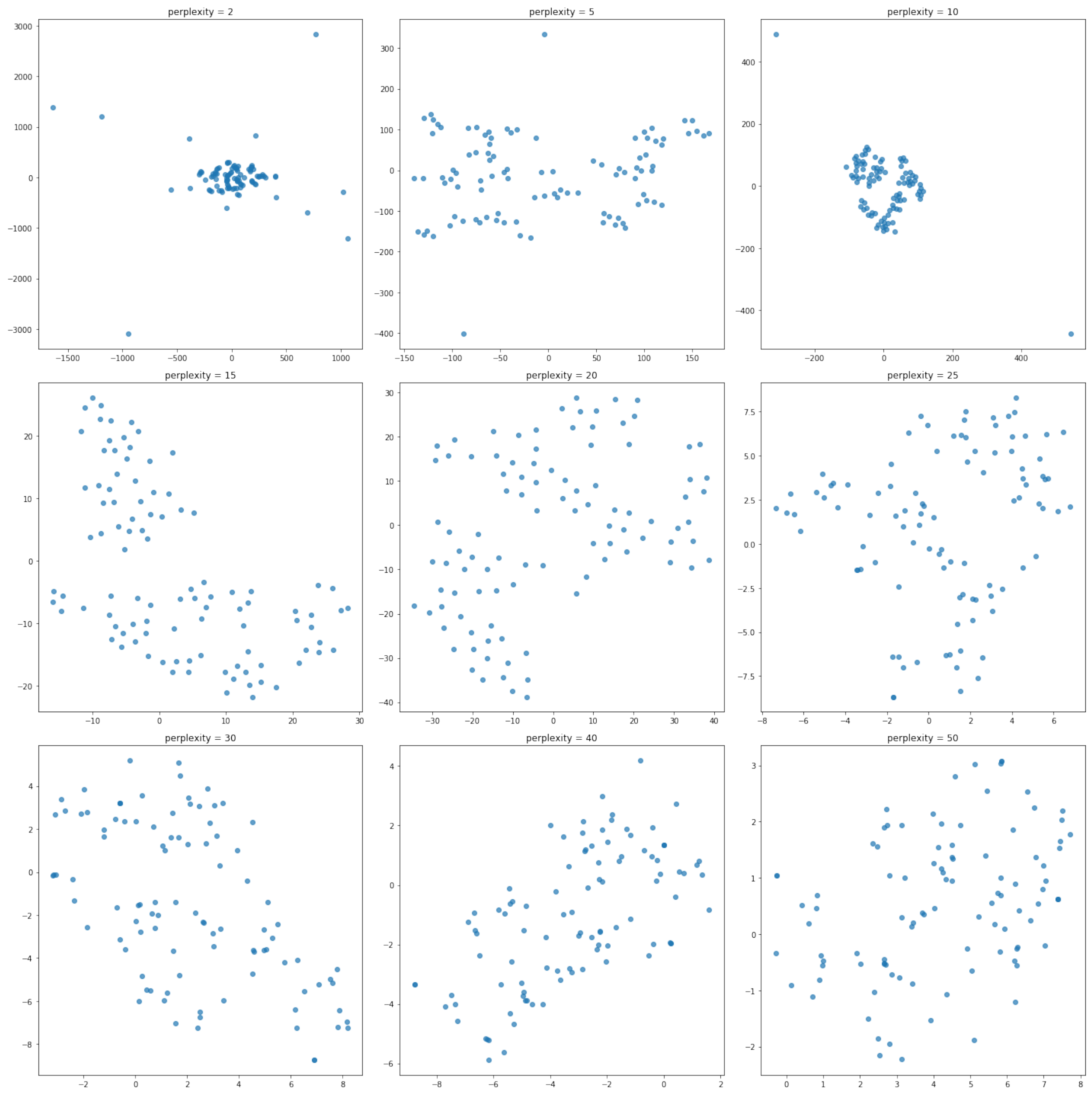

Figure 24: t-SNE on transformed data with multiple perplexity parameters

Finally, I tried the DBSCAN algorithm. The clusters in this case were not very conclusive either with no indication of outliers. In the interest of space the plots for DBSCAN have been left out.

4 Surrogate Functions: Predicting accuracy from hyperparameter values

4.1 Defining a Surrogate Function

For the final part of the project, we will delve into surrogate functions. As defined in [1], surrogate functions, theoretically are functions that output performance (in this case accuracy) for a set of hyperparameter values for a given task. Mathematically,

$$f_t(\theta_0, \theta_1 ... \theta_n) = p_t$$

Where, f is the surrogate function corresponding to task t, θ_i is the i^{th} hyperparameter

and p is the performance value

4.2 Methodology

In practical terms, I used the performance data used in section 2 for building these surrogate functions. The surrogate functions are regression models that take the hyperparameter values as features and predict the accuracy value. Specifically, I used the RandomForestRegressor from scikit-learn and ran a RandomSearch to find best hyperparameters for each surrogate. As mentioned before, the data consists of 42 tasks, each with 2000 hyperparameter settings thus corresponding to a total of roughly 80,000 data points. I made an 80-20% train-test split and trained the Random Forest for each task. Thus we end up with 42 surrogate functions.

As a deliverable, the surrogate functions are pickled and stored at the Github repo . These files can be used in the future for further analysis along this direction. The format of each file is as follows:

”surr_task_(task_no).pickle”

Each pickle file corresponds to a task and contains a RandomForestRegressor object along with the R^2 value on the test-set.

5 Future Work and Concluding Remarks

- **KDE vs Uniform**
  - For the KDE vs Uniform comparison we could look at a a broader section of the hyperparameters and not just the most important ones. However, we are more concerned with finding the best values for the most important hyperparameters. Perhaps we are missing the interaction of parameters and might have to fit multi-variate KDEs over multiple hyperparameters.
  - Also, the analysis should be repeated with more datasets. There was a clear demarcation between datasets that responded to faster convergence with KDEs rather than uniform and those that were slower. In the future when the data for more than 42 tasks is obtained, the characteristics of the faster convergence datasets could be filtered out. This could lead to more effective priors.
- **Dataset Clustering**
  - We should investigate further into the three clusters found and check if these have any relation to algorithm performance. For this we will need the performance data for all the 100 tasks. That way we can link the clusters with performance and not just esoteric meta features.
  - Likewise, for further clustering with increased manual intervention could be tried. For example, building further on the completely unsupervised clustering explored here, we could look at tailored classes that we care about, like class imbalanced could be an important parameter we divide the datasets into, or percentage of categorical features (which is already a cluster we obtained from the t-SNE analysis).

In conclusion, I would like to thank Dr. Jan Nicolaas van Rijn and Prof. Andreas Mueller for their guidance and for the opportunity to work on this fantastic Machine Learning research project. I take away a lot of invaluable skills that include working with the Habanero cluster, the Slurm job scheduling environment, OpenML and most importantly a chance to get my hands dirty with an end to end ML research project. I have thoroughly enjoyed my association with Prof. Mueller and his lab that began with the wonderful Applied ML course in SPring 2018, a Capstone project and this project in the Fall. I would like to express utmost gratitude to Dr. van Rijn for his patience and guidance as I initially grappled with the unfamiliar Habanero cluster environment. It was a pleasure working on this project and I hope that my contribution adds enough value to further this research endeavor. Thank you!

6 References

[1] Jan N. van Rijn, Frank Hutter, ”Hyperparameter Importance Across Datasets”
[2] Open Machine Learning
[3] RandomSearchCV Scikit-learn
[4] Project Github repository
[5] Scipy Gaussian KDE module