Group 5
Han Loo - 2288527L
Quentin Deligny - 2262655D

Machine Learning H Coursework

Report

Task 1: Regression

Introduction

The aim of this task is to predict the performance of 41 unseen CPUs. To achieve this, two regression models are trained on the provided data. The provided data contains 168 CPUs and 6 features each. These are cache memory size, the minimum and maximum number of I/O channels, machine cycle time, and minimum and maximum main memory. We have tested many different kinds of models for this task such as linear, random forest, ridge, lasso, logistic and SVR regression. In the end, we have chosen to submit predictions from random forest regressor and k-neighbours regressor as they showed the most promising results during our initial testing.

Experiment

The first model we have chosen to submit is our random forest regressor. The random forest regressor predicts the unseen CPU performance by fitting a number of decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. For the second model, we have selected k-neighbours regressor. The k-neighbours regressor predicts the unseen CPU performance by local interpolation of the targets associated of the nearest neighbours in the training set. After having tested the performance of various other models (with preprocessing and hyperparameter tuning) through Kaggle and built-in SKLearn metrics, we concluded that random forest regressor and the k-neighbours regressor seemed to have the most potential for further testing.

The next step was to split the dataset for testing and evaluating the model performance. We used train_test_split to split the given X_train and y_train values with a test size of 0.3. With this, we were able to evaluate the model's performance using various performance metrics such as mean squared error and mean absolute error. We then preprocess the data by scaling, using StandardScaler from sklearn. With features that highly vary in range, we believed that normalisation should be performed.

We noticed that a higher value for the number of trees used by the random forest regressor (n_estimators) tended to achieve better Kaggle public score and r2 score. However, given such high parameter values, we assumed that our model was overfitting the data. Therefore, we used grid search and cross-validation to hypertune the parameters of both models. This is very useful in both cases, as this tool matches all possible combinations of parameters within the defined parameter grid in order to determine which combination has the best performance score. For example, this enables our random forest regressor to choose between different ranges of trees and its maximum depth. In addition to that, the parameter grid (see fig 1a) used also includes range of values for the minimum number of samples required to be at a leaf node, the minimum number of samples required to split an internal node,whether to reuse the solution of the previous call (warm_start) and use out-of-bag samples to estimate the R2 on unseen data (oob_score).

We have defined another parameter grid (see figure 1b) to tune our k-neighbours regressor. Some of the compatible combinations of parameters we have tested include the

different algorithms used to compute the nearest neighbours, the leaf size passed to BallTree or KDTree, the number of neighbours to use by default for k neighbours queries, power parameter for the Minkowski metric and the weight function used in prediction. Finally, we have also implemented cross-validation (using 10 folds) to find the most promising set of hyper-parameters in our grid search.

Fig 1a - Random Forest Parameter Grid for Grid Search

```
param_grid = [
   {'max_depth': [None, 3, 5, 7],
    'oob_score' : [True, False],
    'min_samples_leaf':[1, 2],
    'min_samples_split': [2, 3, 4],
    'n_estimators': [100, 500, 1000, 2500, 5000],
    'warm_start': [True, False]}
]
```

Fig 1b - K neighbours Parameter Grid for Grid Search

```
param_grid = [
    {'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size' : [10,15,20,25,30,40,50],
    'n_neighbors' : [2,3,4,5,6,7,8,9,10],
    'p' : [1,2],
    'weights' : ['uniform', 'distance']}
]
```

Evaluation

We conducted basic regression evaluation on the performance of our regression models. The random forest yields different results each time as the order of features considered is randomized at each node, and when two possible splits are then tied, the split to use will depend on which one was considered first. Therefore, we set the random state in the random forest regressor is set to 0 to ensure a consistent outcome. The performance metrics we chose were the R2 score, explained variance, root mean squared error and the kaggle public score. The R2 score is a regression score function which returns the mean accuracy on the given test data and labels. The explained variance measure of how far observed values differ from the average of predicted values.

The results from the experiment yields the following results shown in table 1a. Overall, the k neighbour regressor model seen improvements in the R2 score, explained variance and lower root mean squared error values after hyperparameter tuning and preprocessing data. However on the other hand, the parameter tuned forest with preprocessed data achieved lower performance scores in all 3 metrics. We believed that this was still the better model compared to the base random forest, as the base random forest was overfitting the data.

Table 1a - Evaluation of both regression models (rounded to 3 decimal places)

Regression model and data type **Metric**	Base random forest with normal data	Base K neighbours with normal data	Tuned random forest with scaled data	Tuned K neighbours with scaled data
R2 score	0.962	0.745	0.928	0.871
Explained Variance	0.898	0.745	0.867	0.876
Root Mean squared error	35.677	56.257	42.387	40.041
Kaggle public score			27.755	39.803

Possible improvements

For future improvements, we could use the voting regressor. We have tested that voting regressor using base random forest, gradient boosting and linear regression models achieved a score of 33. We believe that adding more models and hyperparameter tuning the models in the voting regressor could drastically improve the score. We did not include voting regressor in our final submission as we had a limit of 2 classification models.

An attempt was made to remove rows which had values of 0, we concluded that the results from this outcome was inconsistent as it improved the public score for K neighbours regressor but the random forest regressor got worse. We found that removing rows which have "CACH" values of 0 resulted in a significant loss in the training data, it went from 168 to 112 rows. On the other hand, removing the 3 rows which have 0 "CHMIN" or "CHMAX" values would be insignificant. We also tried averaging the minimum and maximum values and calculating the mean difference to overall average. However, these outcomes did not improve the public score for linear regression. Imputation methods would be a promising alternative, but was not implemented due to time constraints.

Conclusion

To conclude, both models have performed reasonably well if we consider the Kaggle Score performance and further evaluation we have conducted. We have chosen the random forest regressor and the k-neighbours model as they are both consistent when given the same parameter settings. This allows for reproducible results should the experiments need to be repeated. We hesitated between SVR and k-neighbours model as they had very close performance score. We opted for k-neighbours as it proved to be slightly more accurate in the final results. We have also attempted to manipulate the input features by meaningfully combining them (average of minimum and maximum value, distance to overall average, the difference between the two…) but these combinations did not improve our final results, hence we decided to not include them. We did not select the random forest regressor model with 100 000 trees as our final submission as we believed it was largely due to overfit. Although our tuned random forest regressor had a worse Kaggle performance score, we decided to select it as it was much more likely to perform better on the hidden dataset.

Task 2: Classification

Introduction

The aim of this task is to classify hundreds of tissue microarray (TMA) samples based on the gathered data. The goal of the implemented classifier models is to distinguish cancerous (or epithelial) and healthy (or stromal) regions within the tissue samples. In this case, the training dataset consists of 200 different data points and 112 features. Each data point can belong to one of the two previously mentioned classes. It is particularly important to build an accurate classifier model while also limiting the false negative predictions for this kind of tasks: the worst outcome is predicting that a patient is healthy while he is truly sick (even if the overall system has high accuracy – it may be a rare disease). We have tested many different kinds of models for this task such as linear classification, random forest or ridge classifier. In the end, we have chosen to submit predictions from the logistic regression classifier and support vector classifier (SVC) models as they showed the most promising results.

Experiment

The first model we have chosen to submit is our SVC model (a different implementation of the same algorithm used by SVM). This is a supervised learning model; this suits our classification task as we have some information regarding the labels for the two types of regions in tissue microarrays. Furthermore, this model is suited for this task as it represents the data as points in space: it maps new data into the same space in order to determine in which category the new data point falls.
For the second model, we have selected logistic regression classification. After having tested the performance of various other models (with preprocessing and hyperparameter tuning) through Kaggle and built-in SKLearn metrics, we concluded that SVC and the logistic regression classifier seemed to have the most potential for further testing. Logistic regression suits our model quite well as our data has a binary classification. This specific classifier builds a prediction hypothesis based on logistic functions; this takes into account all provided features within our dataset, and predicts one of two classes based on the results obtained from the hypothesis equation (given the specific feature values). To do so, it compares the obtained results with a specific decision boundary. Finally, the algorithm utilises a cost function to estimate the error rate between predicted values and expected values. The aim is often to minimise this function (using various methods such as gradient descent).

The next step was to select the most relevant features for both models. With precisely 112 features in the dataset, we believed that selecting the most important features in the dataset might improve the overall classifier model in many different ways:

1. The main goal of feature selection is to remove the noise in our data. Removing it is likely to increase the performance of our overall model (which was proven by our experiment as it nearly doubled the performance score).

2. Reducing the overall number of features also enables quicker training and testing of our model. Although this did not make a major difference considering that our dataset was relatively small, we can imagine that this would be very important if we were to use the same models and very large data sets.
3. There is a chance that feature selection may also reduce overfitting of our model. We are only able to test our model on 30% of the testing data set, therefore there is a potential chance for overfitting. This means that our model could over interpret the data we have given it (and perform well on that data) but fail to adapt to unseen data (ie. the 70% of the test data). This is a common problem in machine learning, and selecting the most important features may counter it to some extent (adding "useless" features to our data increases chances of overfitting).
4. Lastly, this may also reduce the overall complexity of our model, making it easier to interpret for future potential users (or programmers). This is also an important aspect to ensure that our solution is maintainable and scalable.

The most challenging part for this section of the experiment was to select the correct number of most important features, as well as the features themselves (there are different feature selection methods available). We also realised that, based on the feature selection method, the optimal number of selected features may vary; However we did not have the time to test all the potential possibilities for every possible combination. We have approximately covered different types of combinations in terms of selection method and number of selected features. We have used the following feature selection methods for this task:

- **Extra Tree Classifier** is an ensemble learning technique which combines the results of multiple decision trees. The classifier then outputs the classification result of each tested tree and collects the best results. Each tree is constructed from the original training sample: it is given a random sample of n features from the set at every node in order to create the best feature combination following some mathematical criteria (usually the Gini Index). Lastly, this method uses the selected Index to rank the importance of all features. The user is able to select the most important k features from this ranking.
- Applying **Variance Thresholding** to remove a number of features with low variance is another form of feature selection. This method assumes that features with values that have low variation do not bring much information to the classification model. This only applies when there is a high number of features and items, otherwise some important features may be accidentally removed (important features may have a low variation when there is a small number of items). However, our data set is large enough to avoid such issues. SKLearn's implementation of this feature selection method allows the user to set a threshold. All features with a variance that is lower than the defined threshold will be removed from the dataset. The main problem we encountered with this method was that setting a set threshold did not guarantee that the number of selected features would be the same for the testing and training set. For certain thresholds, a different number of features was selected for each set, which caused problems when trying to establish predictions. We had to select specific

thresholds (with associated number of selected features) to ensure that both the training and the testing set had the same number of selected features.

- **PCA** is another method for dimension reduction that we used out of curiosity. It performs feature reduction rather than feature selection, and was very unlikely to outperform other standard feature selection methods. It operates by combining all the features (technically not leaving out any noise or information from the data set) to produce an equivalent set of features (also referred to as components) in lower dimension.

The results of each preprocessing method are discussed in the final section of this report. This section will also detail why we have chosen a specific selection method (and number of selected features) over other alternatives.

We have also used grid search and cross-validation to hypertune the parameters of both models. This is very useful in both cases, as this tool matches all possible combinations of parameters within the defined parameter grid in order to determine which combination has the best performance score. For example, this enables our SVC classifier to choose between different kernels; this implies that it could also perform well on non-linear data by mapping it to higher dimensional spaces. Although it would appear that some features in the provided data for this task may be linear from figure 1 (somewhat resembles a logarithmic function), we cannot exclude the possibility that other features from the same data set are non-linear, in which case performing the kernel trick would improve performance. Therefore, we have also included the RBF kernel (alongside different potential gamma values) in our parameter grid for SVC (figure 2a). We have also selected various strength values (C) for the regularisation parameter of our classifier (see results for outcome of grid search for both SVC and logistic classification).

We have defined a similar parameter grid (see figure 2b) to tune our logistic regression classifier, albeit slightly more complex. Some of the compatible combinations of parameters we have tested include L1 and L2 for the penalty (uses various regularisation methods to shrink the coefficients of less contributive variables when there is a large number of features in cases such as ours), the strength of the regularisation (inversely proportional to the C-value), liblinear and saga solvers (the latter may suit our problem more given the large number of features) as well as the maximum number of iterations taken by the solver to converge (we noticed that it sometimes failed to converge in some cases with the liblinear solver, but increasing the max_iter value had a sizeable impact on the grid search's run-time and would not systematically solve the convergence error). Finally, we have also implemented pipelines as part of our grid search process to allow for cross-validation (using 10 folds) to find the most promising set of hyper-parameters.

Fig 1 - Scatter plot of features `GLCM.Ang..2nd.moment..quick.8.11..Layer.1..all.dir..` and `Ratio.Layer.3`.

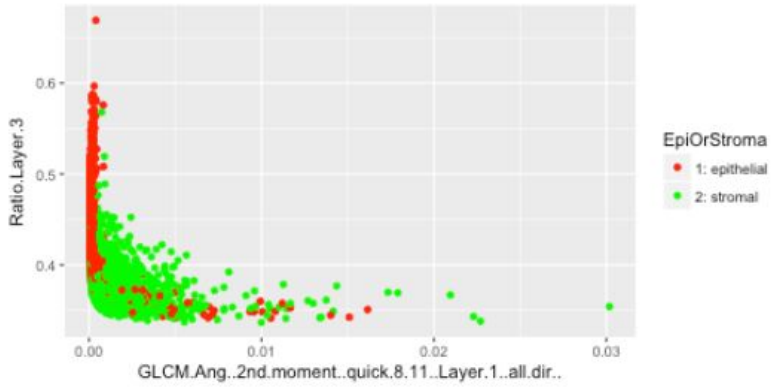


Fig 2a - SVC Parameter Grid for Grid Search

```
param_grid = [
    {'kernel' : ['linear','rbf'],
     'gamma' :[1,0.1,0.001,0.0001],
     'C' : [0.0001, 0.001,0.01, 0.1, 1, 10, 100, 1000]},
]
```

Fig 2b - Logistic Regression Classifier Parameter Grid for Grid Search

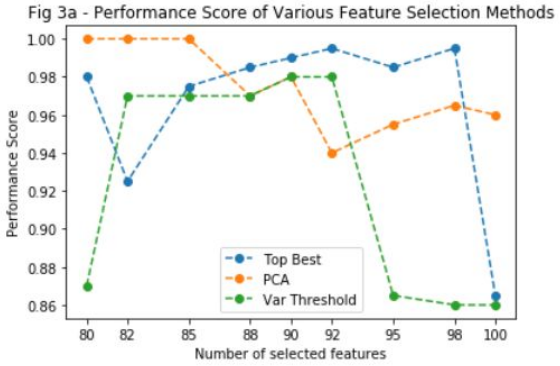
```
param_grid = [
    {'classifier' : [LogisticRegression()],
     'classifier__penalty' : ['l1', 'l2'],
    'classifier__C' : np.logspace(-4, 4, 20),
    'classifier__solver' : ['liblinear', 'saga'],
    'classifier__max_iter' : [2,5,10,15,25,50,100,150,200]},
]
```

Evaluation Methods

For the sake of time, we did not conduct extensive evaluation of all our classifiers and the different parameter combinations/feature selection methods we could have used. Nevertheless, we still had to perform basic evaluation of all our models and some of their combinations in order to determine which had the most potential (to perform well on the hidden testing data) and which should be chosen for the final submission. A metric which proved very useful for a first indication of a model's performance was SKLearn's Score metric. This model evaluation tool provides a single value between 0 and 1, where a higher value is generally better. The advantage of this metric is that we did not need to split our training data in order to compute it (which we only do for models showing good potential to being with). Many other metrics are useful, but require the training data to be split in order to be evaluated: this is because these metrics (such as recall, F1 score, precision, average precision or accuracy) require y_test values (which we did not have, unless we split y_train). Furthermore, we found the basic score metric to be somewhat proportional to the score we would receive from testing our submission on the public data set on Kaggle. All these factors led us to use this basic score metric as our first screening tool to assess the most promising models.

The score of the basic model what not very useful (especially for some cases such as SVC where it was always 1.0). We decided to evaluate this score for both feature selection

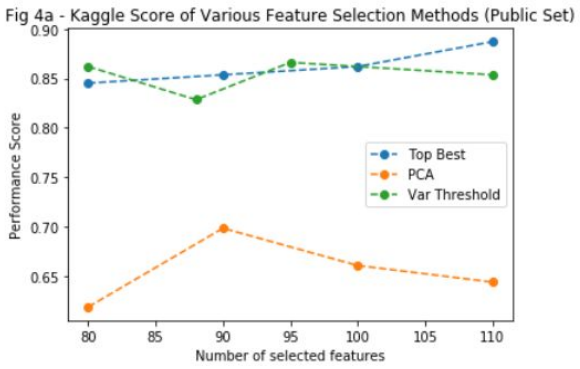
methods we had implemented, as well as PCA. Figure 3a shows the resulting score for selecting different number of features (ranging from 80 to 100 which we deemed to be the most relevant range after having conducted some further testing) with different selection methods for SVC. In each case, the data is preprocessed using the standard scaling method and the model is tuned using the previously mentioned grid to get the best possible results given the feature selection. This preprocessing method (scaling) is useful to normalise our data such that the distribution of each column feature has a mean value of 0 and a standard deviation of 1. The scaled data is easier to manipulate by the model, and has greatly decreased the runtime of grid search for some models. Figure 3b depicts the same measurements applied to the logistic regression classifier (LRC) model.

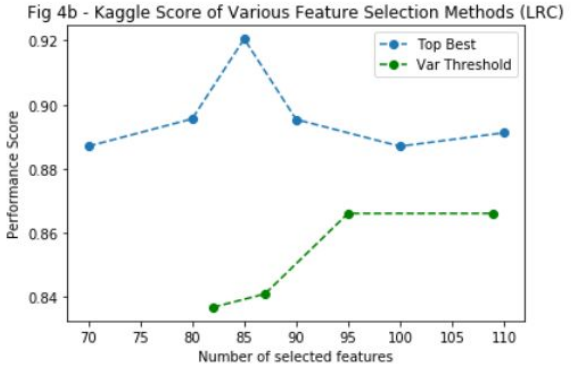


The results from figure 3a were slightly surprising. The first thing to note is that PCA has a score of 1.0 when reducing the number of features to 80, 82 or 85. This is an error from the basic function (although we have not identified the exact cause) which was also encountered when we attempted to evaluate the basic SVC model. We have therefore disregarded these values for PCA. This graph shows a clear drop for both feature selection methods we have implemented when the number of selected features is higher than 92. We are aware that this may not be directly proportional to the score attributed by Kaggle (both private and public), On the other hand, this is the first indication that the optimal number of selected features may lie somewhere between 82 and 92. Figure 3b is not very informative: we understand that the basic score stays relatively constant for PCA and the Variance Threshold methods (regardless of feature count), while it seems to vary unexpectedly for the method which ranks features by importance (ETC). As previously highlighted, this score is a first pointer to what the Kaggle prediction score might be (especially if it is very low or very high), but minor variations are not necessarily translated to the Kaggle scores.

In order to confirm this idea, we decided to submit 4 different prediction files to Kaggle. We have sampled these prediction files from the same models that were used and evaluated to produce figure 3a. Regarding PCA and and the ETC method, we have respectively reduced to and selected 80, 90, 100 and 110 features. We had to choose different numbers of features for the threshold variation method as the previously mentioned numbers did not match for the training and testing data (although the threshold value was the same), preventing us from being able to make any predictions. Thus, we have selected feature

numbers that matched in both data sets: 80, 88, 95 and 110. Figure 4a shows the results (SVC only) we have obtained from these models after having submitted their predictions to Kaggle. Similarly, figure 4b shows our obtained results for the logistic classifier model. We have decided to exclude PCA from the visualisation as initial tests indicated it would perform much worse than the other selection methods (much like in figure 4a). The second reason for not including PCA is that we did not have enough submissions remaining to submit enough tests.





We noticed that we would sometimes get different Kaggle scores for both figures regarding the ETC feature selection method (although the variations were minor). We believe this can be explained by the specific way in which this selection tool operates: the ETC method ranks the features according to some index of importance and does not always produce the same ranking given the same input (although the general order of features is quite similar from one run to the next). This different ranking of features may explain why we did not get the same scores on Kaggle.

Table 2a - Kaggle scores for SVC (as illustrated by Figure 4a)

Method used X Number of features	ETC	PCA	Variation TH
80	0.84518	0.61924	0.86192
88			0.82845
90	0.85355	0.69874	
95			0.8661
100	0.86192	0.66108	
110	0.88702	0.64435	0.85355

This figure proved very interesting, and brought additional information to complement the observations we had made previously. Firstly, we noticed that PCA has generally been much

less accurate than the other two methods regardless of the number of features, despite having good scores in figure 3a. We can therefore assume that PCA is likely to have a worst performance for the secret dataset on Kaggle and rule it out from our list of final model options. The next step took more reflection, as both feature selection methods performed relatively well for various amounts of selected features. The "Top Best" method which uses ETC to select features with the highest importance index had the best overall performance (0.88702 for 110 selected features). However, selecting 110 features from 112 did not seem like a major improvement and we believed that the model would be likely to overfit to the public data (since one of the goals of feature selection is to reduce the risks of overfitting). Furthermore, the Kaggle performance score was positively correlated with the number of features selected by the ETC method (meaning that selecting all 112 features could have potentially yielded an even higher score!) as shown in table 2a. This defeated the purpose of feature selection as it did not give us an ideal number of features to select. This is why we have chosen to submit our final model relying on the variation threshold method. This method shows an improvement from selecting a smaller number of features (and is therefore less likely to overfit). This also correlates with some testing we have done for other models, which seems to indicate that the ideal number of features to select lies in the range 80 to 95. We have chosen to submit our final SVC model with the threshold value set at 0.83 (85 features) as this is a good middle ground between our high performance indicators (80 to 90 features).

From figure 4b, we can see a much clearer difference between the ETC method and the variance threshold selection method. The ideal number of features for this method seemed to lie somewhere between 80 and 90 (initially we had not submitted a version with 85 features); We decided to submit our final model having selected 85 features, which proved to be our best performance on Kaggle as well. Lastly, the fact that the ETC Kaggle performance score is not directly proportional to the number of selected features (like in figure 4a) is reassuring. It implies that we have maximised our score while still being able to remove a significant number of features. Removing 27 features from the initial dataset also reduces the risks that our model may overfit. We did not provide a table for figure 4b as it was less relevant considering the amount of selected features by both methods was not the same. This is due to the variation threshold method not always selecting the same amount of features for the test and the train set.

Comparison of Final Results

In order to evaluate the following metrics, we need to know the true labels of the data for which we make predictions. Since we did not have access to these labels for the given task, the easiest way to approximate these metrics is to split our current training data (for which we know the true labels) into a training and testing set. We have chosen a commonly used randomised 80/20 train/test split to evaluate these metrics for both models. The first metric we will compare for our two submissions is accuracy. This metric measures the amount of correct predictions made by the false (both true and false) from all the predictions

made. Our two models are both highly accurate, with LRC performing slightly better having a score of 0.9. However, it is hard to evaluate the overall performance of a model using this metric only. As mentioned, this is not a good indication given our problem: a model can achieve a very high accuracy but still predict a large number of false negatives (predicting that a sick patient is healthy). This is even more susceptible to occur when there is a large class imbalance (as the model can always predict healthy and still achieve very high accuracy), and is commonly referred to as the accuracy paradox. Therefore, we will also consider other metrics such as precision: this measures the number of positive predictions divided by the total number of positive class values predicted. Once again, our two models have very similar (and relatively good) precision scores which can be interpreted as the exactness of the classifiers (in predicting that a patient has cancerous tissues). We also take into account the recall metric (true positives divided by the sum of true positives and false negatives), where LRC and SVC respectively score 0.993 and 0.867. This underlines the fact that the LRC model is less likely to predict a false negative, which is the worst outcome given the task of identifying cancerous regions within tissue samples. A model who would predict that all the tissues are cancerous would achieve a perfect recall (although it would be useless). Lastly, the last metric we measure is the F1-score which is an indication of the trade-off between a good precision and a good recall. This may be useful when it is equally important for a model to have a good precision and recall (although recall is more important for our task). Nevertheless, our LRC model achieves a better F1-score than our SVC model, which is coherent with the fact that it also achieved the best recall and precision scores. Although SVC also performs quite well overall, this table shows that our LRC model is better suited to the given task. This is the model we would submit if we were required to choose between our two final models.

Table 3

Model Used x Metrics	LRC*	SVC**
Accuracy	0.9	0.875
F1 Score	~0.875	~0.839
Precision	~0.824	~0.813
Recall	~0.933	~0.867

* Used ETC Selection method to pick 85 most important features. Data was preprocessed using the standard scaler. Hyperparameter tuned and cross-validated (10 folds) using gridsearch.

** Used Variance Thresholding selection method with a threshold of 0.87 (or 86 features). Data was preprocessed using the standard scaler. Hyperparameter tuned and cross-validated (10 folds) using gridsearch.

Conclusion

To conclude, both our Logistic Regression classifier and our Support Vector Classifier models have performed well on both our personal evaluations and the public Kaggle Scores. We have achieved our best result (for both Kaggle score and personal evaluation methods) with the LRC model after having selected the 85 most important features using the Extra Trees Classifier Technique (although we noticed that using this technique would not always produce the same predictions given the same inputs). Our second classifier is more consistent, as the predictions are always the same given the same parameterisation of the model and selection techniques. This can be explained by the fact that the variation thresholding technique produces consistent results given the same input features. The performance of our SVC model was slightly worse than our first model for both Kaggle Scores and personal evaluations. Furthermore, we have not selected our best performing Kaggle score for SVC (ETC method selecting 110 most important features) as it more likely to overfit and not perform as well on the hidden dataset. It seemed that removing 2 features from 112 defeated the purpose of feature selection, which is why we have opted for the other selection method.