**Machine Learning Engineer Nanodegree**

**Capstone Project**

James Gan  
May 29th, 2018

**I. Definition**

**Project Overview**

In my job we have propensity models for customer behaviour and these are all based on using a certain machine learning technique (XGBoost). These models predict how likely it is a customer will click on certain products on the company website.

The XGBoost model was chosen as it consistently performed the highest when using a licensed data science tool (DataRobot) which compared dozens of techniques on AUC (a metric for measuring binary classification performance).

I am curious as to whether a stacking classifier method would outperform XGBoost. A stacking classifier is a type of machine learning method that uses many techniques e.g. XGBoost, random forest etc. together to produce a prediction. The main motivation for this project is a comparison of methods which will be explored in the following sections and not so much about the dataset domain. The hope is that the learnings from this project will be useful in my job as I will be able to report back on whether XGBoost can be bettered using a stacking classifier.

Some useful initial links for stacking are provided below:

* <http://blog.kaggle.com/2017/06/15/stacking-made-easy-an-introduction-to-stacknet-by-competitions-grandmaster-marios-michailidis-kazanova/>
* https://rasbt.github.io/mlxtend/user\_guide/classifier/StackingClassifier/

The project is to be based on census data from the US Census Bureau found on the UCI Machine Learning Repository website: <https://archive.ics.uci.edu/ml/datasets/Census-Income+%28KDD%29>

Since this is similar to the datasets I encounter at work. This is typical census data you would expect to be collected and importantly has a column stating whether each person earns over $50k a year or not (this is the feature I will be predicting).

**Problem Statement**

The problem is to investigate whether a stacking classifier produces a higher AUC than an XGBoost model on the census dataset when predicting whether a person earns over $50k in their annual income. I am interested to see how these methods compare since XGBoost has commonly been a very high performing algorithm e.g. in Kaggle competitions.

The steps involved to solve the problem are: cleaning the dataset so it can be used to model off of, building the baseline XGBoost model, building the stacking model by experimenting with combinations of different techniques, comparing the AUC of these two models and concluding which technique is better.

The solution to the problem will be that either a stacking classifier method comprising certain techniques does produce a higher AUC on the prediction of whether somebody earns over $50k. Conversely it could be that XGBoost has a higher AUC than the stacking classifier method.

**Metrics**

AUC will be used as the main metric for measuring success of the models in classifying the data. This is because in relation to benefitting the analysis performed in my work we care most about the ordering/ranking of customers in terms of how likely they are to be classified as one of the two outcomes. Therefore, we are less concerned with e.g. minimizing inaccuracy of the probability that someone will be classified as one of the two outcomes.

To this end AUC is a good metric as it works by first plotting the true positive rate against the false positive rate of the models classifications then calculating the area under the curve generated. Thus the closer this area under the curve is to 1 then the better the model is to correctly classifying all the points in the dataset.

**II. Analysis**

**Data Exploration**

The following points were identified as interesting or needing to be handled in data pre-processing:

* The feature ‘instance weight’ should be removed. As stated in the documentation on the UCI website it was added for stratification to enable more accurate analysis but should not be used for classifier modelling.
* There are no NaN values but there are ‘Not in universe’ and ‘?’ values in a lot of columns which should be considered for imputation at the data pre-processing stage.
* The ‘Target’ feature should be changed to a binary outcome so it is easier to pass to the classifier models, i.e. 1 for >$50k and 0 for <$50k.
* ~94% of the population had <$50k for their ‘Target’ values.
* The industry codes and occupations are basically numerical representations of the major industry and occupation codes but with less detail and can be removed if a non-numerical value can be used in the models instead.
* Education can be banded into more general groupings e.g. below the 9th grade.
* The ‘year’ feature is not needed/should not be considered for this classification problem and can be removed.
* The sunbelt region is the horizontal section across south America.
* There are 3229 duplicate rows of data that could be removed.
* There are a very high number of 0s for ‘capital gains/losses’ and ‘dividends from stocks’. Greater than ~ 90% of rows have 0 values for these features.
* ~94% of ‘wages per hour’ values are also 0.
* There is a high proportion (~50%) of unknowns for major occupation and industry codes and so adds a layer of difficulty to predicting the ‘target’ feature as we don’t know the types of jobs a lot of people have.
* Unfortunately the definition of the ‘veteran benefits’ code was not found and so this column and the questionnaire for veteran’s column should be considered for dropping as it is unclear what they mean.
* Scaling for numerical variables will need to be considered depending on the modelling techniques being applied.
* One-hot encoding (OHE) will need to be considered for categorical variables depending on the modelling techniques to be used.
* <https://archive.ics.uci.edu/ml/machine-learning-databases/census-income-mld/census-income.names> has the distinct values of the features.

These were mostly noticed through the report generated in pandas\_profiling or basic inspection of the Excel file.

A small sample of data is provided below from the first few columns. For a more comprehensive sample see the bottom of the pandas\_profiling.ProfileReport which is executed at the beginning of the Data Exploration section in the Python notebook:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **age** | **class of worker** | **industry code** | **occupation code** | **education** | **wage per hour** | **enrolled in edu inst last wk** | **marital status** | **major industry code** | **…** |
| 73 | Not in universe | 0 | 0 | High school graduate | 0 | Not in universe | Widowed | Not in universe or children | … |
| 58 | Self-employed-not incorporated | 4 | 34 | Some college but no degree | 0 | Not in universe | Divorced | Construction | … |
| 18 | Not in universe | 0 | 0 | 10th grade | 0 | High school | Never married | Not in universe or children | … |

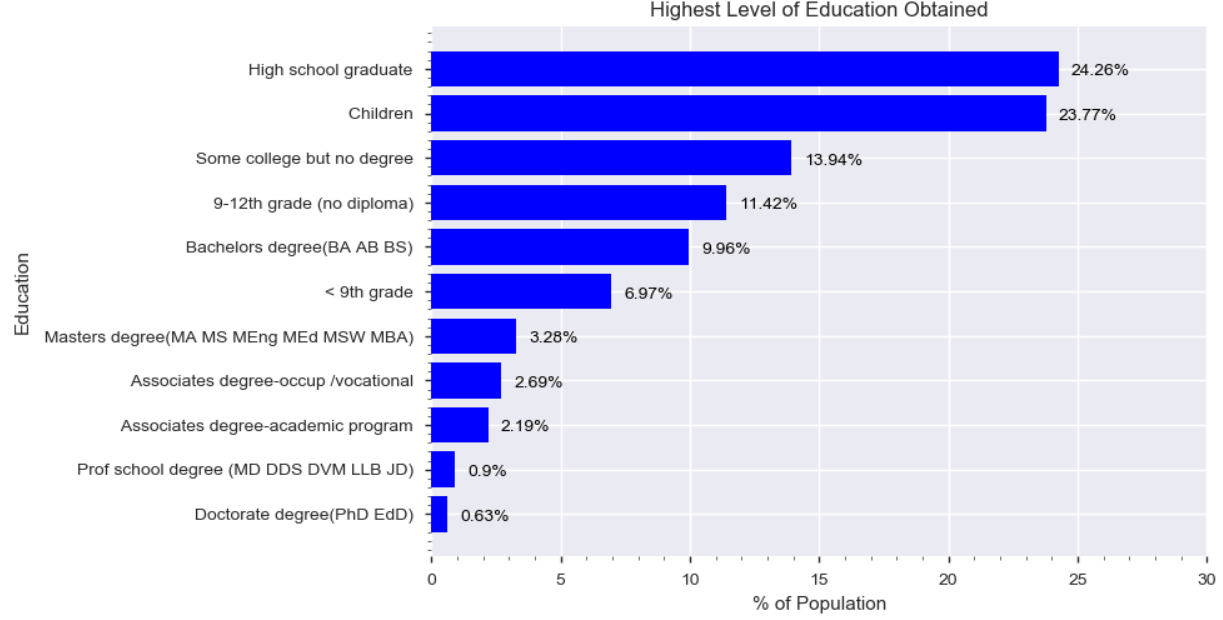
Below is a table describing some basic statistics of the ‘Age’ feature. As we can see from the table the average age of the population is 34 which is quite young by today’s standards but this census was done in 1994/5. This further emphasised by the third quartile being at age 50. Therefore a lot of people are of at least working age in the population.

|  |  |
| --- | --- |
| **Statistic** | **Age** |
| **count** | 199523 |
| **mean** | 34.4942 |
| **standard deviation** | 22.3109 |
| **min** | 0 |
| **25%** | 15 |
| **50%** | 33 |
| **75%** | 50 |
| **max** | 90 |

**Exploratory Visualization**

From the Data Exploration section, we have seen that there are isn’t a high proportion of data in regard to sector/job types of the survey participants. So the next best feature I have decided to analyse is level of education obtained. From looking through the Excel for the education column the ‘Children’ value is for children still in education with other values being the highest form of education received. It is also good to note that for the U.S., high school is grades 9-12.

I collapsed some of the education values into larger groupings to provide a bit more generalisation. For instance banding all those with <9th grade level education together and 9-12th grade with no diploma together.

As we can see from the chart above most people (~24%) are high school graduates and so we would not expect a lot of them to have an income >$50k. The next highest proportion of the population interestingly are children and so obviously should all have an income <$50k. These two groups make up nearly 50% of the population and so it is no surprise that there was such a high bias in the ‘Target’ feature as specified in the previous section towards <$50k earners. In fact <15% of the population has a Bachelors, Masters or Doctorate degree.

Whilst level of education is not the sole influential factor for earning power I think the two will be heavily correlated and so it is likely this will be an important feature in the models.

**Algorithms and Techniques**

The techniques to be used in this project, the key parameter choices, why this technique was chosen and whether the data will be handled in a special way are discussed below.

XGBoost was used as the benchmark model and will also be part of the stacking ensemble in the comparison model. For each technique a basic randomsearch will be run over the parameters to try and optimise the model to some extent before being stacked. This will not be an extensive optimisation due to time and resource constraints. Note that where possible n\_jobs/threading will be used to speed up the modelling.

XGBoost:

XGBoost is an ensemble machine learning technique that works by adding sequential, shallow decision trees to try and minimize the difference between the target feature value and the current predicted value. There is also a regularization term that is considered for each new tree added to help combat overfitting. Boosting is used when adding each new tree (or weak learner) by training on data points that were incorrectly classified in the previous iteration. This is further explained in the following links:

<https://www.youtube.com/watch?v=GM3CDQfQ4sw>

<http://xgboost.readthedocs.io/en/latest/model.html>

The reason to use XGBoost as the benchmark model has already been stated in previous sections. I am interested to see how including this in a stacking ensemble affects the AUC and so will include it in the stacked model to be compared. The key parameters that I want to optimise are the number of estimators i.e. number of trees and the maximum depth. I chose these as they are some of the most influential parameters to augment in terms of minimising overfitting whilst also optimising AUC. Positive scale weight will also be applied as the dataset is very unbalanced.

The data may need to be One-hot encoded to improve results although this technique should be able to handle the data either way. A brief experiment will be tried with both to see which produces the higher AUC.

Random Forest:

Random Forest is an ensemble technique that works similarly to XGBoost above. The key difference is that instead of using boosting Random Forests use bootstrap aggregating or bagging when training on data. A number of decision trees (user specified) are trained in parallel and their predictions are aggregated to form a more robust prediction. Bagging is outlined further in the video below:

<https://www.youtube.com/watch?v=2Mg8QD0F1dQ>

A Random Forest will be chosen as part of the stacking model as I am interested to see how it compares to XGBoost. As per

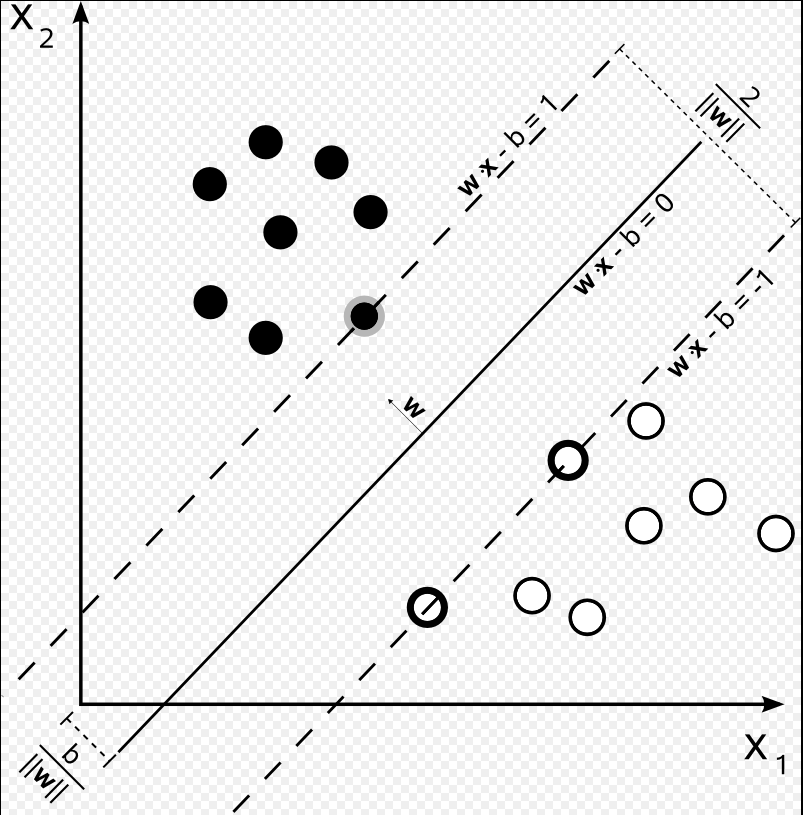
<https://stats.stackexchange.com/questions/173390/gradient-boosting-tree-vs-random-forest>

Random Forest may be better than XGB as it focusses on reducing variance as opposed to bias for boosting. Random Forests also build their trees in parallel as opposed to sequentially in XGB so may be quicker.

Similar to XGB the key parameters I wanted to augment were number of trees but also wanted to change the min sample leaf to try and avoid overfitting. Class weight will also be applied as the dataset is very unbalanced. The data will be handled in a similar way to XGB.

Support Vector Machines:

Support Vector Machines (SVM) work by forming a decision boundary (hyperplane) separating the different classifications of the data. Margins are then established either side of the boundary between the boundary line and the closest datapoint on that side of the boundary. This is depicted below taken from <https://en.wikipedia.org/wiki/Support_vector_machine>



The margin width is then maximised to find the best fit. In the case that the data/relationship is non-linear a kernel trick is applied to transform the data into a linear structure. See the below link for more information on SVMs:

<https://www.youtube.com/watch?v=eUfvyUEGMD8>

SVMs with RBF kernel was chosen to be included in the stacking ensemble as it can handle non-linear data and also has a reputation for being a very powerful/successful technique on sites such as Kaggle. This technique is also good for handling a dataset with many features. One concern is the time it takes to train and this shall be investigated.

The key parameters to augment will be the C and gamma values as these are the most influential in predicting correct results. The class weight again will also be adjusted because of the low number of positives. The size cache may also need to be adjusted to help reduce training time.

In addition to the data pre-processing required for Random Forest and XGB numerical data will also have to be scaled for SVM as it isn’t scale invariant.

Logistic Regression:

Logistic Regression (LR) works by minimizing the log-loss error function of the data points using gradient descent. The points are assigned probabilities of being correctly classified (dictated by distance from the boundary line). Incorrectly classified points are heavily penalized and correctly classified points have a near zero penalization. Further detail is given in the following link: <https://www.youtube.com/watch?v=KayqiYijlzc>

LR was chosen for two reasons to be in the stacking group. The first is that this modelling technique was used by the team at my work before we switched to XGB and so I would be interested to see the impact to the other models in the ensemble. Secondly it is again a powerful modelling technique for classification. Although LR typically doesn’t perform as well on non-linear datasets I would like to experiment with this method especially since there may be a strong linear relationship in the data and LR may therefore combine really well in an ensemble.

Similarly to SVM the key parameter to tune will be the C value as this will help to combat overfitting. Likewise class weights will also be adjusted.

No data preparation in addition to that done for XGB and Random Forest is needed.

<https://elitedatascience.com/machine-learning-algorithms>

Stacking Classifier:

A meta-classifier will be implemented using the mlxtend library. Various different meta-classifiers will be experimented with. For the most part their default parameters will be used apart from the target class balance.

Due to the different data treatments to be used for the different base classifiers the pipeline and column selector methods will be used to run the stacking classifier procedure.

**Benchmark**

The AUC score produced by an XGBoost model will be the benchmark as justified in discussions in previous sections.

The benchmark AUC score is: 0.854

This value was obtained as outlined by the XGBoost method in the ‘Algorithms and Techniques’ section.

**III. Methodology**

**Data Preprocessing**

The following steps were taken to pre-process the data before running the models over them:

* ‘adjusted gross income’, ‘federal income tax liability’, ‘total person earnings’, ‘total person income’ and ‘taxable income amount’ headings were removed from the Excel before loading into the dataframe.
* Removed the following features from the dataset: 'instance weight', 'year', 'veterans benefits', 'fill inc questionnaire for veteran's admin', 'industry code', 'occupation code', 'own business or self employed'
* Replaced all ‘Not in universe’ and ‘?’ values with NAN. ‘Not in universe or children’ and ‘Not in universe under 1 year old’ were left in the dataset as it was not entirely clear what these mean but they were distinguished from ‘Not in universe’.
* The ‘Target’ feature was changed to a binary outcome so it is easier to pass to the classifier models, i.e. 1 for >$50k and 0 for <$50k.
* Education was banded into more general groupings e.g. grouping everything below the 9th grade into one label.
* There were 3,229 duplicate rows of data but after the redundant columns were removed this went up to 49,387 rows. These were removed.
* The ‘<’ symbol was replaced by ‘less than’ in the dataset to avoid later problems with the encoding.
* Encoded all categorical variables to have a numerical value and tested XGB performance on this versus this data that was then subsequently one-hot encoded.
* For SVM modelling the remaining numerical data left after OHE was scaled since SVM is particularly sensitive to relative data scale.

<https://archive.ics.uci.edu/ml/machine-learning-databases/census-income-mld/census-income.names> has the distinct values of the features.

**Implementation**

To implement the base level classifiers for the stacking technique first an appropriate data split needed to be prepared. This was done by first splitting the pre-processed data into two in a ratio of 3:2. The larger portion of data (i.e. 60%) of the initial data was then used for training and hyper-parameter tuning of the base classifiers. The remaining 40% will be used for training and testing of these tuned base classifiers with the meta-classifier running on top of them. The base classifiers were trained separately and individually so that extra detail could be gained about the AUC scores each produced and also to make the training more feasible from a resource/time perspective. Both of these datasets were then split again 4:5 in terms of training:testing. Note that the hyper-parameters for the XGB base classifier were just those mirrored from the benchmark model.

Initially a GridSearch with extensive hyper-parameter ranges was to be run over each base classifier however this took too long with none of the GridSearches finishing. Therefore, a RandomSearch method was used instead over the base classifiers. However, in the case of the SVM classifier this still took too long and would not finish so a GridSearch with a small parameter range was used to at least get some hyper-parameter tuning.

The SVM base classifier also had separate data treatment as the training data needed to be scaled. This scaled data was then joined back on to the encoded dataset used by the other base classifiers for the stacking process. Pipelines were then constructed for first level classifier where the column range of the overall dataset was stated as a parameter (i.e. for SVM the second half of the dataset’s columns were used as opposed as the first half for all the other classifiers).

These pipelines were then fed into the meta-classifier and this stacked ensemble was then trained and tested on the remaining 40% of the original dataset. The time and AUC for each stacking combination was recorded. This was done several times with different meta-classifiers and base classifier combinations used.

**Refinement**

A couple of refinements were made on the XGB baseline model and the conclusions then extended to the other modelling techniques. The first was which data treatment to use for preparing the data for the classifiers. Either encode all categorical data numerically or to perform one-hot encoding. I initially thought that OHE would be far better than just encoding as no numerical ranking would be implied by OHE on categorical features. However, this proved not to be the case as both scored 0.843 AUC when a basic XGB algorithm was fit to the datasets with the encoded data actually scoring 0.0001 better. The key difference however was that the model using encoded data took ~3 seconds to train whereas the OHE data took ~24 seconds, eight times slower.

With Random/Grid Searches to be run on each classifier this would result in a much quicker and feasible experimentation process. Especially given that SVMs are infamous for taking a while to train with processing time of O(number of rows3 x number of features) as per 1.4.4 in

<http://scikit-learn.org/stable/modules/svm.html#complexity>.

The class\_weight/scale\_pos\_weight was also discovered to make a significant improvement to the base XGB model’s AUC score. It increased by ~25% from 0.67 to 0.84. This makes sense as the data is very imbalanced with only ~8% of the target feature being positive (>$50K).

Finally a refinement was made choosing the best stacking combination (a table showing the permutations tried is displayed below with the best AUC and time records highlighted). Over several trials it was noticed that including XGB as one of the base classifiers greatly improved AUC. Also removing SVMs from the stacking process greatly enhanced speed. This led to an optimum base classifier combination of XGB, Random Forest and Logistic Regression. Despite a Logistic Regression meta-classifier being the fastest it produces a lower AUC than an XGB or Random Forest which are only marginally slower.

|  |  |  |  |
| --- | --- | --- | --- |
| **Base Classifiers Used** | **Meta-Classifier Used** | **AUC Score** | **Time Taken to Run Stacking Model (Seconds)** |
| Random Forest, Logistic Regression, SVM | XGB | 0.81714357 | 218.97 |
| Random Forest, Logistic Regression, XGB | SVM | 0.84491662 | 52.02 |
| Random Forest, Logistic Regression, XGB | XGB | 0.84491662 | 34.72 |
| Random Forest, Logistic Regression, XGB | Logistic Regression | 0.84404280 | 32.88 |
| Random Forest, Logistic Regression, XGB, SVM | XGB | 0.82411076 | 257.37 |
| Random Forest, Logistic Regression, XGB | Random Forest | 0.84491662 | 34.44 |

**IV. Results**

**Model Evaluation and Validation**

The final stacking model has XGB, Random Forest and Linear Regression base classifiers and a Random Forest meta-classifier. This produces the best AUC score combined with the fastest training time. However, this AUC score of 0.845 is not better than the benchmark model which is a surprise although the score itself is high which is to be expected given that it is utilizing several modelling techniques. The final parameters are appropriate for the base classifiers as some parameter optimization was done. More parameter augmentation on the base classifier could be have been done and also for the meta-classifier but given that the output AUC score is good I think the parameters are appropriate.

The stacked model was tested on unseen data as designed by splitting the data into train/test sets and so I think the final AUC score is reliable. The tuning of hyper-parameters for the base classifiers was done on a completely separate portion of the data used for training the whole stacked model (apart from the XGB model). This also supports the ultimate AUC score obtained, and thus the model, generalizes well to unseen data. This additional data splitting and training/testing further supports that the results from the model can be trusted, especially since the AUC score close to and thus supported by that of the benchmark model.

The model is robust enough for the problem as I have used several different types of base classifiers and different meta-classifiers to achieve roughly the same AUC. I have changed the input space by using different model combinations in the stacking and these have produced AUC scores within 0.03 or ~3.7% of each other.

One issue I would like to explore having more time would be to investigate why the AUC results from the different stacking combinations generally were lower than the individual AUC score of the best base classifier. This is not enough though for me to suspect the stacked model is untrustworthy.

**Justification**

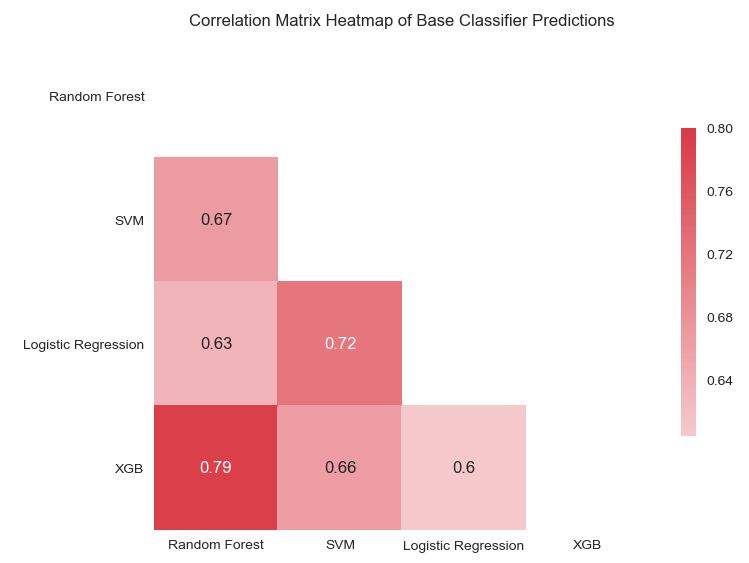
The AUC of the benchmark model is 0.854 and the best stacked model 0.845 thus the benchmark performed ~10.7% better than the stacked model. The time it took to run a RandomSearch was just under 8 minutes and the training time of an optimized model just 23 seconds. Conversely the total time it took to run searches for optimizing the base classifiers was ~96 minutes although 75 minutes of this was for SVM. Fitting the stacked classifier took 33 seconds.

In addition to this it was much more complex preparing the data and augmenting the hyper-parameters for the stacked model compared to just the XGB benchmark model.

Therefore, overall the benchmark is significantly better than the stacked model. Based on AUC score, time and ease of implementation I think the benchmark model solution is significantly better to the extent that it fulfils the purpose of the project investigation. However, the stacked model does have a wider scope to improve. Whilst there are definitely elements of the XGB approach that can be improved e.g. data pre-processing and parameter tuning, due to the nature of the number of permutations of stacking models, parameters and relevant data treatments I think stacking could potentially provide a better solution.

**V. Conclusion**

**Free-Form Visualization**



The above is a heatmap of the correlations between the predicted values of the test portion of the data used for hyper-parameter tuning of the base classifiers. I decided to look at this to see if these models are adding information in the stacked model. For instance, if they were all highly correlated then a stacking classifier would not have a diverse set of predictions to work with and so would be unlikely to perform well.

From the heatmap we can see that XGB and Random Forest are the most highly correlated which isn’t surprising as they both work by using an ensemble of decision trees. It was interesting to see that the next most correlated is SVM and Logistic Regression although again they are roughly similar techniques by creating a boundary line determined by probabilities.

Outside of these two combinations the other correlations between the models isn’t that high which is good. This suggests these are good choices for a collection of ensemble techniques as they should add reasonable levels of different predictions for the meta-classifier to work on.

**Reflection**

A summary of the project process is outlined below:

1. Identify a dataset with similar characteristics as that used in my job.
2. Explore the data to familiarise myself with important features and the general dataset statistics.
3. Pre-process the data ready to be used by various machine learning models.
4. Build a benchmark XGBoost model to provide a baseline AUC score.
5. Using various other machine learning techniques build a stacked classifier with a meta-classifier to produce a comparison AUC score.
6. Compared the stacked classifier against the benchmark based on AUC and time taken to train the models and established that the XGB benchmark performed better.

There were a couple of parts of the project I found particularly interesting. The first was the impact of adjusting the positive/negative weighting in the classifier parameters. Inputting this parameter vastly improved AUC scores and whilst not being surprising that it has an effect the extent to which it did was unexpected. I think there was such a marked improvement because the dataset was very unbalanced. The second interesting element I found was just comparing different machine learning techniques not just in AUC score but also in the way they handle data and their training times.

Related to this second point in terms of what I found most challenging was trying to implement the SVM technique. This technique took by far the longest to train and had more specific data pre-preprocessing requirements. In the end it seemed not to even have the best AUC score. Trying to then integrate this into the stacking classifier required additional thinking and extra steps.

The final solution I would recommend for this type of problem is an XGBoost model by itself and so from this point of view this solution would work in a general setting as demonstrated by its success in Kaggle competitions and even the AUC score in this project. This outcome was a surprise at least for the AUC score. I expected the training time to be longer for the stacking classifier but did not expect it to generate a lower AUC. Ultimately given how much quicker XGB is in addition to it producing a slightly better AUC score makes it the best choice.

**Improvement**

One aspect that could have been improved was the tuning of the meta-classifier. Due to time and resource constraints this was unfortunately not possible in this project but there is a lot that could have been done. For instance, investigating the option of using the parameter that trains the meta-classifier on the dataset along with the predictions of the base classifiers could have improved the end result. Also tuning the hyper-parameters of the meta-classifier by a Grid or Random Search I think could definitely improve the AUC. Both of these parameter augmentations could then be performed on different meta-classifiers to see which one produced the best result.