MIRNA ARIVALAGAN - 220142881

MACHINE LEARNING – SIT720

Trimester 2 2021

Contents

Question 1

What is an ensemble classifier? Name some of the popular ensemble methods (at least three) and which one you prefer and why?

Ensemble learning is a machine learning approach that combines multiple models into one optimal model. An ensemble model works by combining the predictions from multiple models and either getting an average accuracy scoring from each model or implementing a voting classification and getting the scores from the models with the most votes.

There are multiple ways to build ensemble models, such as bagging, boosting and stacking and voting.

Bagging works by providing just a sample of the dataset to each individual model that is embedded in the bagging model for training. Bagging is a row sampling with replacement technique. Once training is done, when test data is passed through, when the test data is passed through the models within the bagging model, the average from each model within the bagging classifier is calculated and provided. Bagging reduced overfitting problems and works well with high dimensional data. Random forest algorithm is an example of a bagging type model. Sklearn also has its own built-in classifier named Bagging Classifier. Bagging works well especially for Decision Tree models that are prone to overfitting.

Boosting is a variation on bagging where it works on algorithms that are in the boosting model that is not performing as well as the other embedded algorithms within it. Boosting works by using all the training data to train the model, but the difference between boosting and bagging is that with boosting, it uses training data to validate or test the model as well. It then identifies data points that are not well predicted, then when the model goes through the next step of training the data, when selecting the data for training, it will select data from the training set, but place an emphasis on data points that has had significant error weighting from the first round of training. The algorithm then builds a model that has a mix of training data, inclusive of the data points that had errors, and then uses the test data to compare the performance. This process repeats until the total number of bags are filled. So really boosting is working hard on the weak learners to improve the performance of the model. Popular boosting algorithms are Adaboost, XGB Boost and Gradient Boosting.

Finally Stacking and Voting ensemble methods are similar. The main difference between these two methods are in how the final aggregation is produced. Voting ensemble can be seen as an extension of the bagging ensemble, here when you have multiple models,

the voting ensemble will calculate the number of correct prediction across the models, and the prediction with the most votes is then used as the final prediction. So for example in our instance, we have 3 models to predict if the customers will purchase an insurance or not, out of the 3 models, 2 predicted will purchase and the other model predicted will not purchase, the final outcome by the voting model will be predicted as purchased.

Stacking on the other hand is the generation of a Meta-model. How this method works is that it trains to make predictions on multiple models that have been passed through the stacking classifier, these predictions once it has gone through the cross validation will then be combined into 1 meta-model. Once the meta-model is constructed, it will then be trained with the additional final estimator model to produce predictions. Popular algorithms for these stacking and voting techniques is Mlxtend's Stacking Classifier and Sklearn's Voting Classifier.

My preference out of the 3 different ensemble methods here would be the stacking and voting method. I believe this will produce a more robust result as we are combining multiple models into one and has a cross validation stage in it that will produce more reliable results.

Question 2

Let's assume we have a noisy dataset. You want to build a classifier model Which classifier is appropriate for your dataset and why?

Noisy dataset can be defined as dataset that has meaningless inputs in them, this can either be features or observations that don't make sense. I would algorithms that are best suited to noisy data sets would be **Random Forest and Support Vector Machine.**

One of the key advantages of Random Forest is that it can handle large amounts of variables which is often the case with noisy data. Random Forest is technically multiple decision trees in one model, and it can also be considered as a bagging ensemble. By utilizing bagging and feature randomness, this algorithm is designed to reduce variances by getting the majority vote, thus making this model suitable to be used with noisy datasets.

Support Vector Machine deals with noisy data using its kernel trick and non linear SVM. The kernel implicitly transforms data points from a 2D space to a 3D space making the data points linearly separable. SVM flexibility with hard and soft margins also works well with noisy datasets which is often not separable by a linear line.

Question 3

Load and pre-process the dataset if necessary. Explain steps that you have taken. Are there any alternative ways for doing that? Explain

The pre-processing steps that I have done were the following:

- Check the data types – all looked ok except for dates. Changed Effective to Date field to time format.
- Created Day and Month column as the year is all 2011. Also did this to standardise the data as we cannot pass date time format fields through the model algorithms.
- Checked data frame for null values, there was no null values to deal with
- Visualised distribution of data on two plots. Noticed that the data is skewed and there is a class imbalance issue with the target variable of 'Response'.
- Mapped Response column to Yes = 1 & No = 0
- Created X and Y data frames – One hot encoded categorical variable
- Split data into train and test datasets (80%-20%)
- Visualised numerical variables on train dataset and identified that there are outliers present in Total Claim Amount, Customer Lifetime Value, Monthly Premium Auto.
- Data Scaling – I decided to use RobustScaler to scale the datasets as there are outlier present and this scaling method works well with outliers.

An alternative pre-processing step that we can consider including in this pre-processing step is balancing the target variables using the SMOTE technique. I have not done it in this instance as it is out of the scope of this assessment. There are also other options with the scaling data and one hot encoding techniques, for instance a MinMax scaler could have been used instead, and instead of using get dummies like I have, there are ordinal encoders or label encoders that could have been used.

Question 4

Analyse the importance of the features for predicting customer response using two different approaches. Explain the similarity/difference between outcomes.

I have used a Logistic Regression model approach and a Random Forest model approach to identify features that are important. I have also decided to do the feature importance on scaled and one hot encoded data as we are getting the feature importance from a

model, and model's don't tend to work well on unscaled data, and with categorical variables that have not been converted.

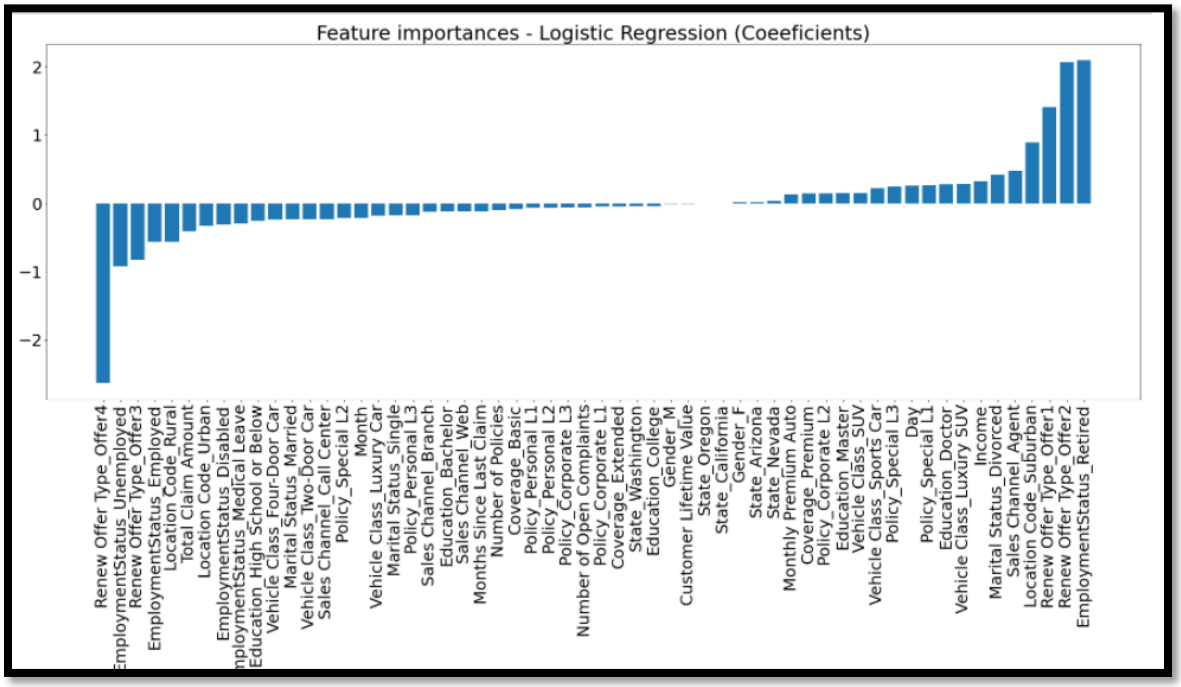
With the logistic regression model, the coefficient value is used to assess the feature importance. I have analysed the top 5 positive coefficient values and top 5 negative coefficient values. The top 5 positive coefficient values were for the following features:

- EmploymentStatus_Retired
- Renew Offer Type_Offer 2
- Renew Offer Type_Offer 1
- Location Code_Suburban
- Sales Channel_Agent

The top negative coefficient values were:

- Renew Offer Type_Offer4
- EmploymentStatus_Unemployed
- Renew Offer Type_Offer 3
- EmploymentStatus_Employed
- Location Code_Rural

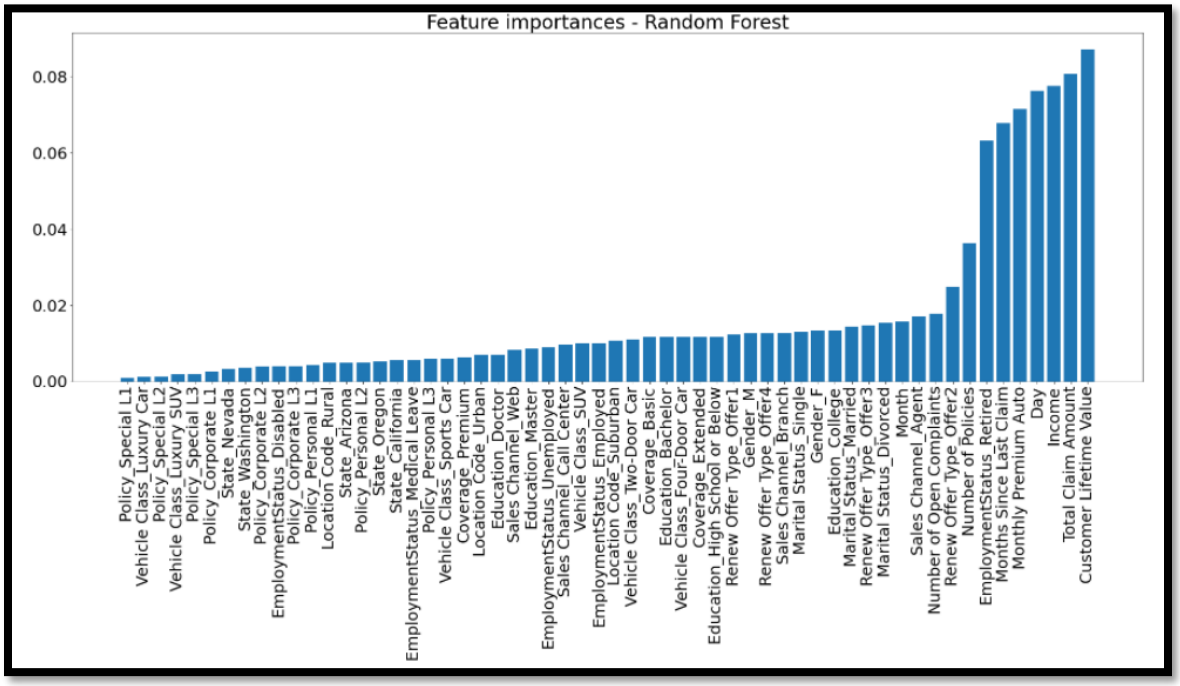
From the above we can conclude that the feature Renewal Offer, Location and Employment status are key predictor for the regression model. This makes sense as we are trying to predict if the customer purchases the insurance or not, employability is a key driver, as you will not be able to afford insurances if you are unemployed and if there are any renewal offers provided, a customer will be more likely in renewing or purchasing addition insurances



My second method is using the Random Forest model. From the output of the model, the following features were the top 10 features:

- Customer Lifetime Value
- Total Claim Amount
- Income
- Day
- Monthly Premium Auto
- Months since Last Claim
- Employment Status_Retired
- Number of Policies
- Renew Offer Type_Offer2
- Number of Open Complaints

There are some similar features across both models, mainly being the Employment Status of Retired and the Renewal Offer Type. The Random Forest model appears to place more importance on numerical features such as Customer Lifetime Value, Total Claim Amount and Income. This makes sense as my assumption would be that there would be a linear relationship between these variables as that as they increase, it increases the likelihood of the customers purchasing or renewing the insurance.



Question 5:

Create three supervised machine learning (ML) models except any ensemble approach for predicting customer response

5A - Report performance score using a suitable metric. Is it possible that the presented result is an overfitted one? Justify.

Initially, I ran all 4 models with the default in built parameter. I will not use Accuracy to assess the model as this metric is not good when handling imbalanced classes and can be misleading, so we will use **Recall, F1 and Matthews Coefficient** as these are better suited when evaluating imbalanced classes models.

Recall:

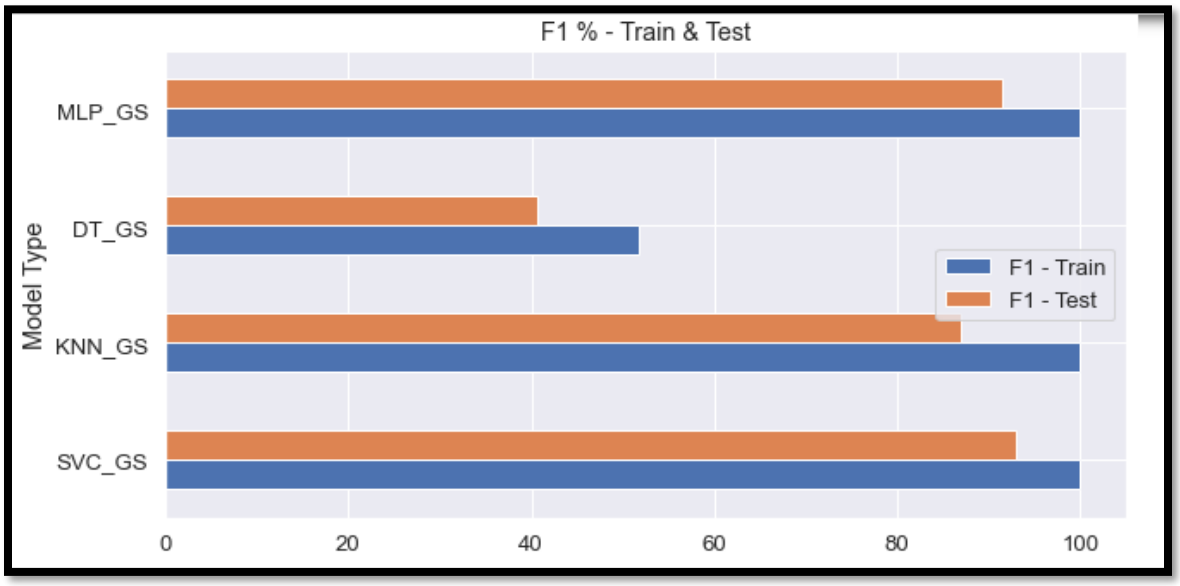
We can observe that there is some overfitting here especially with the MLP and DT models as the difference between the train and test for these 2 models are quite big.

Both DT and MLP had 100% ability in classifying the positive labels but underperformed in the test sets. Decision Tree underperformed by a wider variance, where it was only able to classify the true labels by 65.4%. This indicates some overfitting for the decision tree model.

SVC and KNN had a recall rate of above 95% in both train and test rounds.

We were able to reduce some over fitting for the MLP model after doing grid search to tune the parameters, but the DT model has performed much worse after tuning the hyperparameters and it is underfitting.

F1:



F1 is the harmonic mean between recall and precision. After hyper parameter tuning, the SVC model performed best with its ability to retrieve the correct predictions with a high 92% in the test run. The next best model is the MLP model that scored 91%. Again it appears that KNN is overfitting as well as it scored 100% in the train run but only scored 87% in the test run. DT is underfitting and performing poorly after grid search.

Matthews Coefficient

This is the most appropriate metric to use when dealing with imbalanced datasets. It is a contingency matrix method calculating the Pearson coefficient between actual and predicted values.

After grid searching, the SVM model performed the best and is not over fitting, in the train run, this model scored 99% and test run it scored 91%. KNN appears to have some over fitting issues after hyper tuning, as in the train run, it scored a high 100% MCC score, but in the test run, it only scored 84%. Decision Tree seems to be under fitting after performing the grid search, and for this metric, before the grid search, it scored 100% in the train run and 67% in test run, but after grid search, it scored 47% in train run and 34% in test run.

Model Type	Accuracy - Train	Accuracy - Test	ROC_AUC - Train	ROC_AUC - Test	Recall - Train	Recall - Test	F1 - Train	F1 - Test	Precision - Train	Precision - Test	MCC - Train	MCC - Test
SVC	90.42	87.19	93.97	90.63	98.94	95.49	74.66	68.46	59.94	53.36	72.49	65.30
KNN	96.56	94.85	97.68	95.12	99.23	95.49	89.18	84.39	80.97	75.60	87.80	82.15
DT	100.00	92.39	100.00	81.20	100.00	65.41	100.00	71.46	100.00	78.73	100.00	67.50
MLP	100.00	96.11	100.00	94.45	100.00	92.11	100.00	87.34	100.00	83.05	100.00	85.21
SVC_GS	99.96	97.87	99.98	97.50	100.00	96.99	99.86	92.97	99.71	89.27	99.83	91.83
KNN_GS	100.00	96.00	100.00	94.39	100.00	92.11	100.00	87.03	100.00	82.49	100.00	84.86
DT_GS	88.81	86.26	69.35	63.89	42.13	32.33	51.77	40.66	67.13	54.78	47.40	34.96
MLP_GS	100.00	97.32	100.00	98.12	100.00	99.25	100.00	91.51	100.00	84.89	100.00	90.31

5B – Justify different design decisions for each ML model used to answer this question.

For this assessment I have decided to use the following models to solve the problem:

- Support Vector Machine (SVC)
- K Nearest Neighbours (KNN)
- Decision Tree (DT)
- Multi-Layer Perceptron Classifier (MLP)

The reason why I chose to use SVC is because this model works well in binary classification problems which it is in our instance. Support Vector works by finding the best hyperplane with the largest margin that separates the data points from one class to the other class. I picked this model also because of its flexibility in hyper tuning using a variety of kernels, as the kernels provides with options on data points that do not have a straight boundary line. I also decided to use this because we are not dealing with high dimensional data which does not work well with SVC.

The second model that I have used is KNN. KNN is a distance-based model that classifies data points based on similarities. One of the main reasons why I chose this model is because that there are no assumptions around the data that needs to be made, so this

means this will work well in nonlinear data sets. I also picked this as this model performs well with imbalanced classes, which it is in our instance.

The third model that I have decided to use is Decision Tree. Reason why I chose this model is because I believe this model is scalable when it come to working with real life datasets as missing values don't seem to impact the performance of the model, and with real life datasets, this is often the case. Another reason why I have chosen this algorithm is because it does not require any normalisation, thus again its scalable to real life datasets. This was a test wild card model that I have included. From the experiment, I have observed that this isn't the best model for the task, and it is prone to overfitting.

The fourth model I have chosen to use if MLP. MLP is a neural network method for classification. This algorithm relies on the underlying Neural Network to work. (Nair, n.d.). Reason why I have chosen this method is because it works with non-linear data which can be observed in our instance. This method is also scalable and can work with large amounts of data quite easily.

5C – Have you optimised any hyper-parameters for each ML model? What are they? Why have you done that? Explain.

I have optimised parameters for each of the 4 models via a grid search cross validation. The parameters that I have optimised for each model is:

Support Vector Machine:

- C – this optimises how hard or soft the classification margin is. This allows a bit of flexibility with allowing samples to be in or out of the margin
- Gamma – I've optimised this as too large of a value can cause over fitting.
- Kernel – there is options for radial basis function, linear or poly. By passing this through a grid search, it will use the data to identify what is the best kernel to use as the kernel transforms nonlinear data points to linearly separable data points.

K Nearest Neighbour

- Leaf size – This affects the speed, so finding the best value will reduce in processing time and save on memory.
- Metric – To find the suitable distance metric to use to find the best neighbour
- n_neighbours – This addresses overfitting, if value is too small, it is classifying incorrectly
- p – find if Manhattan distance or Euclidean distance works best for our dataset

- Weights – find if uniform or distance weight measures work best for our dataset. Uniform works that all neighbours are weighted equally, and distance works that the closer distance points have more weighting than points that are far away.

Decision Tree

- Criterion – measures the quality of the split. Gini for impurity and entropy for information gain. Gini tells use how often a random observation would be incorrectly labelled. (Mithrakumar, 2019)
- max_depth – Too high of a value for depth can cause overfitting, so finding optimal value here is important.
- max_features – The number of features to consider when looking to split. This can be used to avoid overfitting as well, as by reducing the number of features, in stabilizes the tree more and reduces variance.
- min_samples_leaf – Indicates the minimum number of samples required to be in each node. Too small of a value can cause overfitting
- min_samples_split - Indicates the number of samples required to split each node, too high of a value can cause underfitting.

Multilayer Perceptron Classifier

- Solver – type of solver to use for weight optimisation. Adam works on large datasets and lbfgs works better on smaller datasets. (RendyK, 2021)
- hidden_layer_sizes – This allows us to set the number of layers and nodes in the classifier. Tasks that are more complex needs more layers
- learning_rate – This controls the step size for the model to reach the minimum loss function, higher learning rate makes the model learn faster, but this can also cause the model to miss things, so need to find a balance to this.
- activation – Activation function for the hidden layer.

5D – Finally, make a recommendation based on the reported results and justify it.

From the four models I have designed, I would say that after hyperparameter tuning the MLP and SVC model are the best models to use for this task.

The SVC model was able to predict 255 of the 266 customers that has a response of yes in the model, which means this model can predict if a customer will purchase the insurance. The recall score for the class 1 was high at 88% and the f1 score is high at 92% as well. We were able to significantly improve the performance of this model after hyper parameter tuning the model.

The MLP model was able to predict 264 of the 266 customers that has a response of yes, or 1 label in the model. The recall score for this model was high at 85% for our 1 class which is the response yes, and f1 score was 92%. Again, after hyper-tuning this model, we have addressed any overfitting issues with the default/base version of the model.

Question 6

Build three ensemble models for predicting customer response

6A – When do you want to use ensemble models over other ML models?

Ensemble model is best used when a model appears to be overfitting. By combining multiple models, this would improve prediction performance by combining the predictions and using the major voted one or the average values of the embedded models. Ensemble models are used to decrease variance and bias thus improving predictions and reducing error rate.

When deciding to use an ensemble model, we need to compare the performance of each model that is embedded into the ensemble individually with the ensemble. If the one of the individual models performs better than the ensemble method, then its advisable to use the individual model instead, but if the ensemble outperforms the individual models, we need to use the ensemble as the final model.

6B – What are the similarities and differences between these models?

For my solution, I decided to use 3 methods – the Bagging classifier, AdaBoost classifier and a stacking classifier that combined my models from question 5 (DT, KNN, SVM & MLP).

There are some similarities between the bagging and boosting classification models. One of the similarities between them is that both models train on training data using random sampling techniques. However, there are slight differences in the random sampling techniques, with bagging random features and samples are selected with replacement from the dataset, and with boosting data is selected without replacement.

Both these techniques also get the N learners from 1 learner, and another similarity is that both these methods give the final prediction by getting the average from the learner's data or getting the majority vote from the learners. The difference between both these methods are that boosting is a sequential process where each embedded model tries to correct the errors from the previous model, this is done by giving higher weighting to the misclassified data points. Another difference between bagging and boosting is around the model building process, with bagging the models are build independently, and with boosting, new models are influenced by the performance of the previous models.

Bagging also tries to solve the over-fitting problems, most common seen in decision trees and boosting tries to reduce bias.

My third is a stacking ensemble model, this technique is slightly different to bagging and boosting in a way that this model uses meta learning approach. Multiple models are combined into one model and then set one separate model as a meta model. The model that has been set as the meta model will take inputs from the other models that have been stacked and will learn from it to make the final prediction. One of the main differences with this approach compared to bagging and boosting is that this approach does not utilise random sampling, thus some sort of cross validation needs to be built into the model. I have used the hyper tuned models from question 5 as my learners and have set the meta model as a logistic regression model.

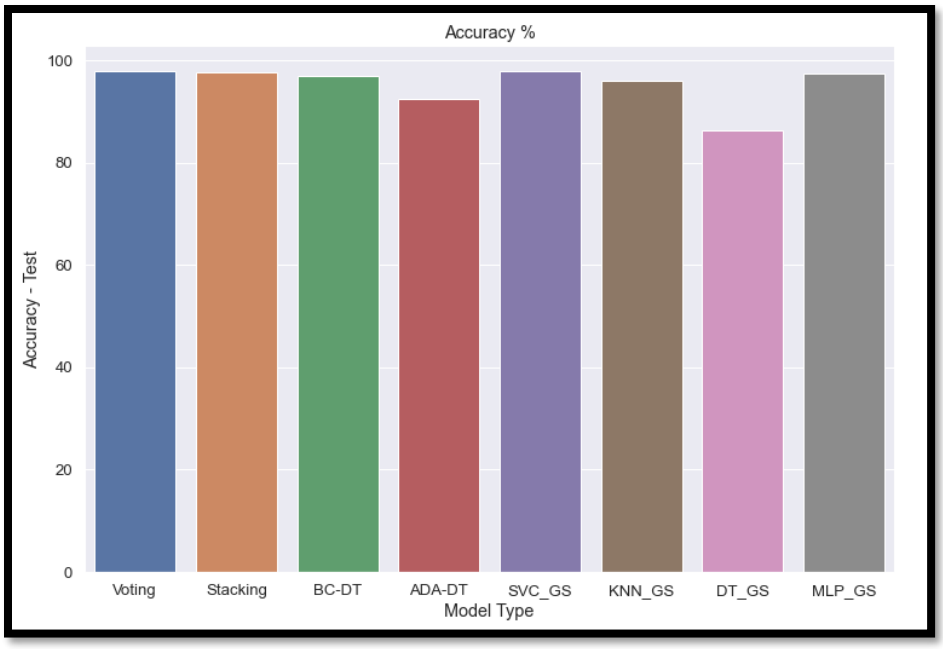
6C – Is there any preferable scenario for using any specific model among set of ensemble models?

Ensemble methods are best used when you have a few models that has variations in its performance. By using ensemble methods in this instance, you are combining your models and optimizing them further by using the average or majority vote from each model, thus providing a much more accurate prediction. Ensemble models are also ideal to use when the models that you are working appears to be overfitting.

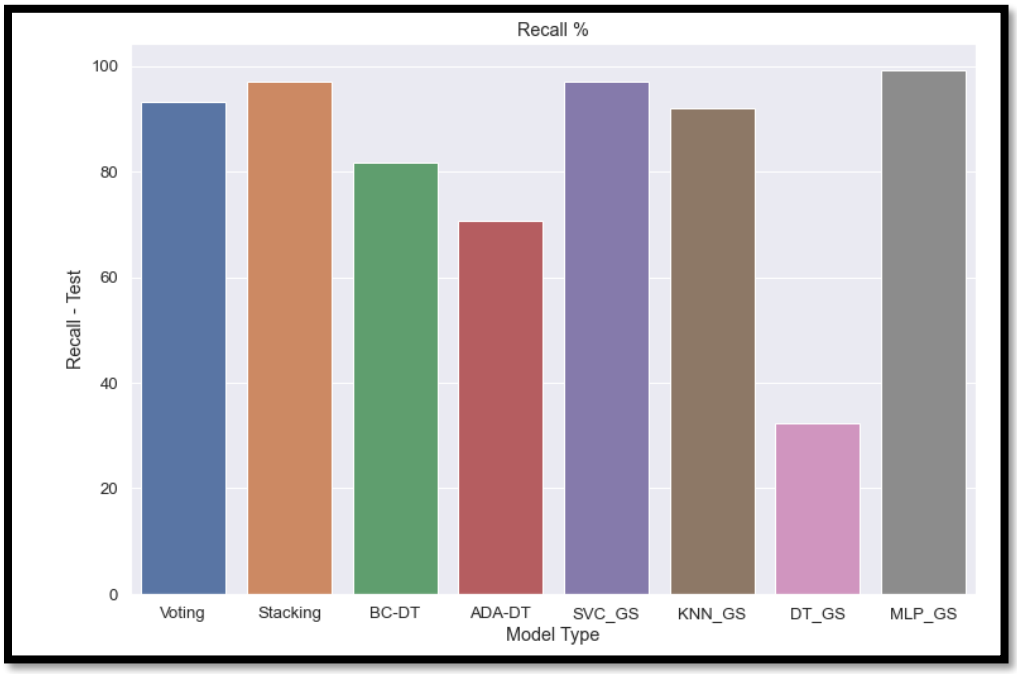
6D – Write a report comparing performances of models built in question 5 and 6. Report the best method based on model complexity and performance.

In terms of model accuracy, most of the models had similar test accuracy around 96% - 97%. However DT_GS (86%) which is the decision tree tuned model and ADA-DT (92%), which is the adaboost decision tree model performed the worse. In terms of the models

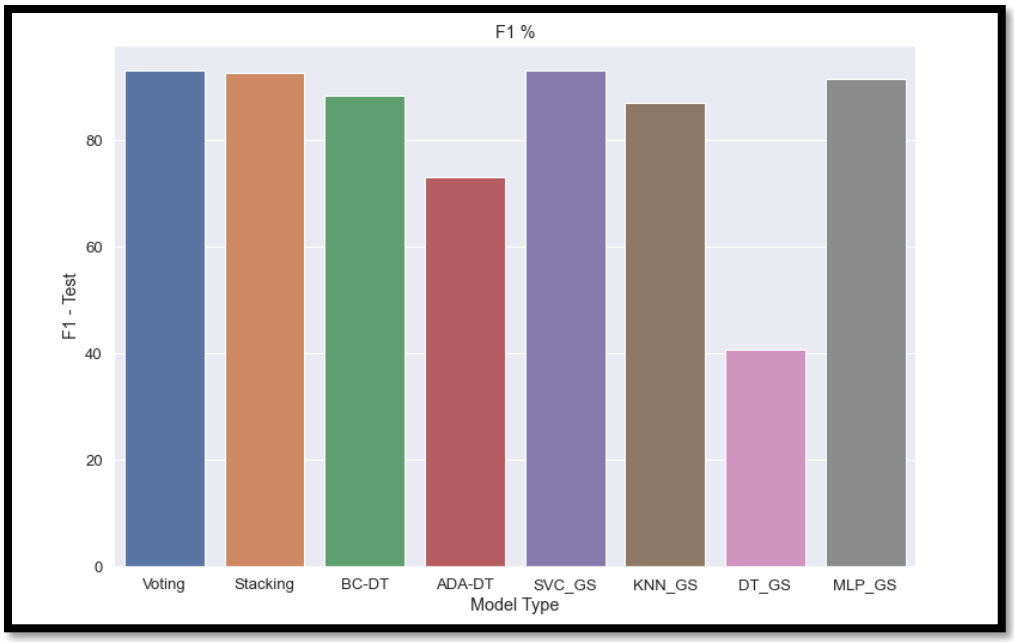
that are low bias and low variance, this would be the Voting and SVC models as both these models have accuracies between train and test that are very close to each other.



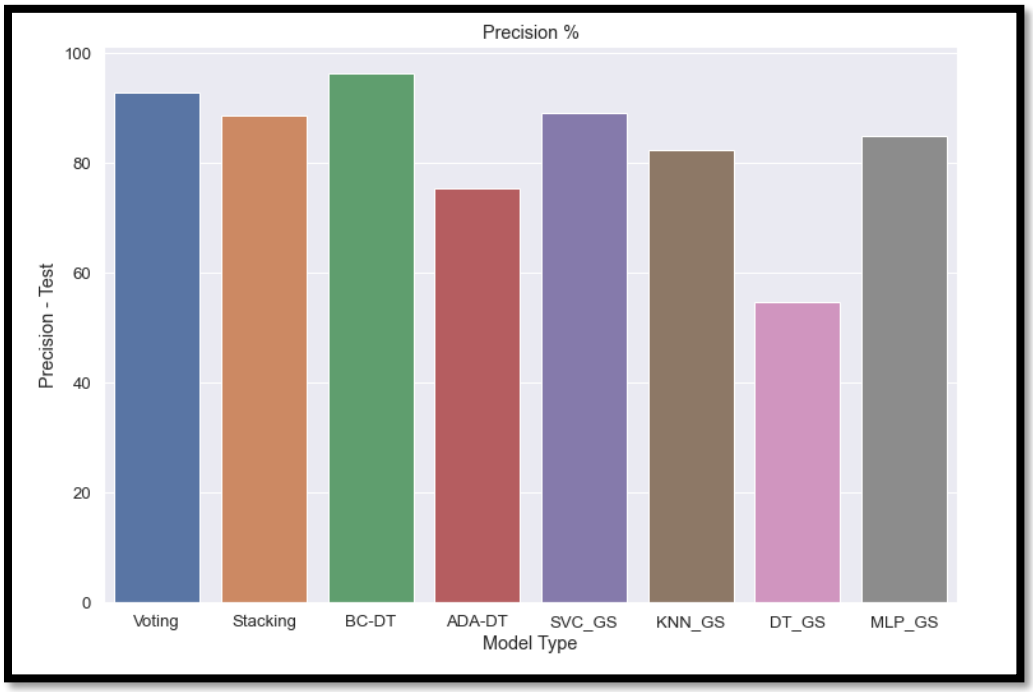
When we look at the recall percentages, is a significant variation between these models. All four ensemble methods performed worse when comparing to the tuned models from question 5 as the MLP (99%) and SVC (96%) models performed well in terms of recall.



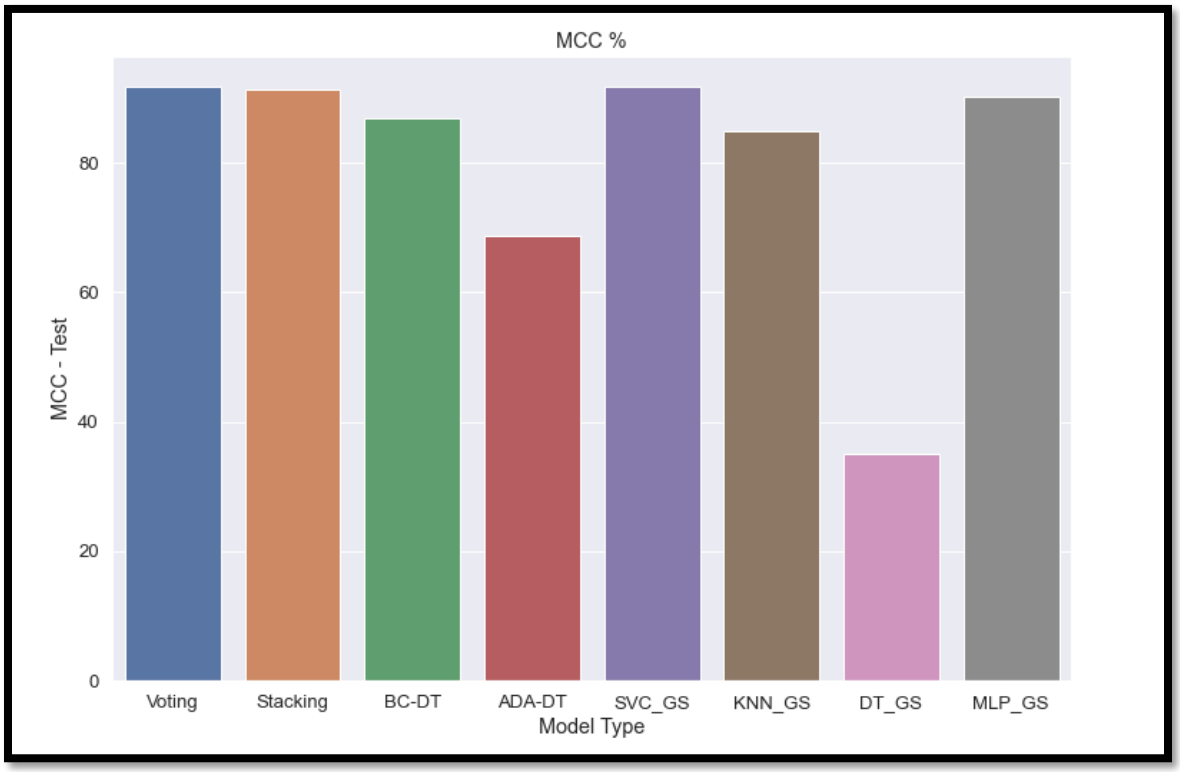
In terms of the F1 score, the voting (93%) ensemble outperformed the rest, and the second best is the SVC model (92%). Decision Tree is consistently performing the worse.



In terms of precision, the BC-DT (96%) performed the best followed by the voting ensemble (92%). There appears to be a large variance with the ADA-DT, KNN and MLP between the train and test results.



When we look at the MCC %, there is some variances, but the Voting (91%) , Stacking (91%) and SVC_GS (91%) all performed well compared to the rest. I believe due to the class imbalance; this is the metric we should be looking at closely when assessing the overall performance of the models.



Based on the performance of these models, not all ensemble methods have outperformed the standalone models, but I would say that out of the four of them, the Voting and Stacking method is consistent and have outperformed the other models.

I believe this is the best model to use, especially when we use it with a set of tuned standalone models. By using these methods, we can combine a few algorithms into one and optimize even further.

The SVC model is a powerful model as well, as it has been performing well across all metrics and does not appear to be over fitting, however this model requires some hyper parameter tuning to optimise its performance, which I have been able to demonstrate by using a grid search, pre tuning, this model performed the worst out of all of them.

Decision Tree models, is not as suitable as a standalone model as it is prone to overfitting, as demonstrated, it requires an ensemble technique to be used along side it to improve its performance.

6E – Is it possible to build ensemble model using ML classifiers other than decision tree? If yes, then explain with an example.

One way of using ensemble learning without a decision tree is using a Voting Classifier. I have used this as my last ensemble model, where I have included baseline KNN, SVC and MLP models to be embedded into the Voting Classifier Model. This Voting Classifier will run each of those models and collate the number of votes, where the majority voted model's prediction is then aggregated and used as the final prediction.

Bibliography

Mithrakumar, M., 2019. *How to tune a Decision Tree.* [Online]
Available at: https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680
[Accessed 26 September 2021].

Nair, A., n.d. *A Beginner's Guide To Scikit-Learn's MLP Classifier.* [Online]
Available at: https://analyticsindiamag.com/a-beginners-guide-to-scikit-learns-mlpclassifier/
[Accessed 26 September 2021].

RendyK, 2021. *Tuning the Hyperparameters and Layers of Neural Network Deep Learning.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/
[Accessed 27 September 2021].