Individual Project – Statistical and Machine Learning Methods

Shabenoor Kamal

Task 1: Explain the mechanism of the algorithms using your own words (e.g. the general idea of the algorithm, the objective function, the algorithm fitting process, pros and cons of the algorithm

The five statistical algorithms that are covered in this report are:

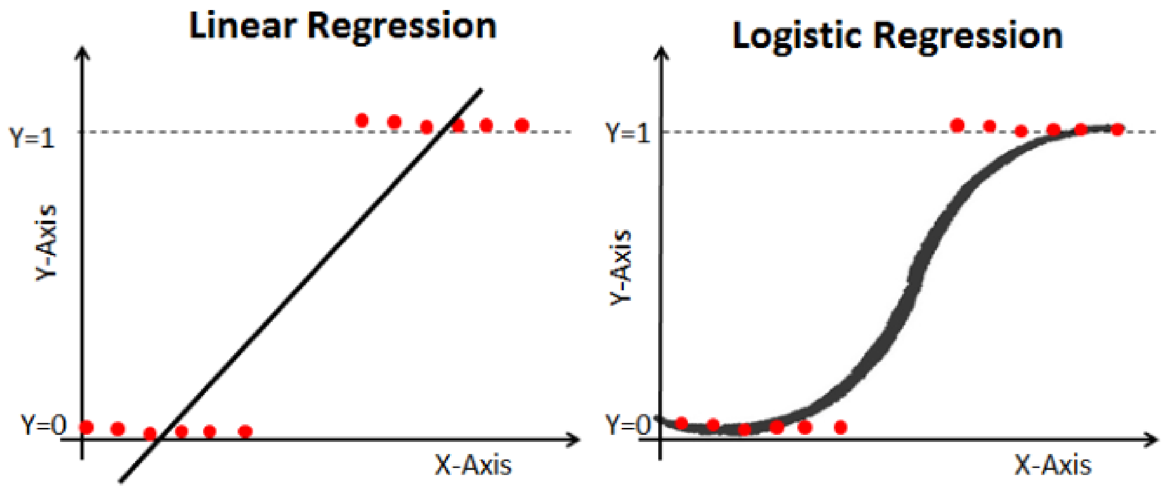
1. Logistic regression
2. Decision tree
3. Random Forest
4. Gradient Boosting
5. SVM

1. Logistic Regression

The logistic regression algorithm is suitable for applying to classification problems. It is a supervised learing algorithm. The algorithm tries to predict the probability of a certain class's existence: such as churn/not churn, win/lose or in this case subscribe/not subscribe. Along with binary problems, which have two classes (as in the examples described above), the algorithm can also be used for multiclass classification problems (where multiple classes exist). An example of this is our Hackathon project where we are trying to predict the industry codes for different countries. We have hundreds of industry codes and thousands of companies and we can use logistic regression to find the probability that each company belongs to a certain industry (industry code).

It is related to Linear Regression but differs from it as it uses a different cost function. Linear cost functions can have a value that is less than 0 or greater than 1. This is fine for linear regression as it works with continuous variables.

The logistic regression uses a sigmoid function. The advantage of this is that it can map any value to another value between 0 and 1. In this way predictions can be turned into probabilities (between 0 and 1) and it allows us to get the probability of belonging to a certain class of the categorical outcome variable. The algorithm classifies the data by considering the outcome variable classes (eg subscribe/not subscribe) on extreme ends (subscribe = 1, not subscribe = 0) and then tries to make a logarithmic line that distinguishes between them. This can clearly be seen in the graphs below.



Difference between linear regression and logistic regression

The sigmoid function is given below:

$$f(x) = \frac{e^x}{1 + e^x}$$

How the algorithm works:

First, the algorithm tries to get the probability P(X) that an observation will belong to Class 1 (in our example: subscribe). The formula for the probability is given below where e is a constant and B0 and B1 are coefficients.

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

To get this is does log odds transformation of the probability p(X) as show below:

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X$$

The logistic regression curve is constructed using the natural log of the "odds" of the target variable, rather than the probability.

How to get the parameters of the model:

To get the parameters of the model Maximum Likelihood Estimation (MLE) is used. MLE helps in determining the parameters of the distribution that best describe the given data. It is a method that will estimate the population parameters (in the case of logistic regression it estimates coefficients of B0 and

B1) from the sample data such that the probability (likelihood) of obtaining the observed data is maximized.

Based on certain values of the coefficients, certain values will be more likely to occur. So basically we have seen the observed data and the logistic regression equation, and now we need to find the values of the coefficients B0 and B1 that are most likely to have produced the data.

MLE assumes we assume that the observations are independent of each other and identically distributed random variables from a normal probability distribution. It then uses the log-likelihood function by taking the log of the likelihood, and tries to maximize this function

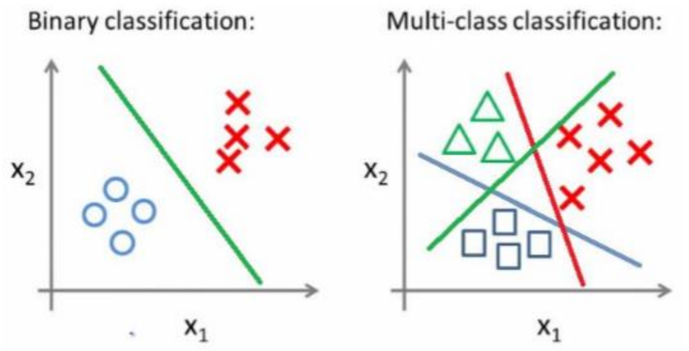
$$\begin{aligned}\ell(\beta) = \log \mathrm{L}(\beta) &= \sum_{i=1}^{N} \{y_i \log p(x_i;\beta) + (1-y_i)\log(1-p(x_i;\beta))\} \\ &= \sum_{i=1}^{N} \{y_i \beta^T x_i - \log(1+e^{\beta^T x_i})\}\end{aligned}$$

To find the maximize the log likelihood function, first the derivate is taken and equated to 0. Then a second derivative of the function is taken to confirm that it is negative. To do this a Gradient Descent optimization algorithm can be used. In this way we can get the values of B0 and B1.

For multi-class logistic regression the process is similar however there are more classes of the categorical target variable therefore the model will have more coefficients: B0, B1, B2 .... Bx. The values of all these coefficients can be found using the maximum likelihood technique described above.

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}} \qquad \log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

The remaining steps for getting the probability p(X) and log odds of the probability remain the same.



Advantages of Logistic regression:

- Most people are familiar with logistic regression and it is a widely used technique. In a business context this is of value as managers are more likely to understand how it works, and to trust the results from it. Additionally the results are easy to interpret as they are probabilities. This makes intuitive sense as it represent the probability of belonging to a certain class.
- It is efficient and does not require large computational resources. Hyper-parameter tuning is not required in this model which further saves computational resources.

- Logistic regression is easy to run as it does not require input features to be scaled. It also does not require model tuning. Additionally its predicted probabilities are well-calibrated and easy to understand.
- Moreover, the predictors do not have to be normally distributed or have equal variance in each group.
- Due to its simplicity and the ease of running it, it can be a good baseline model. This baseline can then be used to compare the performance of other algorithms (eg random forest, gradient boosting) that are more complex. It has low variance, and gives good efficiency despite its simplicity
- Can be used for feature extraction

Disadvantages of Logistic Regression:

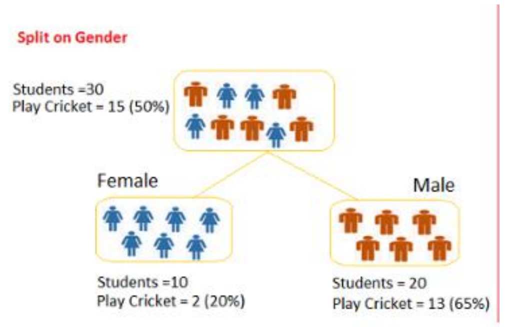
- Careful feature engineering and feature selection is required to get good results with this algorithm. Attributes that are unrelated to the output variable should be removed to improve predictions. Additionally multi-collinearity can be a problem therefore correlation between variables has to be checked, it will not perform well if variables are correlated to each other.
- Cannot handle a large number of categorical variables well
- Logistic Regression can be outperformed by more complex models in most situations as it is a relatively simple model. It may not be flexible enough to capture more complex relationships
- Non-linear problems cannot be solved with logistic regression and non-linear features have to be transformed

2. Decision Tree

Decision Tree is a machine-learning algorithm that can be used for both regression and classification problems. It is a non-parametric supervised learning algorithm.

Decision tree builds classification or regression models in the form of a tree structure. The tree structure is developed as the data is broken down into smaller and smaller subsets. It learns from data by creating a set of decision rules (of the form if-then-else). As the tree grows deeper (model learns more), the decision rules become more complex.

The final result is a tree which has both decision nodes and leaf nodes. A decision node has two or more branches (meaning the tree splits at these nodes). The leaf nodes represent a classification or decision. The tree-creation process starts with the top decision node in the tree which is the best predictor in the data (to predict the target variable). This topmost decision node is called the root node. Each of the other decision nodes is basically a predictor. At each decision node the data splits into smaller subsets based on the predictor variable which represents the decision node – each branch is a possible answer to/value of the variable.



The goal is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules from the training data. Using these rules the tree is developed.

For predicting a new observation we start at the root node of the tree . We can then compare the values of the root predictor with the observation's predictor value. On the basis of comparison, we follow the branch corresponding to the observation's value and move on to the next node. This process continues till a leaf node is encountered, this will provide a classification for that observation.

How the algorithm works

The algorithm aims to divide the predictor space into different non-overlapping regions. The logic is that if an observation falls into region 1 for example the predicted value of that observation will be the mean of the values of the training observations that are also in region 1. The aim is to find regions that will minimize the squared residual error of predictions (RSS – residual sum squared) for regression problems. For classification problems gini index, cross-entropy or classification error rate can be used.

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2.$$

- $\hat{y}_{R_j}$: the mean response of the training observation in R_j box (or region)
- y_i : the response of training observation in R_j box (or region)

There are multiple algorithms that can be used to decide when to split a decision node into two or more branches. The aim of splitting is to increase the 'purity' at each node with respect to the target variable (ideally each sub-node should have results that are the same). If the tree keeps on growing it splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes. Some examples of the algorithms include CART and CHAID.

These algorithms use a top-down 'greedy' approach which means it doesn't look back at the whole data, looks only within the splits to make them smaller regions - solution may therefore not be optimal.

There are 2 main ways for attribute selection (for all nodes): information gain and gini index.

Information gain looks at the information contained by each attribute. It uses entropy to measure the randomness of variables. In a binary classification problem if all observations are the same class (subscribe or not subscribe) entropy will be low, if there is 50% from each class entropy will be high.

Information gain looks at the expected reduction in entropy if split on a certain attribute. The aim is to minimize entropy.

Gini index is a metric that looks at if a random observation was chose how often would it be classified incorrectly. Therefore attributes with lower gini index should be preferred (lower chance of misidentification).

To reduce overfitting and the complexity of the tree pruning has to be done. This can be pre-pruning (set the criteria at which tree should stop growing) or post-pruning (grow a large tree and then prune it back).

Advantages:

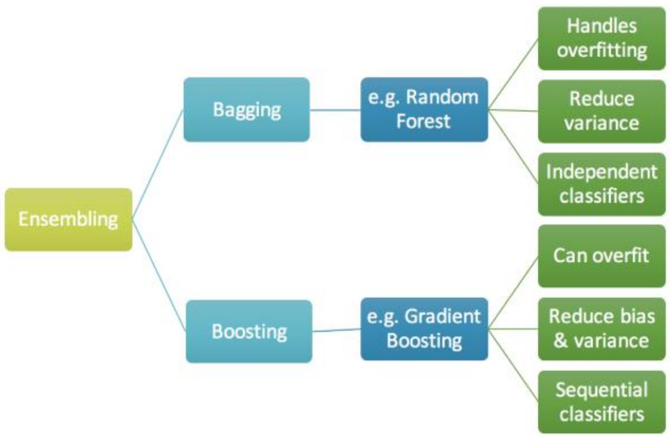
- Simple and useful for interpretation – most average business users can clearly see how it works, therefore it is easy to explain. Decision Tree is one of the easiest and popular classification algorithms to understand and interpret as it is basically a set of rules.
- Can be used for both regression and classification
- Can be improved further by combining multiple trees (explained in next models).
- Decision trees can handle both categorical and numerical data
- Don't have to make dummy variables and don't have to replace missing values
- Very few hyper-parameters to tune

Disadvantages:

- High variance
- Single tree is not good as other models
- Problems with overfitting the data (can happen if tree is not appropriately pruned)
- Low prediction accuracy compared to other ML algorithms.
- Calculations can become complex when there are many class labels

Gradient Boosting and Random Forest

Both Gradient Boosting and Random Forest are tree-based models however they work differently.



3. Random Forest

Random forests are built on decision trees described above. It can be used for classification or regression and is a supervised learning problem. While decision trees are simple models they are also very flexible and has high variance which can lead to overfitting on the training data.

One of the ways to prevent overfitting in a decision tree is to prune it, another way is to combine many decision trees into a single ensemble model which is the random forest. The random forest is made up of hundreds or thousands of trees. An added advantage of this model is that it does random sampling of the training data points when building the trees (Out of bag sampling). The random forest uses bootstrapping to train each tree on different subsets of the sample, then the test prediction is made by averaging the predictions from all of the individual trees. This concept is also called 'bagging', and all the trees are computed independently but in parallel. This addresses the issues with a single decision tree, as in random forest each single tree might have high variance but overall the forest will have lower variance.

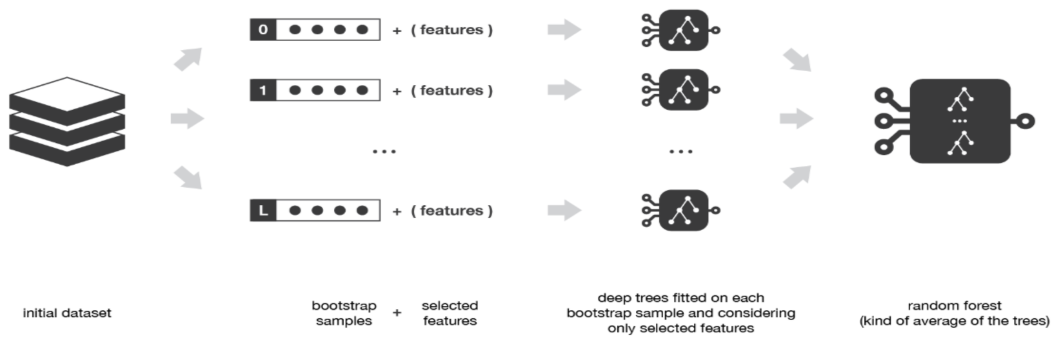
Additionally, it also uses random subsets of variables in the trees for splitting the nodes. Therefore the process of finding the root node and splitting the feature nodes will run randomly. The nodes are split on similar criteria as in the individual decision tree described above.

How the algorithm works:

The random forest algorithm has two stages: the first one is random forest creation. First the algorithm randomly selects a random sample of training points and a subset of features from the total number of features. Then, within the subset of features it finds the best split point amongst the features and then using this feature splits the node into daughter nodes. There is a random selection of a subset of predictors at each split. These steps are repeated until a certain number of nodes have been reached.

The algorithm repeats this entire process for each tree – from selecting a random subset of training data and features to splitting nodes based on best split point. Hundreds or thousands of tree are created to form the forest.

The next step is to make predictions on the test data. The algorithm takes the test features and uses the rules of each decision tree in the forest to predict the outcome for an observation. It stores the predicted outcome (target) from all of these individual decision trees and then calculates the votes for each predicted target. In classification, the predicted target that has the most votes (according to the majority of trees) is considered the final prediction for that test observation. In regression, the results from all individual trees are averaged to give the final prediction.


0 + (features)
1 + (features)
L + (features)
initial dataset
bootstrap samples + selected features
deep trees fitted on each bootstrap sample and considering only selected features
random forest (kind of average of the trees)

Advantages:

- Random forest can be used for both classification and regression
- By using bagging, random forests can decrease variance. They can also decrease overfitting on training data. Additionally, bagging also increases the stability and accuracy of the model.
- Random Forest can handle missing values as there is random sampling of features – only features where data is not missing can be considered
- Random Forest classifier can be modeled for categorical values
- The algorithm can be used for identifying the most important features from the training dataset, therefore it is helpful for feature engineering. It can handle large data sets with higher dimensionality and thousands of variables. Despite this it can identity most significant variables so it can also be used as a dimensionality reduction method.

Disadvantages:

- Interpretability is a problem as there is very little control on what the model does (lots of random selection). When there is a large collection of decision trees it is hard to have an intuitive grasp of the relationships existing in the input data. Different parameters can be tried and a good amount of time will have to be spent on hyperparameter tuning
- It is more complex than decision trees which makes it harder and more time consuming to run, along with a higher computational cost
- Overfitting can easily occur
- Need to choose the number of trees
- Performs better on classification problems compared to regression problems as it does not give precise continuous prediction. For regression, it cannot predict beyond the range in the training data, so there is a chance of overfitting if there is a lot of noise in the data.

4. Gradient Boosting

Gradient boosting is a machine-learning algorithm that can be used for both regression and classification problems. It is a supervised learning algorithm.

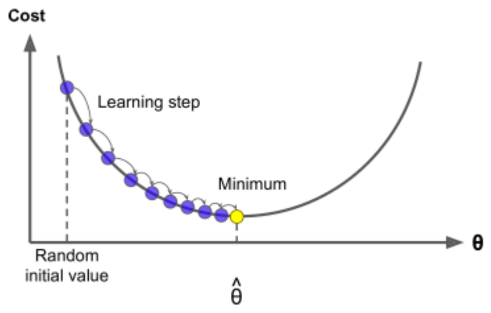
The concept of gradient boosting is similar to bagging in random forest, but works differently. In boosting the individual decision trees are run sequentially, not in parallel. The main aim is to reduce bias and it does this by running the decision trees iteratively – the fit of an individual tree at any step will depend on the trees fitted at the previous steps. Each model in the sequence will be fitted by giving more importance to the train observations that were badly handled by the previous models in the sequence. Therefore, the learning has more focus (or weight) on the most difficult observations.

Gradient boosting creates an ensemble of decision trees and uses boosting to fit them.

How the algorithm works

The algorithm of gradient boosting tries to minimize errors by sequentially fitting trees and giving the errors more weight in the next iteration. In this way it aims to minimize the mean squared error (MSE) loss function for regression. However, it can also work with other loss functions such as mean absolute error (MAE) for classification.

The reason it can work with different loss functions is that it is a gradient descent algorithm. Gradient descent is an optimization algorithm which can help find the optimal solution to a variety of problems. Parameters can adjusted iteratively in order to minimize any cost function. First it measures the local gradient of the loss function (assuming a given set of parameters). Once this is identified it takes steps in the direction of the descending gradient. The size of the steps taken is determined by the 'learning rate'. If the learning rate is a trade-off: if it is too low, then many more iterations will be required to find the minimum, if it is too high then the algorithm may jump over the minimum without finding. The minimum loss is when the gradient is zero.



Since the cost functions may not always be convex in shape stochastic gradient descent can be used. This is represented by the subsample parameter (if < 1 that means it is being used as less than 100% of train data is used in each subset). It samples a subset of the training data without replacement and grows the next tree using that subset. This makes the algorithm faster and adds some random nature in descending the loss function gradient. This can help the algorithm get near the global minimum.

Advantages:

- Gradient Boosting has great predictive accuracy

- The algorithm is very flexible – there are many different hyperparameter tuning options that can be adjusted by the user. There are also options for different loss functions that the model can optimize on.
- Can be used for classification and regression
- No data pre-processing required, works well with categorical and numerical values
- Handles missing data

Disadvantages:

- Gradient Boosting models will continue to improve as long as errors can be minimized. As the observations with errors are given more importance in the next iteration this can lead to overfitting on training data
- Involves the creation of hundreds or thousands of trees. This can take up a lot of time and memory and can be computationally expensive
- A a large grid search is required for tuning all the parameters – this also takes a lot of time
- The output is less interpretable compared to simpler models such as the decision tree

5. SVM – Support Vector Machine

SVM is also a supervised learning algorithm that can be used for both classification (mainly used for this) and regression problems.

SVM algorithm is most commonly used for classification problems. It is fast and can perform well with a limited amount of data. Each observation is plotted as a point in a n-dimensional space (where n is number of features). The value of each feature is represented by the value of the coordinate. To do classification SVM then tries to find the hyper-plane that differentiates the two classes as well as possible.

The SVM classifier is the hyperplane that best separates the two classes in the feature space (the optimal hyperplane). The support vectors are the co-ordinates of individual observations. Support vectors are the data points that are closest to the decision surface (or hyperplane separating the classes). They are therefore the most difficult data points to classify and have a direct influence on the optimal location of the hyperplane. They can change the position of the dividing hyperplane if removed.

How the algorithm works:

Once the hyperplane is created based on what side of the hyperplane they are on. The function of the hyperplane is:

$$f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

Observations are then classified based on what side of the hyperplane they are on. If there are two classes [1 ; -1] and the function for the hyperplane is >0 for an observation it would be on one side of the hyperplane (therefore in a certain class – class 1), if the function is < 0 it would be on the other side (other class – class -1). If the function = 0 then the point is on the hyperplane.

SVM tries to maximize the margin (distance from hyperplane to closest observation) around the hyperplane. The function to do this is specified by the support vectors (a subset of points close to the hyperplane). It is a quadratic optimization problem – the aim being to maximize the margin.

SVM gives a weight for each feature, the linear combination of these predicts the value of a new observation. The process of optimizating the maximum margin helps to reduce the number of weights that are nonzero to just a few that are for the most important features.

Finding the optimal hyperplane is therefore an optimization problem. The optimization algorithm to generate the weights is such that only support vectors determine the weights and the boundary. The optimal hyperplane is the one that has the largest margin. A regularization parameter 'C' can be used, if the value of C is large that means there is a high tolerance for being wrong, while a small C represents low tolerance for error. The soft margin is created by the support vector classifier, and the hard margin is created by the Maximal Margin Classifier.

The concepts above apply for linear SVMs, however SVM can also be non-linear, as a linear SVM may not be able to properly separate the data. For these the kernel can be polynomial instead of linear. The concept is to enlarge the space of features by including transformations, this increases the dimensions. Then the optimal hyperplane is found in the enlarged space. The data is projected from the enlarged space back to original space to find the decision boundary – this results in non-linear decision boundaries in the original space. This process also increases the number of features as polynomials of the original features are added.

SVM also has an option for a radial kernel. It is in an implicit feature space that is high dimensional. It involves a 'gamma' parameter which is a positive constant that represents the size of the radial kernel. If the size of the kernel (gamma) is increased it reduces bias but increases variance, if gamma is decreased it increases bias and reduces variance.

Advantages:

- SVM works well if there is clear margin of separation between the classes
- SVM is more effective in high dimensional spaces (eg sentiment analysis). It is also effective in cases where the number of features is greater than the number of samples
- It takes less time and memory to compute compared to other more complex algorithms
- There is a kernel available to handle non-linearity
- Can work with image data as well

Disadvantages:

- It is not suitable for very large data sets – becomes very computationally expensive
- If the target classes are not clearly separated, or if there is a lot of noise in the data SVM cannot perform well
- The kernel is very sensitive to the parameters selected therefore the model has to be tuned correctly to get the best results
- The optimal kernel has to be selected for the best results

- It gives a non-direct probability prediction – cannot give class probabilities, as it works by putting data points above and below the hyperplane, there is no probabilistic explanation for the classification
- It can lead to overfitting if the number of features is much greater than the number of samples
- It is more complex, and more difficult to explain and understand compared to simpler models like the decision tree

Task 2: Setup the benchmark experiment to compare the 5 selected machine learning algorithms. Describe your setup and the result (e.g. cross-validation method (holdout, k-fold CV, etc.), evaluation metric (AUC, Accuracy, etc.), hyperparameter tuning, variable selection, resampling method (over-sampling, undersampling, etc.), etc.).

For this assignment, I set up two separate experiments to compare the 5 machine learning algorithms I described above. The data cleaning and data processing remained the same for both experiments. Initially feature engineering was done after looking at the variable importance from random forest. After processing the variables different variable selection methods were tried for both cases.

- **Variable selection**

The first one was the Fisher Score which assigns each independent variable a value based on the extent to which they assign similar values to instances in the same class and different values to instances from different classes. The second method tried was stepwise regression (using 'both' – forward and backward), which works by comparing the AIC improvement from dropping and/or adding one variable at each 'step'. It tries to find the most important variables in this way by looking at the best AIC improvement (smallest AIC). For both experiments I tried different numbers of input variables including 20, 30 and 50 from Fisher score and the 23 variables given by stepwise regression.

Among all these combinations the 50 variables selected by Fisher score gave the best results in the models. Additionally I also tried to remove the highly correlated variables (>0.75) before the feature selection to check if this would have any impact on the models performance, however the impact of removing these features was minimal.

Experiment Set Up 1:

Cross validation and Hyper-parameter tuning

For this experiment I used 3-fold or 5-fold cross validation for all of the models. In some cases 3 fold CV was used instead of 5 fold as the models were taking too long to run.

Different hyperparameters were tuned in each model. Since many hyperparamters were tuned in some models, the tune control random was used with between 50 to 150 iteration as a tune control grid was taking too long to run. In models where only a few hyperparameters were tuned (such as SVM) a grid search was used instead of random search. The table below gives more detail on the tuning:

Model	Hyperparameters Tuned

Logistic regression	None
Random Forest	• ntree (values between 100 to 1000 tried) - this hyperparameter refers to the number of trees that should be grown in the forest • mtry (values between 2 and 14 tried) – this hyperparameter refers to the number of variables randomly sampled as candidates for splitting at each node
Gradient Boosting	• Distribution: Bernoulli (logistic regression for 0 1 outcomes) • n.trees (values between 100 and 1000 tried) – represents the total number of trees to be fit/total iterations • interaction depth (values between 2 and 10 tried) – refers to the depth of the tree, smaller depths are more efficient but larger depths allow the capture of unique interactions but also increase chances for overfitting • n.minobsinnode (values between 10 and 80 tried) – refers to the minimum number of observations that should be in each terminal node of the tree, controls the complexity of each tree, higher values can prevent overfitting but lower values are good for imbalanced target classes • shrinkage (values between 0.01 to 0.2 tried) – refers to the 'learning rate', determines the contribution of each tree on the final outcome and controls how quickly the algorithm proceeds down the gradient descent. Smaller values increase model accuracy but require more trees in the sequence • bag.fraction (values between 0.1 to 1 tried) – the proportion of the training set observations that are randomly selected for the next tree in the expansion
XGBoost	• objective: binary logistic • nrounds (values between 200 to 600 tried) – this represents the number of trees to grow/iterations • max_depth (between 3 and 10 tried) – refers to the max depth of a tree, same as gradient boosting. Used to control

	over-fitting as higher depth will allow model to learn relations very specific to a particular sample. • Lambda (between 0.55 and 0.6 tried) – similar to ridge regression, determines L2 regularization on weights • Eta (values between 0.001 and 0.5 tried) – similar to the learning rate as in gradient boosting. It can make the model more robust by shrinking weights on each step • Subsample (values between 0.1 and 0.8 tried) - Same as the subsample of gradient boosting it is the proportion of training data to be randomly sampled for each tree. Lower values can make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting. • min_child_weight (values between 1 to 5 tried) – defines the minimum sum of weights of all observations required in a child. High values can help avoid overfitting, however if values are too high it can lead to underfitting • colsample_bytree (values between 0.2 and 0.8 tried) – It represents the proportion of columns/features to be randomly sampled for each tree
SVM	• Kernal: tried different kernals and linear gave the best results • C – this is the penalty parameter of the error term. It controls the trade-off between a smooth decision boundary and classifying all the training points correctly • Sigma – with a larger sigma, the decision tends to be flexible and smooth. It can make wrong classification when predicting but also avoids overfitting. With a smaller sigma there is higher chance of overfitting

Evaluation Metric and Results

The evaluation metric used for this experiment was AUC. AUC was selected due to its popularity as an evaluation metric for binary classification problems. The models output a probability value for subscribe and then a threshold has to be set (typically at 0.5) – if the probability for an observation is above the threshold it is considered a 1 (subscribe) or if below the threshold it is considered 0 (non-subscribe). An advantage of using AUC is that it considers all possible thresholds and computes the true positive rate and false positive rate at all of them. For a random classifier the AUC score is 0.5 and for a perfect classifier it is 1. Therefore we can compare the models based on how high the AUC score is on the validation and test sets.

	Logistic Regression	Random Forest	Gradient Boosting	XG Boost	SVM
AUC on Test Set	83.1%	75.4%	82.1%	81%	65%

Within this set up the best performing models (based on test set AUC) were logistic regression and gradient boosting. The worst performing model was SVM.

I also wanted to try to reample the data to get a better balance between the subscribe and non-subscribe classes, however I could not use cross-validation with this approach. This is because after resampling the rows with target 1 would be increased in the data and there would be repeats of them. If k-fold CV was used with this approach it could mean that the same rows could be in different folds of the CV (train and test) and because of this the model would get the chance to learn from rows it was supposed to test on. This would lead to information leakage.

Due to this I decided to set up another experiment without k-fold cross validation and hyper parameter tuning, This experiment is described below.

Experiment Set Up 2:

Cross validation and Hyper-parameter tuning

In this experiment I did not use k-fold CV, instead I used the holdout cross validation. This was done by training models on the train set (70% of the data) and testing them on the validation set (15% of the data). Only manual hyperparameter tuning was done by exploring different values for a few parameters in each model and selecting the best ones based on the results from the validation set.

At the end, when all adjustments to the models had been made and the best hyperparameters had been found and incorporated the models were tested on the test et (15% of the data). Only the models that had the best results on the test set were used to predict on the test_holdout set.

I decided to move forward with this approach as there was less overfitting and results on test set were more consistent.

Resampling (using SMOTE)

10 different models were run – 2 each for each of the 5 algorithms (logistic regression, decision tree, gradient boosting, random forest and SVM).

The first model run for each of these was without resampling the data. This was considered as a benchmark model to check if resampling the data would give different results to this benchmark, and if so, how different?

Next, the train set (70% of the data) was resampled using the SMOTE technique. This technique combines the under-sampling of the majority class (0 – non-subscribe) and over-sampling of the minority class (1 – subscribe). Before resampling with SMOTE there were 4329 observations of class 0 and 571 observations of class 1. After resampling with SMOTE there were 2284 observations of class 0 and 1713 observations of class 1 – this makes the target distribution more balanced.

Evaluation Metrics

The evaluation metrics used for all 10 models were AUC and accuracy. The advantage of AUC, as described before, is that it considers all possible thresholds and computes the true positive rate and false positive rate at all of them. The advantage of using accuracy is that it is a commonly used metric that is very simple to understand as it just looks at what proportion of observations are predicted correctly.

Results

The results of the 10 models are given below.

Logistic regression

RESULTS ON TEST SET	Logistic Regression without resampling (benchmark)	Logistic Regression with resampling
Accuracy	88.8%	84.2%
AUC	83.1%	82.6%

The results from logistic regression show that resampling the training data led to a decrease in AUC and Accuracy. The models performance also reduced if the number of variables included were decreased from 50 to 30 or 20. The models performance also decreased when the 23 variables from stepwise regression output were used instead of the 50 features selected based on Fisher score.

Decision Tree

RESULTS ON TEST SET	Decision Tree without resampling (benchmark)	Decision Tree with resampling
Accuracy	88.8%	84.6%
AUC	78.8%	81.8%

The results from the decision tree models show that while the AUC improved after resampling the data using SMOTE, the accuracy of the model decreased, compared to the benchmark. The models performance also reduced if the number of variables included were decreased from 50 to 30 or 20. The models performance also decreased when the 23 variables from stepwise regression output were used instead of the 50 features selected based on Fisher score.

Random Forest

RESULTS ON TEST SET	Random Forest without resampling (benchmark)	Random Forest with resampling
Accuracy	89%	84.2%
AUC	76.9%	80.3%

The results from the random forest models show that while the AUC improved after resampling the data using SMOTE, the accuracy of the model decreased, compared to the benchmark. The models performance also reduced if the number of variables included were decreased from 50 to 30 or 20. However, both models performance increased when the 23 variables from stepwise regression output were used instead of the 50 features selected based on Fisher score.

Gradient Boosting

RESULTS ON TEST SET	Gradient Boosting without resampling (benchmark)	Gradient Boosting with resampling
Accuracy	89.3%	86.5%
AUC	82%	82.2%

The AUC of the gradient boosting model did not improve much after resampling the data, however the accuracy decreased compared to the benchmark. The models performance also reduced if the number of variables included were decreased from 50 to 30 or 20. The models performance also decreased when the 23 variables from stepwise regression output were used instead of the 50 features selected based on Fisher score.

SVM

RESULTS ON TEST SET	SVM without resampling (benchmark)	SVM with resampling
Accuracy	88.4%	84.6%
AUC	65.2%	79%

Based on the results Logistic Regression and Gradient Boosting perform the best on the test set, without resampling the data. Resampling the data is leading to some level of overfitting which is why the accuracy drops for models trained on resampled data.