Assignment Report - Microcredentials UTS Advanced Analytics

Kalyanaraman (Kesh) Kshetrapalapuram (Telstra)

Introduction

The Problem, Inputs & Outputs

The assignment task was to use the provided training dataset to build a robust classifier (classification model) that can accurately predict the QUALIFIED target attribute (binary 0 meaning not qualified or 1 meaning qualified) on unseen data. The ask was to then use the model to predict the QUALIFIED target attribute for the provided test dataset, and write a report summarising the approach, results, and a discussion on learnings.

From the descriptions of the attributes provided, it seems the dataset represents property data, where we are likely attempting to understand when a potential sale is qualified.

Input

A training dataset with examples (rows/records) and attributes (columns / features), along with the target attribute (in this case QUALIFIED).

The training (Assignment-HousingDataset.csv) and testing (Assignment-UnknownDataset.csv) csv files were downloaded from the UTS Canvas site (https://canvas.open.uts.edu.au/courses/973/assignments).

Outputs

- The “best” classification model that can be used to predict the QUALIFIED attribute
- The prediction (QUALIFIED attribute) for the provided test examples
- This report outlining approach, configuration, results and learnings (this report)

While this report does not show source R code, the full code is available from here: https://github.com/kaesava/UTS_Micro_AA/blob/main/assignment.Rmd. An exploration R file is also available here: https://github.com/kaesava/UTS_Micro_AA/blob/main/explore.R

High-level Approach

I decided to use R, so I can learn another programming language, one that seems to be popular among data scientists. I installed the latest versions of R (v 4.0.3) and R Studio

(v1.3.1093), and wrote this assignment as an R Notebook that can be easily compiled into Word and/or PDF.

A a high level, I followed these steps:

- I prepared the R environment by loading the packages I needed.
- I loaded the provided training and testing datasets and explored the data using summary statistics and visualisations to learn more about each attribute in the training dataset (missing values, distribution, distribution relative to target attribute, etc.).
- I split the training data into training (90%) and validation (10%), so that performance measures are reported on new (unseen) data.
- I pre-processed the data by removing redundant attributes, imputing missing values, identifying and removing highly correlated numeric and categorical attributes, testing for outliers, creating new calculated attributes (like date durations), one-hot-encoding categorical attributes and normalising/binning numeric attributes as needed. I found that this step was highly iterative. I built a "reference" random forest model (with default hyper-parameters), and depending on the accuracy (f-score) of the resulting classifier, I re-visited and tuned each pre-processing step several times.
- I ran feature/attribute selection algorithms to determine importance and removed attributes that didn't contribute to the classification.
- I balanced the training dataset so that we had the same number of examples in the 0 and 1 target attributes
- I setup the training (classification) model parameters (like 10-fold cross validation). The random seed for every run of every training algorithm was set to a static number (3433) to minimise uncertainty through randomness.
- I ran 7 different classification algorithms (decision tree, k nearest neighbour, random forest, GBM, SVM, neural network and an ensemble model). For each classifier, I experimented with various hyper-parameters (tuning) to determine the best performing one. I collected several accuracy measures (accuracy, f-score, AUC, time to run).
- I determined the best classifier based on various criteria (accuracy, time to run, etc.) and applied it to the testing dataset, submitting the results to kaggle.

Summary of Results

The GBM classifier was selected. The ideal hyper-parameters used were: n.trees = 450, interaction.depth = 10, shrinkage = 0.1, n.minobsinnode = 10 on full training set.

- Accuracy: 90.83%
- f-score: 90.18%
- AUC: 96.04%
- Time to run: ~ 3 hours

I've outlined reasons for selecting it in the Selecting the best model section below.

Data Exploration, Pre-processing and Attribute Transformation

Environment setup

I loaded R libraries that I needed: like caret (for streamlining model training processes), and ggplot2(for visualisation). ~25 packages were used.

Data Load

The training (Assignment-HousingDataset.csv) and testing (Assignment-UnknownDataset.csv) csv files were downloaded from the UTS Canvas site (https://canvas.open.uts.edu.au/courses/973/assignments) and loaded into R.

They were comma separated, with a header row, and had missing values left blank. I used the read.csv function to read in the csv files.

The training dataset had 75,007 examples and 38 attributes (including the target attribute QUALIFIED). The testing dataset had 32,147 examples and 37 attributes (it did not include the target attribute). The attributes were visually examined. They included strings, dates and numbers, and had attributes with missing values.

A brief explanation of each attribute is available from the UTS Canvas site (https://canvas.open.uts.edu.au/courses/973/assignments/2644) under the Dataset section.

Data Exploration & Pre-Processing

Using a Reference model

A reference random forest model with 10-fold cross validation was used to test whether to (and how to) cleanse and transform attributes. If performance (f-score) deteriorated after a pre-processing step, the transformation was rolled back. Default hyper-parameters and a static random seed were used to minimise variation due to randomness. The validation set was different from the training set used.

Further, all transformation applied to the training dataset were also applied to the validation and testing datasets.

Setting the target attribute as categorical

The target QUALIFIED attribute was converted to a categorical attribute (known as a factor in R), as this is a classification problem (not a regression one).

Dropping attributes that add no value

The following attributes were dropped:

- Row ID: It is a unique arbitrary row identifier and therefore should not participate in prediction
- GIS_LAST_MOD_DTTM: Every example had the same value; it therefore cannot play a role in discriminating between the target attribute levels
- HEAT, STYLE, STRUCT, GRADE, CNDTN, EXTWALL, ROOF, and INTWALL: Each of these descriptive attributes are already coded for by another attribute; the 1:1 between the code and description for each of these attributes was tested before they were dropped.

Splitting the data into Training & Validation

90% of the data was used for training (randomly sampled), and 10% kept aside for validation. This was done to ensure that when we compare the performance of different classification algorithms (or even variants of an algorithm with different hyper-parameters), it is done on a dataset never seen before, to avoid the effects of over-fitting (high variance).

Handling Date attributes

Date fields were first cleansed:

- The SALEDATE was read in as a string, but converted to a date. There were ~15,885 examples with value 01/01/1900. The initial strategy for this field was to mark these as missing (likely to be 0) and then impute by categorising (binning) the date attribute and creating an additional NA category level. But this caused a reduction in both accuracy and the f-score, likely because the interpretation of 1900 as missing may not have been correct. This was therefore reverted.

- The YR_RMDL attribute (year or re-model) had 36,456 missing values (more than half) in the training dataset, suggesting that being missing may itself be important (it likely indicates no property re-modelling). Therefore, instead of imputing with median or mean, these were left as missing and treated (binned) as outlined in the next section. The attribute also included invalid values like 20 and so, any year that was less than 1750 (most likely errors) was set a missing.

- There were two date-year attributes (AYB and EYB) denoting year of build and improvement. They included a handful of invalid (for a year) values like 0 and 20. These, along with missing values (less than ~10) were imputed with the median (median was used instead of mean to eliminate the effects of extremes).

In addition to the date columns, attributes for durations between these dates were tested and found to be useful. Note that it may result in negative numbers too.

- AYB_TO_EYB was calculated as EYB minus AYB
- AYB_TO_SALEDATE was calculated as SALEDATE minus AYB
- AYB_TO_YR_RMDL was calculated as YR_RMDL minus AYB
- EYB_TO_YR_RMDL was calculated as YR_RMDL minus EYB
- EYB_TO_SALEDATE was calculated as SALEDATE minus EYB

- YR_RMDL_TO_SALEDATE was calculated as SALEDATE minus YR_RMDL

The YR_RMDL attribute (and duration attributes that include the YR_RMDL attribute) were then converted to categorical attributes using the optbin function in the BBMisc package. This function finds bin (category) boundaries that maximise prediction of the target attribute (i.e., provide the highest information gain). The missing values were coded with the NA label. The corresponding attributes in the validation and test dataset were also categorised using the same bin boundaries as those in the training dataset.

Due to the large number of missing values, this method produced better accuracy (both accuracy and f-score) compared with raw values with NAs imputed with the median. A more complex strategy for binning (built decision trees that predicted QUALIFIED based on these attributes, and used node decision points to create bins) was attempted, but this produced no better results.

Removing Highly Correlated Date Attributes

Date attributes were tested for pair-wise relatedness using Goodman and Kruskal's tau measure. The measure captures the degree of difference in one attribute that can be explained by another attribute. It is asymmetric (so the influence on x by y is not the same as y on x), which makes it useful in determining which attributes are redundant and can be dropped.

	AYB	YR_RMDL	EYB	SALEDATE	AYB_TO_EYB	AYB_TO_SALEDATE	AYB_TO_YR_RMDL	EYB_TO_YR_RMDL	EYB_TO_SALEDATE	YR_RMDL_TO_SALEDATE
AYB	x	0.077	0.093	0.007	0.129	0.082	0.167	0.088	0.022	0.094
YR_RMDL	0.002	x	0.024	0.011	0.004	0.001	0.689	0.743	0.004	0.818
EYB	0.069	0.273	x	0.017	0.073	0.02	0.252	0.283	0.083	0.272
SALEDATE	0.102	0.178	0.11	x	0.103	0.112	0.154	0.155	0.144	0.242
AYB_TO_EYB	0.144	0.187	0.112	0.009	x	0.027	0.275	0.19	0.019	0.216
AYB_TO_SALEDATE	0.265	0.13	0.048	0.166	0.061	x	0.217	0.126	0.059	0.23
AYB_TO_YR_RMDL	0.006	0.718	0.021	0.008	0.007	0.003	x	0.678	0.003	0.809
EYB_TO_YR_RMDL	0.003	0.757	0.024	0.008	0.004	0.001	0.665	x	0.005	0.807
EYB_TO_SALEDATE	0.027	0.162	0.28	0.177	0.037	0.042	0.139	0.184	x	0.256
YR_RMDL_TO_SALEDATE	0.002	0.687	0.017	0.055	0.004	0.003	0.638	0.65	0.007	x

Ignoring the main diagonal (correlation with self), the table shows that: * YR_RMDL is a strong predictor of YR_RMDL_TO_SALEDATE * AYB_TO_YR_RMDL is a strong predictor of YR_RMDL and YR_RMDL_TO_SALE_DATE * EYB_TO_YR_RMDL is a strong predictor of YR_RMDL and YR_RMDL_TO_SALE_DATE

Therefore, the following attributes were removed from the analysis, as they are accounted for by other attributes: YR_RMDL and YR_RMDL_TO_SALEDATE. A validation on the reference model showed no drop in accuracy.

Inputing Missing values for Numerical attributes

The following attributes had between 20 and 50 examples with missing values: BATHRM, HF_BATHRM, NUM_UNITS, ROOMS, BEDRM, STORIES, KITCHENS, and FIREPLACES. As most examples with these missing values had QUALIFIED set to 0, the missing values were replaced by the median (as opposed to mean to avoid the influence of outliers) of examples where QUALIFIED was 0. The same transformation was applied to the validation and testing datasets too.

These numerical attributes had no missing values: SALE_NUM, BLDG_NUM, PRICE, GBA, LANDAREA.

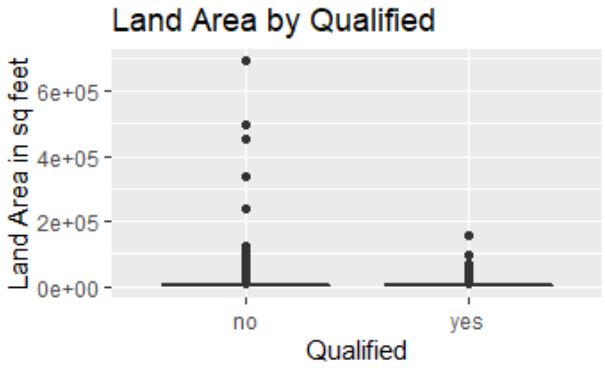
Visualising Numerical attributes

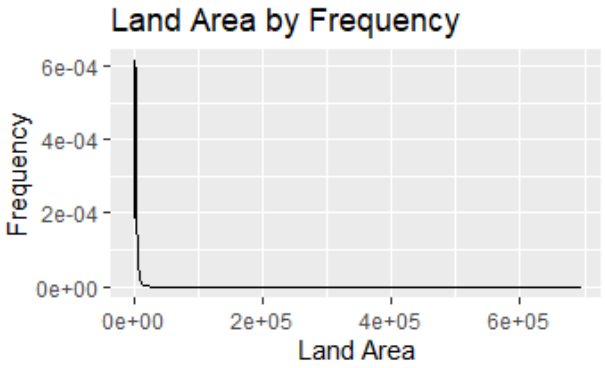
The following numeric attributes were explored: BATHRM, HF_BATHRM, NUM_UNITS, ROOMS, BEDRM, STORIES, SALE_NUM, BLDG_NUM, KITCHENS, FIREPLACES, PRICE, GBA, LANDAREA.

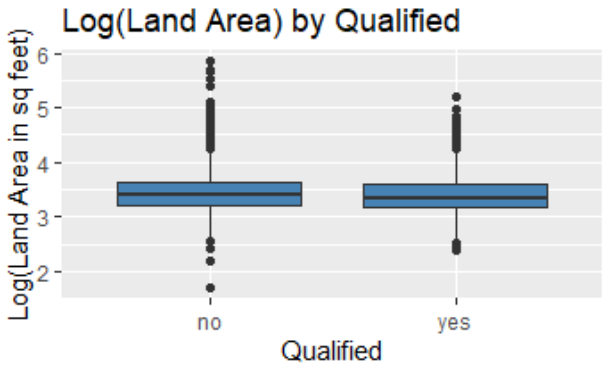
Summary statistics (including number of missing values, mean, standard deviation, and quartiles - 0th, 25th, 50th, 75th and 100th percentiles) were reviewed.

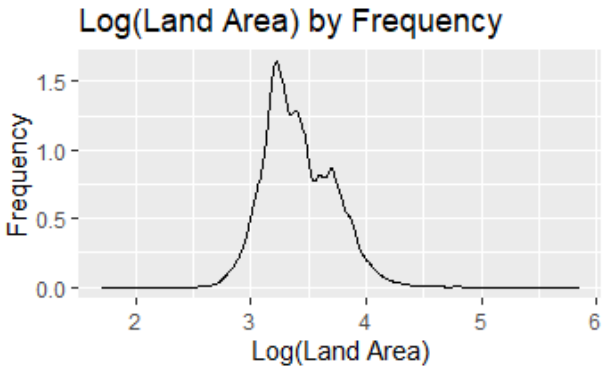
```
-- Data Summary ------------------------
                           Values
Name                       train[, c(numeric_attrs, ...
Number of rows             67507
Number of columns          13
___________________________
Column type frequency:
  numeric                  13
___________________________
Group variables            None

-- Variable type: numeric --------------------------------------------------------------------------------
# A tibble: 13 x 10
   skim_variable n_missing complete_rate       mean          sd    p0   p25    p50    p75      p100
 * <chr>             <int>         <dbl>      <dbl>       <dbl> <dbl> <dbl>  <dbl>  <dbl>     <dbl>
 1 BATHRM                0             1       2.03        1.06      0     1      2      3        24
 2 HF_BATHRM             0             1      0.609       0.618      0     0      1      1        11
 3 NUM_UNITS             0             1       1.19       0.591      0     1      1      1         6
 4 ROOMS                 0             1       7.36        2.34      0     6      7      8       101
 5 BEDRM                 0             1       3.37        1.17      0     3      3      4        54
 6 STORIES               0             1       2.09        3.35      0     2      2      2       826
 7 SALE_NUM              0             1       1.62        1.28      1     1      1      1        15
 8 BLDG_NUM              0             1       1.00      0.0373      1     1      1      1         5
 9 KITCHENS              0             1       1.22       0.624      0     1      1      1        44
10 FIREPLACES            0             1      0.619       0.887      0     0      0      1        13
11 GBA                   0             1      1714.        861.      0  1194   1480   1965     20948
12 LANDAREA              0             1      3412.       5502.      0  1600   2368   4195    691817
13 PRICE             12134         0.820 383106.     569586.       0     0 236000 580000 25000000
```

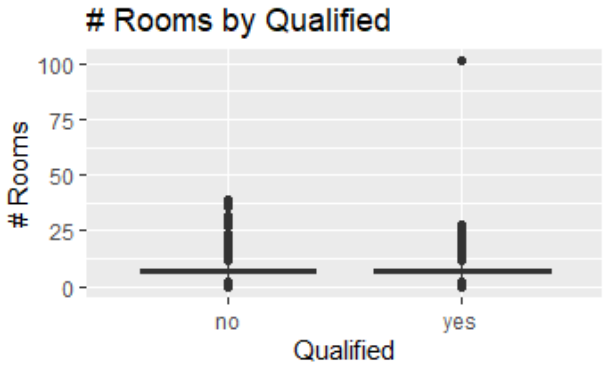
It is clear that most of these attributes are skewed heavily. Various visualisations of each the numeric attributes were explored. An example of a box chart and frequency chart of Land Area and ROOMS is shown below. For visualisation, the raw attribute as well as the log (base 10) of the attribute were charted; the log was used as there was a lot of skewness in the data (i.e., there are very high numbers with most in a smaller range). These visualisations helped guide the missing value treatment and other transformations of these attributes, as outlined below.

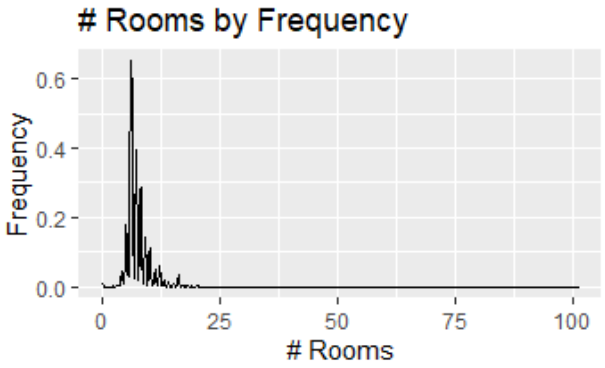


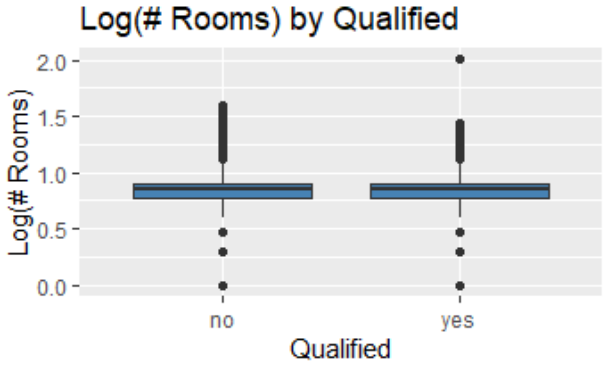


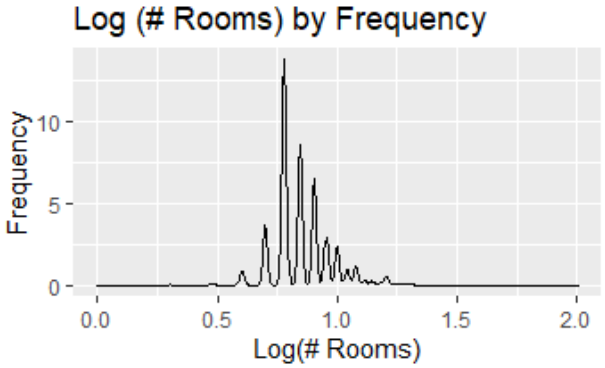












Based on this skew, the numerical attributes were converted to the logarithm (base 10) of these attributes (except SALE_NUM and BLDG_NUM - which didn't have the skew). Given that log (base 10) of 0 is negative infinity, these were replaced with 0. However,

the accuracy and f-score was tested on the reference model, and found to be slightly worse. Therefore, this transformation was not used.

Removing outliers in Numerical attributes

Consideration was given to removing outliers (for example ROOMS has one example with 101 rooms, with the second-highest being 39). To test if a value was an outlier was to test if it was higher (or lower) than 3 standard deviations from the mean. Another method was also tried, where the value was an outlier if it was more than 1.5 times the inter-quartile range (75th minus 25th percentiles) higher than (or lower than) the 75th (or 25th) percentiles respectively.

However, in both cases, not only did removing the outliers make no difference (or in some cases perform worse) on the reference model, it is also possible that there are in fact properties with 100 rooms (as an example). With no more domain knowledge, it felt dangerous to remove them (for example, the testing data might have these), and so outliers were not removed.

Handling ordered categorical attributes as numeric attributes

The Condition and Grade attributes came through as categorical, but were converted to numeric, so their "order" was captured. For example, CONDITION = Excellent (coded as 6) is quantatively better than CONDITION = Poor (coded as 1). They were already coded from worst to best (numbers increasing). They were then normalised (scaled) them to 0 to 1 to ensure comparable contribution with other numerical attributes. 33 of 35 missing values in the training dataset had a target QUALIFIED value 0. Therefore, the missing values were replaced with the median of the training dataset where QUALIFIED = 0.

Calculating Price per square foot

Given this is property data and we have the PRICE and GBA (Gross building area in square feet), it seemed useful to include the Price per square foot.

Imputing the PRICE and PRICE_BY_GBA Attributes

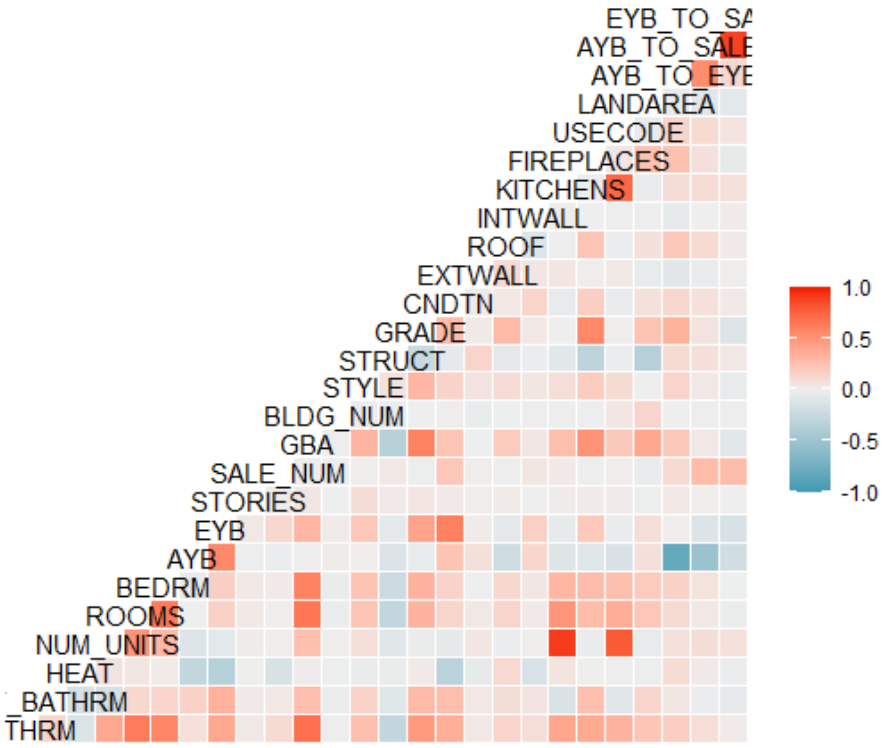
The Price and PRICE_BY_GBA attributes had ~12,134 missing values and ~18,696 with PRICE values under $1000 (mainly $0). Different approaches to imputing missing values were taken:

- Missing values were replaced with zero
- Missing values were replaced with the median
- The attribute was discretised (converted into categories/bins) using a decision tree. The decision tree provided the bin boundaries that maximised prediction of the target attribute

Of all these approaches, the last one provided the best accuracy & f-score, and was therefore used. This attribute seems to be a strong predictor of QUALIFIED, almost 88+% accuracy just with this one attribute, hence the attention paid to various ways of treating this attribute.

Removing Highly Correlated Numeric Attributes

Numeric attributes were tested for pair-wise correlation (relatedness). When two attributes are highly correlated, including them adds little value (extra information) to the classification and in fact can cause un-necessary computation. The findCorrelation method was used to find pair-wise correlation of each numeric attribute in the training dataset which calculates the co-variance between them. A colour-coded visualisation of the correlation (darker means higher correlation, blue means negative, red means positive) helps visualise this.



Based on the findCorrelation function with an absolute threshold of 0.8 (highly correlated), the following attributes were dropped: NUM_UNITS, AYB_TO_EYB, EYB_TO_SALEDATE.

- NUM_UNITS was highly correlated with INTWALL and FIREPLACES
- AYB_TO_SALEDATE was highly correlated with EYB_TO_SALEDATE
- AYB_TO_EYB was highly correlated with AYB.

Attributes that predict the target attribute poorer (checked against the reference model) were dropped (NUM_UNITS, AYB_TO_EYB and EYB_TO_SALEDATE).

Missing values in Categorical attributes

In addition to PRICE, PRICE_BY_GBA, AYB_TO_YR_RMDL, and EYB_TO_YR_RMDL, the following categorical attributes were found in the training dataset (excluding the target): HEAT, AC, STYLE, STRUCT, EXTWALL, ROOF, and INTWALL. Some of them came

through as integers, but were converted to categorical attributes (factors in R). All of them had (the same) 20 examples with missing values, and 19 of these had target QUALIFIED = 0.

While dropping them was considered, it was not possible to drop examples in the test set, so classifying them would have been a challenge. Therefore, the missing values were imputed with the most frequently occurring category corresponding to QUALIFIED = 0 in the training dataset for each of these attributes.

The USECODE attribute came through as a number with 9 unique values, and is simply described as a Property use code, indicating that the attribute might need to be treated as categorical However, when it was converted to a categorical attribute (factor in R), even though the accuracy improved (on the reference model), the f-score was worse. Further, it took significantly longer to run, compared to when it was numeric. Therefore, it was left as a numeric attribute.

Removing Highly Correlated Categorical Attributes

Categorical attributes were tested pair-wise for correlation (relatedness) using the Cramer'r V test (which calculates the level of association between categorical attributes between 0 and 1, the higher, the more related). The Cramer's phi from the top 3 are shown:

- EYB_TO_YR_RMDL & AYB_TO_YR_RMDL: 0.7400530
- PRICE_BY_GBA & PRICE: 0.7043760
- AC & HEAT: 0.5320308

Using a threshold of 0.7 (which was reasonable based on documentation found), AYB_TO_YR_RMDL and EYB_TO_YR_RMDL were highly correlated, and EYB_TO_YR_RMDL was dropped (less predictive using the reference model). Similarly, PRICE_BY_GBA and PRICE were highly correlation, and PRICE_PER_GBA was dropped (less predictive using the reference model).

One-hot-encode categorical attributes

One-hot-encoding is the splitting of a categorical attribute into multiple attributes, one for each factor level, with a 0 or 1 indicating whether that level was set for that attribute. The dummyVars function in R was used to convert categorical attributes to one-hot-encoded attributes.

The 9 categorical attributes resulted in 104 one-hot-encoded attributes since some categorical attributes had over 20 levels. This caused some algorithms (like Random Forest) to take a very long time (one run took over 24 hours with 10-fold cross validation). To reduce the number of attributes levels, the nearZeroVar in the MASS package was used to identify the one-hot-encoded categorical attributes that have a zero or near-zero variance (i.e., they stay “very” constant) - these are less likely to impact the classification outcome. In fact, dropping these attributes actually lead to an improvement in accuracy and f-score (when tested on the reference model), likely

because it was less prone to over-fitting to poorly predictive attributes. It reduced the number of categorical attributes from 104 to 27.

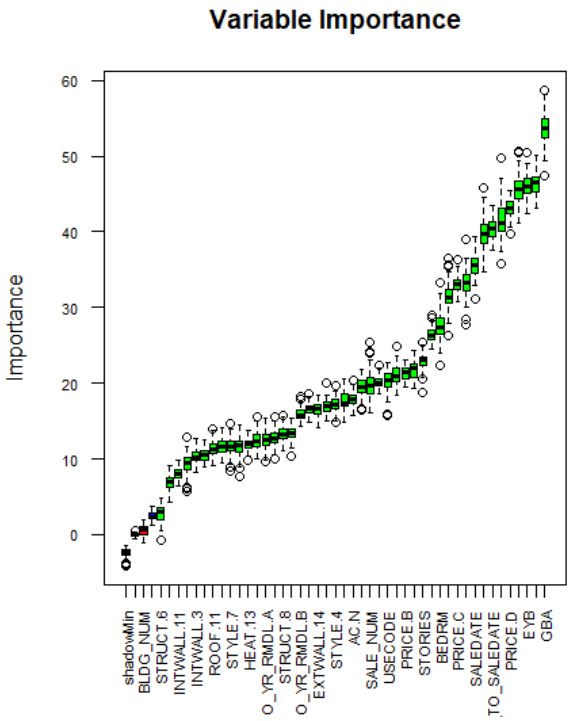
Attribute Importance

Importance is a measure of the predictive power of the attribute on the target attribute. Having attributes that don't contribute meaningfully causes un-necessary computation. Given that some algorithms were very expensive to run (where the complexity is a function of the number of attributes), removing ones that are not needed was important.

Forward (and backward) attribute selection algorithms let you select an optimal set of attributes starting with none (or all) of the attributes and building (or culling) them in steps. Here, backward selection was used using the glmStepAIC function in the MASS package that builds an underlying GLM (generalised linear model) model and keeps culling attributes using AIC - which is an estimate the relative information loss, as attributes are removed.

Another approach also taken, using the Baruta package in R to build 100 random forests and examine attributes used (and not used) to determine importance.

The top 3 most useful attributes in both cases were: PRICE, GBA, and CNDTN. It was re-assuting to see the level of agreement between the two approaches.



The following 16 attributes were found to be in the bottom 25 of both methods (when attributes ordered by importance descending), and were therefore dropped: HEAT.1, HEAT.7, HEAT.13, AC.N, AC.Y, STYLE.7, STRUCT.1, STRUCT.6, EXTWALL.22, ROOF.1,

ROOF.2, ROOF.6, ROOF.11, AYB_TO_YR_RMDL.B HF_BATHRM, SALE_NUM. Checks on the reference model suggested no noticable drop in performance (f-score).

Balancing the training dataset

In the training dataset, there were 29,090 examples with the target QUALIFIED attribute set to 1, and 38,417 examples with target set to 0. The dataset was therefore not fully balanced, it was skewed to value 0. I applied the SMOTE (Synthetic Minority Over-Sampling Technique) over-sampling technique to the 1 examples in the training dataset to generate “synthetic” minority-class examples (i.e. QUALIFIED=0) so that the training dataset ended up with the same number of 0 and 1 values in the target attribute. I chose not to under-sample the majority class so as not to lose examples. The parameter for the number of nearest neighbours used to generate “likedness” was set to the default 5.

The result was a balanced training dataset with the same number of examples with target class 0 and 1 (38,417). Having a balanced dataset makes it more reliable to interpret the accuracy measure.

Model Build & Accuracy Testing

Classification techniques used

The following classification techniques were used. Details of parameter settings are covered in each sub-section below.

- Decision Tree
- K-Nearest Neighbour
- Random Forest
- Gradient Boost (GBM)
- Support Vector Machines (SVM)
- Neural Network
- Ensemble (combination)

Training Setup

Accuracy Measures

The following measures were captured with a 10-fold cross-validation model build and prediction test on validation data:

- F1 (or f-score) - harmonic mean of the precision (proportion of positive predictions that were correct) and recall (proportion of positive examples that were correctly predicted), 1 implying perfect precision and recall.
- AUC (Area under the curve) - (0 to 1 - higher the better) is summary metric for the ROC curve and was reported. It captures the relation between true-positive rate and false positive rate.

- Time to run - elapsed time in hours (on my work laptop running R - single processor)

Note that without any further domain-specific understanding of the QUALIFIED target attribute, it was not clear whether to preference a false positive or false negative. However, given that the kaggle submission uses the F1 metric, it was decided that the F1 metric will be used. The caret package in R, uses accuracy by default, but allowed the use of a provided function to maximise. I coded the f-score (by calculating precision and recall and calculating the harmonic mean (twice the product divided by the sum)).

A combination of these measures were considered when determining the “best” model, as outlined in the sections below.

Cross-validation

All models were build with 10-fold cross-validation (which is considered good practice). Here, the model training process is repeated 10 times, each time with 9/10 of the data used for training and the remaining 1/10 for validation. Every example therefore participates in the testing dataset (reducing the potential of over-fitting to a sample that the model happens to do well on). The combined accuracy is reported and is more reliable than running the model training once. This way, the chance of the algorithm getting “lucky” with the training dataset and not being able to generalise to an unknown dataset is reduced.

Classification algorithms in R (caret package) included a parameter called trControl that can be used to control model training, where these options were set. The classProbs option was also set to TRUE so class probabilities were captured in addition to predicted class - this allowed ROC curves to be drawn and ensemble models to be built more robustly (weighted by probability rather than the final class).

Further, the tuning parameter was set to 10. This allowed 10 combination of hyper-parameters relevant to the model will be attempted and the best one automatically selected. The algorithm had default starting values for these.

Decision Tree Model

The C5.0 algorithm is becoming the industry standard for decision trees, and so was used instead of the ID3. It grows a full decision tree (using information gain and entropy to determine splitting criteria) and then post-prunes (cull branches after trees are built) over-fitted branches of the tree. The C50 package in R was used.

Parameter Tuning

There are three tuning parameters available: trials (number of boosting iterations), model (one of tree - default, or rules if the tree should be decomposed into a rule-based model) and winnow (FALSE - default, or TRUE if predictor feature selection is to be used).

The tuneLength parameter in the train function in the caret package was set to 10, meaning 10 different combinations of the above parameters were attempted before determining the one with the best performance.

The best fitting parameters were: trials = 80, model = rules, winnow = FALSE on full training set.

Performance

- Accuracy: 89.87%
- f-score: 88.90%
- AUC: 94.68%
- Time to run: ~ 2 hours

K-Nearest Neighbour

The K-Nearest Neighbour algorithm classifies an example based on the most frequent target attributes of k examples closest to it, where distance is the Euclidean distance (square root of the sum of the square of the differences between each attribute). The knn package was used. Training was quick, but predictions took time, as it needed to calculate distances in the training dataset “on-the-fly”.

Parameter Tuning

There is one tuning parameters available: k (number of nearest neighbours to consider).

The tuneLength parameter in the train function in the caret package was set to 10, meaning 10 different combinations of this parameter were attempted before automatically determining the one with the best performance.

The best fitting parameters were: k = 5 on full training set.

Performance

- Accuracy: 88.80%
- f-score: 87.98%
- AUC: 93.63%
- Time to run: < 20 minutes (prediction took longer as expected)

Random Forest Model

The Random Forest algorithm is an example of bagging, where it uses an algorithm like decision trees to builds multiple deep trees in parallel (each has low bias but high variance) and them combines them to lower the variance. The rf package in R was used.

Parameter Tuning

The Random Forest model has the mtry parameter, which is the number of attributes randomly collected to be sampled at each split. The default is the square root of the number of attributes.

The tuneLength parameter in the train function in the caret package was set to 10, meaning 10 different values of the mtry were attempted before determining the one with the best performance.

The best fitting parameters were: mtry = 83.

Accuracy

- Accuracy: 88.14%
- f-score: 87.27%
- AUC: 93.05%
- Time to run: ~4-5 hours

GBM (Gradient Boost Model)

The GBM algorithm is an example of a boosting algorithm where multiple trees are built sequentially, with subsequent trees focusing on examples that were previously mis-classified (by increasing their weights). The gbm package was used.

Parameter Tuning

There are 4 tuning parameters available: n.trees (number of trees), interaction.depth (maximum nodes per tree), shrinkage (also known as the learning rate), n.minobsinnode (minimum number of observations in trees' terminal nodes).

The tuneLength parameter in the train function in the caret package was set to 10, meaning 10 different combinations of this parameter were attempted before automatically determining the one with the best performance.

The best fitting parameters were: n.trees = 450, interaction.depth = 10, shrinkage = 0.1, n.minobsinnode = 10 on full training set.

Performance

- Accuracy: 90.83%
- f-score: 90.18%
- AUC: 96.04%
- Time to run: ~ 3-4 hours

Support Vector Machine (SVM)

The SVM (Support Vector Machine) algorithm works by checking for hyperplanes (higher dimension than attribute space) that can create good margins between classes

of data by using kernel functions to transform the data. The svmRadial package was used; it uses a radial kernel function.

The numeric attributes were normalised for better performance, and only 50% of the examples in the training dataset (randomly drawn without replacement) were used, since training on the full dataset took too long.

Parameter Tuning

The Radial SVM model has two parameters: C (penalty imposed on the model for making an error) and sigma (how dependent is the SVM boundary to just the closest points).

The tuneLength parameter in the train function in the caret package was set to 10, meaning 10 different values of the mtry were attempted before determining the one with the best performance.

The best fitting parameters were: sigma = 0.0273, C = 4.

Accuracy

- Accuracy: 89.72%
- f-score: 88.89%
- AUC: 93.40%
- Time to run: ~8 hours

Neural Networks

Neural networks are popular data mining algorithm (especially in the imaging space) and were roughly based on neurons, where examples “flow” through nodes and depending on where they land (i.e., error relative to the actual target), weights in the nodes are adjusted to “learn” from their mistake. This process is repeated for each example. The nnet package provides a feed-forward single-hidden-layer neural network algorithm, which was used. The numeric attributes were normalised for better performance.

Parameter Tuning

There are two tuning parameters available: size (number of units in the hidden layer) and decay (regularisation parameter for weight decay used to avoid over-fitting).

The tuneLength parameter in the train function in the caret package was set to 10, meaning 10 different combinations of this parameter were attempted before automatically determining the one with the best performance.

The best fitting parameters were: size = 9, decay = 0.001 on full training set.

Performance

- Accuracy: 89.87%
- f-score: 89.00%

- AUC: 94.68%
- Time to run: ~ 4 hours

Ensemble

The Random Forest model (example of Bagging) and the GBM model (example of Boosting) are specialised ensemble models, that build and aggregate decision tree models in parallel and sequentially (respectively). They both performed well, and so a manual ensemble was attempted using all the models built (with the best hyper-parameter tuned for each).

The predictions (probabilities of class “yes”) across all these models were used to calculate an average probability, and the target attribute was set to yes if it exceeded the threshold 0.5, otherwise no. The average was also weighted based on the relative f-scores of the three models (i.e., better performing models were given more weight).

While numerous combinations of weights (adding up to 1) were attempted to optimise the overall f-score, and various cut-offs (to determine when a prediction is a yes) were attempted, ultimately, the GBM algorithm out-performed all of them. Further, the best cut-off was found to be 0.5.

Selecting the best model

The selected model was the GBM model, with hyper-parameters: n.trees = 450, interaction.depth = 10, shrinkage = 0.1, n.minobsinnode = 10 on full training set.

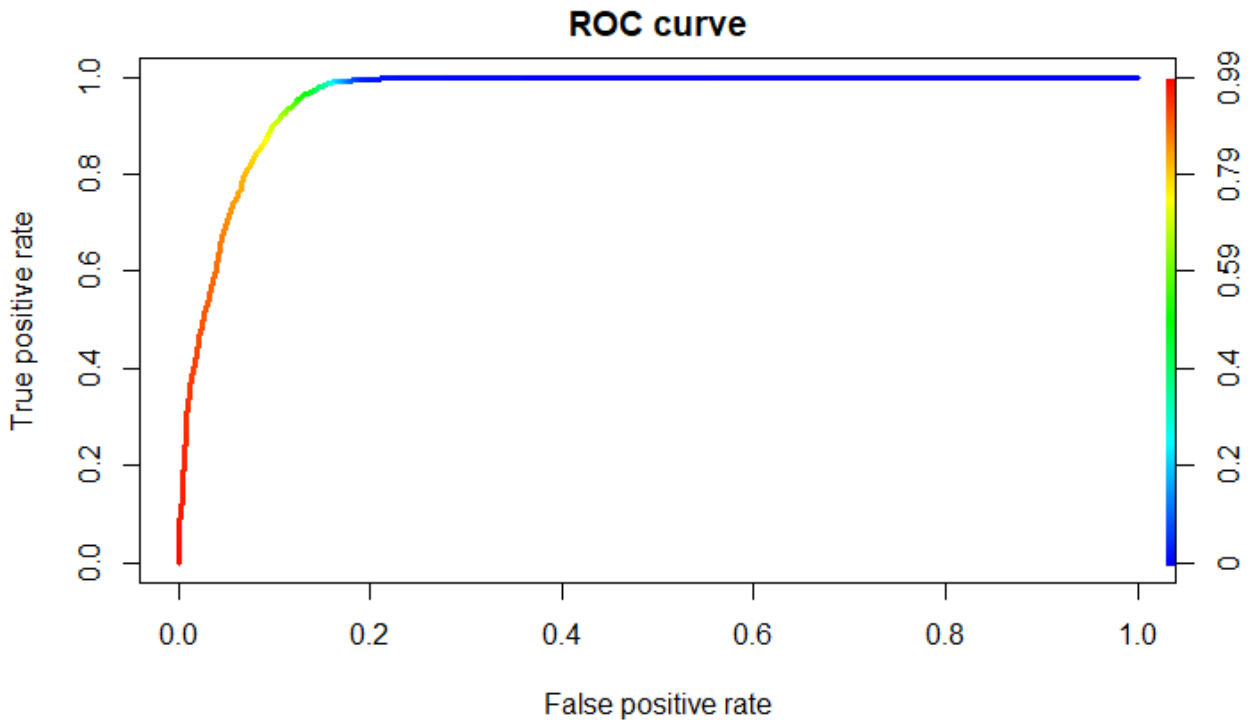
There were several considerations when selecting the best model:

- Accuracy (specifically, the f-score) - While training accuracy is good, it was important for the model to generalise well (i.e., not overfit - low variance). The GBM model reduces variance through boosting.
- The algorithm should also not be too simple (for example, using a linear regression model for a complex problem will result in high bias) and be robuse (deal with noise). The GBM algorithm does so.
- Speed / Time to run: This is a very important consideration given limited resources (i.e., not a very powerful laptop). It is the elapsed time taken during model build and prediction. For example, the SVM algorithm was promising, but took too long to run, and was therefore not preferred over the GBM algorithm.
- Interpretability: In some domains, it is critical to be able to explain the “rule set” behind the underlying model (for example, if used in determining if prisoners should get parole, you want to understand how it’s using race, gender, etc.). In this case however, it seemed less important. Decision trees aid interpretability, while neural networks for example, don’t. Associated with interpretability is the concept of goodness, where a simpler set of rules is preferred to a more substantive set even at the cost of some accuracy.

Based on all these considerations, the selected model was the GBM. Intuitively, this made sense because it seems like all the algorithms were able to classify ~88-90% of the cases fairly consistently, but struggled with the last 10-12%, and so a boosting algorithm that focused in on those mis-classified examples in subsequent iterations did better.

- It had the highest f-score
- It is much faster than SVM for example
- It is not interpretable, but in this domain, it may not matter as much.

The ROC curve for the best model (GBM)



Confusion Matrix:

	Reference	
Predicted	No	Yes
No	3,703	565
Yes	114	3,118

The model was mis-classifying yes more often than it was mis-classifying no, suggesting a higher specificity than sensitivity.

Other statistics:

- Accuracy: 90.95%
- No Information Rate : 0.5089 (accuracy if random)

- P-Value [Acc > NIR] : < 2.2e-16 (value less than 0.05 suggests it significantly outperformed randomness)
- Sensitivity : 0.8466
- Specificity : 0.9701
- Pos Pred Value : 0.9647
- Neg Pred Value : 0.8676
- Prevalence : 0.4911
- Detection Rate : 0.4157
- Detection Prevalence : 0.4309

Submitting Predictions

All the pre-processing steps applied on the training dataset was applied on the testing dataset and the best model was used to predict QUALIFIED attribute. the output was collated in the format needed by Kaggle and submitted successfully.

Reflections & Learnings

I've captured my reflections here, including what I've learnt, and what I would do differently if I did it again.

Test and Learn Quickly

I spent a lot of time applying transformations to the data that I thought made sense. I initially completed all my pre-processing before testing, and was horrified to get a really poor accuracy score when tested against the first algorithm! I then decided to build a "reference" model that I would use to test each transformation (not once just once at the end) to see if it significantly improved performance (and also to test one type of transformation over another - like how to treat missing values).

In the future, I will definitely be taking the latter approach.

It's science with art

As I researched different methods for pre-processing, setting parameters, etc., a lot of recommendations that I found on the internet were "it depends", suggesting that there is no "one-size-fits-all", and that experimentation is needed since every dataset is different. Therefore there is art as well as science in the process, and no "silver bullet" solution. With more experience, I suppose choices will become easier, but when starting, it just seems like there are so many possibly ways of approaching the problem, that one can get lost.

I also found it curious, that you can get to a decent accuracy fairly quickly, but making significant improvement from there was very difficult! In fact, you can quickly get bogged down by delving deep into a hypothesis to improve performance, and many many hours

later, coming out with little or nothing to show for it. I learnt that the time would have been better spent on exploring more broadly.

I under-estimated the effort

I did not expect algorithms to run for hours, with the longest one I had running for 24 hours on my laptop! The learning was to test my algorithm on subsets of data an prove it out before running it on a full 10-fold cross-validation across the entire dataset, with multiple hyper-parameters being tested at the same time!

One of my practical learnings was to consistently back-up my models and clearly document the parameters that I used to run them (apply version control). Initially, I'd get a good result, but I'd lose it and not be able to re-create it (or re-creating it would take another 24 hours!) I therefore started using github to back-up my work.

If I had more time (or was doing it again), I'd definitely do more work earlier, rather than procrastinating and trying to finish things quickly!

Need more statistical background

While I can get away with running these algorithms, it's clear that a strong statistics background will help cement concepts better, and also provide a stronger basis for making one decision over another. I intend to pursue basic statistics courses so I can appreciate the use of commonly applied statistics (like anova).

Ultimately, I learnt a lot

When I started this assignment, I didn't realise how much I will learn, from a brand new programming language (R), concepts in advanced analytics, to practical application through code. I explored tons of R packages to perform transformations and apply statistical methods and data mining algorithms. Because of the popularity of R among data scientists, I believe that this investment in learning R will pay off in my career.

The advanced analytics concepts in the class became much clearer when I applied them through this assignment. For example, the concept of bias and variance, when applied (for example through bagging - random forests), became a lot clearer, and I appreciate this trade-off a lot more now, than I might have otherwise.

In addition to going through the course notes and listening to all the lectures and workshops, doing the assignment also got me to research a lot on Google. StackOverflow was my best friend!