COMP3009: Data Mining Assignment

Nina Kumagai (19389905)

1.0 Summary

* Please note that a pdf of the jupyter notebook is also provided in **mypython.pdf** to illustrate which functions, packages I used. **run.py** is just the converted version of the original jupyter notebook.

1.1 Major Findings

1.11 Data Preparation

- Att14 and att17 are irrelevant because whether they are included in the analysis or not will not make a difference to the outcome.
- Att3 and att9 are categorical attributes with very few missing values so these missing values were imputed by using the mode. Att25 and att28 however are numerical with a few missing values so these were imputed using the mean. Att13 and att19 had a very large number of missing values so these attributes were removed.
- Att8 and att24 are duplicate columns so att24 was removed as this would be redundant information for the classification. There were no duplicate rows.
- Att1 to att12 which had alphabetical labels were changed into categorical datatype by engineering dummy variables. The other categorical variables which had numerical values were simply converted into categorical datatype without producing dummy variables.
- Att18, att25 and att28 are all skewed to the right. Thus, a log transformation was applied.
- Training and validation data (1000 instances) were split off from the test data (100 instances).
- Standardisation was applied to the training/valid set and then also applied to the test set with the same mean.
- Multicollinearity check was applied to the dataset and there was no multicollinearity present. This means LDA can now be conducted.
- Principal component analysis was applied to the dataset. From 65 attributes, 50 principal components were produced.
- Data balancing was applied using SMOTE, balanced bagging classifier and stratified cross validation (stratified cross validation ensured balanced distribution both for training and validation sets).

1.12 Classification

- Hyperparameters were tuned for all models, including KNN, Naïve Bayes, Decision Tree, Logistic Regression, Linear Discriminant Analysis (LDA) and Random Forest. Some hyperparameters which were adjusted include K in KNN, max depth in random forest and decision trees.
- Stratified 10-fold cross validation was used to evaluate the effectiveness of the classifier.
- Average accuracy was derived as well as the variances for each model. F-scores were also produced alongside confusion matrices for each model using scikit learn.
- Overall looking at recall and precision it can be seen that recall is much higher for all the models compared to precision. This signifies that most models are more likely to predict a 1 as 0 than it is to predict a 0 as a 1.
- Logistic Regression (with accuracy of 0.761, variance of 0.001393 and F-score of 0.788) and Random Forest (with accuracy of 0.766, variance of 0.001796 and F-score of 0.795) were chosen as the best two classification schemes because of their high accuracy, lower variances and higher F-scores.

1.2 Lessons Learned

1.21 Data Preparation

- Sometimes without domain knowledge and context of a dataset, it is hard to make decisions in the pre-processing step.
- Deciding when to split the training, validation and test is an important decision. In the current dataset, there were very few missing values to impute, so it was not necessary to split the test from the training data prior to imputation to prevent data leakage. When to split the dataset is a decision that needs to be made depending on the type of dataset that is analysed.
- Dummy variables, although not always necessary, are a good way to change numerical data to categorical format (binary values) and feature engineer new columns. This may also improve the PCA because each level of a factor may influence the model with different weightage.
- Data pre-processing is arguably one of the most important skills for classification and the decisions that are made will affect the outcome of the classification models heavily.

1.22 Classification

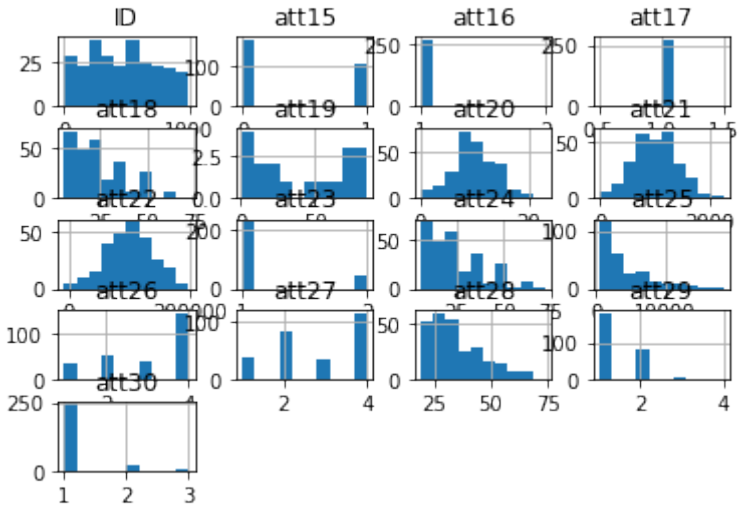
- Balancing dataset is very important since training on an unbalanced dataset will mean it will automatically predict the majority class more often than the minority and result in bias in the model.
- Initial accuracies on oversampled dataset were too high. Overfitting was found to be the cause of this high accuracy. Overfitting can result from balancing data (resampling from the minority class) due to duplicates. SMOTE and balanced bagging classifiers as well as stratified sampling in cross validation is one way to prevent this problem.
- Important to tune hyperparameters manually in classification modelling as it cannot be pre-determined. It can only be selected based on classifier outcomes such as with accuracy or the F-score.
- When finding the optimal K for KNN, it is necessary to find the error rate that is small enough so that only the samples which are relevant are present but also big enough that there is no overfitting. Determining the value of K in KNN is a difficult problem. K = 1 showed the least mean error but this is most likely due to overfitting. It is difficult to access which K value is optimal just looking at the accuracy of the model or mean error rates.
- Important to use whole training dataset (1000 instances) to train the final model and predict the test class labels.
- Weka is a powerful tool because it allows better insight into hyperparameter tuning. Thus, even though python is very powerful, it probably is easier to tune hyperparameters quickly on Weka to get some insight.

2.0 Methodology

2.1 Data preparation:

Exploratory Data Analysis (EDA):

- Histogram of numerical attributes show varying distributions



- A rough pair plot at the beginning shows that no single attribute is good at distinguishing between the classes. This means that classification will depend on a group of attributes. Note however this plot is not so reliable given that data cleaning and pre-processing has not taken place yet - just a graph for initial insight into the dataset!

Irrelevant attributes: this data set is known to have irrelevant attributes. Describe what you think irrelevant attributes are.

Attributes which are irrelevant have no influence on the classification model and thus these attributes are redundant in nature (in actual fact, it could even discourage an optimal model from being constructed).

For each attribute, carefully examine it and decide whether it is irrelevant. If so, give a brief explanation and remove the attribute.

Att14 and att17 are irrelevant because whether they are included in the analysis or not will not make a difference. This is because att14 and att17 both have the same response for every instance and thus will not make a difference to the classification model. The ID column was also removed as this is irrelevant information for the classification model.

Missing entries

Which attributes/instances have missing entries?

```
ID          0
Class     100
att1        0
att2        0
att3        4
att4        0
att5        0
att6        0
att7        0
att8        0
att9        5
att10       0
att11       0
att12       0
att13    1028
att15       0
att16       0
att18       0
att19    1034
att20       0
att21       0
att22       0
att23       0
att24       0
att25       3
att26       0
att27       0
att28       6
att29       0
att30       0
```

Attributes which have missing values include att3, att9, att25, att28, att13 and att19.

For those attributes/instances, how many missing entries are present and for each attribute/instance with missing entries, make a suitable decision, justify it, and proceed.

Small Number of Missing Data:

att3:

- Type: Categorical
- Issue(s): four missing values.
- Decision: impute the values with the most frequent occurring category for this attribute.

- Reason: The number of missing entries is much smaller than the total number of instances and it is assumed the data is non-dependency oriented. Categorical means that mode is used for imputation.
- Filter used: “df['att3'].fillna(df['att3'].mode()[0], inplace=True)”

att9:

- Type: Categorical
- Issue(s): five missing values.
- Decision: impute the values with the most frequent occurring category for this attribute.
- Reason: The number of missing entries is much smaller than the total number of instances and it is assumed the data is non-dependency oriented. Categorical means that mode is used for imputation.
- Filter used: “df['att9'].fillna(df['att9'].mode()[0], inplace=True)”

att25:

- Type: Numerical
- Issues: three missing values.
- Decision: the average values of the available data in att25 will be calculated and this average will be recorded in the three missing points.
- Reason: The number of missing entries is much smaller than the total number of instances and it is assumed the data is non-dependency oriented. Numerical means that the mean can be used for imputation.
- Filter used: “df['att25'].fillna(df['att25'].mean(), inplace=True)”

att28:

- Type: Numerical
- Issues: six missing values.
- Decision: average values of the available data in att25 will be calculated and this average will be recorded in the missing points.
- Reason: The number of missing entries is much smaller than the total number of instances and it is assumed the data is non-dependency oriented. Numerical means that the mean can be used for imputation.
- Filter used: “df['att28'].fillna(df['att28'].mean(), inplace=True)”

Large Number of Missing Data:

att13:

- Type: Categorical
- Issues: 1028 missing values
- Decision: Remove the column
- Reason: There are far too many missing values (1028) in order for us to impute or accept the missing values.
- Filter used: “del df['att13']”

att19:

- Type: Numerical
- Issues: 1034 missing values
- Decision: Remove the column
- Reason: far too many missing values (1034) in order for us to impute or accept the missing values.

- Filter used: “del df['att19']”

Duplicates. Detect if there are any duplicates (instances/attributes) in the original data?

From simple observation we can confirm there are attribute duplicates. More in-depth check was run via python (using “getDuplicateColumns”) and duplicate columns were removed.

For each attribute/instance with duplicates, make a suitable decision, justify it, and proceed.

Att8 and att24 are duplicate columns so att24 was removed as this would be redundant information for the classification. There were no duplicate rows.

Data type: **For each attribute, carefully examine the default data type (e.g. Numeric, Nominal, Binary, String, etc.) that has been decided when Weka loads the original CSV file.**

Original Attributes:

```
Class    float64
att1      object
att2      object
att3      object
att4      object
att5      object
att6      object
att7      object
att9      object
att10     object
att11     object
att12     object
att15      int64
att16      int64
att18      int64
att20      int64
att21      int64
att22      int64
att23      int64
att25    float64
att26      int64
att27      int64
att28    float64
att29      int64
att30      int64
```

If the data type of an attribute is not suitable, give a brief explanation and convert the attribute to a more suitable data type. Provide detailed information of the conversion.

Reclassified Attributes:

* Note that object is a string, float64 is a decimal value, int64 is an integer in pandas.

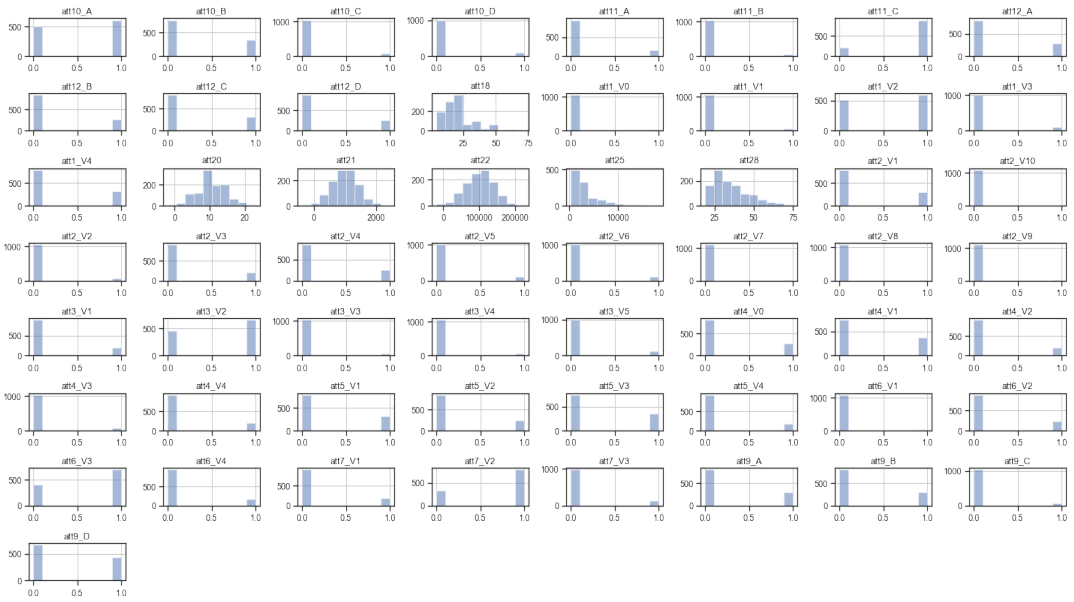
```
Attribute    Current     Reclassified
ID           int64       categorical
Class       float64      categorical      (binary)  1 or 0
```

```
att1          object      categorical
att2          object      categorical
att3          object      categorical
att4          object      categorical
att5          object      categorical
att6          object      categorical
att7          object      categorical
att9          object      categorical
att10         object      categorical
att11         object      categorical
att12         object      categorical
att15          int64      categorical     (binary)   1 or 0
att16          int64      categorical.    (binary)   1 or 2
att18          int64      numerical
att20          int64      numerical
att21          int64      numerical
att22          int64      numerical
att23          int64      categorical     (binary)   1 or 2
att25        float64      numerical
att26          int64      categorical                1,4,3,2
att27          int64      categorical                4,3,2,1
att28        float64      numerical
att29          int64      categorical                1,2,3,4
att30          int64      categorical                1,2,3
```

Float values did not have any decimal values attached and thus were converted to integers instead to keep all attributes consistent. Attributes which should be categorical range from attributes 1 to 12 and also class, att15, att16, att23, att26, att27, att29 and att30. Note that att1 to 12 are alphabetical (e.g. V1, V2, A, B, C etc) whilst att15, att16, att23, att26, att27, att29 and att30 contain digits (e.g. 0, 1, 2, 3, 4).

The alphabetical attributes were first converted into categorical format and all other attributes left as numerical. This was to ensure that when applying the dummy variable function in python to the dataset, dummy variables were only created for att1 to 12.

Dummy variables are necessary to allow the machine learning models to process the data (traditionally, alphabetical values result in error with scikit learn). Dummy variables work by making additional attributes to ensure that all selected attributes become binary in nature (categorical integer representing the data).

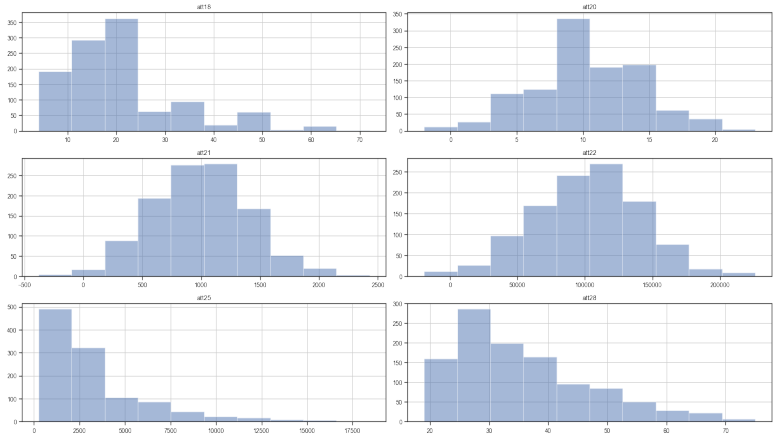


As can be visualised with the histogram of each attribute, dummy variables have been successfully created with binary values of 1 or 0.

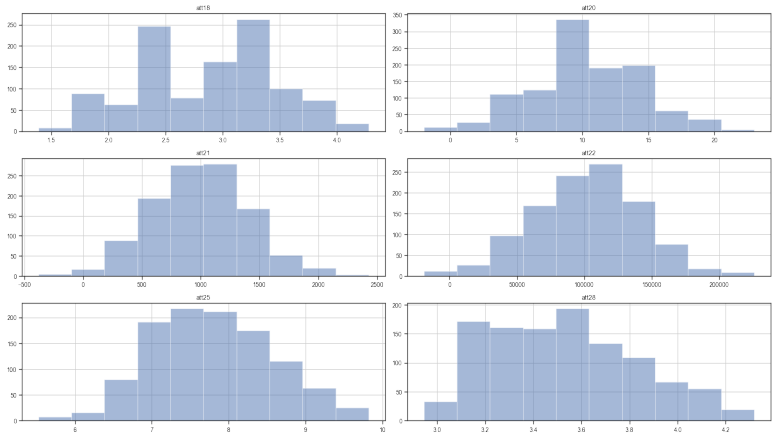
Once these dummy variables were created, the conversion to categorical data type was continued for the other attributes which were initially in digit form: class, att15, att16, att23, att26, att27, att29 and att30. All other attributes were converted to integers as they were numerical in nature. This included float values as it was found that all floats did not have decimal placed values.

Log Transformation:

Visualising all numerical features with histograms, it can be seen that att18, att25 and att28 are all skewed to the right. Thus, a log transformation was applied. Note that a log transformation was applied prior to the standardisation on purpose.



Afterwards it was seen with the plot below that the log transformation was successful at removing the skew:



Training, Validation, and Test Sets: suitably divide the prepared data into training, validation and test sets.

Training and validation data (1000 instances) were split off from the test data (100 instances).

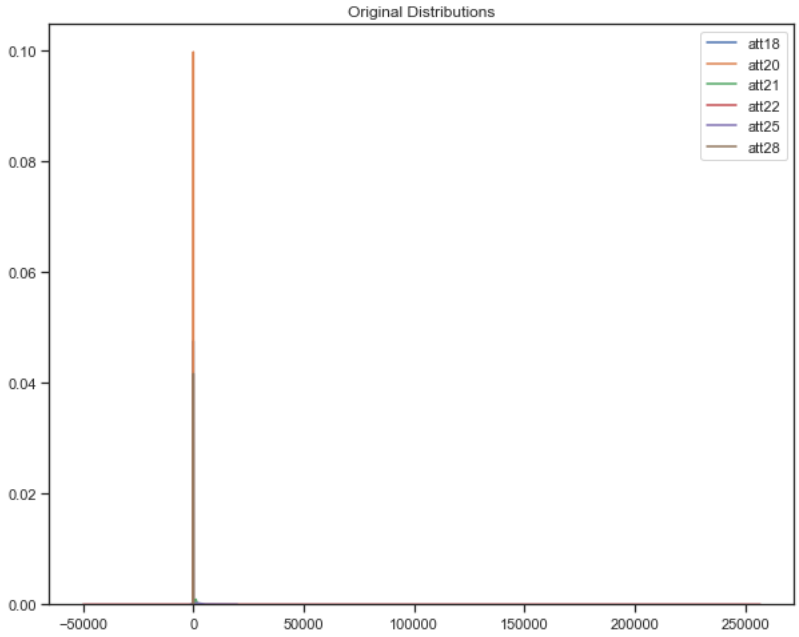
Applying the split at this point was an important decision to maximise the reliability and accuracy of our results. For example, this split was applied after the pre-processing to impute missing data. This was because there were very few missing datapoints to be imputed – otherwise it may have been important to split the dataset prior to imputation (to prevent data leakage into the test set). Also note that the dataset was split after the logarithmic transformation because it needed to apply both to the training and test sets.

The split was also applied prior to standardisation because standardising should be applied to the training set first and then the same mean from training should be used to standardise the test set. This is because when the mean is similar (consistent) between training and test, mean from the training should be more reliable to use as it contains more datapoints.

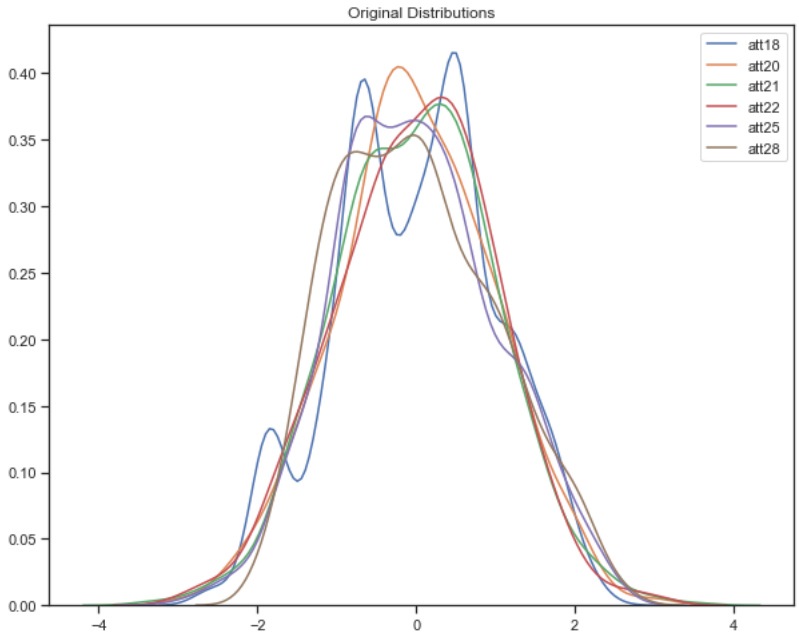
Scaling and standardisation:

For each numeric attribute, decide if any pre-processing (e.g. standardisation) is required. Give a brief explanation why it is needed (this should be discussed in relation to the subsequent classification task).

Initially when all numerical points were plotted, there was only a thin bar. This signifies that the values of the numerical attributes vary quite a lot.



Once standardisation was applied to the numerical attributes (att18, att20, att21, att22, att25, and att28), the curvature could be visualised with mean equalling 0 and a standard deviation of 1.



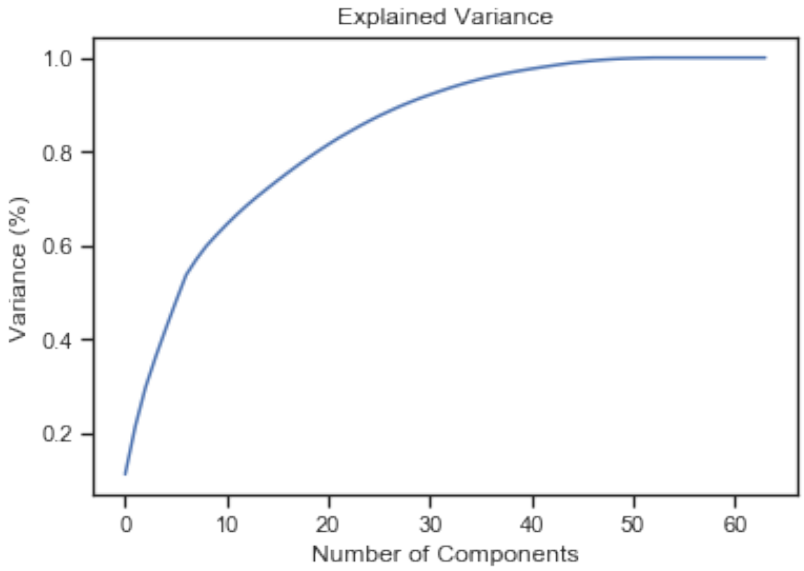
The above plot shows that the blue line has a slightly strange distribution and therefore a trial run of the classification models were run without this attribute. However, it was found that including this attribute lead to better results and thus this attribute was kept. Note that normalisation was also trialed as can be found in **mypython.pdf** but it was found that accuracy and F-score was better when standardisation was applied.

Feature engineering: you may also come up with attributes derived from existing attributes. If this is the case, give an explanation of the new attributes that you have created.

*Note to Sonny: I have already discussed dummy variables previously and this will also fall under feature engineering.

Another way in which feature engineering was applied to the dataset was to use principal component analysis. Principal component analysis (PCA) uses orthogonal transformations to try and grasp the principal components with the highest variance (highest representation) of the datapoints. In doing so, it converts correlated variables into uncorrelated variables. Principal component analysis is also important to remove the likelihood of overfitting to the training dataset as noise is removed from the data.

Using "sklearn.decomposition import PCA" and fitting the PCA algorithm with our Data, principal components were derived. Here is a plot illustrating the cumulative summation of explained variance:



Feature/Attribute selection: if applicable, clearly indicate which attributes you decide to remove in addition to those (obviously) irrelevant attributes that you have identified above and give a brief explanation why.

Initially there were 65 attributes in the dataset but with the cumulative summation of explained variance shown with the plot above, it can be seen that only about 50 principal components explain the bulk of the variance in the dataset. Thus only 50 components (comprising of dummy variables) were kept, and this formed the new dataset to feed into the classification models.

Data instances: if you decide to make changes to the data instances with class labels (this may include selecting only a subset of the data, removing instances, randomizing/reordering instances, or synthetically injecting new data instances to the training data, etc.), provide an explanation.

The answer will be addressed in the data imbalance section of the report.

Data imbalance: **the data set is known to have more samples from one class than the other. If you employ any strategy to address the data imbalance issue, describe it thoroughly.**

Many different ways to balance the dataset. A combination of data balancing methods was applied in the end.

1. **Oversampling from minority group**
    - "df_class_0_over = df_class_0.sample(400, replace=True)"
    - It was found that oversampling the minority group led to very high accuracy on the validation dataset. As duplicate values lead to much higher chance of a model fitting the validation due to overfitting, this may signify that the model actually performs poorly on new unclassified data.
2. **Undersampling from majority group**
    - "from imblearn.under_sampling import ClusterCentroids".
    - This reduces the accuracy because there were much fewer datapoints which the model was being trained on.
3. **Both oversampling from minority and undersampling from majority**
    - Good but SMOTE is better.
4. **Using SMOTE**
    - "from imblearn.over_sampling import SMOTE"
    - Works very well! Accuracy and F-score improved a lot (approximately 5% boost with SMOTE than without).
    - SMOTE uses over-sampling to balance dataset but instead of duplicating minority group instances, it actually creates synthetically similar instances to the minority class instances keeping reference to the neighbouring records.
5. **Using Balanced Bagging Classifier (from imblearn in Python)**
    - "from imblearn.ensemble import BalancedBaggingClassifier"
    - Balanced Bagging Classifier is a bagging classifier with additional balancing.
    - Bagging is a machine learning algorithm which is used to increase the stability and the accuracy of classification models. It also helps to reduce variance by fitting the classification model on random subsets of the dataset and aggregating the predictions from each fit (via voting) to form the final prediction.
    - Usually most important for decision trees and random forest models but can also be applied to other models.
6. **Using Stratified Cross Validation**
    - "from sklearn.model_selection import StratifiedKFold, KFold"
    - Stratified k-fold cross validation ensures that there is an equal representation of each class in the separate samples.

Overall, SMOTE, Balanced Bagging Classifier (imblearn package), and stratified cross validation was used. In terms of addressing the issue of class imbalance for the training data, balanced bagging classifier was used. This ensured that bagging would be applied whilst keeping the training samples balanced. SMOTE together with stratified cross validation was also used to ensure balance in the training and validation set as well. This is an important assumption to satisfy given that the test data contains exactly 50 samples from each class. If the model is trained without equal proportion of both classes, the model would be biased towards finding more of the majority class. Thus data balancing is a very important step in this project.

Others: describe other data-preparation steps not mentioned above.

Checking for Multicollinearity:

```
VIF Factor     features
```

```
0       100.8   Intercept
1       1.1     Class
2       1.1     att15
3       1.1     att16
4       2.0     att18
5       1.0     att20
6       1.0     att21
7       1.0     att22
8       1.0     att23
9       2.2     att25
10      1.3     att26
11      1.1     att27
12      1.2     att28
13      1.1     att29
14      1.0     att30
```

All the relevant VIF factors are below 5 so there exists no multicollinearity. This is important in order to run linear discriminant analysis later.

2.2 Data classification:

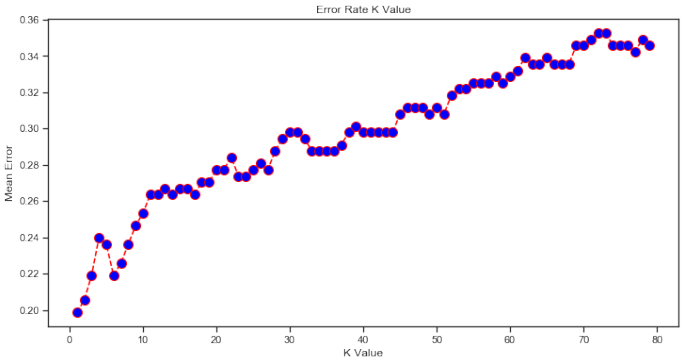
Classifier selection: you will need to select at least three (3) classifiers that have been discussed in the workshops: k-NN, Naive Bayes, and Decision Trees (J48). Other classifiers, including meta classifiers, are also encouraged. Every classifier typically has parameters to tune. If you change the default parameters to achieve higher cross-validation performance, clearly indicate what the parameters mean, and what values you have selected.

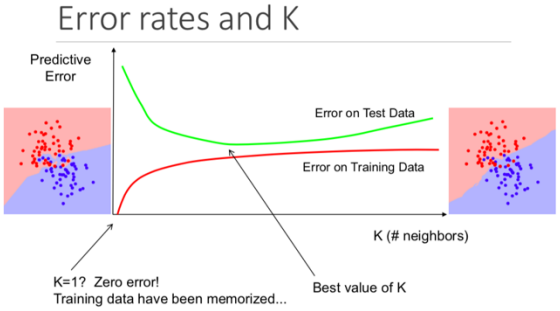
Classifiers which were tested:

- KNN: ("from sklearn.neighbors import KNeighborsClassifier").
- Naïve Bayes: ("from sklearn.naive_bayes import GaussianNB").
- Decision Tree: ("from sklearn.tree import DecisionTreeClassifier").
- Logistic Regression: ("from sklearn.linear_model import LogisticRegression").
- Linear Discriminant Analysis: ("from sklearn.discriminant_analysis import LinearDiscriminantAnalysis").
- Random Forest: ("from sklearn.ensemble import RandomForestClassifier").

Parameters & Hyperparamater Tuning:
KNN:

- Tuning the K Nearest Neighbour hyperparameter:





- It can be seen that K=1 result in the smallest error for the balanced dataset with figure 1. However, K = 1 should not be used in the KNN classifier because it can result in overfitting. One reason K = 1 is shown as the optimal parameter here is because of the fact that SMOTE was used to oversample synthetic data mimicking the original minority class and may have resulted in overfitting (the training data may have been memorised). As shown with the second figure above, it is important to find a compromise for the value of K which results in the lowest error for both the test and validation sets (where validation is taken from training via cross validation). The compromise between overfitting and inaccurate predictions, resulted in using K = 7 as it shows the lowest mean error whilst not being close to K = 1.

Naïve Bayes:

- Naïve Bayes used the Bayesian theorem to classify the labels.
- Var_smoothing = 1e^-9. Var_smoothing which looks at the portion of the largest variance of all features that is added to variances.

Decision Trees:

- max_depth = 3. Max-depth controls the tree's maximum depth. In other words, it is the longest path length from the root the leaf of the tree. A variety of max depths were tested, and it was found that a max depth of three returned optimal results.

Logistic Regression:

- solver='liblinear'. Liblinear was used because the dataset is quite small and thus liblinear will iterate through the dataset quickly.
- multi_class='ovr'. Seeing as the data class is binary, ovr was used here.

Linear Discriminant Analysis (LDA):

- solver='svd', default was used because this does not compare the covariance matrix and thus is suitable for most datasets with many features.
- Shrinkage= none. No shrinkage was applied to the model.
- n_components was set to the minimum number between the number of classes – 1 and the number of features.

Random Forest:

- max_depth=2. A variety of max depths were tested, and it was found that a max depth of two returned optimal results.
- Criterion = "gini". Criterion that was used was the gini index which is the function that is used to understand the quality of a split.

Cross validation:

How to evaluate the effectiveness of a classifier on the given data?

Use accuracy and F-score. Variance of the accuracies could also give an indication of the robustness of the model. Sensitivity and specificity could be used but for the current purpose, the context of the

dataset is not known (it is not clear whether the value of false positive or false negatives is more detrimental to the problem) so this was not taken into account when evaluating effectiveness.

How to address the issue of class imbalance in the training data?

As discussed above, in terms of addressing the issue of class imbalance for the training data, balanced bagging classifier was used. This ensured that bagging would be applied whilst keeping the samples balanced. SMOTE together with stratified cross validation was also used to ensure balance in both training and validation sets as well. This is an important assumption to satisfy given that the test data contains exactly 50 samples from each class and not having the same distribution in the validation or training could bias the models to that distribution.

What is your choice of validation/cross-validation?

Stratified 10-fold cross validation was used to evaluate the effectiveness of the classifier. This is equivalent to the "Leave One Out" strategy. Choosing a K = 10 meant that there would be plenty of data to train the model on whilst also having enough datapoints (more than 100) for the validation. Note that when choosing the value of K, there is a bias-variance trade-off. K = 10 has been shown to yield a test error rates that isn't on extremes of a spectrum (very high variance nor high bias).

For each classifier that you've selected, what is the validation/cross-validation performance? Give an interpretation of the confusion matrix.

Using the package: "from sklearn.metrics import confusion_matrix"

Before analysing confusion matrix, here is a list of definitions around interpretation indexes for confusion matrices:

- **Accuracy:** how frequently is the classifier correct?
  - (TP+TN)/total
- **Error Rate:** how frequently is it wrong?
  - (FP+FN)/total (which is same as 1 minus accuracy)
- **True Positive Rate:** When it's yes, how frequently does it predict yes?
  - TP/actual 0 (same as "Sensitivity" or "Recall")
- **False Positive Rate:** When it's no, how frequently does it predict yes?
  - FP/actual 1
- **True Negative Rate:** When it's no, how frequently does it predict no?
  - TN/actual 1 (same as 1 minus False Positive Rate and also known as "Specificity")
- **Precision:** When it predicts yes, how often is it correct?
  - TP/predicted 0
- **Prevalence:** How frequently does the yes condition occur in the sample?
  - actual 0/total

KNN:

	pred: 0	**pred: 1**
true: 0	73	0
true: 1	25	48

Error rate: 0.235
Recall: 1
Precision: 0.745

*Recall of 1 may signify overfitting.

Naïve Bayes:

	pred: 0	pred: 1
true: 0	69	4
true: 1	25	48

Error rate: 0.255
Recall: 0.945
Precision: 0.734

Decision Tree:

	pred: 0	pred: 1
true: 0	55	18
true: 1	25	48

Error rate: 0.267
Recall: 0.753
Precision: 0.689

Logistic Regression:

	pred: 0	pred: 1
true: 0	59	14
true: 1	17	56

Error rate: 0.239
Recall: 0.808
Precision: 0.776

LDA:

	pred: 0	pred: 1
true: 0	61	12
true: 1	21	52

Error rate: 0.242
Recall: 0.836
Precision: 0.744

Random Forest:

	pred: 0	pred: 1
true: 0	61	12
true: 1	18	55

Error rate: 0.234
Recall: 0.836
Precision: 0.0.772

Overall looking at recall and precision it can be seen that recall is much higher for all the models compared to precision. This signifies that most models are more likely to predict a 1 as 0 than it is to predict a 0 as a 1. Now let's move on to the F-score which is an average between precision and recall.

F-scores:

Package used: "from sklearn.metrics import f1_score"

- F-score for KNN: 0.829
- F-score for Naive Bayes: 0.801
- F-score for Decision Tree: 0.767
- F-score for Logistic Regression: 0.788
- F-score for LDA: 0.774
- F-score for Random Forest:0.795

As can be seen, F-score for KNN is highest at 0.829, followed by Naïve Bayes at 0.801. The particular models of interest here are Logistic Regression and Random Forest because they had the highest accuracies. They gave a F-score of 0.788 and 0.795 respectively. Accuracy is an important measure, but it applies equal weighting to both positive and negative results, thus F-score was used to measure accuracy whereby the weighted harmonic means of both precision (true positive rate) and recall are taken into account.

For each classifier that you've selected, what is the estimated classification accuracy on the actual test data?

Accuracy:

Packaged used: "from sklearn.metrics import accuracy_score"

- Overall Accuracy for KNN with 10-fold cross-validation: 0.765
- Overall Accuracy for Naive Bayes with 10-fold cross-validation: 0.745
- Overall Accuracy for Decision tree with 10-fold cross-validation: 0.733
- Overall Accuracy for Logistic Reg with 10-fold cross-validation: 0.761
- Overall Accuracy for LDA with 10-fold cross-validation: 0.758
- Overall Accuracy for Random Forest with 10-fold cross-validation: 0.766

In terms of accuracy, logistic regression, KNN and random forest returned the highest with 0.761, 0.765, and 0.766 respectively. Variance was quite low for all models signifying robustness of the models.

Variance:

Package used: "import pandas as pd"

- Variance for KNN Accuracy: 0.000976
- Variance for NB Accuracy: 0.006399
- Variance for Decision Tree Accuracy: 0.000497
- Variance for Logistic Regression Accuracy: 0.001393
- Variance for LDA Accuracy: 0.000704
- Variance for Random Forest Accuracy: 0.001796

Classifier comparison:

Compare the classification performance between difference classifiers. You need to select at least two (2) evaluation metrics, for example F-measure and classification accuracy, when comparing them. Your comparison must take into account the variation between different runs due to cross-validation. Based on the comparison, select the best two (2) classification schemes for final prediction. Note that the two classification schemes can be one type of classifier, but

with two different parameters. Clearly indicate the final choice of parameters if they are not the default values.

Logistic Regression (with accuracy of 0.761, variance of 0.001393 and F-score of 0.788) and Random Forest (with accuracy of 0.766, variance of 0.001796 and F-score of 0.795) were chosen as the best two classification schemes because of their high accuracy, lower variances and higher F-scores. KNN, despite high accuracy, F-score and low variance was not chosen as the optimal model because of the difficulty that was experienced in selecting the best K value. This led us to precaution that overfitting might be occurring in the KNN model and thus decided against using it as the top classification scheme. Predicted class labels for both these attributes are stored in "predict.csv" and also shown in the appendix below.

3.0 References: list any relevant work that you refer to.

- https://medium.com/@saeedAR/smote-and-near-miss-in-python-machine-learning-in-imbalanced-datasets-b7976d9a7a79
- https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
- https://imbalancedlearn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html
- https://stats.stackexchange.com/questions/49540/understanding-stratified-cross-validation
- https://machinelearningmastery.com/k-fold-cross-validation/
- https://medium.com/datadriveninvestor/k-fold-and-other-cross-validation-techniques-6c03a2563f1e
- https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets
- https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/
- https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60
- https://stackabuse.com/classification-in-python-with-scikit-learn-and-pandas/

6.0 Appendices: important things not mentioned above.

Predict.csv:

ID	logReg	randomForest
1001	0	0
1002	0	0
1003	0	0
1004	0	0
1005	0	0
1006	1	0
1007	1	1
1008	1	0
1009	1	1
1010	0	0
1011	0	0
1012	1	1
1013	1	1
1014	0	0
1015	1	1

```
1016    0       1
1017    0       0
1018    0       0
1019    1       1
1020    0       1
1021    1       1
1022    1       1
1023    1       0
1024    1       1
1025    0       1
1026    1       1
1027    0       0
1028    0       0
1029    1       0
1030    1       0
1031    1       1
1032    1       1
1033    1       1
1034    0       0
1035    0       0
1036    1       1
1037    0       0
1038    0       0
1039    0       0
1040    1       1
1041    0       1
1042    0       0
1043    1       1
1044    1       1
1045    0       1
1046    0       0
1047    0       0
1048    0       1
1049    1       1
1050    0       0
1051    1       1
1052    0       0
1053    1       1
1054    1       1
1055    1       1
1056    1       1
1057    1       0
1058    1       1
1059    0       1
1060    0       0
1061    1       1
1062    1       1
1063    0       0
1064    0       0
1065    0       0
1066    1       1
1067    1       0
1068    1       1
1069    1       1
1070    0       1
1071    0       0
1072    1       1
1073    0       0
```

```
1074    1       1
1075    0       0
1076    1       1
1077    1       1
1078    1       1
1079    0       1
1080    1       1
1081    1       0
1082    1       1
1083    0       1
1084    1       1
1085    0       0
1086    1       1
1087    0       0
1088    0       0
1089    1       1
1090    1       0
1091    1       1
1092    0       0
1093    1       1
1094    1       1
1095    0       0
1096    0       0
1097    0       0
1098    0       0
1099    0       1
1100    1       1
```