Using Supervised Machine Learning Algorithm to Predict Customer Response

Modelling and Evaluation

Sony JUFRI, Sydney, 19 May 2020

Setup

In stage one of the project, the Bank Marketing dataset was explored. There were 41,188 examples and 20 attributes in the dataset. Ten of those attributes were numericals, whilst the rest were categoricals, with one output variable indicating whether a customer subscribed to a term deposit. The dataset was imbalanced because only 11% customers subscribed to a term deposit. It also had many missing values and outliers. Accordingly, it was cleaned, normalized and up-sampled to address those issues.

In stage two, four machine learning algorithms were used to fit the data and create models to predict customer response to a marketing campaign. The aim was to answer the following research question:

Q: "Is the recommended Random Forest model better than the alternatives considered?"

Hypothesis testing would be used to answer this question with the following hypotheses to be tested:

H_0: The recommended model's f1-score is not statistically better than the alternatives.

H_a: The recommended model's f1-score is statistically better than the alternatives.

Because the dataset was imbalanced, accuracy was not the right performance metric to use (89% of the bank's customers responded 'No' to the campaigns, so predicting 'No' all the time would easily give an 89% accuracy, even without a model). Instead, the classification f1-score[2] was used to measure the effectiveness of the models. The f1-score measures the harmonic mean of the model's precision and recall. Precision is the accuracy of the positive predictions, whereas recall is the ratio of positive instances that are properly detected by the model. A high f1-score (close to 1) can only be achieved if both recall and precision are high (ie if the model has a high accuracy of predicting the 'Yes' and many of the 'Yes' in the data can be detected), making it a good performance metric for our imbalanced dataset.

Once the f1-scores had been calculated, significance testing using the Wilcoxon signed-rank test[3] was performed to test the null hypothesis. The Wilcoxon test is a non-parametric statistical test used to compare two related samples and assess if their population mean ranks differ. Unlike the Student's t-test[4], it does not assume normally distributed populations, and the standard deviations of the two samples do not have to be equal (hence, it was appropriate for this project). However, it requires at least 20 samples, so all models were trained 20 times to meet this requirement. A significance level (α) of 0.01 was used, which means the probability of incorrectly rejecting the null hypothesis even when it was true (called 'Type I error') was fairly small at 1%, showing the strong reliability of this experiment.

Approach

In this project, four machine learning models were used to predict customer response and see if they are likely to subscribe to a term deposit in response to a marketing campaign (class 'Yes' or class 'No').

Logistic Regression

The first machine learning model considered was the Logistic Regression. Logistic Regression[5] is a binary classifier that can be used to estimate the probability of an instance belonging to a particular class (in this case, the probability of a customer signing up to a term deposit). The algorithm computes the weighted sum of the input variables and outputs the logistic of the result using the sigmoid function. If the estimated probability is higher than 50%, the model would predict 'Yes', otherwise it would predict

'No'. The model learns from the data and sets the weights of the input variables so it can estimate high (low) probabilities for positive (negative) instances by minimizing the cost function.

Because Logistic Regression is easy to implement, efficient to train, works well with high-dimensional data (no dimensionality reduction was performed when fitting this model) and can produce good predicted probabilities, it was used as the first benchmark model in this project.

k-Nearest Neighbor (kNN)

k-Nearest Neighbor[5] classifies the class of an instance by finding its k-nearest training instances (using a distance measure such as Euclidean distance) and makes predictions based on the majority class of those k-nearest neighbors. The model learns from the data by memorizing the entire training instances and performs classification by comparing the attributes of the new instance and the training instances.

kNN is simple, easy to implement and often produces accurate predictions. Therefore, it was used as a second benchmark model in this project. However, because kNN does not perform well in high dimension and the PCA analysis showed less than 50% attributes is sufficient to explain more than 80% of the variance in the data, the number of attributes was reduced to eight when fitting this model.

Decision Trees and Random Forest Classifier

Decision Trees[5] use a tree-like graph to classify an instance. It does so by traversing the tree from the root to the leaf node, testing the values of the attributes along the way. It then returns the class of the leaf node as its prediction ('Yes' or 'No'). Random Forest Classifier[5] on the other hand, is an ensemble method that train a group of Decision Trees to predict the class of an instance. Each Decision Tree is trained on a different random subset of the training data using a different subset of attributes. Random Forest would then aggregate those predictions and predict the class that gets the most votes (eg, if 100 trees are used to classify an instance and 80 of them predict 'Yes', Random Forest would predict 'Yes' for that instance). The performance of Random Forest depends on the performance of each individual trees and the correlation between those trees. Therefore, to generate diversity and reduce correlation between the trees, bagging (data sampling with replacement) and random feature selection were used when fitting this model. Random Forest usually performs well, efficient, robust to overfitting and can identify important features in the data. Accordingly, it was chosen as the proposed machine learning model in this experiment, with a single Decision Tree being used as the third benchmark model.

Train and Test Splits

Firstly, the train_test_split class from SciKit Learn was used to split the dataset into training and test sets. The training set was used to build the model and the test set was used to evaluate it (later on, the training set was split further into training and validation sets to tune the model's hyper-parameters). Because the Bank Marketing dataset was imbalanced, stratified splitting using the StratifiedKFold from sklearn was performed to ensure that each class is represented with approximately equal proportions in both data sets. Subsequently, the SMOTE sampling method from imblearn library was used to up-sample the training set so the number of 'Yes' are equal to the number of 'No'. This would help the models learn both classes ('Yes' and 'No') better in training and yield better predictions on unseen data.

Grid-Search and Cross Validation

Overfitting and underfitting are common issues in machine learning. Underfitting happens when a model is too simple that it is not able to learn the true relationship between the attributes and the output variable, and overfitting is when a model captures too much specific patterns in the training data to the extent that it performs poorly on unseen data[2]. In order to avoid these problems, the sklearn GridSearchCV with 10-fold cross validation was used to tune the model hyper-parameters:

Model	Parameters	Description
Logistic Regression	C = 0.01, 0.1, 1, 10, 100	Smaller C means the model is more regularized, lowering the chance of overfitting and vice versa.
k-Nearest Neighbor	k = 1, 3, 5, 10	The number of neighbors (k) greatly affects kNN performance. If k is too small, the model is susceptible to overfitting and vice versa.
Decision Tree and Random Forest Classifier	No. of estimators = 10, 50, 100, 200 (RF only) max leaf nodes = 32, 64, 128, 256 max features = 5, 10, 20	The number of estimators specifies the number of trees to build in the forest. Higher number of trees can give better performance but slower to train, and vice versa. The max leaf nodes sets the maximum number of leaf nodes allowed in the model. Higher number of nodes increases the model complexity and the chance of overfitting, and vice versa. Lastly, the max features sets the maximum number of features provided to each tree in the forest.

A wide range of parameter values were tested to ensure the best parameter combinations were used. For each model, the GridSearchCV split the training data into 10 sets of approximately equal size. Then, for each parameter combination, it built the model 10 times (each time trained using nine sets and then evaluated on the last remaining set) and output the best f1-scores along with the best parameters. Subsequently, StratifiedKFold 20-fold cross validation was used to re-build the model 20 times using the whole training data and the best parameters, and then tested on the test data to output the final f1-scores. For completeness, the ROC (receiver operating characteristic) curve showing the true positive rate vs false positive rate, the AUC (the area under the ROC curve) along with the Confusion Matrix and Classification Report (showing a summary of the precision, recall and f1-score) were also produced.

The Wilcoxon signed-rank test

Once the f1-scores had been calculated, the stats.wilcoxon class from scipy was used to compare the models' f1-scores and test the null hypothesis. If the p-values from the Wilcoxon tests were lower than the significance level (0.01), it would suggest strong evidence to reject the null hypothesis[6]. Once it was ascertained that the recommended model (the one with the highest f1-score) was statistically better than the alternatives, it was then recommended to the bank to be used for future campaigns.

Results

The GridSearchCV and the StratifiedKFold 20-fold cross validation output the following best parameters and scores for each model (Note: class 0 = 'No' and class 1 = 'Yes'):

Model	Best Parameters	f1-score	Precision	Recall
Logistic Regression	C = 10	Class 0: 0.92 Class 1: 0.58	Class 0: 0.98 Class 1: 0.43	Class 0: 0.86 Class 1: 0.86
k-Nearest Neighbor	k = 1	Class 0: 0.93 Class 1: 0.53	Class 0: 0.96 Class 1: 0.45	Class 0: 0.90 Class 1: 0.66
Decision Tree	maximum leaf nodes = 256 maximum features = 20	Class 0: 0.94 Class 1: 0.64	Class 0: 0.98 Class 1: 0.51	Class 0: 0.90 Class 1: 0.85
Random Forest	number of estimators = 200 maximum leaf nodes = 256 maximum features = 10	Class 0: 0.94 Class 1: 0.65	Class 0: 0.98 Class 1: 0.51	Class 0: 0.89 Class 1: 0.88

Overall, all models produced high precision, recall and f1-scores when predicting 'No'. However, due to the imbalanced dataset, they all produced lower scores when predicting 'Yes'. The GridSearchCV chose

C=10 as the optimal regularization parameter for the Logistic Regression. A high C tends to lower the regularization strength, allowing the model to increase its complexity and overfit the data. However, in this case, the Logistic Regression seemed to perform reasonably well. It had good recall value for class 1 (0.86), which means it was able to detect 86% of the actual 'Yes' in the data. However, the precision was quite low at 0.43, which means more than 50% of the 'Yes' prediction was incorrect. Of the four models considered, kNN produced the lowest precision, recall and f1-score. As mentioned, kNN is susceptible to high dimensionality, so even after the number of features was reduced to eight using PCA, the resulting model remained unsatisfactory for this dataset (this could also be exacerbated by the use of k=1 as the optimal parameter, which might cause the model to overfit the training data). For Decision Tree and Random Forest, the GridSearchCV chose the highest max leaf nodes of 256 as the optimal parameter. Whilst this might increase the chance of overfitting, they both seemed to perform reasonably well here. The Decision Tree used 20 features to fit the best model, whereas Random Forest used a combination of 200 Trees (higher number of trees usually gives better performance though slower to train) and 10 features. As a result, the Random Forest produced the highest f1-score score (0.65), with the Decision Tree model came close in second place (0.64). As expected, a group of Decision Trees in Random Forest performed better than a single Tree. It was able to detect 88% of the Yes' in the data (compared to 85% only in a single Tree) and more than 50% of the 'Yes' predictions were correct.

The question remained however, whether the Random Forest's f1-score was statistically better than the other models. The table below shows the p-values from Wilcoxon signed-rank test:

	vs. Logistic Regression	vs. k-Nearest Neighbor	vs. Decision Tree
Random Forest	8.8574e-05	8.8574e-05	0.0001

All the p-values were lower than 0.01, suggesting strong evidence to reject the null hypothesis. In other words, the evidence suggested that the recommended Random Forest model's f1-score was statistically better than the other three models. At a significance level of 0.01, there was only a 1% chance that the null hypothesis was incorrectly rejected. This indicates the strong reliability of this experiment.

Despite the good outcome, the precision (0.51) and the f1-score (0.65), even for the recommended model, were still relatively low when predicting 'Yes'. This is because the dataset was imbalanced (only 11% customers in the dataset subscribed to a term deposit). Whilst this situation is quite realistic (in practice, many customers would not usually sign up when contacted in a campaign), there are ways that we believe can be done to improve the outcome of this experiment: (1) use larger dataset so the model can learn from more examples; (2) use more relevant and less correlated attributes, such as customer's income and expenses; or (3) use more complex model such as neural network. Typically, it can produce better performance, but, sometimes the model produced by neural network can be harder to interpret.

Conclusion

In this project, end-to-end data science process was performed. In stage one, the business requirement and the dataset were explored and analyzed, and then the dataset was pre-processed for the modelling. In stage two, four machine learning models were trained to predict customer response. The chosen Random Forest model produced reasonable f1-score at 0.65 and the Wilcoxon test confirmed this was statistically better than the alternatives. The model was able to detect 88% of the 'Yes' in the test data and more than 50% of the 'Yes' predictions were correct. Considering the imbalanced dataset, this was a really good outcome and should be recommended to the bank to be implemented in future campaigns.

I find this project very useful and practical. Overall, I feel I have learned and improved a lot by doing this project, and I am excited to implement what I have learned here to other subjects and in real life.

References

1. Bank Marketing Data Set, UCI Machine Learning Repository, viewed 14 March 2020, https://archive.ics.uci.edu/ml/datasets/Bank+Marketing
2. Geron A 2019, Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow, O'Reilly
3. "Wilcoxon signed-rank test", Wikipedia, viewed 28 April 2020, https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test
4. "Student's t-test", Wikipedia, viewed 28 April 2020, https://en.wikipedia.org/wiki/Student%27s_t-test
5. Tan, Steinbach, Karpatne, Kumar 2020, Introduction to Data Mining (Global Edition), Pearson NY
6. Grus J 2019, Data Science from Scratch, O'Reilly

Appendix A – Examples of Model Outputs

Best Parameters and Scores from GridSearchCV

```
Parameter grid:
{'C': [0.01, 0.1, 1, 10, 100]}
Test set score: 0.59
Best parameters: {'C': 10}
Best cross-validation score: 0.88
Best estimator:
LogisticRegression(C=10, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=1000, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)

Parameter grid:
{'n_neighbors': [1, 3, 5, 10]}
Test set score: 0.35
Best parameters: {'n_neighbors': 1}
Best cross-validation score: 0.90
Best estimator:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')

Parameter grid:
{'n_estimators': [10, 50, 100, 200], 'max_leaf_nodes': [32, 64, 128, 256], 'max_features': [5, 10, 20]}
Test set score: 0.65
Best parameters: {'max_features': 10, 'max_leaf_nodes': 256, 'n_estimators': 200}
Best cross-validation score: 0.93
Best estimator:
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balanced',
                       criterion='gini', max_depth=None, max_features=10,
                       max_leaf_nodes=256, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Confusion Matrix and Classification Reports showing final scores

Logistic Regression

```
Logistic Regression:

Confusion Matrix:
[[1545  253]
 [  31  194]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.86      0.92      1798
         1.0       0.43      0.86      0.58       225

    accuracy                           0.86      2023
   macro avg       0.71      0.86      0.75      2023
weighted avg       0.92      0.86      0.88      2023


roc-auc score: 0.86
```

k-Nearest Neighbor

```
k-Nearest Neighbor:

Confusion Matrix:
[[1613  185]
 [  76  149]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      0.90      0.93      1798
         1.0       0.45      0.66      0.53       225

    accuracy                           0.87      2023
   macro avg       0.70      0.78      0.73      2023
weighted avg       0.90      0.87      0.88      2023


roc-auc score: 0.78
```

Decision Tree

```
Decision Tree Classifier:

Confusion Matrix:
[[1614  184]
 [  34  191]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.90      0.94      1798
         1.0       0.51      0.85      0.64       225

    accuracy                           0.89      2023
   macro avg       0.74      0.87      0.79      2023
weighted avg       0.93      0.89      0.90      2023


roc-auc score: 0.87
```

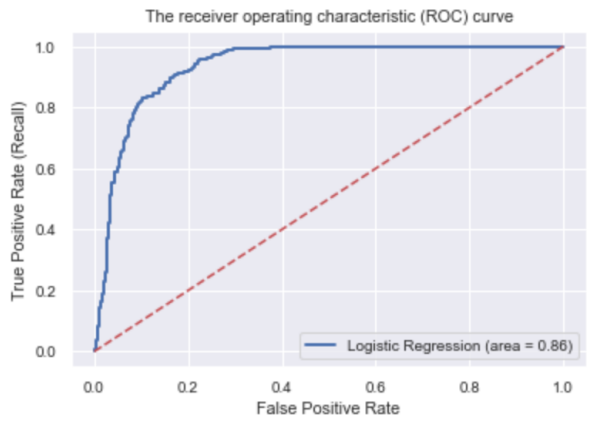
Random Forest

```
Random Forest Classifier:

Confusion Matrix:
[[1608  190]
 [  27  198]]

Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.89      0.94      1798
         1.0       0.51      0.88      0.65       225

    accuracy                           0.89      2023
   macro avg       0.75      0.89      0.79      2023
weighted avg       0.93      0.89      0.90      2023


roc-auc score: 0.89
```

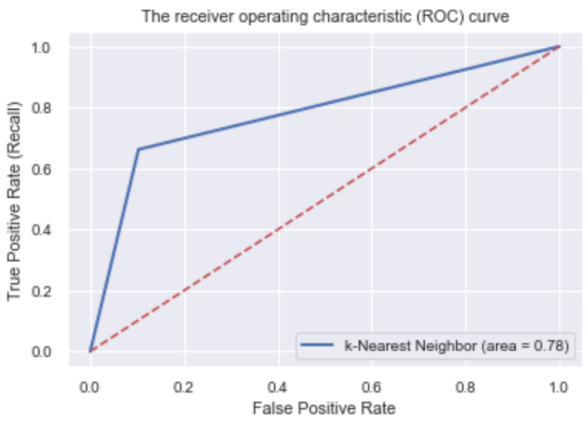
Notice the Random Forest produced the highest f1-score score, precision and recall. It was able to detect 88% of the Yes' in the data and more than 50% of the 'Yes' prediction was correct.

<u>Receiver Operating Characteristic (ROC) Curve</u>

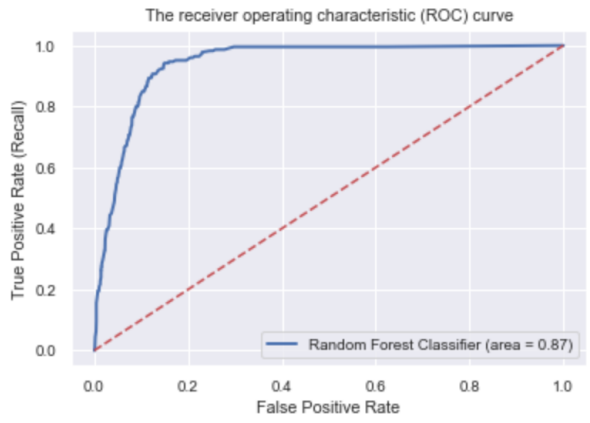
Logistic Regression



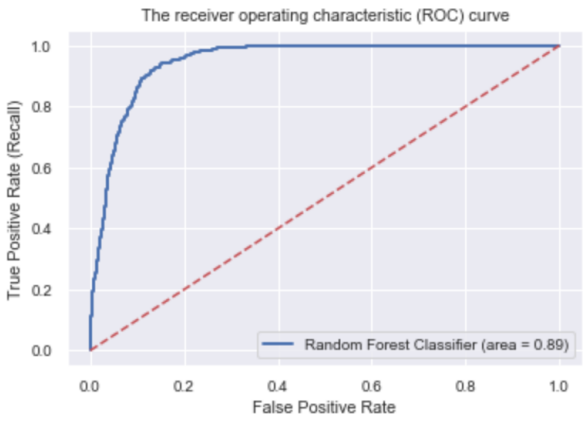
k-Nearest Neighbor



Decision Tree



Random Forest



Notice the Random Forest produced the highest true positives and true negatives, edging closer to the top-left corner of the chart, followed closely by Decision Tree, Logistic Regression and kNN, consistent with the highest f1-scores produced by these models.

Appendix B - Examples of Python code

Example 1. Applying GridSearchCV to tune the model hyper-parameters

```
# using grid search to find the best parameters for our logistic regession model
param_grid = {'C': [0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(param_grid))

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
grid_search_lr = GridSearchCV(LogisticRegression(max_iter=1000,class_weight='balanced',), param_grid, cv=10,
                          scoring='f1',return_train_score=True)
grid_search_lr.fit(X_train, y_train)

print("Test set score: {:.2f}".format(grid_search_lr.score(X_test, y_test)))
print("Best parameters: {}".format(grid_search_lr.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search_lr.best_score_))
print("Best estimator:\n{}".format(grid_search_lr.best_estimator_))

# save grid search result
from sklearn.externals import joblib
joblib.dump(grid_search_lr.best_estimator_, 'grid_search_lr.pkl')
```

Example 2. Using StratifiedKFold 20-fold cross validation to re-build the model 20 times using the whole training data and the best parameters, and then test them on the test data to output the final f1-scores

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression

kf = StratifiedKFold(n_splits=20,shuffle=True,random_state=42)
pred_test_full = 0
r_score_lr = []
f_score_lr = []

for train_index,test_index in kf.split(X,y):
    X_train, X_test = X.loc[train_index], X.loc[test_index]
    y_train, y_test = y.loc[train_index], y.loc[test_index]

    # SMOTE balancing on the training data
    oversample = SMOTE()
    X_train, y_train = oversample.fit_resample(X_train, y_train)

    #model
    lr = LogisticRegression(class_weight='balanced', C=10, max_iter=1000)
    lr.fit(X_train, y_train)
    y_pred_lr = lr.predict(X_test)
    rscore_lr = roc_auc_score(y_test, y_pred_lr)
    fscore_lr = f1_score(y_test, y_pred_lr)
    r_score_lr.append(rscore_lr)
    f_score_lr.append(fscore_lr)

print('ROC AUC score (Mean):', round(np.mean(r_score_lr),2))
print('ROC AUC score (Standard Deviation):', round(np.std(r_score_lr),2))

print('f1-score (Mean):', round(np.mean(f_score_lr),2))
print('f1-score (Standard Deviation):', round(np.std(f_score_lr),2))
```

Example 3. Using the Wilcoxon test to compare the models' f1-scores and test the null hypothesis

```
# logistic regression vs random forest classifier
# paired student's t-test
student_ttest_f2 = stats.ttest_rel(f_score_lr, f_score_rfc).pvalue*0.5
student_ttest_r2 = stats.ttest_rel(r_score_lr, r_score_rfc).pvalue*0.5
# Wilcoxon signed-rank test
wilcoxon_test_f2 = stats.wilcoxon(f_score_lr, f_score_rfc).pvalue
wilcoxon_test_r2 = stats.wilcoxon(r_score_lr, r_score_rfc).pvalue

print("Paired Student's t-test (f1-score):", student_ttest_f2)
print("Wilcoxon Signed-Rank test (f1-score):", wilcoxon_test_f2)

print("Paired Student's t-test (roc-auc-score):", student_ttest_r2)
print("Wilcoxon Signed-Rank test (roc-auc-score):", wilcoxon_test_r2)
```

See the python code submission for complete coverage of the codes