Data Glacier

Your Deep Learning Partner

Bank Marketing Strategy

<Improve Targeting Efficiency>

<2023.2.10>

Agenda

- Executive Summary
- Problem Statement
- Approach
- Model Selection
- Model Tuning & Validation
- Metric Evaluation & Result

Executive Summary

- Clean and preprocess the dataset for machine learning
- Select machine learning models to trial
- Build pipelines to deal with scale of the features & class imbalance
- Feature reduction & hyperparameter tuning
- Result evaluation

Problem Statement

- This project aims to fix the problem that there are too many potential clients associate with a Portuguese banking institution. The bank marketing campaigns are primarily based on phone calls. However, there are too many potential clients, and it is impossible to call all of them. Hence, some machine learning techniques are needed to classify the customers and predict if they are going to subscribe the term deposit or not. By implementing ML models, we are able to limit the number of ‘potential clients’ to an acceptable number that make sure the marketing strategies can be carried on.

Approach

- Data Loaded & Cleaning & Preprocessing
- Data Imputation
  - Simple imputer
  - Iterative imputer
  - KNN imputer

```
x_iter = IterativeImputer(n_nearest_features=50).fit_transform(x)
x_iter_df = pd.DataFrame(x_iter)
```

```
x_iter_df.head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	58.0	2143.0	5.0	1.0	-1.0	0.0	-0.001759	93.460615	-37.686446	3.690600	5166.507093
1	44.0	29.0	5.0	1.0	-1.0	0.0	0.013415	93.471578	-38.045953	3.665590	5165.584233
2	33.0	2.0	5.0	1.0	-1.0	0.0	-0.012448	93.451250	-38.446562	3.637786	5166.385384
3	47.0	1506.0	5.0	1.0	-1.0	0.0	0.008979	93.502296	-38.150717	3.642176	5163.373385
4	33.0	1.0	5.0	1.0	-1.0	0.0	-0.024682	93.495518	-37.526338	3.675858	5162.429888

```
x_simple = SimpleImputer(strategy='mean').fit_transform(x)
pd.DataFrame(x_simple).head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	58.0	2143.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487
1	44.0	29.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487
2	33.0	2.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487
3	47.0	1506.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487
4	33.0	1.0	5.0	1.0	-1.0	0.0	0.081922	93.57572	-40.502863	3.621293	5167.03487

Approach

- Machine Learning Algorithms
- Feature engineering & Hyperparameter tuning
- Result explanation

Logistic Regression

Most widely used binary classification linear technique

L1 Penalized

Very good precision of 96%

```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.7864857380227084
```

```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.9589578872234118
```

```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.8037690696978762
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[2687  656]
 [ 115  153]]
```

Logistic Regression

- **RFE instead of L1**
- **Another way of feature reduction**
- **Very good in all of the metrics**

```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.9277208529493215
```

```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.9331084879145587
```

```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.993119521387974
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3320   23]
 [ 238   30]]
```

KNN

- **Make decision based on the nearest neighbors given a certain distance metric**
- **Especially good in precision of 93.4%**

```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.910828025477707
```

```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.934176487496407
```

```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.9721806760394855
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3250   93]
 [ 229   39]]
```

Random Forest

- **Bagging Tree**
- **Ensembled Method**
- **Improve learning once**
- **Low accuracy but high precision**

```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.875934644142896 7
```

```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.957057151878749 6
```

```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.906670655100209 4
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3031  312]
 [ 136  132]]
```

XGBoost

- **Ensemble Method**
- **Boosting Method**
- **Improve learning sequentially**
- **Not an ideal model for this case**

```
acc = accuracy_score(y_test_num, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.8726114649681529
```

```
pre = precision_score(y_test_num, y_pred)
print(f'The precision score is {pre}')
```

```
The precision score is 0.281818181818181 8
```

```
rec = recall_score(y_test_num, y_pred)
print(f'The recall score is {rec}')
```

```
The recall score is 0.462686567164179 1
```

```
cm = confusion_matrix(y_test_num, y_pred)
print(cm)
```

```
[[3027  316]
 [ 144  124]]
```

SVM

- **Kernel based method**
- **RBF means KNN**
- **Only use influential data points**
- **Long training time, require more computation resource**

```
acc = accuracy_score(y_test, y_pred)
print(f'The accuracy score is {acc}')
```

```
The accuracy score is 0.929824425366934
```

```
pre = precision_score(y_test, y_pred,  pos_label='no')
print(f'The precision score is {pre}')
```

```
The precision score is 0.935665914221 2189
```

```
rec = recall_score(y_test, y_pred,  pos_label='no')
print(f'The recall score is {rec}')
```

```
The recall score is 0.991923422075 9796
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[3316   27]
 [ 228   40]]
```

Summary

- According to the models above and the metrics we used, logistic regression with recursive feature elimination and support vector machine with RBF kernel had the greatest accuracy while considerable precision/recall score. When converting these ML metrics into business metrics, this filtering model that select potential clients could significantly improve our targeting efficiency. Reducing the number of targeted clients while still convincing a lot of them to subscribe, the targeting efficiency could be boosted.

Results

- **Optimal Models: Logistic Regression (RFE Based), KNN, SVM**
- **Precision-weighted: Logistic Regression (l1-penalized), Random Forest**
- **Machine Learning is just AI model that help people make decision. Not even one model is ‘correct’, but some of them are useful.**

Thank You