main

March 2, 2025

Data Preprocessing

I conducted the exploratory analysis (EDA) in R (see eda.Rmd), and this part of the notebook deals with preparing the data using what I found from the EDA.

[1]:
```python
from src.data_main import Data

data = Data();
```

Creating an instance of the Data class reads the data csv.

[2]:
```python
data._Data__convert_response();
```

```
C:\Users\ivori\Documents\id5059\deposits-predictor\src\data_main.py:49:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  self.data["response"] = self.data["y"].replace({"no": 0, "yes": 1});
```

I convert the response variable to boolean.

[3]:
```python
data._Data__create_day_ids();
```

The EDA revealed this data has a significant temporal component. Given the data is ordered and I have days of week, I can create a helper column called “new_day” to mark changes in the day of week, and cumulatively sum over it to get a “day_id” or the number of days elapsed since data collection started.

[4]:
```python
data._Data__convert_to_categorical();
```

I convert all string columns to categorical values. The EDA identified that some numerical variables would be inappropriate to represent as continuous (i.e previous), so I convert them to categorical as well.

[5]:
```python
data._Data__merge("loan", "housing");
data._Data__merge("poutcome", "previous");
```

EDA identified that these columns need to be merged for linear models because some levels of this category are perfectly multcolinear with each other.

[6]:
```
data._Data__bin_continuous();
data._Data__bin_categorical();
```

```
C:\Users\ivori\Documents\id5059\deposits-predictor\src\data_main.py:119:
FutureWarning: The behavior of Series.replace (and DataFrame.replace) with
CategoricalDtype is deprecated. In a future version, replace will only be used
for cases that preserve the categories. To change the categories, use
ser.cat.rename_categories instead.
  self.data["default_group"] = self.data["default"].replace({"unknown":
"unknown_or_yes", "yes": "unknown_or_yes"});
```

EDA showed densities of certain non-linear continuous variables are neatly seperated by class at some thresholds (age, pdays, campaign) so linear models can use them to seperate between classes. Similarly, certain categoricals (default) have some unecessary levels that can be removed to reduce their complexity and standard error.

[7]:
```
data._Data__encode_categorical();
```

I perform one-hot encoding of all categorical variables to make it easier for models to interpret.

[8]:
```
data._Data__remove_unfair_predictors();
```

Some predictors (duration) are only known after a recording is complete, so need to be removed.

[9]:
```
data._Data__split_data();
```

The EDA identified a significant temporal component to the data. To produce genuine predictions in the future, a successful model needs to be able to infer temporal patterns in the data using day_id. I choose to split the data based on day_id. The idea here is to train the model on a lot of historical data with different temporal patterns (e.g the strategy shifts identified in EDA) so the model learns these different patterns, then test them on the most recent data. Given the most recent days contain the fewest records, a balance needs to be struck between showing the model enough of the most recent temporal pattern and restricting the size of the training data to reduce overfitting.

[10]:
```
data.split_day
```

[10]:
```
np.float64(260.0)
```

[11]:
```
data.test_prop
```

[11]:
```
0.06958337379819365
```

[12]:
```
data.train_prop
```

[12]:
```
0.9304166262018063
```

This proportion of training is higher than normal (0.8), so models trained with this split have a tendancy to overfit to the training data, but is necessary for the model to learn about the most recent data.

```
[13]: data.data["response"].value_counts()
```

```
[13]: response
      False    36548
      True      4640
      Name: count, dtype: int64
```

The data as a whole has a class imbalance, but EDA identified that the class balance becomes more equal for more recent observations. Data split using the above splitting schema will have different balances for training and testing data, so I oversample the True class in the training data so a) it resembles the split of the testing data and b) it learns characteristics of True at the same level as False.

Linear models (e.g GAM fitted during EDA) sometimes work best with non-linear continuous by binning and need some adjustments to avoid perfect multicolinearity, whereas machine learning models (e.g DecisionTree, RandomForest, SGD) work best by inferring the bins themselves and need no such adjustments. I create two sets of data: “sensitive” for linear models and “insensitive” for ML.

```
[14]: data.insensitive_train_X.columns
```

```
[14]: Index(['age', 'campaign', 'pdays', 'emp.var.rate', 'cons.price.idx',
             'cons.conf.idx', 'euribor3m', 'nr.employed', 'job_admin.',
             'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
             'job_management', 'job_retired', 'job_self-employed', 'job_services',
             'job_student', 'job_technician', 'job_unemployed', 'job_unknown',
             'marital_divorced', 'marital_married', 'marital_single',
             'marital_unknown', 'education_basic.4y', 'education_basic.6y',
             'education_basic.9y', 'education_high.school', 'education_illiterate',
             'education_professional.course', 'education_university.degree',
             'education_unknown', 'default_no', 'default_unknown', 'default_yes',
             'housing_no', 'housing_unknown', 'housing_yes', 'loan_no',
             'loan_unknown', 'loan_yes', 'contact_cellular', 'contact_telephone',
             'month_apr', 'month_aug', 'month_dec', 'month_jul', 'month_jun',
             'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep',
             'day_of_week_fri', 'day_of_week_mon', 'day_of_week_thu',
             'day_of_week_tue', 'day_of_week_wed', 'previous_0', 'previous_1',
             'previous_2', 'previous_3', 'previous_4', 'previous_5', 'previous_6',
             'previous_7', 'poutcome_failure', 'poutcome_nonexistent',
             'poutcome_success'],
            dtype='object')
```

```
[15]: data.sensitive_train_X.columns
```

```
[15]: Index(['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m',
             'nr.employed', 'job_admin.', 'job_blue-collar', 'job_entrepreneur',
             'job_housemaid', 'job_management',
             …
```

```
        'campaign_group_2', 'campaign_group_3', 'campaign_group_4',
        'campaign_group_5', 'campaign_group_6', 'campaign_group_7',
        'campaign_group_8', 'campaign_group_9+', 'default_group_no',
        'default_group_unknown_or_yes'],
       dtype='object', length=126)
```

Finally, I create validation sets from a random sample of 20% of the training data to tune hyper-parameters and decision thresholds with.

Modelling and Evaluation

[16]:
```python
from src.models_main import Models

models = Models(data);
```

Creating an instance of the Models class initialises the three classifier I chose: a Stochastic Gradient Descent (SGD), a Decision Tree (DT), and a Random Forest (RF).

Every model I chose has "hyperparameters", or characteristics about the model that are not changed when trained on different data.

[17]:
```python
"""
Print a given list of objects in a user-friendly format
@param list_dict: list of dictionaries
"""
def print_list(list_dict):
    for dictionary in list_dict:
        print(dictionary);
```

[18]:
```python
print_list(
    models.get_param_grids()
);
```

```
{'SGD': {'estimator__loss': ['hinge', 'log_loss', 'modified_huber'],
'estimator__penalty': ['l2', 'l1', 'elasticnet'], 'estimator__alpha': [0.0001,
0.001, 0.01], 'estimator__max_iter': [1000, 2000], 'estimator__tol': [0.001,
0.0001], 'method': ['sigmoid'], 'cv': [5], 'estimator__random_state': [42]}}
{'Decision Tree': {'max_depth': [None, 20, 40, 60], 'max_leaf_nodes': [None, 20,
50, 100], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4],
'criterion': ['gini', 'entropy']}}
{'Random Forest': {'n_estimators': [100, 200, 300], 'max_depth': [None, 10, 20,
30], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]}}
```

I search through every possible combination of these hyperparameters (GridSearchCV) and choose the combination of hyperparameters that produce a model with the highest f1 score when trained on the validation set.

[19]:
```python
models.tune(load_path = "src/hyperparameters.json");

print_list(
    models.get_params()
);
```

```
{'SGD': {'cv': 5, 'estimator__alpha': 0.0001, 'estimator__loss': 'hinge',
'estimator__max_iter': 1000, 'estimator__penalty': 'elasticnet',
'estimator__random_state': 42, 'estimator__tol': 0.0001, 'method': 'sigmoid'}}
{'Decision Tree': {'criterion': 'entropy', 'max_depth': None, 'max_leaf_nodes':
None, 'min_samples_leaf': 1, 'min_samples_split': 2}}
{'Random Forest': {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}}
```

I train the models using these hyperparameters. All chosen models are trained on the "insensitive" training sets and produce probabilities of "response" being True (including SGD, which I wrap in a Platt scaler to convert distance to the decision boundary to a probability).

[20]:
```
models.train();
```

```
100%|                          | 3/3 [00:18<00:00,  6.18s/it]
```

Once trained, I select a "decision threshold", above which a produced probability prediction will be classed as "True", to maximise F1.

[21]:
```
print_list(
    models.get_thresholds()
);
```

```
{'SGD': np.float64(0.0)}
{'Decision Tree': np.float64(0.01)}
{'Random Forest': np.float64(0.47000000000000003)}
```

Note the strangely low thresholds of SGD and DT.

[22]:
```
models.evaluate();
```

Hyperparameters and decision thresholds have been selected to maximise F1, and I evaluate my models using F1 and an ROC curve. I choose F1 because a false-positive is equally bad as a false-negative, and these metrics are robust to our class imbalance (e.g unlike Youden's J).

[23]:
```
print_list(
    models.get_f1s()
);

print_list(
    models.get_confusion_matrices()
);
```

```
{'SGD': np.float64(0.6740689336109184)}
{'Decision Tree': np.float64(0.4602692140686062)}
{'Random Forest': np.float64(0.44105011933174226)}
{'SGD': {'true_positive': np.int64(1457), 'false_positive': np.int64(1409),
'true_negative': np.int64(0), 'false_negative': np.int64(0)}}
```

```
{'Decision Tree': {'true_positive': np.int64(530), 'false_positive':
np.int64(316), 'true_negative': np.int64(1093), 'false_negative':
np.int64(927)}}
{'Random Forest': {'true_positive': np.int64(462), 'false_positive':
np.int64(176), 'true_negative': np.int64(1233), 'false_negative':
np.int64(995)}}
```

SGD's high F score is misleading as it always predicts "True" meaning it has failed. The remaining F1 scores are disappointing.

```
[24]: print_list(
          models.get_pred_ranges()
      );
```

```
{'SGD': [np.float64(0.1874011373959083), np.float64(0.7203909017508012)]}
{'Decision Tree': [np.float64(0.0), np.float64(1.0)]}
{'Random Forest': [np.float64(0.06), np.float64(0.93)]}
```

SGD does not produce predictions distributed across the entire probability space [0,1] unlike DT or (mostly) RF.

```
[25]: print_list(
          models.get_roc_integrals()
      );
```

```
{'SGD': np.float64(-0.07393706457717245)}
{'Decision Tree': np.float64(0.08048696125210984)}
{'Random Forest': np.float64(0.14037842751366153)}
```

From the two working models, DT has the higher performance (F1 score) the optimised threshold but RF has the highest performance across all possible thresholds (area underneath ROC curve).

```
[26]: print_list(
          models.get_precisions_recalls()
      );
```

```
{'SGD': [np.float64(0.5083740404745289), np.float64(1.0)]}
{'Decision Tree': [np.float64(0.6264775413711584),
np.float64(0.363761153054221)]}
{'Random Forest': [np.float64(0.7241379310344828),
np.float64(0.31708991077556625)]}
```

DT has a lower precision but a higher recall than RF at their optimal thresholds, but this situation weights false positives and false negatives equally so cannot be a basis for model selection. I choose the model with the highest harmonic mean between precision and recall, DT.

```
[27]: print_list(
          models.get_train_f1s()
      );

      print_list(
          models.get_train_confusion_matrices()
      );
```

```
{'SGD': np.float64(0.6666666666666666)}
{'Decision Tree': np.float64(0.9960457502444832)}
{'Random Forest': np.float64(0.9960457502444832)}
{'SGD': {'true_positive': np.int64(35139), 'false_positive': np.int64(35139),
'true_negative': np.int64(0), 'false_negative': np.int64(0)}}
{'Decision Tree': {'true_positive': np.int64(35139), 'false_positive':
np.int64(279), 'true_negative': np.int64(34860), 'false_negative': np.int64(0)}}
{'Random Forest': {'true_positive': np.int64(35139), 'false_positive':
np.int64(279), 'true_negative': np.int64(34860), 'false_negative': np.int64(0)}}
```

SGD also predicted "yes" naively during training, but the other models have nearly perfectly overfitted to the data.

[28]:
```
# Retrain models on data without days
data_no_day = Data();
data_no_day.preprocess(remove_day_ids = True);
models_no_day = Models(data_no_day);
models_no_day.tune(load_path = "src/no_day_hyperparameters.json");
models_no_day.train()
```

```
C:\Users\ivori\Documents\id5059\deposits-predictor\src\data_main.py:49:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  self.data["response"] = self.data["y"].replace({"no": 0, "yes": 1});
C:\Users\ivori\Documents\id5059\deposits-predictor\src\data_main.py:119:
FutureWarning: The behavior of Series.replace (and DataFrame.replace) with
CategoricalDtype is deprecated. In a future version, replace will only be used
for cases that preserve the categories. To change the categories, use
ser.cat.rename_categories instead.
  self.data["default_group"] = self.data["default"].replace({"unknown":
"unknown_or_yes", "yes": "unknown_or_yes"});
100%|                      | 3/3 [00:36<00:00, 12.07s/it]
```

[29]:
```
# Check that models have no access to day_id
try:
    print(models.models[0].train_X["day_id"]);
except KeyError:
    print("day_id removved successfully");
```

```
day_id removved successfully
```

[33]:
```
# Evaluate no_day models
models_no_day.evaluate()
print_list(
    models_no_day.get_f1s()
)
print_list(
    models_no_day.get_train_f1s()
)
print_list(
    models_no_day.get_confusion_matrices()
)
```

```
{'SGD': np.float64(0.6740689336109184)}
{'Decision Tree': np.float64(0.45021645021645024)}
{'Random Forest': np.float64(0.5445887445887446)}
{'SGD': np.float64(0.6666666666666666)}
{'Decision Tree': np.float64(0.9960457502444832)}
{'Random Forest': np.float64(0.9960457502444832)}
{'SGD': {'true_positive': np.int64(1457), 'false_positive': np.int64(1409),
'true_negative': np.int64(0), 'false_negative': np.int64(0)}}
{'Decision Tree': {'true_positive': np.int64(520), 'false_positive':
np.int64(333), 'true_negative': np.int64(1076), 'false_negative':
np.int64(937)}}
{'Random Forest': {'true_positive': np.int64(629), 'false_positive':
np.int64(224), 'true_negative': np.int64(1185), 'false_negative':
np.int64(828)}}
```

SGD without day ID still fails. RF and DT without day IDs still overfit extremely in training, but produces better testing performance than including them. Random forest becomes the better model.