model_build

August 5, 2025

step 2: prediction model building

The following imports are done because all previous functions were copied to python script files for easier navigation between notebooks, in this second step we will focus on using the data we cleaned to model predictions of the 'risk_score' parameter.

[107]:
```python
from src.pipeline import pipeline
from src.map import data_mapping
from src.preprocess import num_data_prep, drop_useless
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import joblib
import pandas as pd
import numpy as np
```

[3]:
```python
data_path = '../data/raw/data_chunck.parquet'
```

[46]:
```python
df = pd.read_parquet(data_path).sample(n=50_000, random_state=2)

df = data_mapping(df)
df = num_data_prep(df)
df = drop_useless(df)

df_y = df['risk_score']
df =df.drop(columns='risk_score')

print(df.shape)
print(df_y.shape)

X_train, X_val, y_train, y_val = train_test_split(df, df_y)
```

```
number of cols to drop: 26
data shape before dropping: (36856, 79)
26 columns dropped successfuly
data shape after dropping: (36856, 67)
(36856, 66)
```

```
(36856,)
```

0.1 Introduction to model building

Let's get started with model building, as we are dealing with very high dimensionality data, a lot of non linearity and mixed-type features, the optimal choice for us will probably be a tree based model, We will use a Random Forest Regressor, which is an application of bagging to decision trees with the additional "random subspace" trick: we create a large number of trees and each tree is trained on a bootstrap sample of the data chosen randomly with replacement, and at each split only a portion of the features are randomly chosen without replacement, generally $\sqrt{n}$ features for classification or $n/3$ for regression for n features.

This will save a ton of computing power while reducing the variance of our model.

In fact, if each tree result is a random variable, let's say we have n random variables X_i equally distributed but not necessarily independant, our model output $\tilde{X}$ would then be the average of all the results of the different trees: $\tilde{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ if we let $Var(X_i) = \sigma^2$ for all i and $Cov(X_i, X_j) = \rho\sigma^2$ for all $i \neq j$ (ρ being the coeificient of correlation),

Then, by a simple variance calculation we have that,

$$Var(\tilde{X}) = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

As n grow larger, $\frac{1-\rho}{n}\sigma^2 \xrightarrow[n\to\infty]{} 0$, we are then left with $Var(\tilde{X}) = \rho\sigma^2 \leq \sigma^2$ as $\rho \leq 1$.

So, the more trees we have, the more we reduce our model's output variance, this variance reduction is crucial for improving model stability, especially when evaluating the model across different economic scenarios to assess its robustness.

0.2 Additional Note:

- In classification, the Random Forest aggregates by majority voting (which is equivalent to averaging the 0/1 votes and then thresholding) or by averaging the class probabilities. The variance formula applies to the probability estimates, which can then be thresholded to get the class label. This variance reduction in the probability estimates leads to more stable predictions.

[3]:
```python
#let's start with the baseline sklearn model
rf = RandomForestRegressor(random_state=42, n_jobs=-1)
pipe_rf = pipeline(rf)

pipe_rf.fit(X_train, y_train)
print('fitting done')
y_pred = pipe_rf.predict(X_val)
print('predictions computed')
print("MAE:", mean_absolute_error(y_val, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_val, y_pred)))
print("R² Score:", r2_score(y_val, y_pred))
```

```
[ColumnTransformer] … (1 of 5) Processing scale, total=   1.0s
frequency data encoded successfuly
```

```
[ColumnTransformer]  (2 of 5) Processing frequency_cols, total=   0.1s
employement lenght encoded successfuly
[ColumnTransformer]  (3 of 5) Processing employement_lenght, total=   0.0s
Rows with invalid 'earliest_cr_line': 0
Remaining rows: 625254, earliest cr data encoded successfuly
[ColumnTransformer] … (4 of 5) Processing account_age, total=   0.1s
[ColumnTransformer] … (5 of 5) Processing cat_left, total=   0.5s
[Pipeline] … (step 1 of 2) Processing features, total=   2.1s
[Pipeline] … (step 2 of 2) Processing model, total=15.2min
frequency data encoded successfuly
employement lenght encoded successfuly
Rows with invalid 'earliest_cr_line': 0
Remaining rows: 208418, earliest cr data encoded successfuly
MAE: 0.070548273181778893
RMSE: 0.30444092574704473
R² Score: 0.9464900825847499
```

We now compare with a dummy model

[5]:

```
dummy = DummyRegressor(strategy='mean')             # or 'median'
pipe_dummy = pipeline(dummy)
pipe_dummy.fit(X_train, y_train)


y_dummy_pred = pipe_dummy.predict(X_val)


print("DUMMY MAE:",  mean_absolute_error(y_val, y_dummy_pred))
print("DUMMY RMSE:", np.sqrt(mean_squared_error(y_val, y_dummy_pred)))
print("DUMMY R²:",   r2_score(y_val, y_dummy_pred))
```

```
[ColumnTransformer] … (1 of 5) Processing scale, total=   1.6s
frequency data encoded successfuly
[ColumnTransformer]  (2 of 5) Processing frequency_cols, total=   0.1s
employement lenght encoded successfuly
[ColumnTransformer]  (3 of 5) Processing employement_lenght, total=   0.1s
Rows with invalid 'earliest_cr_line': 0
Remaining rows: 625254, earliest cr data encoded successfuly
[ColumnTransformer] … (4 of 5) Processing account_age, total=   0.1s
[ColumnTransformer] … (5 of 5) Processing cat_left, total=   0.6s
[Pipeline] … (step 1 of 2) Processing features, total=   3.1s
[Pipeline] … (step 2 of 2) Processing model, total=   0.0s
frequency data encoded successfuly
employement lenght encoded successfuly
Rows with invalid 'earliest_cr_line': 0
Remaining rows: 208418, earliest cr data encoded successfuly
DUMMY MAE: 0.8851774609818346
DUMMY RMSE: 1.316092837674032
DUMMY R²: -2.9110487000938434e-06
```

We now check with cross validation

[7]:
```
scores = cross_val_score(pipe_rf, X_train, y_train, cv=5, scoring='r2',␣
    ↪n_jobs=-1)
print("CV R²:", scores, "Mean:", scores.mean())
```

```
CV R²: [0.9454963  0.94811823 0.94592827 0.94508505 0.94532304] Mean:
0.9459901781102609
```

The results on cross vqlidqtion are very consistent, training-set R^2 was ~0.9465 and CV R^2 mean is ~0.9460—almost identical.

Very little spread between folds means your model's performance isn't depending heavily on any one subset of data.

We now can start hyperparameters tuning, and as the last cells were really long to run, we will use a subset of the data as this doesn't really change what params are the best.

[ ]:
```
#we'll use 200k rows, I think it is the sweet spot between good results and␣
    ↪reasonable computing time
data_for_search = pd.read_parquet(data_path).sample(n=200_000, random_state=2)
#we run the prep functions
data_for_search = data_mapping(data_for_search)
data_for_search = num_data_prep(data_for_search)
data_for_search = drop_useless(data_for_search)
#take the target variable appart
data_for_search_y = data_for_search['risk_score']
data_for_search =data_for_search.drop(columns='risk_score')
```

```
number of cols to drop: 26
data shape before dropping: (147695, 79)
26 columns dropped successfuly
data shape after dropping: (147695, 67)
```

[97]:
```
#we track time for a single fit for a 2/3 of the data (grid search will be␣
    ↪using cross val)
rf = RandomForestRegressor(random_state=42, n_jobs=-1)
pipe_rf = pipeline(model=rf, ver=False)
X_train = data_for_search.sample(frac=0.66, random_state=2)
y_train = data_for_search_y.sample(frac=0.66, random_state=2)

import time
start = time.time()
pipe_rf.set_params(model__n_estimators=100, model__max_depth=20)
pipe_rf.fit(X_train,y_train)
print("One-fit time:", time.time() - start)
scores = cross_val_score(pipe_rf, data_for_search, data_for_search_y, cv=3,␣
    ↪scoring='r2', n_jobs=-1)
print(f"cv score : {scores.mean()}")
```

```
frequency data encoded successfuly
employement lenght encoded successfuly
Rows with invalid 'earliest_cr_line': 0
Remaining rows: 97479, earliest cr data encoded successfuly
One-fit time: 57.89259648323059
cv score : 0.9315072315016438
```

[98]:
```
cut = int(len(data_for_search_y)*0.66)
data_for_search_y.iloc[cut:].describe()
```

[98]:
```
count    50217.000000
mean         0.520023
std          1.327595
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          5.000000
Name: risk_score, dtype: float64
```

That means for a 4x3x3x2 grid we have 72 combinations, so, 576 x 3 seconds of runtime which is about 28.8 minutes

[99]:
```
for depth in [None, 10, 20]:
    for min_samples_split in [2, 5, 10]:
        model = RandomForestRegressor(
            max_depth=depth,
            min_samples_split=min_samples_split,
            n_estimators=100,
            random_state=42,
            n_jobs=-1
        )

        pipe = pipeline(model=model, ver=False)

        scores = cross_val_score(pipe, data_for_search, data_for_search_y,␣
 ↪cv=3, scoring='r2', n_jobs=-1)

        print(f"depth={depth}, min_samples_split={min_samples_split} → average␣
 ↪r2 R²={scores.mean():.4f}")
```

```
depth=None, min_samples_split=2 → average r2 R²=0.9313
depth=None, min_samples_split=5 → average r2 R²=0.9316
depth=None, min_samples_split=10 → average r2 R²=0.9314
depth=10, min_samples_split=2 → average r2 R²=0.9138
depth=10, min_samples_split=5 → average r2 R²=0.9138
depth=10, min_samples_split=10 → average r2 R²=0.9137
depth=20, min_samples_split=2 → average r2 R²=0.9315
depth=20, min_samples_split=5 → average r2 R²=0.9316
```

```
depth=20, min_samples_split=10 → average r2 R²=0.9314
```

best are depth=20 and min_samples_split=5 (depth=None gives the same same result but may use more computing power so we stick with 20)

[103]:
```python
for n in [50,100,150,200,250]:
    for max_features in ['sqrt', 'log2', None]:
        model = RandomForestRegressor(
            max_depth=20,
            min_samples_split=5,
            n_estimators=n,
            max_features=max_features,
            random_state=42,
            n_jobs=4
        )

        pipe = pipeline(model=model, ver=False)

        scores = cross_val_score(pipe, data_for_search, data_for_search_y,␣
 ↪cv=3, scoring='r2', n_jobs=-1)

        print(f"n_estimators={n}, max_features_per_tree={max_features} →␣
 ↪average r2 R²={scores.mean():.4f}")
```

```
n_estimators=50, max_features_per_tree=sqrt → average r2 R²=0.8784
n_estimators=50, max_features_per_tree=log2 → average r2 R²=0.8588
n_estimators=50, max_features_per_tree=None → average r2 R²=0.9310
n_estimators=100, max_features_per_tree=sqrt → average r2 R²=0.8800
n_estimators=100, max_features_per_tree=log2 → average r2 R²=0.8610
n_estimators=100, max_features_per_tree=None → average r2 R²=0.9316
n_estimators=150, max_features_per_tree=sqrt → average r2 R²=0.8807
n_estimators=150, max_features_per_tree=log2 → average r2 R²=0.8613
n_estimators=150, max_features_per_tree=None → average r2 R²=0.9317
n_estimators=200, max_features_per_tree=sqrt → average r2 R²=0.8808
n_estimators=200, max_features_per_tree=log2 → average r2 R²=0.8616
n_estimators=200, max_features_per_tree=None → average r2 R²=0.9318
n_estimators=250, max_features_per_tree=sqrt → average r2 R²=0.8808
n_estimators=250, max_features_per_tree=log2 → average r2 R²=0.8619
n_estimators=250, max_features_per_tree=None → average r2 R²=0.9318
```

So best combination is n_estimators=20, max_features_per_tree=10, depth=20, min_samples_split=5 So let's train a model on the full dataset with these parameters.

[104]:
```python
full_data = pd.read_parquet(data_path).sample(frac=1, random_state=1)

full_data = data_mapping(full_data)
full_data = num_data_prep(full_data)
full_data = drop_useless(full_data)
```

```
full_data_y = full_data['risk_score']
full_data =full_data.drop(columns='risk_score')

X_train, X_val, y_train, y_val = train_test_split(full_data, full_data_y, test_size=0.25, random_state=1)
```

```
number of cols to drop: 26
data shape before dropping: (1668529, 78)
26 columns dropped successfuly
data shape after dropping: (1668529, 66)
```

[106]:
```
rf_model = RandomForestRegressor(n_estimators=250, max_features=None,max_depth=20,min_samples_split=5, random_state=1, n_jobs=4)
final_pipe = pipeline(model=rf_model, ver=True)

final_pipe.fit(X_train, y_train)
```

```
[ColumnTransformer] … (1 of 5) Processing scale, total=   2.2s
frequency data encoded successfuly
[ColumnTransformer]  (2 of 5) Processing frequency_cols, total=   0.2s
employement lenght encoded successfuly
[ColumnTransformer]  (3 of 5) Processing employement_lenght, total=   0.1s
Rows with invalid 'earliest_cr_line': 0
Remaining rows: 1251396, earliest cr data encoded successfuly
[ColumnTransformer] … (4 of 5) Processing account_age, total=   0.2s
[ColumnTransformer] … (5 of 5) Processing cat_left, total=   0.8s
[Pipeline] … (step 1 of 2) Processing features, total=   4.2s
[Pipeline] … (step 2 of 2) Processing model, total=87.6min
```

[106]:
```
Pipeline(steps=[('features',
                 ColumnTransformer(transformers=[('scale', RobustScaler(),
<sklearn.compose._column_transformer.make_column_selector object at
0x000001B00747C0E0>),
                                                 ('frequency_cols',
FunctionTransformer(func=<function freq_encoding at 0x000001B03A57C9A0>),
                                                  ['purpose', 'addr_state']),
                                                 ('employement_lenght',
FunctionTransformer(func=<function emp_lenght…
FunctionTransformer(func=<function earliest_to_date at 0x000001B03A57BBA0>),
                                                  ['earliest_cr_line']),
                                                 ('cat_left', OneHotEncoder(),
                                                  ['term',
                                                   'debt_settlement_flag',
                                                   'initial_list_status',
                                                   'verification_status',
                                                   'home_ownership'])],
                                   verbose=True)),
                ('model',
```

```
                RandomForestRegressor(max_depth=20, max_features=None,
                                      min_samples_split=5, n_estimators=250,
                                      n_jobs=4, random_state=1))],
         verbose=True)
```

[109]:
```
joblib.dump(final_pipe,'../model/final_pipeline.joblib')
```

[109]:
```
['../model/final_pipeline.joblib']
```