Capstone Project

Data Science
Nanodegree

Benjamin Payne

June 6, 2022

Mortgage Underwriting AI Analyst

1. Project Definition

Project Overview

Using mortgage loan data from 2019, establish an AI-powered dashboard that consumes data points related to mortgage underwriting and provides analytics for predicted mortgage performance.

Problem Statement

Minimum guidance as well as FI overlays, provide strict guidelines for underwriting mortgage applications. However, outside of these guidelines, there could be additional insights into the performative nature of a mortgage based on Machine Learning techniques. The approaches used in this project help alleviate the problem of qualified mortgages becoming nonperforming by adding an AI / Machine Learning overlay to the underwriting process. The intent isn't to supersede the guidelines or FI overlays but to augment them.

Metrics

Because a false positive in this instance can be very costly, the Precision of the final model will be used to evaluate the performance. The below table is the evaluated performance using the classification report from Scikit learns metrics:

	Precision	recall	f1-score	support
0	0.93	0.94	0.94	33702
1	0.81	0.78	0.80	10899
accuracy			0.90	44601
macro avg	0.87	0.86	0.87	44601
weighted avg	0.90	0.90	0.90	44601

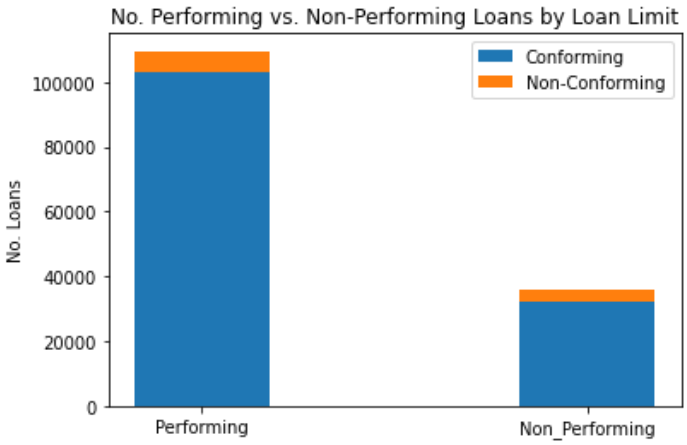
The average precision of 90% is enough for a proof of concept however for business-critical tasks it is too low. More data with better labeling would help increase this number to >95%.

2. Analysis

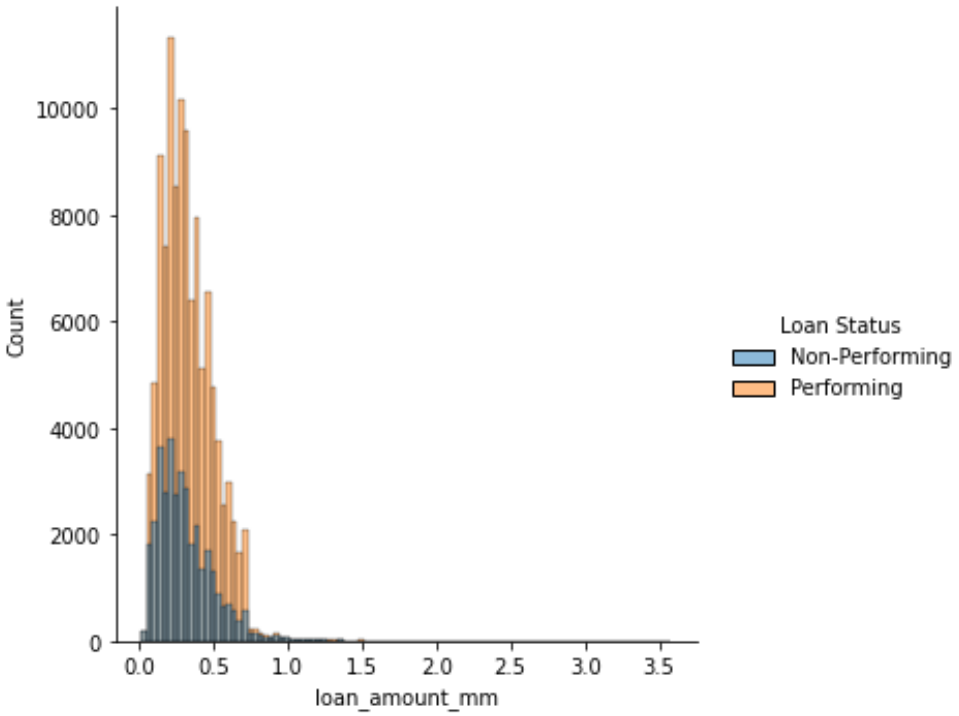
Data exploration and visualization

There are 34 feature vectors in the retail lending dataset. Although all vectors are used in the model, here we'll analyze just the key vectors used in underwriting.

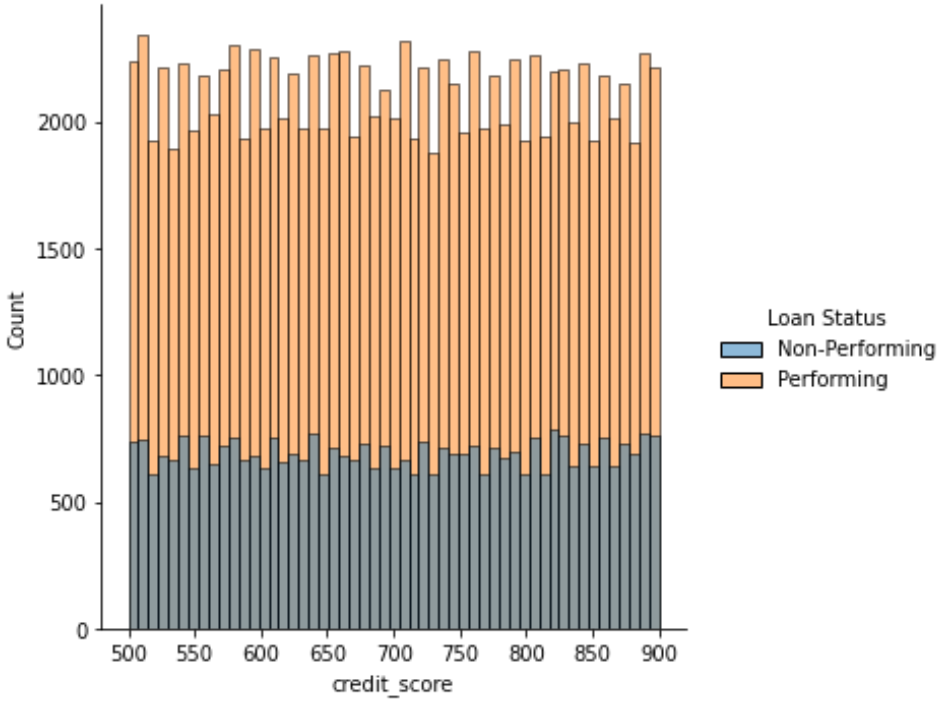
1. Loan limit – Conforming or Non-Conforming loan limit. Depending on which program and the FIs overlays loans over a certain limit are considered non-Conforming. Below is the breakdown of Conforming and Non-Conforming loans within the feature set:



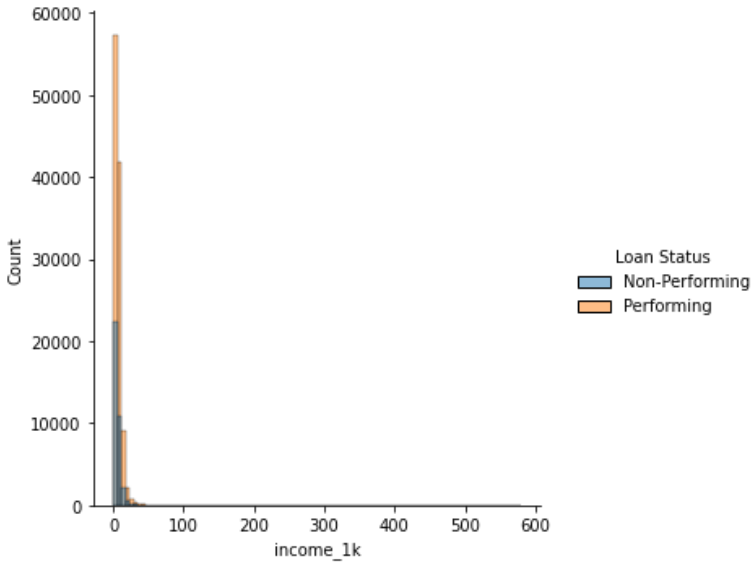
   a. Most files are performing, conforming loans with about equal distribution of conforming to non-conforming limits
2. Loan amount – Most loan amounts are below $1mm with the majority of those being between $100k and $500k. Performing vs non-performing grouping shows relatively the same distribution of loans between the two subsets indicating there is little correlation between loan amounts and performance.



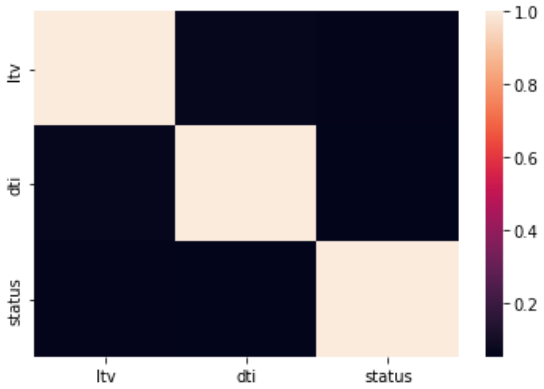
3. Credit Score – Credit also shows an even distribution between performing and non-performing loans with no real significant variance between the two subsets:



4. Income – Again with income, there is a similar distribution vs. performance:

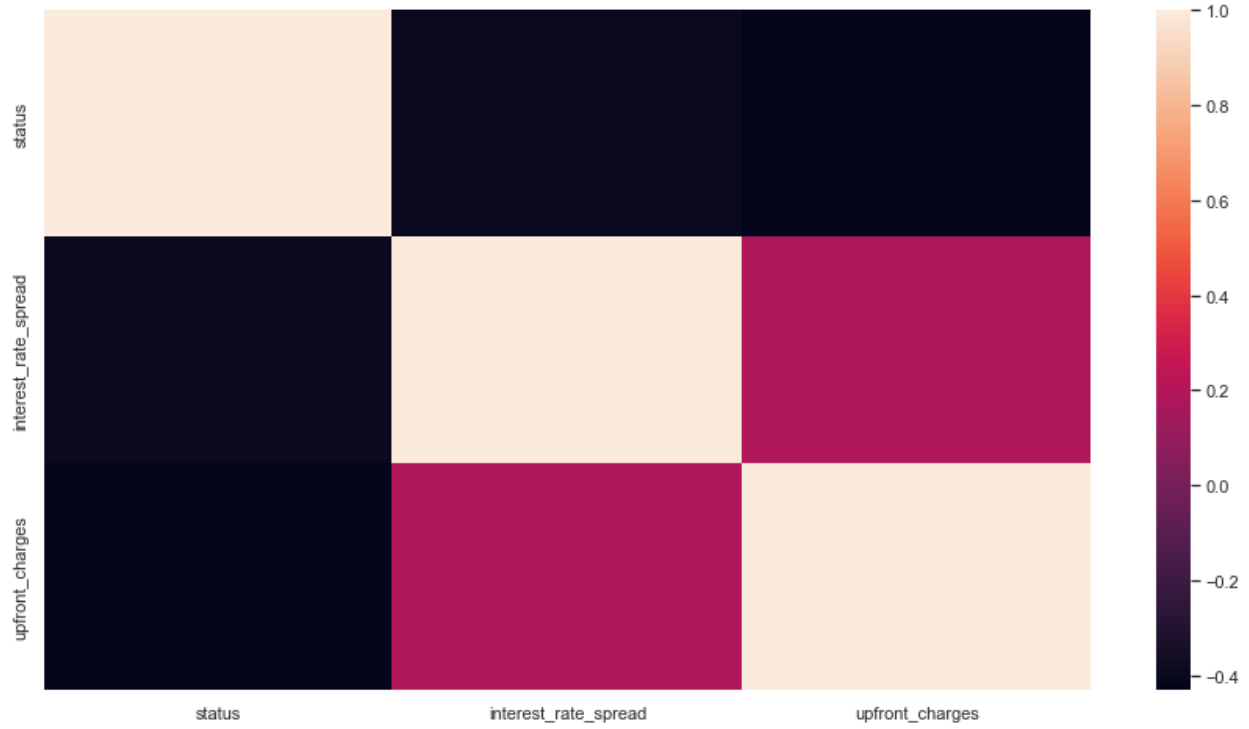


5. LTV and DTI – There also doesn't appear to be a strong correlation between DTI and LTV and Loan Status:



Further exploration

The strongest correlation between status and another vector is the interest rate spread and upfront charges amount:



This shows a strong negative correlation so as these vectors increase the more likely the loan is to be non-performant. Unfortunately, there is not enough information in this dataset to tie the interest rate spread and upfront charges to another vector to determine if a specific loan type (e.g. ARMs) or purpose is tied to a higher rate of default.

3. Methodology

Data preprocessing

Many vectors had incorrect data types that needed to be coerced and imputed to Int64 or floats to be properly modeled. The pre-processed data will be mapped to values in dropdowns in the corresponding web application that will send new data to the model to be evaluated and scored. Below is a breakdown of the pre-processing steps that were taken to clean up the dataset.

ID – This is a unique identifier column that is not necessary for the model.

Year – All data is from the year 2019 so this column was dropped as well.

Loan limit – Unique labels were cf, nan, and ncf. These were interpreted to mean Conforming, NaN, and Non-Conforming. Conforming and Non-Conforming were replaced by 1 and 2 respectively and NaN was changed to 0.

PreApproval – Unique labels were nopre, pre and NaN. These were interpreted to be No Preapproval, Preapproval and NaN. No Preapproval and Preapproval were replaced with 0 and 1 respectively and NaNs were also set to 0.

Loan type – There were no NaNs in the dataset, however, an impute step was added for future use. The 'type' prefix was dropped from the vectors and then coerced to an int64

Loan purpose – NaNs were imputed with 'p0' then the 'p' prefix was dropped from all labels and then coerced to int64.

Credit worthiness – Native values were l1 and l2. Without a data map, it was unclear what these values represented. There was no significant correlation found between this vector and status, so the column was dropped

Open credit – The native values were nopc and opc. These were interpreted to mean no open credit and open credit. There were no NaNs however an impute step was added for future use. Open credit was replaced with 1 and NaN and No Open Credit were replaced by 0.

Business or commercial – This column was renamed to commercial_loan. The native values of nob/c and b/c were replaced with 0 and 1 respectively. NaNs were imputed to 0.

Loan amount – This is a continuous vector with no NaNs so no pre-processing was needed.

Interest rate – This is a continuous vector and is required however there were 75,792 rows with missing interest rates. This was too many to drop so further exploration was needed. The interest rate description of each status sub-group was the following:

Status 0:

	rate_of_interest
count	112031.000000
mean	4.044931
std	0.561356
min	0.000000
25%	3.625000
50%	3.990000
75%	4.375000
max	8.000000

Status 1:

	rate_of_interest
count	200.000000
mean	4.350500
std	0.495546
min	3.125000
25%	3.990000
50%	4.312500
75%	4.750000
max	5.500000

The median was also calculated for each status 1 and 0 as 4.3125 and 3.99 respectively. Because the interest rate is continuous the median was used to impute the missing values.

Interest rate spread – Not a required vector so this was imputed to 0.0

Upfront charges – Not a required vector so this was imputed to 0.0

Term – The majority of both status 1 and 0 terms are 360 so imputed NaNs with that.

Negative amortization – Not_neg and neg_amm were understood to mean No Negative Amortization and Negative Amortization. Replaced not_neg and neg_amm with 0 and 1. Imputed NaNs with 0.

Interest only – Not_int and int_only were understood to mean Not interest only and interest only. Not_int and int_only were replaced with 0 and 1 respectively. NaNs were imputed with 0.

Lump sum payment – Not_lpsm and lpsm were understood to mean Not lump sum payment and Lump sum payment. Not_lpsm and lpsm were replaced with 0 and 1 respectively. NaNs were imputed with 0.

Property value – This is a required field for underwriting however 15,098 rows are missing a value for this vector. Because property value is continuous NaNs are imputed using the median value:

Status 1

```
count    112029.000000
mean     505606.066286
std      342784.462653
min        8000.000000
25%      288000.000000
50%      428000.000000
75%      638000.000000
max     9268000.000000
Name: property_value, dtype: float64
```

Status 0

```
count      21543.000000
mean      457786.009377
std       436255.032008
min         8000.000000
25%       228000.000000
50%       348000.000000
75%       558000.000000
max     16508000.000000
Name: property_value, dtype: float64
```

Construction type – Sb and mh were understood to mean single borrower and multi-home. Sb and Mh were replaced with 1 and 2 respectively. NaNs were imputed with 0.

Occupancy type – Pr, sr and ir were understood to mean primary residence, secondary residence and investment residence. These were replaced by 1, 2, and 3 respectively. NaNs were imputed with 0.

Property type – This could also be the collateral type. Home and Land were replaced with 1 and 2 respectively. NaNs were imputed with 0

Units – The number of units..up to 4. Imputed NaNs with 0.

Income – Continuous vector that cannot be NaN. Imputed NaNs with the median of each status.

Credit type / CoBorrower credit type – Map EXP, EQUI, TRANS, CIB, and CRIF to 1,2,3,4, and 4 respectively.

Credit score – No action needed

Submission of application – Unclear what this feature vector represents but seemed related to HMDA / URLA Demographic information on how the application was taken. To_inst and not_inst were mapped to 1 and 2 respectively. NaNs were imputed to 0.

LTV – Cannot be null but since the loan amounts and property values were imputed earlier set any missing LTVs to be $(loan\_amount \ / \ property\_value) \ * \ 100$

Deposit type – Unclear what this feature vector represents however direct and Idriect(sic) were mapped to 1 and 2 respectively. NaNs were imputed to 0.

DTI – Continuous feature vector that cannot be null. There are 24,121 missing labels. Impute NaNs to the mode for this vector:

Status 1:

	dtir1
count	20329.000000
mean	39.597324
std	12.716828
min	5.000000
25%	32.000000
50%	42.000000
75%	49.000000
max	61.000000

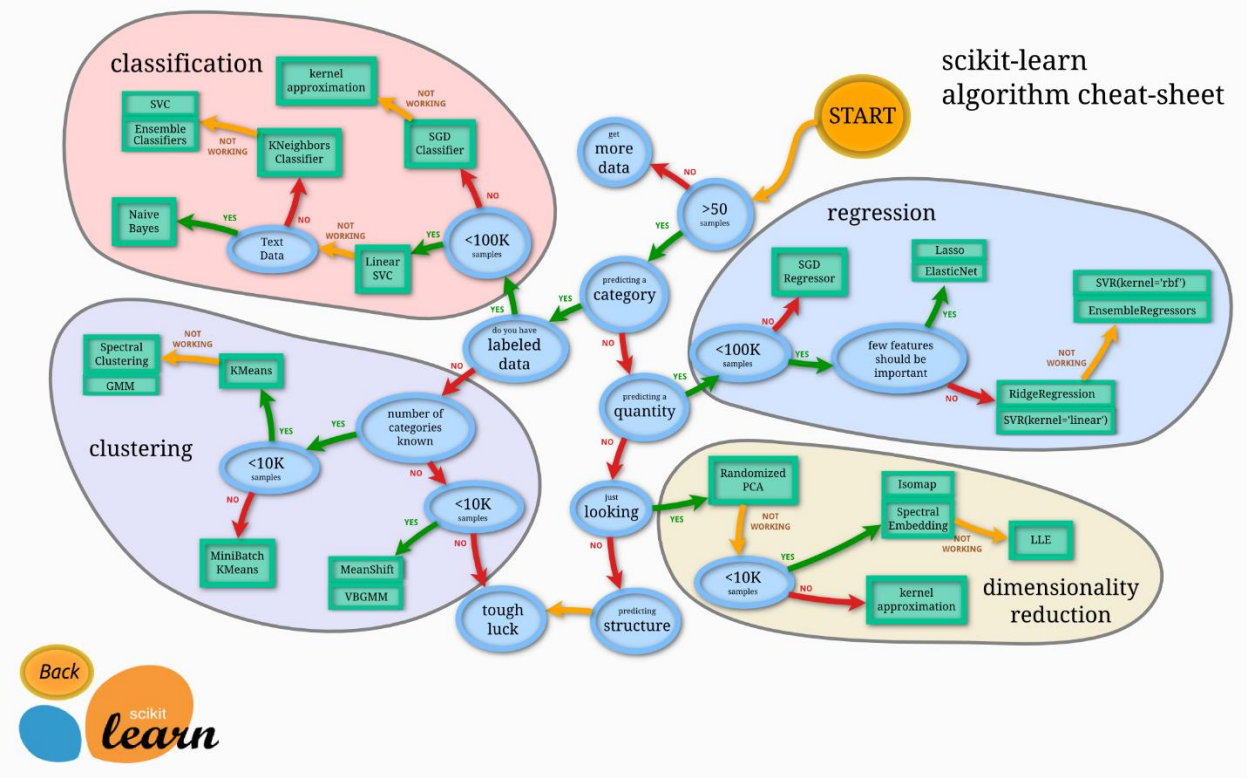
Status 0:

	dtir1
count	104220.000000
mean	37.369267
std	10.027197
min	5.000000
25%	31.000000
50%	38.000000
75%	44.000000
max	61.000000

Status – No action needed.

Implementation

Choosing the right classifier is one of the most difficult parts of solving machine learning problems however this chart from ScikitLearn comes in handy:



Following the flow outlined above, there are more than 50 samples, the goal is to predict a category, the data is labeled, and there are more than 100k samples. The first classifier to try is Stochastic Gradient Descent, however, this performed poorly and was only about 78% accurate. After further research, the Multi-Layer Perceptron classifier was implemented and consistently classified the status with ~91% accuracy. Further improving the model with Grid Search improved the score to ~95% accuracy.

Build model:

```
# Load etl_data
dataset.to_csv('processed_data.csv', index=False)

dataset_processed = pd.read_csv('processed_data.csv')

# Prep for load
y = dataset_processed["status"].copy()
X = dataset_processed.drop(["status", ], axis=1, inplace=False).copy()

# Split into training and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=40)

# Rest indices
X_train.reset_index(inplace = True, drop = True)
X_test.reset_index(inplace = True, drop = True)
y_train.reset_index(inplace = True, drop = True)
y_test.reset_index(inplace = True, drop = True)

# Scale continous data 0..1
scale_columns = [
    'loan_amount',
    'ltv',
    'upfront_charges',
    'property_value',
    'income',
    'interest_rate',
    'term',
    'credit_score',
    'dti'
]

train_scaler = StandardScaler(with_mean=True, with_std=True)
test_scaler = StandardScaler(with_mean=True, with_std=True)

train_scaler = train_scaler.fit(X_train[scale_columns])
X_train[scale_columns] = train_scaler.transform(X_train[scale_columns].copy())

test_scaler = test_scaler.fit(X_test[scale_columns])
X_test[scale_columns] = test_scaler.transform(X_test[scale_columns].copy())

# Train classifier
max_iter = 10000

classifier = MLPClassifier(
    hidden_layer_sizes = (15, 15), # number of neurons in the perceptron
    early_stopping = True, # after n_iter_no_change epochs stop fitting
    n_iter_no_change = 50, # number of epochs to stop after
    max_iter = max_iter # total number of epochs
)

print (classifier.fit(X_train.to_numpy(), y_train.to_numpy()))

print (f"iterations ran: {classifier.n_iter_}")
print (f"Train score: {classifier.score(X_train.to_numpy(), y_train.to_numpy())}")
print (f"Test score: {classifier.score(X_test.to_numpy(), y_test.to_numpy())}")
```

Model results:

```
MLPClassifier(early_stopping=True, hidden_layer_sizes=(20, 20), max_iter=10000,
              n_iter_no_change=50)
iterations ran: 98
Train score: 0.9999519549529639
Test score: 0.9997085267146476
```

Complications

The biggest hurdle was understanding the labels. For example, the credit_type vector has two unique labels, CIB and CRIF that further research couldn't determine what they correlate to. The other three labels are well-known credit reporting agencies.

The data is also not very well balanced being only from a single year and not having a uniform amount of performing and non-performing mortgages. There are also a lot of NaN values in the dataset, and key vectors are missing a lot of labels. Imputing these values worked for this, but further refinement of the data is needed for a production scenario.

All the key vectors with missing labels are continuous so to impute the values the median was used to overcome this issue. Additionally, when training the model, the vectors were scaled in the test and training datasets to improve accuracy:

LTV

```
dataset.loc[dataset['ltv'].isna(), 'ltv'] = ((dataset['loan_amount'] / dataset['property_value']) * 100)
```

Property value

```
status0_median = dataset[(dataset['status'] == 0) & (~dataset['property_value'].isna())]
['property_value'].median()
status1_median = dataset[(dataset['status'] == 1) & (~dataset['property_value'].isna())]
['property_value'].median()
```

Income

```
status0_median = dataset[(dataset['status'] == 0) & (~dataset['income'].isna())]['income'].median()
status1_median = dataset[(dataset['status'] == 1) & (~dataset['income'].isna())]['income'].median()

dataset.loc[(dataset['status'] == 0) & (dataset['income'].isna()), 'income'] = status0_median
dataset.loc[(dataset['status'] == 1) & (dataset['income'].isna()), 'income'] = status1_median
```

Interest rate

```
status0_median = dataset[(dataset['status'] == 0) & (~dataset['rate_of_interest'].isna())]
['rate_of_interest'].median()
status1_median = dataset[(dataset['status'] == 1) & (~dataset['rate_of_interest'].isna())]
['rate_of_interest'].median()

dataset.loc[(dataset['status'] == 0) & (dataset['rate_of_interest'].isna()), 'rate_of_interest'] = status0_median
dataset.loc[(dataset['status'] == 1) & (dataset['rate_of_interest'].isna()), 'rate_of_interest'] = status1_median
```

Fair credit regulations

There were also several vectors that needed to be removed for this implementation because of regulatory requirements. Credit issuance cannot be based on gender, or age and cannot have a disparate impact on specific regions. Because of this the gender, age, and region vectors were dropped from the dataset for this project. However, for other projects, these vectors could be useful for stress testing an FIs portfolio to see how credit decisions align with regulations and to determine if underwriting procedures need to be adjusted.

Refinement

To further improve the model the hidden_layer_sizes, learning_rate_init and tol(tolerance) parameters were searched with the best parameters being 20, 20, 0.001, and 0.00005 respectively:

```
parameters = {
    'hidden_layer_sizes': [(20, 20), (25, 25)],
    'learning_rate_init': [.001, .003, .004],
    'tol': [1e-05, 5e-05, 1e-04],
}

cv = GridSearchCV(classifier, param_grid=parameters, cv=3, verbose=5, n_jobs=-1, return_train_score=True)

print(cv.fit(X_train.to_numpy(), y_train.to_numpy()))
print (f"recommended estimator: {cv.best_estimator_}")
print (f"recommended parameters: {cv.best_params_}")
print (f"best score: {cv.best_score_}")

classifier_improved = cv.best_estimator_
print (f"Train score: {classifier_improved.score(X_train.to_numpy(), y_train.to_numpy())}")
print (f"Test score: {classifier_improved.score(X_test.to_numpy(), y_test.to_numpy())}")
```

The initial model fit with about 99% accuracy and using Grid Search didn't improve the score much.

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     33702
           1       1.00      1.00      1.00     10899

    accuracy                           1.00     44601
   macro avg       1.00      1.00      1.00     44601
weighted avg       1.00      1.00      1.00     44601

R-squared score: 0.9997571532671935
```

4. Results

Model Evaluation and Validation

GridSearchCV (cross-validation) was used to find the best parameters:

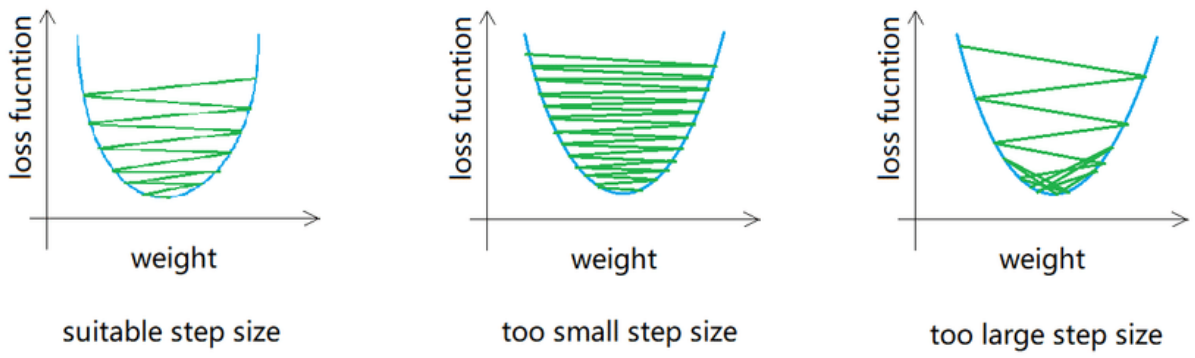
```
parameters = {
    'hidden_layer_sizes': [(20, 20), (25, 25)],
    'learning_rate_init': [.001, .003, .004],
    'tol': [1e-05, 5e-05, 1e-04],
}

cv = GridSearchCV(classifier, param_grid=parameters, cv=3, verbose=5, n_jobs=-1, return_train_score=True)

print(cv.fit(X_train.to_numpy(), y_train.to_numpy()))
print (f"recommended estimator: {cv.best_estimator_}")
print (f"recommended parameters: {cv.best_params_}")
print (f"best score: {cv.best_score_}")

classifier_improved = cv.best_estimator_

print (f"Train score: {classifier_improved.score(X_train.to_numpy(), y_train.to_numpy())}")
print (f"Test score: {classifier_improved.score(X_test.to_numpy(), y_test.to_numpy())}")
```

Hidden layer sizes: The number of layers in the first and second layers was recommended to be 25. The default of 100 was too high for the number of features in the dataset and would have resulted in overfitting. Further research revealed that a good rule of thumb is:

1. The number of hidden neurons should be between the size of the input layer and the size of the output layer.
2. The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
3. The number of hidden neurons should be less than twice the size of the input layer.

2/3 of the size of the input layer is 19 so rounding up to 20 is the first option in the CV search. Ultimately 25 was determined to fit best.

Learning rate init: The step size for backpropagation. Because SGD was used there needs to be a proper stepping rate for when to adjust the weights:



If the step size is too small it will overfit, and if it's too large it will underfit. Because the algorithm is not following the curve exactly using batches ensuring the stochastic steps properly fit the descent will result in a more accurate model.

Tol: Combined with a constant learning rate and an initial number of 'no-change' iterations of 50…once no change within this tolerance is hit then stop training. This prevents overfitting and keeps the training performant.

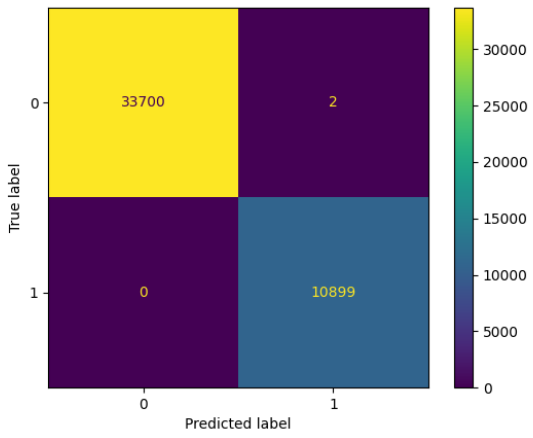
The classification report shows strong precision scores which is the most important for this model since false positives are the more costly confusion.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	33702
1	1.00	1.00	1.00	10899
accuracy			1.00	44601
macro avg	1.00	1.00	1.00	44601
weighted avg	1.00	1.00	1.00	44601

R score

```
R-squared score: 0.9997571532671935
```

Confusion matrix



Justification

While not perfect this is a good first step and provides solid proof of concept for further exploration. The SGD classifier used initially offered decent performance but couldn't get above 97% accuracy. Using the MLPClassifier resulted in 99% accuracy:

```
sgd_classifier = SGDClassifier(max_iter=1000, tol=1e-3)

print (sgd_classifier.fit(X_train.to_numpy(), y_train.to_numpy()))

print (f"iterations ran: {sgd_classifier.n_iter_}")
print (f"Train score: {sgd_classifier.score(X_train.to_numpy(), y_train.to_numpy())}")
print (f"Test score: {sgd_classifier.score(X_test.to_numpy(), y_test.to_numpy())}")

SGDClassifier()
iterations ran: 21
Train score: 0.9720666096532109
Test score: 0.9729826685500325
```

```
# batch size
max_iter = 10000

classifier = MLPClassifier(
    hidden_layer_sizes = (15, 15), # number of neurons in the perceptron
    early_stopping = True, # after n_iter_no_change epochs stop fitting
    n_iter_no_change = 50, # number of epochs to stop after
    max_iter = max_iter # total number of epochs
)

print (classifier.fit(X_train.to_numpy(), y_train.to_numpy()))

print (f"iterations ran: {classifier.n_iter_}")
print (f"Train score: {classifier.score(X_train.to_numpy(), y_train.to_numpy())}")
print (f"Test score: {classifier.score(X_test.to_numpy(), y_test.to_numpy())}")

MLPClassifier(early_stopping=True, hidden_layer_sizes=(20, 20), max_iter=10000,
              n_iter_no_change=50)
iterations ran: 98
Train score: 0.9999519549529639
Test score: 0.9997085267146476
```

5. Conclusion

Reflection

The goal was to take the loan default dataset and set up a web application with a dashboard where an underwriter could enter key metrics and using a machine learning model get a result back with a recommendation score and get feedback on the entered data. The interesting part of this project was starting from scratch and having to put all the pieces together…ETL, data engineering, ML processing and learning, application development, and deployment.

Improvement

The data is overfitting and without a more diverse set of training data, there isn't much that can be done about that. Future improvements for the data would be to get a more balanced set of mortgages and use that to train the model. The web app is also just a proof of concept and not fully fleshed out. Authentication, state, and a database backend for historical reporting are some improvements that would be beneficial.

Sources:

Dataset: https://www.kaggle.com/datasets/yasserh/loan-default-dataset

Step size image: https://www.researchgate.net/figure/The-Comparison-of-Different-Step-Size-Therefore-the-general-gradient-descent-algorithm_fig2_324751265

Classifier docs: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html