



**Dr. Christian Tsoungui Obama**

## **ML Cheat sheet #2**

# **Your roadmap to build a deployable classification model**

A Foundational Guide for Aspiring Data Scientists

Based on  
"Hands-On Machine Learning with Scikit-learn, Keras & TensorFlow"  
by

Aurélien Géron

## Step 1: Properly Frame the Problem

- Imagine a bank reviewing loan applications and trying to answer the question: "Given a person's income and credit history, can we predict whether their loan should be approved?"
- Now think of a factory monitoring its machines. Someone asks: "Based on vibration and temperature readings, can we predict whether a machine is operating normally or about to fail?"

Addressing such questions using machine learning (ML) requires following the road map illustrated in the "[ML Cheat sheet #1](#)" of this ML series (see Figure 1).

Practical steps to build [classification models](#) are described in what follows.

## Step 2 : Acquire the data

Possible data sources (see [ML Cheat sheet #1](#)).

```
1 import kagglehub
2 from kagglehub import KaggleDatasetAdapter
3
4 # Load a DataFrame with a specific version of a CSV
5 loan = kagglehub.dataset_load(
6     KaggleDatasetAdapter.PANDAS,
7     "taweilo/loan-approval-classification-data/versions/1",
8     "loan_data.csv",
9 )
```

Example code to import "Loan approval" dataset from Kaggle<sup>1</sup>

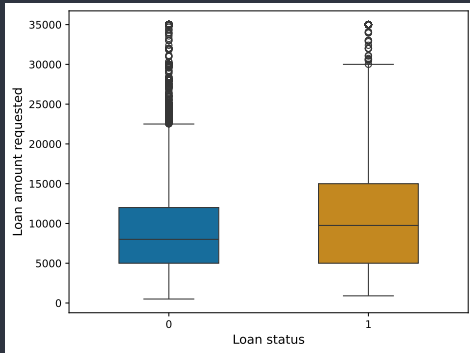
---

<sup>1</sup>[www.kaggle.com/datasets/taweilo/loan-approval-classification-data](https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data)

## Step 3: Explore the data

### Exploratory data analysis (EDA)

1. split data into train & test sets
2. investigate linear correlation & target class imbalance (numeric attributes)
3. transform data using pipelines (e.g., imputation, one-hot-encoding)



**Target class imbalance:** attribute potentially not discriminative (large overlap)



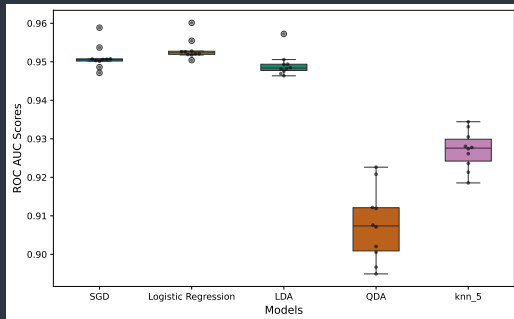
**Correlation matrix:** values quantify strength and direction of linear correlation

## Step 4: Train & evaluate candidate models

Some candidate models for classification:

Models	Advantages	Disadvantages
Logistic Regression	simple, interpretable, fast to train	assumes linear relationship between features and log-odds & sensitive to multicollinearity
Linear Discriminant Analysis (LDA)	works well with normally distributed features & computationally efficient	assumes normality and equal covariance matrices, sensitive to outliers & multicollinearity
K-Nearest Neighbors (KNN)	simple, non-parametric & models complex decision boundaries	computationally expensive for large datasets, sensitive to irrelevant features and scaling

```
1 # Training Logistic regression classifier
2
3 from sklearn.model_selection import
4   cross_val_score
5
6 from sklearn.linear_model import
7   LogisticRegression
8
9 score = "roc_auc"
10
11 # Logistic regression
12 log_reg = LogisticRegression()
13 log_roc_auc_scores = cross_val_score(
14     log_reg, loan_prepared,
15     loan_labels, scoring=score, cv=10
16 )
```



Models performance: ROC area under the curve scores (AUC – higher is better)

## Step 5.1 : Fine tune best candidate model (Logistic regression)

Model hyperparameters are fine-tuned to improve its performance:

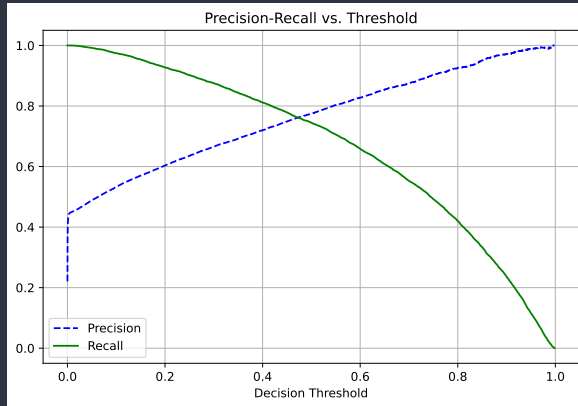
1. explore large space with Random search
2. fine tune search with Grid search
3. use cross-validation in both cases

```
1 # Fine tuning model parameters with grid search
2 # Pipeline to define iterations inside cross-validation
3 pipe_crossval = Pipeline([('log_reg', LogisticRegression(max_iter=2000))])
4 param_grid = {
5     'log_reg__C': [0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19],
6     'log_reg__penalty': ['l1'],
7     'log_reg__solver': ['liblinear']
8 }
9 grid_search = GridSearchCV(estimator=pipe_crossval,
10                             param_grid = param_grid,
11                             cv=5,
12                             scoring='roc_auc',
13                             n_jobs=-1,
14                             verbose=2)
15
16 # Fit model for all combinations of parameters
17 grid_search.fit(loan_prepared, loan_labels)
18
19 # Picking best model from grid search
20 best_model = grid_search.best_estimator_
21
```

## Step 5.2 : Be aware of Precision-Recall trade-off

depending on the application, one might want either of:

- high precision, i.e., if model predicts positive class it is highly likely to be correct,
- high recall, i.e., if true class is positive, model is highly likely to predict it as such.



**Precision-Recall trade-off:** for loan approval precision is preferable to recall (decision threshold: 60%)

## Step 6: Assess generalization of fine tuned model on test set

Prepare test set with same pipeline as the training set

```
1 # Defining test dataset
2 X_test = strat_test_set.drop("loan_status", axis=1)
3 y_test = strat_test_set["loan_status"].copy()
4
5 X_test_prepared = full_pipeline.fit_transform(X_test)
6
```

Evaluate model performance

```
1 # Running predictions on test set
2 # Predicted probabilities for the positive class (1 = loan approved)
3 from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
4
5 y_proba_test = best_model.predict_proba(X_test_prepared)[: , 1]
6 threshold = 0.6
7 y_pred_test = (y_proba_test >= threshold).astype(int)
8
9 print("ROC AUC score:", roc_auc_score(y_test, y_pred_test))
10
```



## Step 7: Save model for possible deployment

```
1 # Saving the model as joblib file
2 def save_model(model, name="model", model_path="."):
3     """
4     Save the model with a given name to a specific location.
5
6     Parameters
7     -----
8     model :
9         The model trained.
10    name:
11        The file name with which to save the model
12    model_path:
13        The path in which the model is saved
14    Example:
15        "./models/"
16
17    """
18    os.makedirs(model_path, exist_ok=True)
19    model_name = name + ".pkl"
20    pkl_path = os.path.join(model_path, model_name)
21    joblib.dump(model, pkl_path)
22
23 MODEL_PATH = os.path.join("models", "loan")
24 save_model(best_model, name="LogisticRegression_V1.1", model_path=MODEL_PATH)
```



**Did you find this useful?**



 **Get complete notebook:** » [GitHub repository](#) «

 **Comment below:** Can you think of an example where recall is more important than precision?

 **Follow me:** for the next cheat sheet in this series.

 **Like this post:** to help it reach more people.

**Let's Connect!**

 [/in/christiantsoungui](#)  [@chrisTs](#)