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).

```python
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Load a DataFrame with a specific version of a CSV
loan = kagglehub.dataset_load(
                                KaggleDatasetAdapter.PANDAS,
                                "taweilo/loan-approval-classification-data/versions/1",
                                "loan_data.csv",
                              )
```

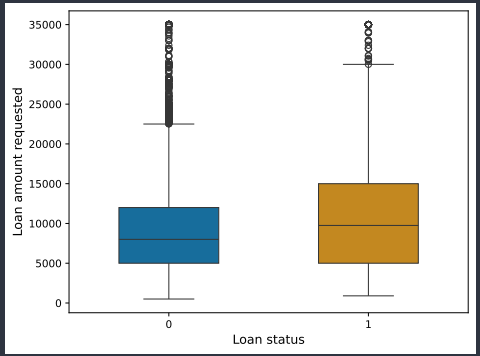
Example code to import "Loan approval" dataset from Kaggle[1]

[1] 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)

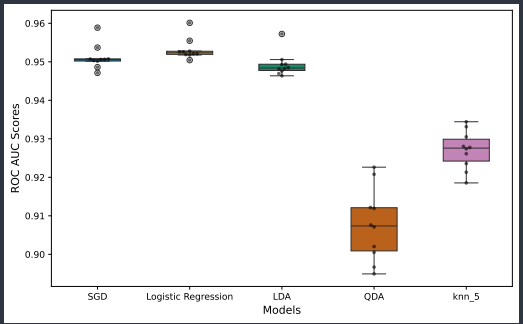
Correlation matrix: values quantify strength and direction of linear correlation

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
       cross_val_score
4  from sklearn.linear_model import
       LogisticRegression
5
6  score = "roc_auc"
7
8  # Logistic regression
9  log_reg = LogisticRegression()
10 log_roc_auc_scores = cross_val_score(
11                     log_reg, loan_prepared,
       loan_labels, scoring=score, cv=10
12                     )
13
```



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:

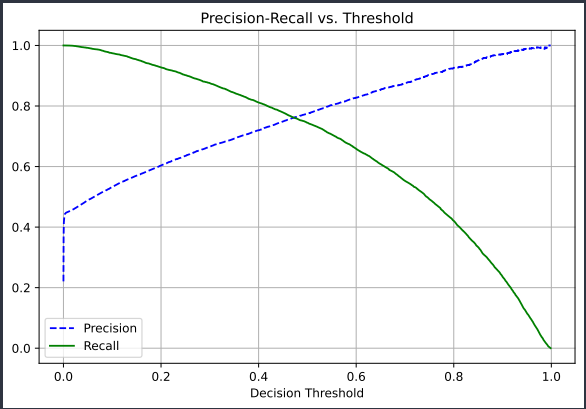
1. explore large space with Random search
2. fine tune search with Grid search
3. use cross-validation in both cases

```
# Fine tuning model parameters with grid search
# Pipeline to define iterations inside cross-validation
pipe_crossval = Pipeline([('log_reg', LogisticRegression(max_iter=2000))])
param_grid = {
    'log_reg__C': [0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19],
    'log_reg__penalty': ['l1'],
    'log_reg__solver': ['liblinear']
            }
grid_search = GridSearchCV(estimator=pipe_crossval,
                           param_grid = param_grid,
                           cv=5,
                           scoring='roc_auc',
                           n_jobs=-1,
                           verbose=2)

# Fit model for all combinations of parameters
grid_search.fit(loan_prepared, loan_labels)

# Picking best model from grid search
best_model = grid_search.best_estimator_
```

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

```
# Defining test dataset
X_test = strat_test_set.drop("loan_status", axis=1)
y_test = strat_test_set["loan_status"].copy()

X_test_prepared = full_pipeline.fit_transform(X_test)

```

Evaluate model performance

```
# Running predictions on test set
# Predicted probabilities for the positive class (1 = loan approved)
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

y_proba_test = best_model.predict_proba(X_test_prepared)[:, 1]
threshold = 0.6
y_pred_test = (y_proba_test >= threshold).astype(int)

print("ROC AUC score:", roc_auc_score(y_test, y_pred_test))

```

Step 7: Save model for possible deployment

```python
# Saving the model as joblib file
def save_model(model, name="model", model_path="./"):
    """
    Save the model with a given name to a specific location.

    Parameters
    ----------
    model :
        The model trained.
    name:
        The file name with which to save the model
    model_path:
        The path in which the model is saved
        Example:
            "./models/"

    """
    os.makedirs(model_path, exist_ok=True)
    model_name = name + ".pkl"
    pkl_path = os.path.join(model_path, model_name)
    joblib.dump(model, pkl_path)

MODEL_PATH = os.path.join("models", "loan")
save_model(best_model, name="Logistic_Regression_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 @TsounguiChris