Logistic Regression

Kannan Singaravelu, CQF

Logistic Regression

Logistic Regression is one of the most widely used algorithms for classification that maps quantitative data onto categorial variables. Unlike Linear Regression, where `y` is an outcome variable, we use a function of `y` called the logit.

Logit can be modelled as a linear function of the predictor

$$Logit = log(odds) = w_0 + w_1x_1 + w_2x_2 + ... + w_qx_q$$

and can be mapped back to a probability which in turn can be mapped to a class.

Problem Statement

Objective is to predict the underlying trend based on classification algorithm and formulate a trading strategy. In this lab, we'll use Logistic Regression to predict market direction and devise a trading strategies and analyse the results.

Import Libraries

[ ]:
```
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Base Libraries
import pandas as pd
import numpy as np
from functools import partial
import matplotlib.pyplot as plt

# Classifier
from sklearn.linear_model import LogisticRegression

# Preprocessing
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.pipeline import Pipeline
```

```
from sklearn.model_selection import (
                                    train_test_split,
                                    TimeSeriesSplit
                                    )

# Metrics
from sklearn.metrics import (
                            accuracy_score,
                            classification_report,
                            RocCurveDisplay,
                            ConfusionMatrixDisplay,
                            log_loss
                            )
```

1 Collect & Load Data

```
[ ]: df = pd.read_csv('../data/niftyindex.csv', index_col=0, parse_dates=True,␣
      ↪dayfirst=True)
     df.head(2)
```

```
[ ]: # Get Info
     df.info()
```

```
[ ]: # Visualize data
     plt.plot(df['Close']);
```

2 EDA of Original dataset

```
[ ]: # Descriptive statistics
     df.describe().T
```

3 Cleaning & Imputation

Data is already cleaned. No further processing or imputation required.

```
[ ]: # Check for missing values
     df.isnull().sum()
```

4 Feature Engineering

Features or Predictors are also known as an independent variable which are used to determine the value of the target variable. We will derive a features set from the original dataset.

4.1 Feature Specification

[ ]:
```
# Create Features
df['HC'] = df['High'] - df['Close']
df['RET'] = np.log(df['Close'] / df['Close'].shift(1))
df['MA7'] = df['Close'] / df['Close'].rolling(7).mean()
df['VMA'] = df['Volume'] / df['Volume'].rolling(7).mean()

df['OC_'] = df['Close'] / df['Open'] - 1
df['OC'] = df['OC_'].rolling(7).mean()
df['OC'] = df['OC_'].rolling(14).mean()

df['HC_'] = df['High'] / df['Low'] - 1
df['HC'] = df['HC_'].rolling(7).mean()

df['GAP_'] = df['Open'] / df['Close'].shift(1) - 1
df['GAP'] = df['GAP_'].rolling(7).mean()

df['STD'] = df['RET'].rolling(7).std()
df['UB'] = df['Close'].rolling(7).mean() + df['Close'].rolling(7).std() * 2

df.dropna(inplace=True)

features = df.drop(['Open', 'High', 'Low', 'Close', 'Volume', 'OC_', 'HC_',␣
↪'GAP_'], axis=1)
features.head(2)
```

[ ]:
```
# Specific X
X = features.values
```

4.2 Label Specification

Label or the target variable is also known as the dependent variable. Here, the target variable is whether the underlying price will close up or down on the next trading day. If the tomorrow's closing price is greater than the 0.995 of today's closing price, then we will buy the underlying, else we will sell it.

We assign a value of +1 for the buy signal and 0 otherwise. The target can be described as :

$$y_t = \begin{cases} +1, & \text{if } p_{t+1} > 0.995 * p_t \\ -1, & \text{if } p_{t+1} \text{Otherwise} \end{cases}$$

whre, p_t is the current closing price of the underlying and p_{t+1} is the 1-day forward closing price of the underlying.

[ ]:
```
# Specify y
y = np.where(df['Close'].shift(-1)>0.995*df['Close'],1,0)
```

[ ]:
```
# Check Class Imbalance
pd.Series(y).value_counts()
```

4.3 Base Model

We now build a base model with default parameters using Pipelines. Dataset needs to be scaled for the model to work properly and all the features should have a similar scale. The scaling can be accomplished by using the `StandardScaler`.

Split Data

[ ]:
```
# Split the Data into Training and Testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪shuffle=False)
```

[ ]:
```
# Define a Baseline Model
classifier = Pipeline([
        ("scaler", StandardScaler()),
        ("classifier", LogisticRegression(class_weight='balanced'))
    ])

classifier.fit(X_train, y_train)
```

[ ]:
```
# Verify Class Labels
classifier.classes_
```

[ ]:
```
# Predict the Class Labels
y_pred = classifier.predict(X_test)
y_pred[-20:]
```

[ ]:
```
# Predict Probabilities
y_proba = classifier.predict_proba(X_test)
y_proba[-20:]
```

[ ]:
```
# Get the Scores
acc_train = accuracy_score(y_train, classifier.fit(X_train, y_train).
↪predict(X_train))
acc_test = accuracy_score(y_test, classifier.predict(X_test))
print(f'Baseline Model -- Train Accuracy: {acc_train:0.4}, Test Accuracy:␣
↪{acc_test:0.4}')
```

4.4 Prediction Quality

Confusion Matrix Confusion matrix is a table used to describe the performance of a classification model on a set of test data for which the true values are known.

Outcome	Position[1]
True Negative	upper-left
False Negative	lower-left
False Positive	upper-right
True Positive	lower-right

True Positive is an outcome where the model correctly predicts the positive class. Similarly, a true negative is an outcome where the model correctly predicts the negative class.

False Positive is an outcome where the model incorrectly predicts the positive class. And a false negative is an outcome where the model incorrectly predicts the negative class.

Note: In a binary classification task, the terms ''positive" and ''negative" refer to the classifier's prediction, and the terms ''true" and ''false" refer to whether that prediction corresponds to the external judgment (sometimes known as the ''observation") and the axes can be flipped. Refer Scikit-Learn Binary Classification for further details.

[ ]:
```
# Display confussion matrix
disp = ConfusionMatrixDisplay.from_estimator(
        classifier,
        X_test,
        y_test,
        # display_labels=model.classes_,
        cmap=plt.cm.Blues
    )
plt.title('Confusion matrix')
plt.show()
```

Receiver Operator Characterisitc Curve (ROC) The area under the ROC curve (AUC) is a measure of how well a model can distinguish between two classes. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds.

[ ]:
```
# Display ROCCurve
disp = RocCurveDisplay.from_estimator(
            classifier,
            X_test,
            y_test,
            name='Baseline Model')
plt.title("AUC-ROC Curve \n")
plt.plot([0,1],[0,1],linestyle="--", label='Random 50:50')
plt.legend()
plt.show()
```

Classification Report A classification report is used to measure the quality of predictions from a classification algorithm.

[ ]:
```
# Classification Report
print(classification_report(y_test, classifier.predict(X_test)))
```

Macro Average Average of precision (or recall or f1-score) of different classes.

Weighted Average Actual Class1 instance * precision (or recall or f1-score) of Class1 + Actual Class2 instance * (or recall or f1-score) of Class2.

5 Hyperparameter Tuning

Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes. It is possible and recommended to search the hyper-parameter space for the best cross validation score. Any parameter provided when constructing an estimator may be optimized in this manner.

5.1 Cross-validation of Time Series

Time series data are sequential in nature and are characterised by the correlation between observations. Classical cross-validation techniques such as KFold assume the samples are independent and identically distributed, and would result in poor estimates when applied on time series data.

To preserve the order and have training set occur prior to the test set, we use **Forward Chaining** method in which the model is initially trained and tested with the same windows size. And, for each subsequent fold, the training window increases in size, encompassing both the previous training data and test data. The new test window once again follows the training window but stays the same length.

We will tune the hyperparameters to select the K-Best Neighbor by **TimeSeriesSplit** from scikit-learn. This is a forward chaining cross-validation method and is a variation from the KFold. In the kth split, it returns first k folds as train set and the (k+1)th fold as test set. Unlike standard cross-validation methods, successive training sets are supersets of those that come before them.

[ ]:
```
# Get Params list
classifier.get_params()
```

[ ]:
```
# Use Optuna for Tuning
import optuna
```

[ ]:
```
# Define Objective Function
def optimize(trial, x, y):

    # specify params range
    tolerance = trial.suggest_float("tol", 0.001, 0.01, log=True)
    regularization = trial.suggest_float('C', 0.001, 1, log=True)

    model = Pipeline([
        ("scaler", StandardScaler()),
```

```
        ("model", LogisticRegression(
            C=regularization,
            tol=tolerance,
            class_weight='balanced'))
    ])

    tscv = TimeSeriesSplit(n_splits=2, gap=1)
    ll = []

    for idx in tscv.split(x):
        train_idx, test_idx = idx[0], idx[1]
        xtrain = x[train_idx]
        ytrain = y[train_idx]

        xtest = x[test_idx]
        ytest = y[test_idx]

        model.fit(xtrain, ytrain)
        preds = model.predict(xtest)

        ll.append(log_loss(ytest, preds))

    return -1.0 * np.mean(ll)
```

[ ]:
```
# Create a Study
study = optuna.create_study(
    study_name='hp_lr',
    direction='minimize'
)
```

[ ]:
```
# Specify Optimization function
optimization_function = partial(optimize, x=X, y=y)
study.optimize(optimization_function, n_trials=20)
```

[ ]:
```
# Get the Best Params
print(f'Best Params: {study.best_params}, Best Value: {study.best_value}')
```

5.2 Visualize Optimization

[ ]:
```
# plot Optimization History
optuna.visualization.plot_optimization_history(study)
```

[ ]:
```
# Plot Param Importances
optuna.visualization.plot_param_importances(study)
```

[ ]:
```
# plot accuracies for each HP trail
optuna.visualization.plot_slice(study)
```

[ ]:
```
# plot the surface
optuna.visualization.plot_contour(study, params=['tol', 'C'])
```

[ ]:
```
# plot parallel coordinates
optuna.visualization.plot_parallel_coordinate(study)
```

5.3 Tuned Model

We now build a tuned model with the best parameters using Pipelines. Dataset needs to be scaled for the model to work properly and all the features should have a similar scale. The scaling can be accomplished by using the `StandardScaler`.

[ ]:
```
# Scale and fit the model
clf = Pipeline([
    ("scaler", StandardScaler()),
    ("estimator", LogisticRegression(
        tol=study.best_params['tol'],
        C=study.best_params['C'],
        class_weight='balanced',
        ))
])

clf.fit(X_train, y_train)
```

[ ]:
```
# Predict Class Labels
y_pred = clf.predict(X_test)

# Predict Probabilities for upside
# y_proba = model.best_estimator_.predict_proba(X_test)[:,1]

# Measure Accuracy
acc_train = accuracy_score(y_train, clf.predict(X_train))
acc_test = accuracy_score(y_test, y_pred)

# Print Accuracy
print(f'\n Training Accuracy \t: {acc_train :0.4} \n Test Accuracy \t\t:␣
 ↪{acc_test :0.4}')
```

Confusion Matrix

[ ]:
```
# Display confussion matrix
disp = ConfusionMatrixDisplay.from_estimator(
        clf,
        X_test,
        y_test,
        # display_labels=model.classes_,
        cmap=plt.cm.Blues
    )
plt.title('Confusion matrix')
plt.show()
```

Receiver Operator Characterisitc Curve (ROC)

[ ]:
```
# Display ROCCurve
disp = RocCurveDisplay.from_estimator(
            clf,
            X_test,
            y_test,
            name='Tuned Model')
plt.title("AUC-ROC Curve \n")
plt.plot([0,1],[0,1],linestyle="--", label='Random 50:50')
plt.legend()
plt.show()
```

Classification Report

[ ]:
```
# Classification Report
print(classification_report(y_test, y_pred))
```

Observation

1. Accuracy, Recall and Precision improved marginally.
2. Tuning improved prediction for upside marginally.

6 Trading Strategy

Let's now define a trading strategy. We will use the predicted signals for trades. We then compare the result of this strategy with the buy and hold and visualize the performance of the strategy built using Logistic Regression.

[ ]:
```
df2 = pd.read_csv('../data/niftyindex.csv', index_col=0, parse_dates=True,␣
 ↪dayfirst=True)
df2 = df2.iloc[13:,:]

# Get Prediction
df2['Signal'] = clf.predict(X)

# Define Entry Logic
df2['Entry'] = np.where(df2['Signal']==1, df2['Close'], 0)

# Defining Exit Logic
df2['Exit'] = np.where((df2['Entry'] != 0) & (df2['Open'].shift(-1) <=␣
 ↪df2['Close']),
                         df['Open'].shift(-1), 0)

df2['Exit'] = np.where((df2['Entry'] != 0) & (df2['Open'].shift(-1) >␣
 ↪df2['Close']),
                         df2['Close'].shift(-1), df2['Exit'])

# Calculate MTM
df2['P&L'] = df2['Exit'] - df2['Entry']

# Generate Equity Curve
df2['Equity'] = df2['P&L'].cumsum() + df2['Close'][0]

# Calculate Benchmark Return
df2['Returns'] = np.log(df2['Close']).diff().fillna(0)

# Calculate Strategy Return
df2['Strategy'] = (df2['Equity']/df2['Equity'].shift(1) - 1).fillna(0)
df2 = df2.iloc[:-1]
```

[ ]:
```
# Generate HTML Strategy Report
# Refer HTML file for report
import quantstats as qs
qs.reports.html(df2['Strategy'], df2['Returns'])
```

[ ]:
```python
# Can also use pyfolio for analysis
import pyfolio as pf

df3 = df2.copy()
df3.index = df3.index.tz_localize('utc')
pf.create_returns_tear_sheet(df3['Strategy'], live_start_date='2020-04-07',␣
 ↪benchmark_rets=df3['Returns'])
```

7 References

- TimeSeriesSplit
- Cross-validation
- GridSearchCV
- Hyperparameters Tuning
- K-Neighbors Classifier

Notes 1. Sckit-learn format. One may also use a difference convention for axes.

Python Labs by Kannan Singaravelu.