Become a Kaggle Master-HW3

Team ZYMAA

AYARI Mohamed Aziz
YBEGGAZENE Zakaria

Université Paris Dauphine-PSL | M2 IASD en apprentissage

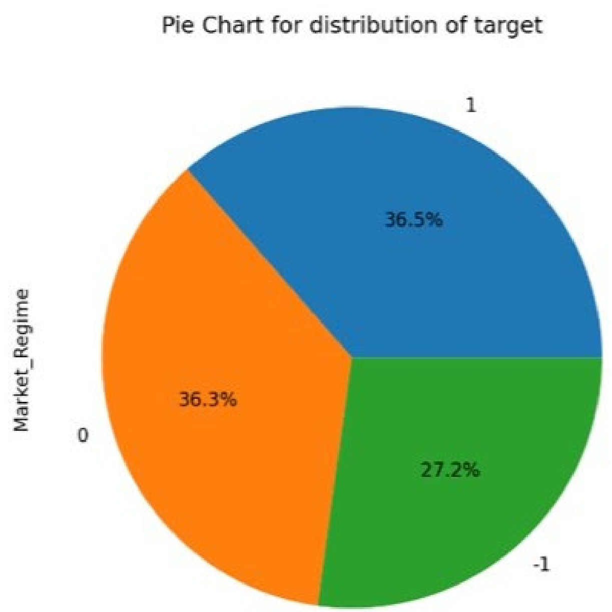
Objective : Determine the economic regime predicted by the best economists

PLAN

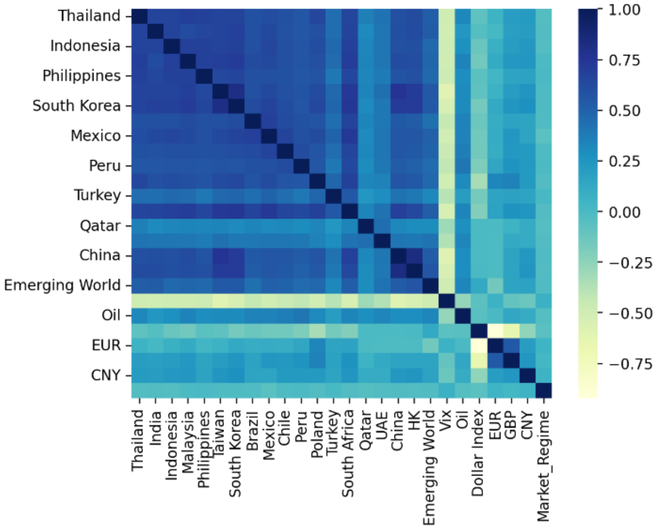
1. Data visualization
2. Pre-processing and feature engineering
3. Modeling and validation
4. Results

Data visualization

Check imbalanced classes



- Plotting the heat map showed that features are not highly correlated.
- Some variable are correlated such as (HK, China)



- There are no missing values

Data pre-processing and Feature engineering

Feature engineering

We calculated rolling average (with a 7 days window size) and lag features (with 1, 3, 5 and 10 lag periods) for each column except 'Date' and 'Market_Regime' (the target)

```
# Define the rolling window size for rolling average
window_size = 7

# Calculate rolling average for each column except 'Date' and 'Market_Regime'
for col in df.columns:
    if col not in ['Date', 'Market_Regime']:
        df[col + '_RollingAvg'] = df[col].rolling(window=window_size).mean()

# Create lag features
lag_periods = [1, 3, 5, 10]  # You can adjust the lag periods as needed
for col in df.columns:
    if col not in ['Date', 'Market_Regime']:
        for lag in lag_periods:
            df[col + '_Lag' + str(lag)] = df[col].shift(lag)
```

Feature engineering

Concerning the ‘Date’ column, we split it into multiple features like the ‘year’, ‘month’, ‘day’, ‘day_of_week’ etc. and drop the ‘Date’ column

```
# Add new features such as year month and semester
df['Date_year'] = df['Date'].dt.year
df['Date_month'] = df['Date'].dt.month
df['Date_day'] = df['Date'].dt.day
df['Date_dow'] = df['Date'].dt.dayofweek
df['Date_week'] = df['Date'].dt.week
df['quarter'] = df['Date'].dt.quarter
df['semester'] = np.where(df['quarter'].isin([1,2]), 1, 2)
```

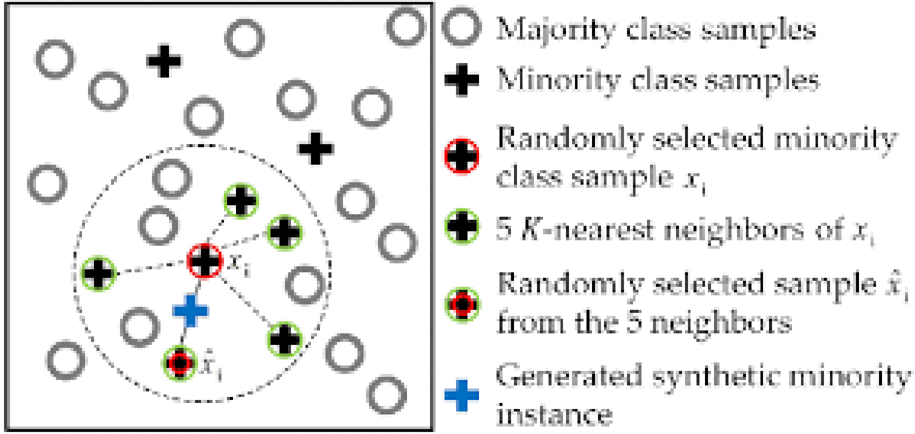
Feature engineering

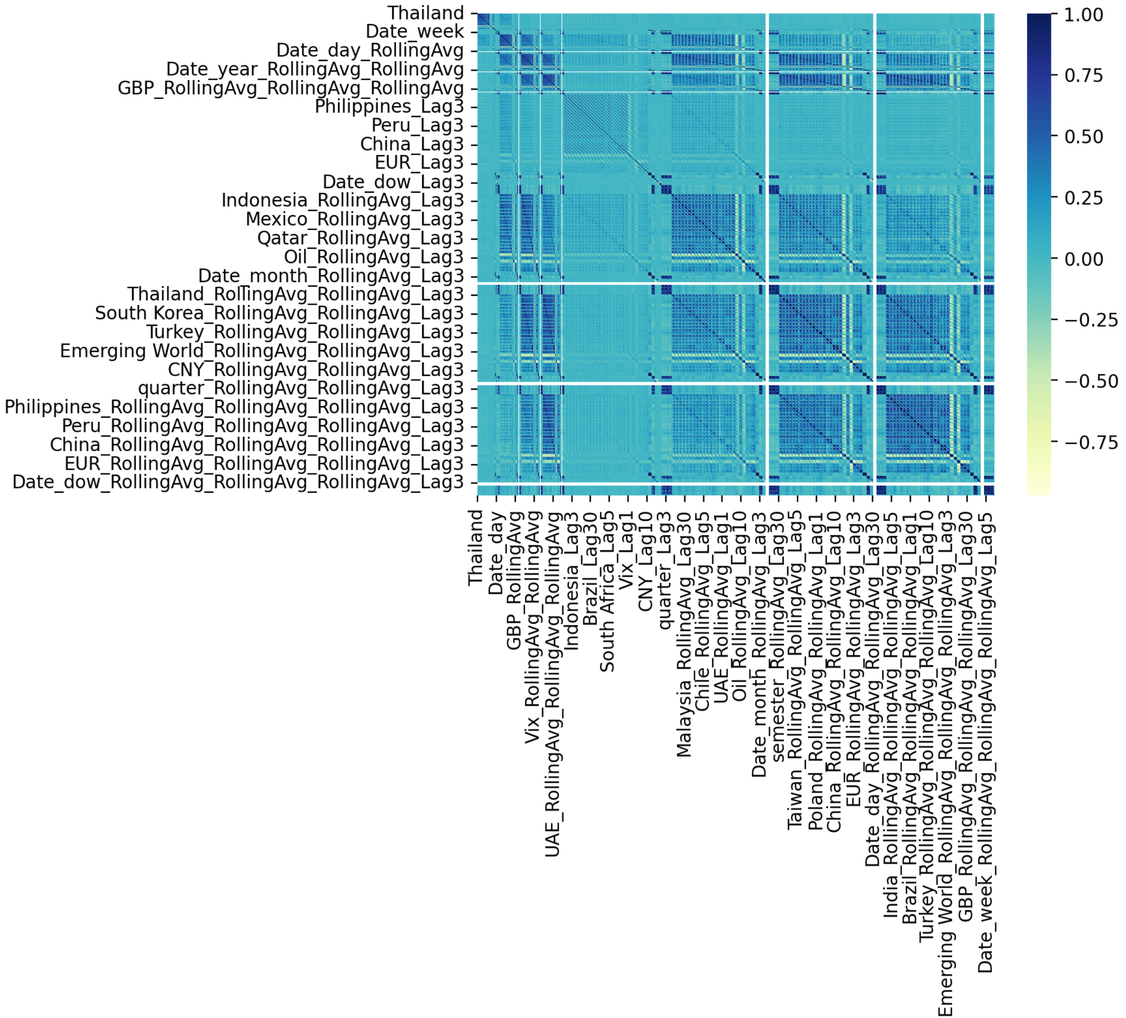
The target variable takes its values in the set {-1, 0, 1} which is not very adapted to all models that do multi-class classification.
We use a Label Encoder to transform the values into the set {0, 1, 2} respectively as follows

```
# Encode the target variable
label_encoder = LabelEncoder()
df['Market_Regime'] = label_encoder.fit_transform(df['Market_Regime'])
```

Feature engineering

SMOTE for the imbalanced classes





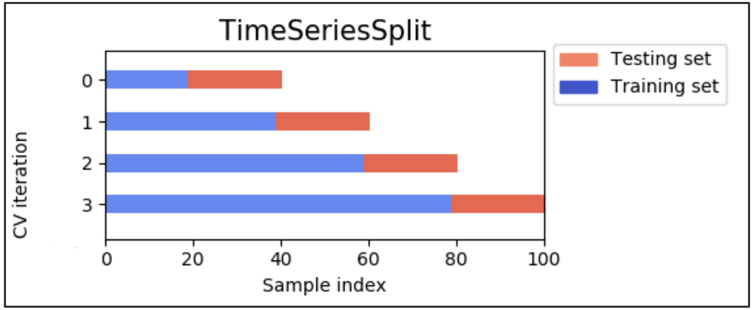
Our feature engineering resulted in highly correlated variables so we decided to remove them using a correlation threshold 0.9

Modeling and validation

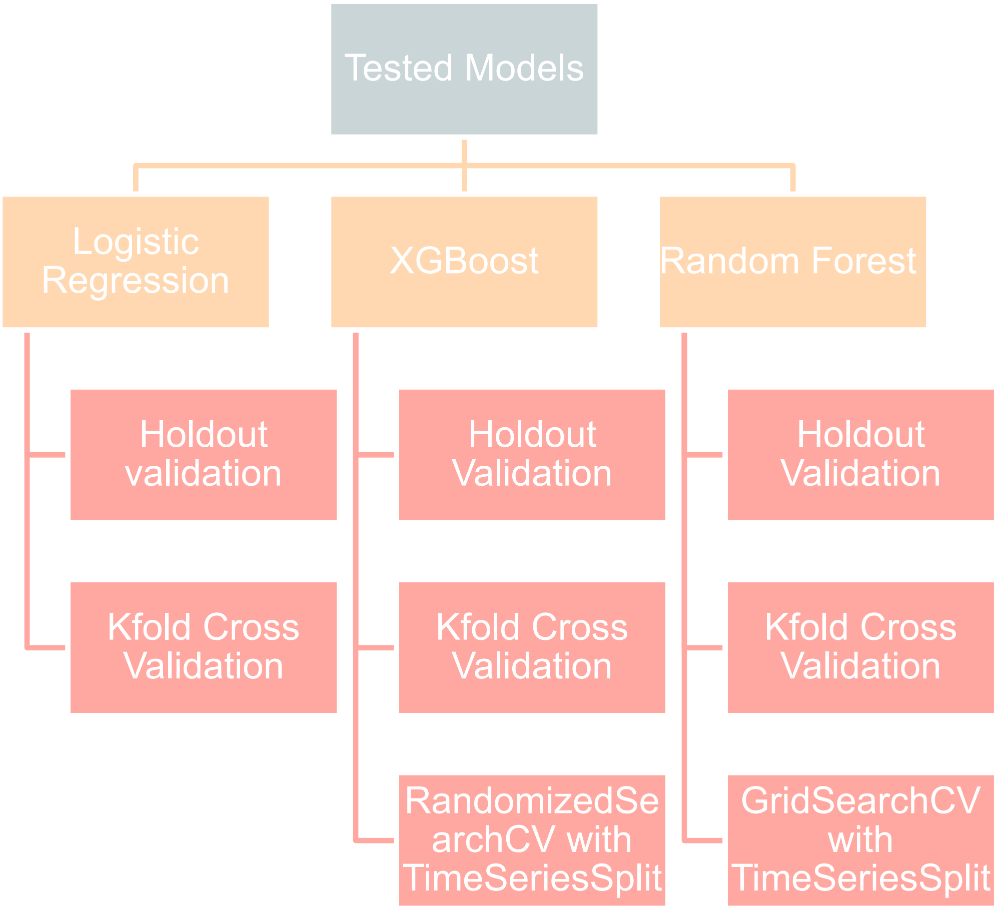
Time Series data => data points are **not i.i.d.**

We first define a validation strategy:

- We started with holdout validation (80%-20% split with shuffle = False)
- Then we used a KFold cross validation with 5 folds
- Finally, we opted for TimeSeriesSplit also with 5 folds



Combined with GridSearchCV and RandomizedSearchCV for hyperparameter tuning


Tested Models
Logistic Regression
XGBoost
Random Forest
Holdout validation
Kfold Cross Validation
Holdout Validation
Kfold Cross Validation
RandomizedSearchCV with TimeSeriesSplit
Holdout Validation
Kfold Cross Validation
GridSearchCV with TimeSeriesSplit

Logistic Regression Classifier

We optimize our Logistic Regression classifier on:

- Multinomial (softmax) objective function:

```
# Build the logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
```

- One-vs-Rest (separate binary logistic regression models are trained for each class against the rest):

```
# Build the logistic regression model
model = LogisticRegression(multi_class='ovr', solver='lbfgs')
```

- And evaluate on the requested evaluation metric (ROC AUC):

```
auc_roc = roc_auc_score(y_val_encoded, y_pred_proba, multi_class='ovr')
```

XGBoost Classifier

XGBoost offers the possibility to define a custom objective function. Therefore, we take advantage of this and use the ROC AUC objective:

```
def roc_auc_obj(preds, dtrain):
    labels = dtrain.get_label()
    preds = 1.0 / (1.0 + np.exp(-preds)) # convert predictions to probabilities
    grad = preds - labels
    hess = preds * (1.0 - preds)
    return grad, hess
```

```
# Create an instance of TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)  # specify the number of splits as needed

# Create an instance of XGBClassifier for multi-class classification
xgb_2 = xgb.XGBClassifier(params, objective=roc_auc_obj, eval_metric='auc', num_class=3)

# Perform time series cross-validation with ROC AUC as the evaluation metric
roc_auc_scores = cross_val_score(xgb_2, X_train, y_train, cv=tscv, scoring='roc_auc_ovr')
```

The RandomizedSearch CV yields the best hyperparameters for this model:

{'**reg_lambda**': 0.1, '**reg_alpha**': 0.5, '**n_estimators**': 100, '**min_child_weight**': 3, '**max_depth**': 9, '**learning_rate**': 0.3, '**gamma**': 1, '**colsample_bytree**': 0.7}

Random Forest Classifier

Using GridSearch CV, we achieved a similar validation score as with XGBoost.

The best hyperparameters for this model were:

{'**n_estimators**': 500, '**min_samples_split**': 5, '**min_samples_leaf**': 2, '**max_features**': 'sqrt', '**max_depth**': None, '**bootstrap**': True}

Ensemble methods - Averaging

- **Majority Voting**: we used majority voting with the best versions of the 3 models presented earlier (Logistic Regression, Random Forest, XGBoost)

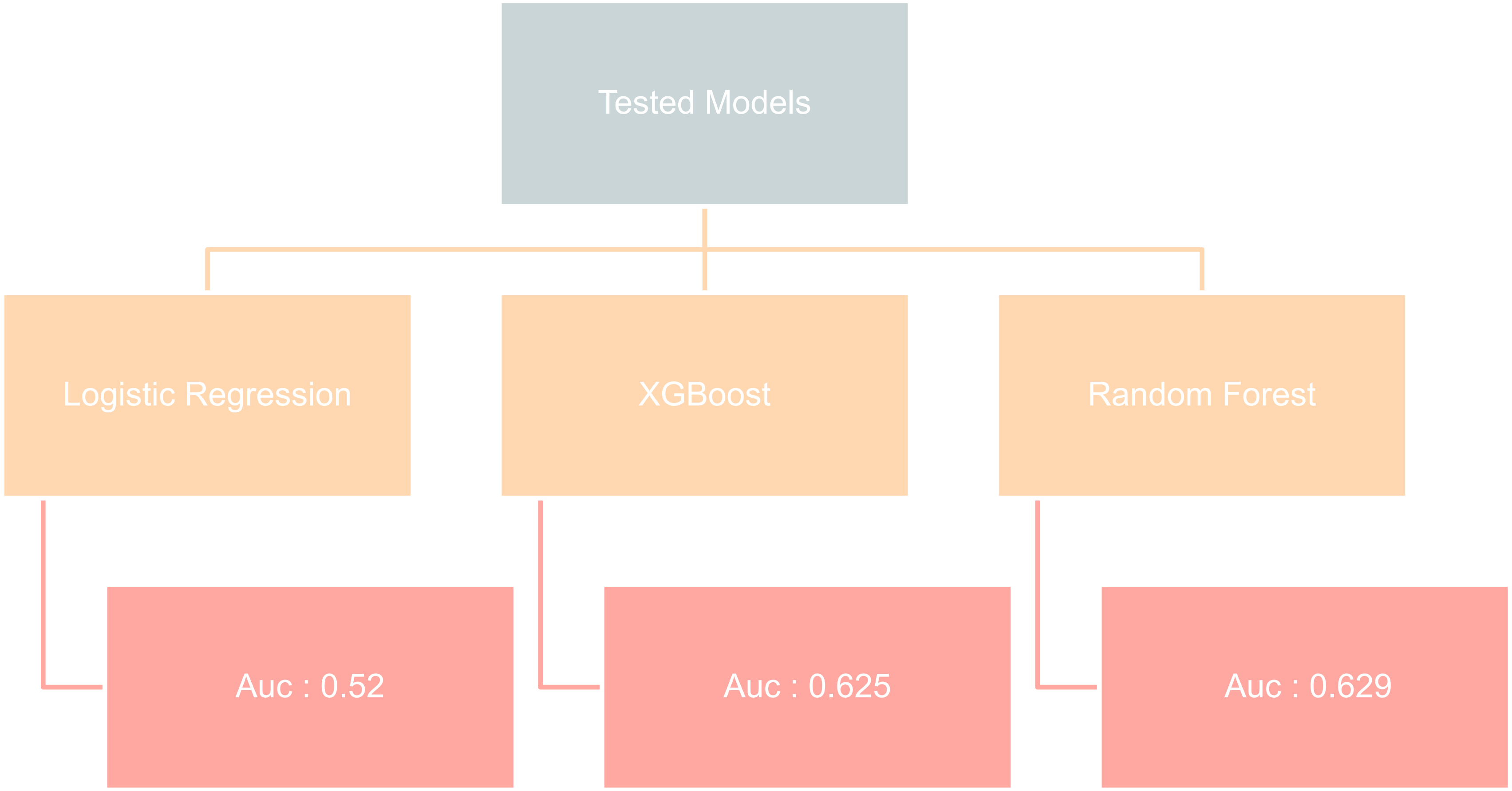
- The Logistic Regression classifier has a lower predictive power than the other models

- We therefore tried averaging with the **(weighted) arithmetic average** on Random Forest and XGBoost (on the probas).

- We choose the weights of the average by:

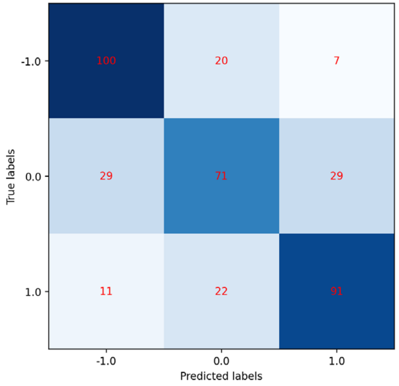
1. Manually testing different combination and using the public leaderboard as feedback
2. Relying on the scores on the out-of-fold predictions

Results


Tested Models
Logistic Regression
XGBoost
Random Forest
Auc : 0.52
Auc : 0.625
Auc : 0.629

Understanding the scores

- We could plot confusion matrices for each fold in our cross-validation to understand which classes are the hardest to predict (below is one confusion matrix over one fold of the KFold CV):



- We also used the out-of-fold predictions during cross-validation to plot a confusion matrix on all the folds (hard with TimeSeriesSplit)

Best Models
(on the public leaderboard)

Ensemble Methods
(0.5*Random Forest + 0.5*XGBoost Classifier)
Score : 0.66184

Conclusion

- Features engineering is very important for improving model performance and even more effective than improving models with other techniques.
- Use techniques such as Smote to get balanced classes help to improve the model performance
- Voting techniques can be used as a final step combining the more efficient models tested.

Thanks for your attention