Module 5 - Homework

Task description

In the exercise of this week you will be working with financial data in order to (hopefully) find a portfolio of equities which outperform SP500. The data that you are going to work with has two main sources:

- Financial data from the companies extracted from the quarterly company reports (mostly extracted from macrotrends so you can use this website to understand better the data and get insights on the features).
- Stock prices, mostly extracted from morningstar, which basically tell us how the stock price is evolving so we can use it both as past features and the target to predict.

Before going to the problem, let's comment some of the columns of the dataset:

- `Ticker`: a short name to identify the equity (that you can use to search in macrotrends).
- `date`: the date of the company report (normally we are going to have 1 every quarter). This is for informative purposes but you can ignore it when modeling.
- `execution_date`: the date when we would have executed the algorithm for that equity. We want to execute the algorithm once per quarter to create the portfolio, but the release dates of all the different company reports don't always match for the quarter, so we just take a common execution_date for all of them.
- `stock_change_div_365`: what is the % change of the stock price (with dividends) in the FOLLOWING year after execution_date.
- `sp500_change_365`: what is the % change of the SP500 in the FOLLOWING year after execution_date.
- `close_0`: what is the price at the moment of execution_date.
- `stock_change__minus_120`: what is the % change of the stock price in the last 120 days.
- `stock_change__minus_730`: what is the % change of the stock price in the last 730 days.

The rest of the features can be divided between financial features (the ones coming from the reports) and technical features (coming from the stock price). We leave the technical features here as a reference:

```
technical_features = ['close_0', 'close_sp500_0', 'close_365', 'close_sp500_365',
       'close__minus_120', 'close_sp500__minus_120', 'close__minus_365',
       'close_sp500__minus_365', 'close__minus_730', 'close_sp500__minus_730',
       'stock_change_365','stock_change_div_365', 'sp500_change_365',
       'stock_change__minus_120', 'sp500_change__minus_120',
       'stock_change__minus_365', 'sp500_change__minus_365',
       'stock_change__minus_730', 'sp500_change__minus_730',
       'std__minus_365', 'std__minus_730', 'std__minus_120']
```

The problem that we want to solve is basically find a portfolio of `top_n` tickers (initially set to 10) to invest every `execution_date` (basically once per quarter) and the goal is to have a better return than SP500 in the following year. The initial way to model this is to have a binary target which is 1 when `stock_change_div_365` - `sp500_change_365` (the difference between the return of the equity and the SP500 in the following year) is positive or 0 otherwise. So we try to predict the probability of an equity of improving SP500 in the following year, we take the `top_n` equities and compute their final return.

Componentes de la solución:

Crear una Pull Request (PR) en vuestro repositorio personal en la que resolvais la tarea propuesta. En la descripción de la PR deberás añadir una o varias imágenes con los resultados obtenidos así como una breve descripción con conclusiones o comentarios añadidos.

We have trained the first models for all the periods for you, but there are a lot of things which may be wrong or can be improved. Some ideas where you can start:

- Try to see if there is any kind of data leakage or suspicious features.
- If the training part is very slow, try to see how you can modify it to execute faster tests.
- Try to understand if the algorithm is learning correctly.
- We are using a very high level metric to evaluate the algorithm so you may need to use some more low level ones.
- Try to see if there is overfitting.
- Try to see if there is a lot of noise between different trainings.
- To simplify, why not only keep the first tickers in terms of Market Cap?
- Change the number of quarters to train in the past.

Initial code

The following variables and functions are given to you to accelerate the development work:

```
import pandas as pd
import re
import numpy as np
import lightgbm as lgb
from plotnine import (ggplot, geom_histogram, aes, geom_col, coord_flip,
                      geom_bar, scale_x_discrete, geom_point, theme, element_text)

# number of trees in lightgbm
n_trees = 40
minimum_number_of_tickers = 1500
# Number of the quarters in the past to train
n_train_quarters = 36
# number of tickers to make the portfolio
top_n = 10
```

Load and filter the dataset:

```
data_set = pd.read_feather("data/financials_against_return.feather")

# Remove quarters with less than minimum_number_of_tickers tickers
df_quarter_lengths = (data_set.groupby(["execution_date"])
                              .size()
                              .reset_index()
                              .rename(columns={0: "count"}))
data_set = pd.merge(data_set, df_quarter_lengths, on=["execution_date"])
data_set = data_set[data_set["count"] >= minimum_number_of_tickers]

# Create the target
data_set["diff_ch_sp500"] = (data_set["stock_change_div_365"]
                             - data_set["sp500_change_365"])
data_set.loc[data_set["diff_ch_sp500"] > 0, "target"] = 1
data_set.loc[data_set["diff_ch_sp500"] < 0, "target"] = 0
```

The main metric function: given predictions with probabilities for each equity, sort in descending order of probability, pick the `top_n` ones, and compute their weighted return:

```
def get_weighted_performance_of_stocks(df, metric):
    df["norm_prob"] = 1 / len(df)
    return np.sum(df["norm_prob"] * df[metric])

def get_top_tickers_per_prob(preds):
    if len(preds) == len(train_set):
        data_set = train_set.copy()
    elif len(preds) == len(test_set):
        data_set = test_set.copy()
    else:
        assert ("Not matching train/test")
    data_set["prob"] = preds
    data_set = data_set.sort_values(["prob"], ascending=False)
    data_set = data_set.head(top_n)
    return data_set

# Main metric to evaluate: average diff_ch_sp500 of the top_n stocks
def top_wt_performance(preds, train_data):
    top_dataset = get_top_tickers_per_prob(preds)
    return "weighted-return", get_weighted_performance_of_stocks(
        top_dataset, "diff_ch_sp500"), True
```

Train/test split function based on execution_date:

```
def split_train_test_by_period(data_set, test_execution_date,
                               include_nulls_in_test=False):
    train_set = data_set.loc[
        data_set["execution_date"] <= (pd.to_datetime(test_execution_date)
                                       - pd.Timedelta(350, unit="day"))]
    train_set = train_set[~pd.isna(train_set["diff_ch_sp500"])]
    execution_dates = train_set.sort_values("execution_date")["execution_date"].unique()
    if n_train_quarters is not None:
        train_set = train_set[
            train_set["execution_date"].isin(execution_dates[-n_train_quarters:])]
    test_set = data_set.loc[data_set["execution_date"] == test_execution_date]
    if not include_nulls_in_test:
        test_set = test_set[~pd.isna(test_set["diff_ch_sp500"])]
    test_set = (test_set.sort_values("date", ascending=False)
                        .drop_duplicates("Ticker", keep="first"))
    return train_set, test_set
```

Columns to remove (irrelevant or target-related features):

```
def get_columns_to_remove():
    columns_to_remove = [
        "date", "improve_sp500", "Ticker", "freq", "set",
        "close_sp500_365", "close_365", "stock_change_365",
        "sp500_change_365", "stock_change_div_365", "stock_change_730",
        "stock_change_div_730", "diff_ch_sp500", "diff_ch_avg_500",
        "execution_date", "target", "index", "quarter", "std_730", "count"]
    return columns_to_remove
```

Main modeling function — trains a LightGBM classifier. You are not expected to change the algorithm but feel free to tune its hyperparameters:

```
import warnings
warnings.filterwarnings('ignore')

def train_model(train_set, test_set, n_estimators=300):
    columns_to_remove = get_columns_to_remove()
    X_train = train_set.drop(columns=columns_to_remove, errors="ignore")
    X_test  = test_set.drop(columns=columns_to_remove, errors="ignore")
```

```
    y_train = train_set["target"]
    y_test  = test_set["target"]

    lgb_train = lgb.Dataset(X_train, y_train)
    lgb_test  = lgb.Dataset(X_test, y_test, reference=lgb_train)
    eval_result = {}

    params = {
        "random_state": 1,
        "verbosity": -1,
        "n_jobs": 10,
        "n_estimators": n_estimators,
        "objective": "binary",
        "metric": "binary_logloss",
    }
    model = lgb.train(
        params=params,
        train_set=lgb_train,
        valid_sets=[lgb_test, lgb_train],
        feval=[top_wt_performance],
        callbacks=[lgb.record_evaluation(eval_result=eval_result)],
    )
    return model, eval_result, X_train, X_test
```

Run model for a single execution_date:

```
def run_model_for_execution_date(execution_date, all_results,
                                 all_predicted_tickers_list, all_models,
                                 n_estimators, include_nulls_in_test=False):
    global train_set, test_set
    train_set, test_set = split_train_test_by_period(
        data_set, execution_date,
        include_nulls_in_test=include_nulls_in_test)
    train_size, _ = train_set.shape
    test_size, _  = test_set.shape
    model = X_train = X_test = None

    if train_size > 0 and test_size > 0:
        model, evals_result, X_train, X_test = train_model(
            train_set, test_set, n_estimators=n_estimators)
        test_set["prob"] = model.predict(X_test)
        predicted_tickers = test_set.sort_values("prob", ascending=False)
        predicted_tickers["execution_date"] = execution_date
        all_results[execution_date] = evals_result
        all_models[execution_date]  = model
        all_predicted_tickers_list.append(predicted_tickers)
    return all_results, all_predicted_tickers_list, all_models, model, X_train, X_test
```

Main training loop — iterates over all execution dates:

```
all_results = {}
all_predicted_tickers_list = []
all_models = {}

execution_dates = np.sort(data_set["execution_date"].unique())

for execution_date in execution_dates:
    print(execution_date)
    (all_results, all_predicted_tickers_list, all_models,
     model, X_train, X_test) = run_model_for_execution_date(
        execution_date, all_results, all_predicted_tickers_list,
```

```
        all_models, n_trees, False)

all_predicted_tickers = pd.concat(all_predicted_tickers_list)
```

Parse evaluation results and plot:

```
def parse_results_into_df(set_):
    df = pd.DataFrame()
    for date in all_results:
        df_tmp = pd.DataFrame(all_results[(date)][set_])
        df_tmp["n_trees"] = list(range(len(df_tmp)))
        df_tmp["execution_date"] = date
        df = pd.concat([df, df_tmp])
    df["execution_date"] = df["execution_date"].astype(str)
    return df

test_results  = parse_results_into_df("valid_0")
train_results = parse_results_into_df("training")

test_results_final_tree  = (test_results.sort_values(["execution_date", "n_trees"])
                                        .drop_duplicates("execution_date", keep="last"))
train_results_final_tree = (train_results.sort_values(["execution_date", "n_trees"])
                                         .drop_duplicates("execution_date", keep="last"))

# Test weighted-return per execution date
(ggplot(test_results_final_tree)
 + geom_point(aes(x="execution_date", y="weighted-return"))
 + theme(axis_text_x=element_text(angle=90, vjust=0.5, hjust=1)))
```

Feature importance helper:

```
def draw_feature_importance(model, top=15):
    fi = model.feature_importance()
    fn = model.feature_name()
    feature_importance = pd.DataFrame(
        [{"feature": fn[i], "imp": fi[i]} for i in range(len(fi))])
    feature_importance = (feature_importance
                          .sort_values("imp", ascending=False)
                          .head(top)
                          .sort_values("imp", ascending=True))
    plot = (ggplot(feature_importance, aes(x="feature", y="imp"))
            + geom_col(fill="lightblue")
            + coord_flip()
            + scale_x_discrete(limits=feature_importance["feature"]))
    return plot
```