Analyzing FOMC Impact: Leveraging NLP for Sentiment Scores and Swap Rate Predictions

The Federal Open Market Committee (FOMC) minutes provide essential insights into the Federal Reserve's monetary policy and economic outlook. By analyzing NLP-derived sentiment and thematic scores from these minutes, this example seeks to explore whether these qualitative measures correlate with and potentially predict quantitative movements in federal swap rates. The research utilizes a linear statistical model to assess the predictive power of NLP scores, aiming to provide insights into how market expectations of future interest rates are influenced by FOMC communications. The results could offer valuable implications for enhancing prediction models and decision-making processes in financial markets.

More details can be found in Local Functions.

```
close; clear; clc; warning('off');
```

Data Preprocessing

In this example, we use the FOMC minutes released between March 2000 and July 2024. In the following local functions, we also retrieve the FOMC text data in the HTML form from webpage directly.

```
[data, reldate] = clean(2000:2024);
```

Sentiment Analysis on FOMC minutes

To avoid the runtime, we load the trained sentiment scores. If you are the first time running FinBERT model, you will need to download the pretrained model. The instructions can be found here.

```
todo = false;
if todo
    % load the FinBert
    mdl = finbert;
    tokenizer = mdl.Tokenizer;
    parameters = mdl.Parameters;
    scores = fnbt(jdata, mdl, tokenizer, parameters);
else
    load("scores.mat"); % the scores we got
end
```

Regression Analysis

This example is to investigate the swap rate using the statistical methods.

We hypothesize that the future swap rate is influenced by the current swap rate and the VIX rates from the latest two terms. Based on the assumption, we develop a regression model. To further enhance the model, we incorporate sentiment scores derived from the FOMC minutes, aiming to illustrate their additional impact on the model's explanatory power.

Swap Rates

In this example, we used 2-year daily/weekly federal swap rates from 2 sources, due to the limited access. In the following steps, we have to merge the 2 pieces together.

- Source 1: starting from September 30 2009: Bloomberg (swap-libor.xlsx).
- Source 2: starting from July 3 2000: Federal Reserve Bank St.Louis (DSWP2.csv & WSWP2.csv).

We used source 1 to fill the time periods not covered by source 2.

The dataset scores is less than rates, due to different frequencies. To match the length of the dataset rates, we applied the forward filling, replacing the missing values with the last observed value. By switching `d` and `w`, we can obtain the daily and weekly basis data. You may find some details of pairing in the local functions section.

```
Dregress = merge(scores, "d");
Wregress = merge(scores, "w");
```

Statistical Analysis

The data obtained from "merge" function is a matrix.

For "Dmdl_2" and "Wmdl_2", the first column contains the dates, the second column contains the latest updated swap rates, the third column contains latest updated sentiment scores, the fourth column contains the one previous swap rates, the fifth column contains the latest updated VIX rates, the last column contains the one previous VIX rates.

For "Dmdl_1" and "Wmdl_1", the columns are identical to "Dmdl_2" and "Wmdl_2", except that they do not include the column of sentiment scores.

Daily Basis

```
regress = zeros(size(Dregress(:, 2:end)));
regress(:, 1) = Dregress(:, 2);
regress(:, 2) = Dregress(:, 4);
regress(:, 3) = Dregress(:, 3);
regress(:, 4:end) = Dregress(:, 5:end);
regress = array2table(regress);
regress.Properties.VariableNames = {'Future_Swap_Rate', 'Sentiment_Score',
'Current_Swap_Rate', 'Latest_VIX_Rate', 'Second_Latest_VIX_Rate'};
Dmdl_1 = fitlm(regress,'Future_Swap_Rate ~ Current_Swap_Rate + Latest_VIX_Rate +
Second_Latest_VIX_Rate', Intercept = false)
Dmdl_2 = fitlm(regress,'Future_Swap_Rate ~ Sentiment_Score + Current_Swap_Rate +
Latest_VIX_Rate + Second_Latest_VIX_Rate', Intercept = false)
```

Weekly Basis

```
clear regress % clear regress
regress = zeros(size(Wregress(:, 2:end)));
regress(:, 1) = Wregress(:, 2);
regress(:, 2) = Wregress(:, 4);
regress(:, 3) = Wregress(:, 3);
regress(:, 4:end) = Wregress(:, 5:end);
regress = array2table(regress);
```

```
regress.Properties.VariableNames = {'Future_Swap_Rate', 'Sentiment_Score',
'Current_Swap_Rate', 'Latest_VIX_Rate', 'Second_Latest_VIX_Rate'};
Wmdl_1 = fitlm(regress,'Future_Swap_Rate ~ Current_Swap_Rate + Latest_VIX_Rate +
Second_Latest_VIX_Rate', Intercept = false)
Wmdl_2 = fitlm(regress,'Future_Swap_Rate ~ Sentiment_Score + Current_Swap_Rate +
Latest_VIX_Rate + Second_Latest_VIX_Rate', Intercept = false)
```

Model Comparison

We compared the adjusted R^2 for each model and performed the Ftest to compare the fitness of two nested models fit with least-square regression.

```
Models = {'Daily Models w.o. Sentiment Scores';'Daily Models w.t. Sentiment
Scores';'Weekly Models w.o. Sentiment Scores';'Weekly Models w.t. Sentiment
Scores'};
AdjRSquare =
[Dmdl_1.Rsquared.Adjusted;Dmdl_2.Rsquared.Adjusted;Wmdl_1.Rsquared.Adjusted;Wmdl_2.R
squared.Adjusted];
TResults = table(Models, AdjRSquare)
[Df, Dt] = ft(Dmdl_2, Dmdl_1) % daily basis model
[Wf, Wt] = ft(Wmdl_2, Wmdl_1) % weekly basis model
```

From the results, we can see that the adjusted coefficients of determination R^2 show that the models with sentiment scores perform better than models without them. Moreover, the p-values obtained from F test show that the models with sentiment scores are statistically significantly different from models without them. That means NLP results derived from FOMC minutes do enhance the prediction of future Federal Swap Rates.

Local Functions

FOMC minutes Data Preprocessing

This function retrieves the texts of FOMC minutes directly from web pages. The process begins by obtaining the URLs for these files. For the most recent files, the function retrieves the URLs from https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm. For earlier files, the function accesses various websites year by year to gather the necessary URLs.

Once the websites are read, the function searches within the HTML content for URLs of the FOMC HTML files. These URLs are embedded within HTML anchor tags. After collecting all the relevant URLs, the function fetches the content from these links and processes the HTML to extract the text of the FOMC minutes.

Moreover, FOMC holds at most 8 meetings within a year, this information is used to increase efficiency.

Fetch the FOMC

```
function txts = fetching(years)
    links = cell(length(years)*8, 1); % 8 meetings within a year at most
    year = 0;
    httpsUrl = "https://www.federalreserve.gov/monetarypolicy/fomchistorical"; %
obtain earlier minutes first
    p1 = '<a href=\"\/fomc/minutes/.{0,8}.htm\">'; % possible HTML anchors
```

```
    p2 = '<a href=\"\/monetarypolicy/fomc.{0,8}.htm\">';
    p3 = '<a href=\"\/monetarypolicy/fomcminutes.{0,8}.htm\">';
    pattern = strcat(p1, "|", p2, "|", p3); % merging the patterns

    for i = years
        Url = strcat(httpsUrl, string(i), ".htm");
        try
            data = webread(Url);
        catch ME % once not valid, break the loop, switch the httpsUrl and fetch
the latest files instead
            break;
        end

        matchEnd = regexp(data, pattern, 'match', 'all');

        for id = 1:length(matchEnd)
            link = matchEnd{id};
            new_link = strcat('https://www.federalreserve.gov', link(10:(end -
2))); % build the URLs for HTML files
            links{id + year*8} = new_link;
        end
        year = year + 1;
    end
    links = links(~cellfun(@isempty, links));

    httpsUrl = "https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm"; %
fetch the latest files
    data = webread(httpsUrl);
    matchEnd = regexp(data, pattern, 'match', 'all');
    matchEnd = matchEnd(end:-1:1);
    link = cell(length(matchEnd), 1);
    i = 1;
    while i <= length(matchEnd)
        sig = min(i + 7, length(matchEnd));
        match = matchEnd(sig:-1:i);

        link(i:sig) = match;
        i = i + 8;
    end

    for i = 1:length(link)
        l = link{i};
        link{i} = strcat('https://www.federalreserve.gov', l(10:(end - 2)));
    end

    urls = [links; link];

    txts = cell(length(urls), 1); % get the texts
    for i = 1:length(urls)
        url = urls{i};
```

```
        code = webread(url);
        tree = htmlTree(code);
        txt = splitlines(extractHTMLText(tree));
        txts{i} = txt;
    end
end
```

Text Cleaning

Those texts are initially split by paragraphs. All that we need to do is to find certain transition sentences as signals, we then obtain the indices of the paragraphs where the signals are in and get the useful paragraphs we need.

How to find those signals is up to you. In this project, we focus on the views on economics and finance. The name of the candidates, what are they planning and how well their works are done are unrelated to our goals. We also obtain the releasing dates of those minutes during the process, which are the earliest accessible time of the FOMC minutes rather than the convening dates.

```
% reldate contains the releasing dates of the minutes
% jdata contains the rejoined cleaned data
function [jdata, reldate] = clean(years)
    txts = fetching(years);
    reldate = zeros([length(txts), 1]);
    data = cell(length(reldate), 1); % data contains the cleaned data split by
paragraphs
    for j = 1:length(txts)
        txt = txts{j};
        reldate(j) = time(txt); % this is a utility function for the releasing dates

        txt = txt(txt ~= "");
        be1 = find(contains(txt, "The information reviewed", "IgnoreCase", true),
1, "first");
        be2 = find(contains(txt, "The information provided", "IgnoreCase", true),
1, "first");
        be3 = find(contains(txt, "The information received", "IgnoreCase", true),
1, "first");
        be4 = find(contains(txt, "The information available", "IgnoreCase", true),
1, "first");
        be5 = find(contains(txt, "The data available", "IgnoreCase", true), 1,
"first");
        be6 = find(contains(txt, "Staff Review of the Economic Situation",
"IgnoreCase", true), 1, "first") + 1;
        be7 = find(contains(txt, "Consumer spending", "IgnoreCase", true), 1,
"first");
        be = min([be1, be2, be3, be4, be5, be6]);
        if isempty(be)
            be = be7;
        end

        if sum(contains(txt, "At the conclusion of", "IgnoreCase", true)) > 1
```

```
            t = find(contains(txt, "At the conclusion of", "IgnoreCase", true)) - 1;
            for h = 1:length(t)
                if t(h) <= be
                    continue;
                else
                    t = t(h);
                    break;
                end
            end
        else
            t = find(contains(txt, "At the conclusion of", "IgnoreCase", true), 1,
"last") - 1;
        end

        if sum(contains(txt, "Committee Policy Action", "IgnoreCase", true)) == 1
            m = find(contains(txt, "Committee Policy Action", "IgnoreCase", true),
1, "last") + 1;
        elseif sum(contains(txt, "discussion of monetary policy", "IgnoreCase",
true)) > 1
            m = find(contains(txt, "discussion of monetary policy", "IgnoreCase",
true));
            for h = 1:length(m)
                if m(h) <= be
                    continue;
                else
                    m = m(h);
                    break;
                end
            end
        else
            m = find(contains(txt, "discussion of monetary policy", "IgnoreCase",
true), 1, "last");
        end
        en = min([t, m]);

        txt = txt(be:en);
        data{j} = txt(strlength(txt) > 100);
    end

    jdata = cell(length(data), 1); % rejoin the lines to score one file as a whole
    for i = 1:length(data)
        jdata{i} = strjoin(data{i}, " ");
    end
end
```

Assign the Sentiment Scores using FinBERT model

FinBERT loads a pretrained BERT transformer model for sentiment analysis for financial text.

`[sentiment, scores] = finbert.sentimentModel(X, parameters)` classifies the sentiment of the input `1-by-numInputTokens -by-numObservations` array of encoded tokens with the specified parameters. The output sentiment includes a categorical array with categories "positive", "neural"

```
function scores = fnbt(data, mdl, tokenizer, parameters)
    scores = zeros([1, length(data)])';
    for i = 1:length(data)
        str = data{i};
        tokens = encode(tokenizer, str);
        X = padsequences(tokens, 2, "PaddingValue", mdl.Tokenizer.PaddingCode);
        [~, scrs] = finbert.sentimentModel(X, parameters);
        scores(i) = sum(scrs);
    end
end
```

Releasing Dates Extraction

```
function release = time(txt)
    ind = find(contains(txt, "last update:", "IgnoreCase", true), 1, "last");
    str = txt(ind);
    pattern = '(\w+ \d+, \d{4})';
    matches = regexp(str, pattern, 'match');
    release = convertTo(datetime(matches{1}), "yyyymmdd");
end
```

Combining

This function is used for combining 2 pieces of swap rates with different frequencies. It will return a matrix that contains the dates and the corresponding swap rates. You can set freq as `w` for weekly data and `d` for daily data.

```
function swaps = combine(file1, file2, freq) % combine 2 pieces of swap rates
together
    swaps1 = readtable(file1);
    swaps2 = readtable(file2);

    swaps1.DATE = convertTo(swaps1.DATE, "yyyymmdd");
    swaps2.Date = convertTo(swaps2.Date, "yyyymmdd");
    swaps2 = swaps2(height(swaps2):-1:1, 1:2);

    if freq == "w" % you may choose weekly data
        swaps2 = swaps2(1:7:end, 1:2);
    end

    if freq == "w"
        mgind = find(swaps2.Date >= (swaps1.DATE(end) + 7), 1, "last");
    else
        mgind = find(swaps1.DATE <= swaps2.Date(1), 1, "last"); % use source 2
after September 30 2009
    end
    dates = [swaps1.DATE(1:mgind); swaps2.Date];
```

```
    if freq == "w"
        rates = [swaps1.WSWP2(1:mgind); swaps2.LastPrice]; % combined swap rates
    else
        rates = [swaps1.DSWP2(1:mgind); swaps2.LastPrice];
    end

    rates = fillmissing(rates, "knn", 5);
    swaps(:, 1) = dates;
    swaps(:, 2) = rates;
end
```

Pairing

This function is used for pairing the predictors to the corresponding dates, it returns the latest values of the predictor corresponding to those dates. By choosing back = "true", you may add a "one previous" column of the predictor to the return matrix.

```
function pfactor = pair(factor, resp, back)
    % factor: a matrix that contains the dates and the values of a predictor
    % resp: a matrix that contains the dates and the values of the swap rate
    dates = factor(:, 1);
    params = factor(:, 2);
    r1 = resp(:, 1);
    pfactor = zeros([length(r1), 1]);

    for i = 1:length(r1)
        idx1 = find(dates < r1(i), 1, "last");
        idx2 = idx1 - 1;
        pfactor(i, 1) = params(idx1);
        if back % you may choose to obtain the one previous data
            pfactor(i, 2) = params(idx2);
        end
    end
end
```

Merging

This function is used for merging the columns, including the dates, the latest updated swap rates and the predictors.

```
% regress are the paired data for regression analysis
function regress = merge(scores, freq)
    if freq == "w"
        file1 = "WSWP2.csv";
        VIX = readtable("WVIX.csv");
    else
        file1 = "DSWP2.csv";
        VIX = readtable("DVIX.csv");
    end
    file2 = "swap-libor.xlsx";
```

```
    swaps = combine(file1, file2, freq);
    vix(:, 1) = convertTo(VIX.Date, "yyyymmdd");
    vix(:, 2) = fillmissing(VIX.Adj_Close, "knn", 5);

    pscores = pair(scores, swaps(2:end, :), false);
    pvix = pair(vix, swaps(2:end, :), true);
    regress = [swaps(2:end, :), swaps(1:(end - 1), 2), pscores, pvix];
end
```

Using the F-test to Compare Two models

This function is used for F-test. It returns the test result and its p-value.

If the models have different numbers of parameters, the formula becomes:

$$F = \frac{(SSE_1 - SSE_2)/(df_1 - df_2)}{SSE_2/df_2}.$$

SSE_1 is the sum of squares of the simpler model, and SSE_2 is the sum of squares of the more complicated model.

```
function [f, p] = ft(mmdl, mdl)
    f = ((mdl.SSE - mmdl.SSE)/(mdl.DFE - mmdl.DFE))/(mmdl.SSE/mmdl.DFE);
    p = 1 - fcdf(f, (mdl.DFE - mmdl.DFE), mmdl.DFE);
end
```