Technical Documentation

The Trio

2025-04-18

I. Introduction

1. Project overview

Macroeconomic forecasting plays a critical role in shaping national policy, but it faces a major challenge where real-time GDP data is often revised over time, making forecasting decisions difficult. Research (Croushore and Stark, 2001) has shown that forecasts using real-time data tend to be less accurate than those using revised data whereby traditional models may not reflect the actual economic situation.

Our project addresses this issue by developing a web-based interactive application that allows users to benchmark various time-series forecasting models; using real-time and revised GDP data. Our application enables users to visualize and compare the accuracy and robustness of different models across data vintages and forecast horizons.

Our main goal is to help policymakers and economists evaluate how forecasts would have performed in real-time, quantify the impact of data revisions, and identify which models remain reliable under changing economic conditions.

2. Overall design

Our application consist of two main tabs: Dataset and Model. The Dataset Tab is where we allow user to either work with our provided sample dataset or upload their own. Here user will also set up their interest forecasting period. We also provide a preview of the cleaned data from the starting period, along with visualizations to help users to examine the differences in current and vintage data.

The Model tab allows users to select forecasting models for evaluation and customize them by specifying key parameters and features. This interactive setup ensures that users can experiment with different models and directly observe their impact on forecasting performance. Error metrics and visualization will be provided to assess the results.

3. Data and methodology

Our primary data source is available at https://www.philadelphiafed.org/surveys-and-data/real-time-data-research/routput We also use the FRED API to source additional economic data for model enhancement.

In this project, we evaluate model performance using two different data settings: vintage data (data available at the time of forecasting) and latest vintage data (revised data). This comparison helps quantify the impact of data revisions on forecast accuracy.

We focus on three models: - Autoregressive (AR): A benchmark model that uses only past values of GDP growth to generate forecasts. - Autoregressive Distributed Lag (ADL): Extends the AR model by incorporating additional economic indicators. - K-Nearest Neighbors (KNN): A non-parametric machine learning model that relies on historical patterns

Model performance is assessed using two standard forecasting error metrics: Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). These metrics allow us to rank models consistently based on both accuracy and robustness across different data vintages.

II. Application Backend

1. Application Framework

Before moving into building the application, it is necessary to go through our logic. The framework of how our forecast works is as follow: - Current Vintage Prediction: For each point we want to forecast, we will use the vintage data at that point to train the model and generate forecast. For instance, a forecast for 2000Q1 GDP growth will use the 2000Q1 vintage data to generate forecast. This is the real-time data that we have in 2000Q1 about all other dates. - Latest Vintage Prediction: We will use the latest vintage data for training our model (2025Q1), however, we use only use data up until the time of prediction. Thus, a 2000Q1 forecast with the latest data will only use GDP Growth up until 1999Q4. - Evaluation: We will then generate two sequences of forecasts, and calculate the error metrics of the sequences with the real value in the latest vintage data. The reason is that this most recent data is what closest to the truth by constantly going through revision.

2. Data Preparation

For data selection, users can either work with our provided sample dataset or upload their own. For sample dataset, we are using the real-time dataset available here for download: https://www.philadelphiafed.org/surveys-and-data/real-time-data-research/routput

For upload option, we currently support uploads in csv, xlsx, or json formats. Additionally, the uploaded dataset must follow a structure similar to the FRED data format to ensure compatibility with our processing.

Our initial plan was to incorporate both quarter vintages and monthly vintages. In order to do this, we need to detect the frequency of the dataset that users chose. This is where we define a function that can detect the frequency of dataset as below.

```r
detect_frequency <- function(data) {
    # Get first vintage column name
    first_col <- names(data)[2]

    if (str_detect(first_col, "M\\d+$")) {
      return("monthly")
    } else if (str_detect(first_col, "Q\\d$")) {
      return("quarterly")
    } else {
      stop("Unsupported Uploaded File!")
    }
  }
```

After we have select our data, we start our data cleaning process. We begin by cleaning the column names, which are in the format "ROUTPUT65Q1", to extract the correct year and quarter. For each subsequent column, we compare the last two digits of the year with the previous one: If the new year is greater than or equal to the previous, we keep the current prefix.

If it's smaller, this signals a rollover into the next century, so we increment the prefix by 1 (e.g., from "19" to "20").

For example, from "98" to "99", we stay in the 1900s. When it shifts from "99" to "00", we move to the 2000s.

```
## Get the right century for the year
  clean_columns <- function(data) {

    data_cols <- names(data)[-1] %>%
      str_remove(pattern = "ROUTPUT")

    yy <- str_sub(data_cols, start = 1, end = 2)
    prev_yy <- yy[1]
    century = 19
    complete_year <- c()
    for (i in 1:length(yy)) {
      cur_yy <- yy[i]
      if (as.numeric(cur_yy) < as.numeric(prev_yy)) {
        century = century + 1
      }
      complete_year[i] <- paste0(century, cur_yy)
      prev_yy <- cur_yy
    }
    return(complete_year)
  }
```

This function will get us the right prefix for each column. As a result, we can use this function and concatenate the prefix with the year and quarter/month and get a better column name ,for instance, "ROUTPUT65Q1" to "1965Q1".

Next, we clean the dataset by reshaping into long format, allowing us to extract and organize the columns into year, quarter, v_year (vintage year), v_quarter/v_month (vintage quarter/month) and the corresponding GDP value. This structure makes it easier to filter for the corresponding vintage in future analysis.

```r
clean.data <- function(data, vintage_freq = "quarterly") {

    q <- names(data)[-1] %>% str_remove(pattern = "ROUTPUT") %>% str_sub(start = 1)
    clean_cols <- paste0(clean_columns(data), q)
    names(data)[-1] <- clean_cols
    total_col <- length(names(data))
    ## Clean the data
    clean_data <- data %>%
      mutate(across(2:total_col, as.numeric)) %>%
      pivot_longer(cols = -1, names_to = "vintage",
                   values_to = "current_vintage") %>%
      mutate(year = str_sub(DATE, 1,4),
             quarter = str_sub(DATE, 7,7),
             v_year = str_sub(vintage, 1,4),
             log_current_vintage = log(current_vintage)) %>%
      drop_na()

    if (vintage_freq == "quarterly") {
      final_data <- clean_data %>%
        mutate(v_quarter = str_extract(vintage, pattern = "(?<=Q).*$")) %>%
        select(year, quarter, v_year, v_quarter,
               current_vintage, log_current_vintage) %>%
        mutate(across(1:6, as.numeric)) # Turn all value to numeric
    }
    else {
      final_data <- clean_data %>%
        mutate(v_month = str_extract(vintage, pattern = "(?<=M).*$")) %>%
        select(year, quarter, v_year, v_month, current_vintage, log_current_vintage) %>%
        mutate(across(1:6, as.numeric))
    }


    return(final_data)
  }
```

As the project progressed, we chose to focus exclusively on quarterly vintage data. This decision simplifies the modeling process while still aligning with our primary goal: quantifying the impact of data revisions. With that being said, it is completely possible to extend our model to accompany monthly vintage data as the structure would be very similar to how we handle quarterly data.

Moving on, we define a filtering function to extract data for a specific vintage, based on the selected vintage year and quarter. This function will take in the vintage year and vintage quarter and output the filter table with the same structure, with the additional columns that we need such as current vintage gdp level (current_vintage) and vintage growth (current_growth)

```r
filter_function <- function(v_year1, v_quarter1) {
    cleaned_data() %>% filter(v_year == v_year1,
                              v_quarter == v_quarter1) %>%
      mutate(lag_current_vintage = lag(current_vintage,1),
             log_lag_current_vintage = log(lag_current_vintage),
             current_growth = 400*(log_current_vintage - log_lag_current_vintage))

  }
```

Due to missing data from earlier years in some vintages, we begin all analyses from 1965, which is also the first available vintage in our dataset. However, this introduces a challenge: if a user selects a forecast starting point close to 1965, the model will have limited historical data to train on. This lack of training data may negatively impact model performance and forecast reliability. One solution would be to restrict the range of vintages that users can choose to allow a certain level of training size. This would be a potential area for further research to determine the suitable range.

Model Construction

1. Autoregressive (AR)

We choose our baseline model to be the AR model. The AR model serves well as the baseline model because it is a relatively intuitive model. The idea behind the choice is that past growths (lags) of GDP could be used to predict GDP growth in the next period.

First, we define a function to fit an AR model with lag p. Since we are using its own lag for regression, the function will only need to take in the data for the target variable Y, accompany with p for the number of lags and h for forecast horizon.

```
fitARp=function(Y,p,h){

    #Inputs: Y- predicted variable,  p - AR order, h -forecast horizon
    aux=embed(Y,p+h) #create p lags + forecast horizon shift (=h option)
    y=aux[,1] #  Y variable aligned/adjusted for missing data due to lags
    X=as.matrix(aux[,-c(1:(ncol(Y)*h))]) # lags of Y corresponding to forecast horizon
    if(h==1){
      X.out=tail(aux,1)[1:ncol(X)] #retrieve last p observations if one-step forecast
    }else{
      X.out=aux[,-c(1:(ncol(Y)*(h-1)))] #delete first (h-1) columns of aux,
      X.out=tail(X.out,1)[1:ncol(X)] #last p observations to predict T+1
    }

    model=lm(y~X) #estimate direct h-step AR(p) by OLS
    coef=coef(model) #extract coefficients
    #make a forecast using the last few observations: a direct h-step forecast.
    pred=c(1,X.out)%*%coef

    return(list("pred"=pred))
  }
```

After we have implement our function, we define another function to run our forecast precedure through all the forecasting point

```r
run_ar <- function(h, p) {
    # A dataframe to store the result
    results <- data.frame('year' = numeric(0),
                          'quarter' = numeric(0),
                          'cur_pred' = numeric(0),
                          'latest_pred' = numeric(0),
                          'latest_growth' = numeric(0))


    for (i in 1:nrow(forecast_list())) {
      item <- forecast_list()[i,]
      year_f <- item$year
      quarter_f <- item$quarter
      # Filter for correct vintage data
      final_data <- filter_function(v_year1 = year_f, v_quarter1 = quarter_f) %>%
        select(year, quarter, current_growth) %>%
        right_join(latest_vintage_data(), by = c('year', 'quarter')) %>%
        filter(year >= 1965, year <= year_f) %>% drop_na() %>%
        select(year,quarter, current_growth, latest_growth)
      # Data process to input into model
      Y_cur <- as.matrix(final_data %>% pull(current_growth))
      Y_latest <- as.matrix(final_data %>% pull(latest_growth))
      # Fitting the model
      model_cur <- fitARp(Y = Y_cur, p = p, h = h) # Current Vintage
      model_latest <- fitARp(Y = Y_latest, p = p, h = h) # Latest Vintage
      pred_cur <- model_cur$pred[1]
      pred_latest <- model_latest$pred[1]
      # Assemble the result
      result_df <- data.frame("year" = year_f, "quarter" = quarter_f,
                              "cur_pred" = pred_cur, "latest_pred" = pred_latest)
      result_df <- result_df %>% left_join(actual_data(), by = c("year", "quarter")) %>%
        rename("latest_growth" ="value")
      results <- rbind(results, result_df)
    }
    return(results)
  }
```

2. Autoregressive Distributed Lag (ADL)

3. K Nearest Neighbours (KNN)

Our third model is K-Nearest Neighbors (KNN). KNN regression has been widely studied for time series forecasting, and research has consistently shown it to perform well in capturing nonlinear patterns in the data (Lora et al., 2007; Zhang et al., 2017) We choose KNN to explore the selfexplanatory power of GDP growth and compare this with parameter tuning models. We utilized the tsfknn package for using KNN regression for time-series forecast. More information on the package can be found here: https://github.com/franciscomartinezdelrio/tsfknn

To generate an h-step ahead forecast, we use data only up to time T-h. This approach ensures that our model does not unintentionally peek into the future. We construct lag-based features by using the first four lags of the target variable as autoregressive inputs. Since we are forecasting multiple steps ahead (rather than just the next time point), we also need to specify a multi-step ahead strategy. In our case, we use the Multiple Input Multiple Output (MIMO) method, which allows the model to generate all future predictions in one step. This avoids the compounding error problem commonly encountered in recursive forecasting. MIMO is therefore more stable and suitable for economic forecasting tasks like GDP growth, where forecast horizons often extend several quarters ahead. Then we extract the latest forecast which will be at time T for our purpose. The loop will run through all forecast point in the forecast list, generate forecast and assemble the result for evaluation.

```r
run_prediction_knn <- function(h, k, cf) {
    results = data.frame(year = numeric(0), quarter = numeric(0), cur_forecast = numeric(0), la
    for (i in 1:nrow(forecast_list())) {
      item = forecast_list()[i,]
      year_f = item$year
      quarter_f = item$quarter
      data <- filter_function(v_year1 = year_f, v_quarter1 = quarter_f) %>% left_join(latest_vi
        select(year, quarter, current_growth, latest_growth) %>% filter(year >= 1965)
      data <- data[1:(nrow(data)-h+1),] # Extract data up until T - h
      train_data_cur <- data %>% select(-latest_growth)
      train_data_latest <- data %>% select(-current_growth)
      ts_train_cur <- ts_transform(train_data_cur)
      ts_train_latest <- ts_transform_latest(train_data_latest)
      model_current <- knn_forecasting(ts_train_cur, h = h, lags = 1:4, k = k, msas = "MIMO", c
      cur_pred <- tail(model_current$prediction, 1)
      model_latest <-  knn_forecasting(ts_train_latest, h = h, lags = 1:4, k = k, msas = "MIMO"
      latest_pred <- tail(model_current$prediction, 1)
      results[i, 'year'] = year_f
      results[i, 'quarter'] = quarter_f
      results[i, 'cur_forecast'] = cur_pred
      results[i, 'latest_forecast'] = latest_pred

    }
    return(results)
  }
```

III. Application Frontend