COPOM Minutes Analysis.

Iven da S. Valpassos

My work is in the field of economics and finance and it is amazing how we can extract useful information from almost any type of data using the available technology. But with such a variety of tools at our disposal, what is the best option?

A current discussion among some of my fellow researchers is about “Python vs R”, what language is the most complete for data science (in a very borad sense) practitioners?

Although you can do almost everything sticking with only one option, to my mind (and it is my very humble opinion) the use of the best features of each language can give us better results.

When it comes to machine learning, I think that sklearn and some other Python tools are better options than those we have in R. R, however, is my choice for data munging, inferential statistics and (some) plots.

Luckyly enough, we have at our disposal, for free, tools that make using both languages simultaneously almost a walk in the park, why should we don’t do it?

In this example, we are going to make an analysis of the Brazilian Central Bank Monetary Policy Committee (COPOM) minutes using both Python and R.

1 - We convert COPOM minutes (pdf files) into a corpus;

2 - Word clouds of different periods;

3 -Counting words in each minute;

4- Most common words for each presidency;

5 -Logistic regression to classify minutes as dovish or hawkish;

6- SVM to classify minutes as dovish or hawkish;

7- Conclusions.

The package “reticulate” is loaded. It allows us to work with Python and R interchangeably. Also, we install Python packages required for the analysis.

Now, we load all R packages needed to perform our analysis.

library(tm)  
library(pdftools)  
library(qdap)  
library(RWeka)  
library(wordcloud)  
library(e1071)  
library(readxl)  
library(tidyverse)  
library(data.table)  
library(tidytext)  
library(lubridate)  
library(scales)

The minutes are transformed into a VCorpus.

setwd("C:/atas\_bcb")  
  
directory<- c("C:/atas\_bcb/texts")  
  
copom\_corpus <- VCorpus(DirSource(directory),   
 readerControl = list(reader = readPDF, language = "English"))

Now we have a corpus and we can start cleaning our data. The process involves changing words capitalization, removing punctuation, change number for text format and removing words that don’t aggregate information (stop wors).

copom\_corpus<- tm\_map(copom\_corpus, content\_transformer(tolower))  
  
copom\_corpus<- tm\_map(copom\_corpus,  
 content\_transformer(removePunctuation))  
  
copom\_corpus<- tm\_map(copom\_corpus, content\_transformer(removeNumbers))  
  
copom\_corpus<- tm\_map(copom\_corpus, stripWhitespace)  
  
custom\_stop\_words<- c("committee","jan", "january","feb","february","mar","march","apr","april","may","jun","june", "jul","pp", "july","aug","august","sep","september", "oct","october","nov","november","dec", "december","bps","basis","month","months","bank","central", "brasil", "brazil","index","month","year","yearly","compared","banco", "central","information","fourweek","gerinbcbgovbr","gcibacenbcbgovbr","gerin","fourweek","question",  
 "questions","due",  
 "disclosure", "billion", "million","intended", "intention", "copom", "communicate","bcbgovbr","bind","according", "gcibacenbcbgovbr", "meet", "meeting","th","comments","\_head","quarter","ended","members","ended","discussed","focus survey","actions bcbgovbr","relevant horizons","deputy","governor","-head","gcibacenbcbgovbr", "yearoveryear","monthtomonth","head","-head","baseline","conduct","-","fabio","kanczuk","fernanda","bruno","joao","manoel","carolina","fernandes", "de","campos","neto","feitosa","mello", "maurício","nechio", "inflation","increase","increased", "brazilian","department","prices","price", "twelve","monetary","policy","monthonmonth","occurs", "occurrence","percent")  
  
  
stop\_words<- c(custom\_stop\_words,stopwords("en"))  
  
copom\_corpus<- tm\_map(copom\_corpus, removeWords, stop\_words)

You may wonder how do I reach to the conclusion about which words should be removed in addition to the standard stop words. Well, I am saving you some time as I have already messed around with this data dozens of times, but feel free to include or exclude some tokens.

Moving to the next step, we extract each word or combination of words present in the minutes. I choose tokens of 1 word and combinations of 2 words. In addition, I have made some changes to column names, removing “.pdf” from them and converting the column names from text to numbers.

The matrix we built is a TermDocumentmatrix (TDM). The absolute frequency of the words is wheighted by the inverse of the number of documents the word occurs (TfIdf).

#function to tell how we extract tokens from text. In this case, tokens of 1 word and 2 words.  
  
tokenizer<- function(x){  
   
 NGramTokenizer(x, Weka\_control(min = 1, max = 2))  
}  
  
copom\_tdm\_weighted<- TermDocumentMatrix(copom\_corpus, control = (list(weighting = weightTfIdf,tokenize = tokenizer))) #creates a TDM  
  
copom\_tdm\_matrix\_weighted<- as.matrix(copom\_tdm\_weighted)  
  
new\_names<- str\_remove\_all(colnames(copom\_tdm\_matrix\_weighted),".pdf")  
  
new\_names<- as.numeric(new\_names)  
  
colnames(copom\_tdm\_matrix\_weighted)<- new\_names  
  
x<- seq(from = 1, to = 165, by= 1)  
  
z<- as.character(x)  
  
y<- copom\_tdm\_matrix\_weighted[,z]  
  
copom\_tdm\_matrix\_weighted<-y  
  
indices<- rownames(copom\_tdm\_matrix\_weighted)#this indexes will be used later

Now, things become more exciting, we convert a R matrix into a Python data frame.

But before we do it, we are loading all Python tools we need.

import pandas as pd  
from wordcloud import WordCloud  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LogisticRegression  
from sklearn.linear\_model import SGDClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import confusion\_matrix  
from sklearn.model\_selection import GridSearchCV  
from sklearn.svm import SVC  
import numpy as np

indexes = list(r.indices)   
copom\_tdm\_matrix\_weighted\_py = pd.DataFrame(r.copom\_tdm\_matrix\_weighted,index=indexes)

print(copom\_tdm\_matrix\_weighted\_py.iloc[0:6,0:10])

## 0 1 2 3 ... 6 7 8 9  
## – – 0.000645 0.000000 0.00000 0.000000 ... 0.0 0.0 0.0 0.0  
## – ’s 0.000000 0.000000 0.00000 0.000000 ... 0.0 0.0 0.0 0.0  
## – absence 0.000000 0.000000 0.00000 0.000000 ... 0.0 0.0 0.0 0.0  
## – accumulated 0.000000 0.000000 0.00000 0.000000 ... 0.0 0.0 0.0 0.0  
## – adjustments 0.000000 0.000000 0.00000 0.000000 ... 0.0 0.0 0.0 0.0  
## – advisor 0.000202 0.000173 0.00019 0.000193 ... 0.0 0.0 0.0 0.0  
##   
## [6 rows x 10 columns]

copom\_tdm\_matrix\_weighted\_py['word\_freq']= copom\_tdm\_matrix\_weighted\_py.sum(skipna=True, axis=1)  
  
freq = copom\_tdm\_matrix\_weighted\_py["word\_freq"]  
  
freq.sort\_values(ascending=False)

## unrestricted foreign 0.187775  
## unrestricted 0.187775  
## exchange actions 0.187775  
## reforms 0.164171  
## goods 0.128710  
## ...   
## analyzed 0.000000  
## date 0.000000  
## comply 0.000000  
## comply targets 0.000000  
## expectations 0.000000  
## Name: word\_freq, Length: 117371, dtype: float64

w = WordCloud(width=400, height=300, mode='RGBA', background\_color="white", max\_words=200).fit\_words(freq)

plt.imshow(w)

plt.show()

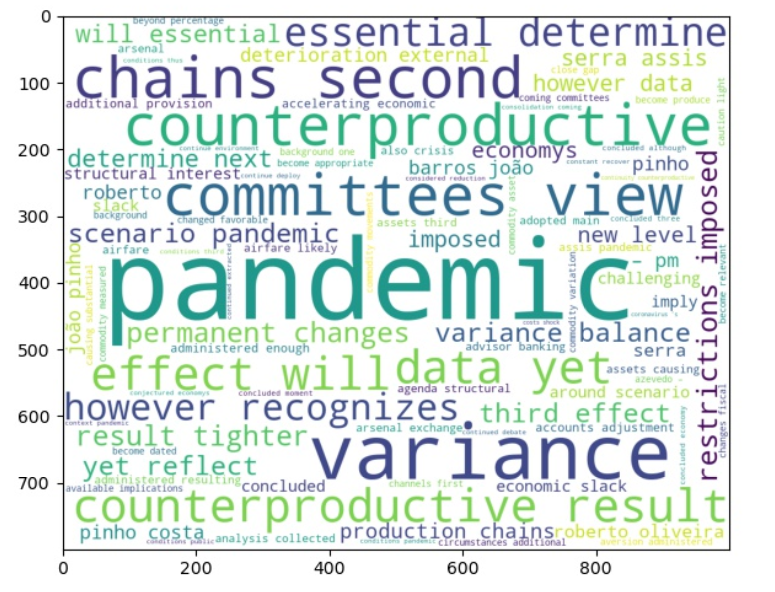


Our data comprises COPOM metings from Dec/2001 to Mar/2020. We can see that “exchange policy”, “reforms”, “administered contracts” are strong terms during the years under analisys. Also, confirming our status of small open economy “foreign” is also highlighted.

What about COPOM’s last meeting? Is there a concern about corona virus? We can check it.

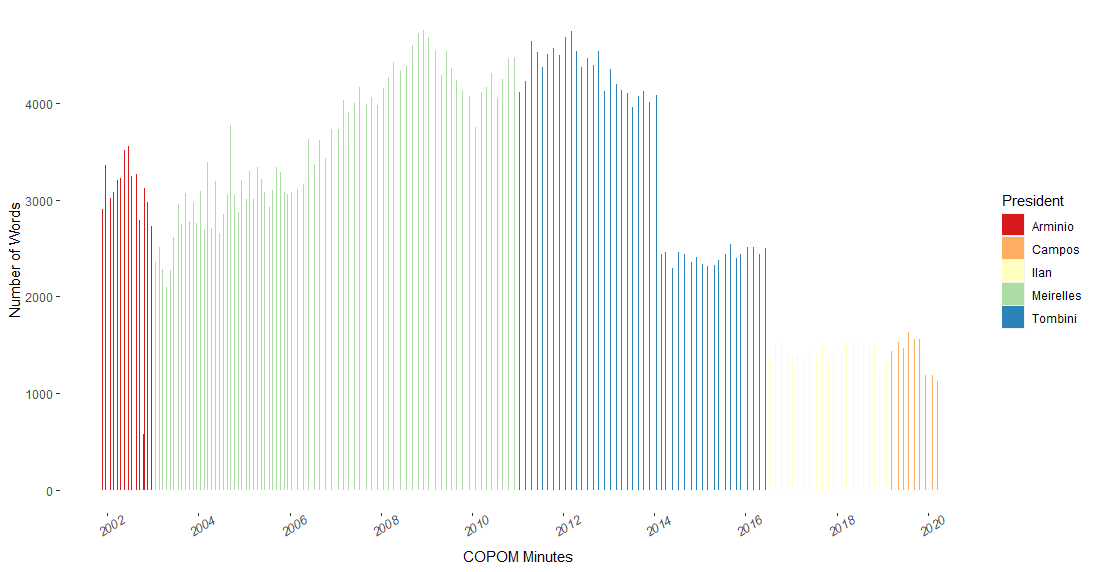
copom\_df<- data.frame(copom\_tdm\_matrix\_weighted, check.names = FALSE)  
  
corona\_copom<- copom\_df%>%select("165")  
  
colnames(corona\_copom)<- c("freq")

corona\_py = pd.DataFrame(r.corona\_copom)  
  
w\_corona = WordCloud(width=1000, height=800, mode='RGBA',  
background\_color="black", max\_words=100).fit\_words(corona\_py["freq"])



Well, the image is self-explanatory.

counter<- colSums(copom\_tdm\_matrix\_weighted!=0)  
  
counter<- data.frame(counter)  
  
extra\_info<- read\_xlsx("Class.xlsx")  
  
names(extra\_info)<- c("COPOM\_Minute","Classification\_dov\_hawk","Date\_Meeting","President\_CBB")  
  
copom\_data<- cbind(counter,extra\_info)  
  
copom\_data$President\_CBB <- as.factor(copom\_data$President\_CBB)  
  
copom\_data$year<- as.Date(copom\_data$Date\_Meeting)  
  
copom\_data$Date\_Meeting <- as.character(copom\_data$Date\_Meeting)  
  
g1<- ggplot(data = copom\_data, aes(x = year ,y = counter, fill = President\_CBB)) + geom\_col() + scale\_x\_date(name = "COPOM Minutes ", date\_breaks = "2 years",labels = date\_format('%Y')) + scale\_fill\_brewer(palette = "Spectral") + theme(axis.text.x = element\_text(angle = 30, face = "italic", size = 10)) + labs( y = "Number of Words",fill = "President") +theme(panel.background = element\_blank())  
  
g1



The previous plot depicts the number of words on each COPOM Minute. We during the aftermath of 2008 crisis CBB has a lot to tell and comunicate about its actions. Under Ilan and Campos, CBB comunication became more direct and concise, although we must take into consideration the different global and comestic economic scenario.

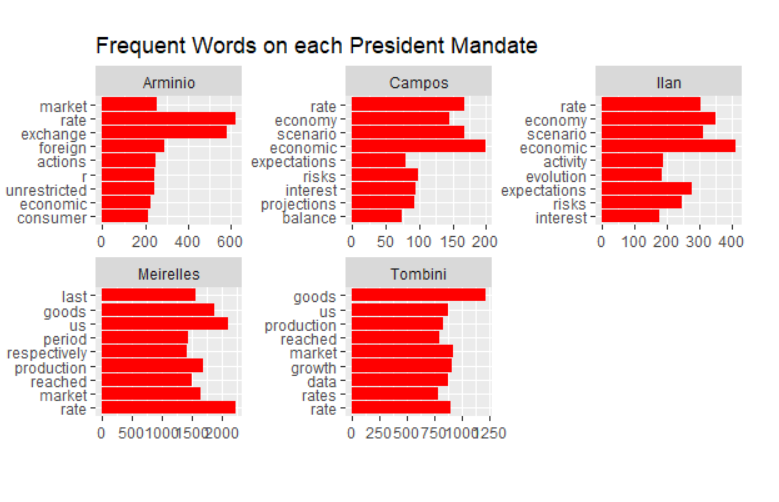
We are now interested in the most common words under the leadership of each Central Bank of Brasil (CBB) president. I will re-create the database using a different method. It gives us opportunity to revise a new method to create the database and it will make data manipulation easier on a later stage.

#creating links  
  
minutes\_links<- list()  
  
  
for (i in 1:165) {  
   
 minutes\_links[i] = print(paste("C:/Users/ivenv/OneDrive/Documentos/atas\_bcb/texts/",i,".pdf", sep = ""))  
   
}

minutes\_data<- unlist(minutes\_links)

copom\_text<- data.frame(presidente=copom\_data$Pres,stringsAsFactors = FALSE) %>%  
   
 mutate(text= map(minutes\_data,pdf\_text)) %>% unnest(text) %>%   
   
 group\_by(presidente) %>% mutate(page=row\_number()) %>%  
   
 ungroup() %>%  
   
 mutate(text=strsplit(text,"\r")) %>%   
   
 unnest(text) %>%   
   
 mutate(text=gsub("\n","",text)) %>%  
   
 group\_by(presidente) %>%   
   
 mutate(line=row\_number()) %>%   
   
 ungroup() %>%   
   
 select(presidente,line,page,text)

copom\_words<- copom\_text%>%select(presidente,page,line,text)%>% unnest\_tokens(word,text)  
  
copom\_words%>%mutate(word = gsub("[^A-Za-z ]","",word)) %>%  
   
 filter(word!="") %>%  
   
 filter(!word %in% stop\_words) %>%  
   
 group\_by(presidente) %>%  
   
 count(word, sort = TRUE)%>%  
   
 mutate(rank = row\_number()) %>%  
   
 ungroup() %>%  
   
 arrange(rank, presidente) %>%  
   
 filter(rank < 10) %>%  
   
 ggplot(aes(y=n, x = fct\_reorder(word,n))) +  
   
 geom\_col(fill = "red") + coord\_flip() +   
   
 facet\_wrap(~presidente, scales = "free") + labs(x = "", y = "", title = "Frequent Words on each President Mandate")



The minutes of Arminio’s period is marked by two terms “exchange” and “rate”. Nothing more natural considering that the “Argentine” crisis and the “Lula fear” crisis were both reflected in those COPOM Minutes. Tombini’s COPOM minutes have “growth” as one of the most frequent words. This is expected as under his chairmanship we experienced one of the worst domestic recessions in history.

Previously, in order to build a word cloud, we transformed our text data into a Term Document Matrix. Now, our aim is to predict the “tone” of the minute, that is, we want to know if the central bank is hawkish or dovish. For this purpose, we work with the same data, however, we transpose the matrix. The Document Term Matrix is now what we want.

tokenizer\_2<- function(x){  
   
 NGramTokenizer(x, Weka\_control(min = 1, max = 1))  
}  
  
copom\_dtm\_weighted<- DocumentTermMatrix(copom\_corpus, control = (list(weighting = weightTfIdf,tokenize = tokenizer\_2)))  
  
copom\_dtm\_m<- as.matrix(copom\_dtm\_weighted)  
  
new\_row\_names<- str\_remove\_all(row.names(copom\_dtm\_m),".pdf")  
  
new\_row\_names<- as.numeric(new\_row\_names)  
  
rownames(copom\_dtm\_m)<- new\_row\_names  
  
x<- seq(from = 1, to = 165, by= 1)  
  
z<- as.character(x)  
  
y<- copom\_dtm\_m[z,]  
  
copom\_dtm\_m<-y  
  
copom\_dtm\_df<- data.frame(copom\_dtm\_m, check.names = FALSE)  
  
copom\_dtm\_df[,"Class\_dov\_hawk"]<- extra\_info$Classification\_dov\_hawk

As I said before, “sklearn” is my favourite framework when it comes to machine learning. To make good use of it, we are now converting our data to Python format so we can run Logistic Regression and SVM to predict if the COPOM minute is hawkish or dovish. It must rest clear to the reader that this work is merely for learning purposes and there is no academic rigour on it. Our dataset has some drawbacks that make it harder to get to strong conclusions, the main issues are:

1. We have only 165 observations and both algorithms perform poorly with such a short sample;
2. The classification used to train the algorithm was made based on interest rates movements and by reading some of the minutes, but here the researcher bias can be strong.

copom\_dtm\_py = pd.DataFrame(r.copom\_dtm\_df)

y = copom\_dtm\_py.Class\_dov\_hawk  
  
X = copom\_dtm\_py.drop('Class\_dov\_hawk', axis=1)

y = copom\_dtm\_py.Class\_dov\_hawk  
  
X = copom\_dtm\_py.drop('Class\_dov\_hawk', axis=1)  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=20)  
  
lr = LogisticRegression()  
  
searcher = GridSearchCV(lr, {"C":[0.001,0.01,0.1,1,10]})  
  
searcher.fit(X\_train,y\_train)

## GridSearchCV(cv=None, error\_score=nan,  
## estimator=LogisticRegression(C=1.0, class\_weight=None, dual=False,  
## fit\_intercept=True,  
## intercept\_scaling=1, l1\_ratio=None,  
## max\_iter=100, multi\_class='auto',  
## n\_jobs=None, penalty='l2',  
## random\_state=None, solver='lbfgs',  
## tol=0.0001, verbose=0,  
## warm\_start=False),  
## iid='deprecated', n\_jobs=None,  
## param\_grid={'C': [0.001, 0.01, 0.1, 1, 10]},  
## pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=False,  
## scoring=None, verbose=0)

print("The best parameter value is", searcher.best\_params\_)

## The best parameter value is {'C': 0.001}

best\_lr = searcher.best\_estimator\_  
coefs = best\_lr.coef\_  
print("Number of features:", coefs.size)

## Number of features: 6881

print("Selected features:", np.count\_nonzero(coefs))

## Selected features: 6240

log\_reg = LogisticRegression(C=0.001).fit(X\_train,y\_train)  
  
y\_predicted = log\_reg.predict(X\_test)  
  
print('Train set accuracy: ', log\_reg.score(X\_train, y\_train))

## Train set accuracy: 0.5652173913043478

print('Test set accuracy: ', log\_reg.score(X\_test, y\_test))

## Test set accuracy: 0.5

print('Test set score accuracy: ', accuracy\_score(y\_test, y\_predicted))

## Test set score accuracy: 0.5

print('Confusion matrix: \n', confusion\_matrix(y\_test, y\_predicted)/len(y\_test))

## Confusion matrix:   
## [[0.5 0. ]  
## [0.5 0. ]]

Most important words.

inds\_ascending = np.argsort(log\_reg.coef\_.flatten())   
inds\_descending = inds\_ascending[::-1]  
  
words = list(X.columns)

Logistic regression allows us to see the words more related with a hawkish instance of COPOM (positive coeficients) and the words more associated with a dovish instance (negative coefficients).

print("Most hawkish words: ", end="")

## Most hawkish words:

for i in range(5):  
 print(words[inds\_descending[i]], end=", ")

## turn, realignment, changed, tends, evaluates,

print("\n")

print("Most dovish words: ", end="")

## Most dovish words:

for i in range(5):  
 print(words[inds\_ascending[i]], end=", ")

## reforms, slack, judges, agreed, agenda,

print("\n")

log\_reg = LogisticRegression().fit(X, y)  
  
print('Accuracy of logistic regression: ', log\_reg.score(X, y))

## Accuracy of logistic regression: 0.5454545454545454

prob\_class = log\_reg.predict\_proba(X)  
  
print(prob\_class)

We have seen that logistic regression is useful but due to our small sample the results are not so convincing. It is possible to try other classification algorithm. Doing this way, we can compare results and, more important, learn another approach.

svm = SVC(probability=True)  
  
parameters = {'gamma':[0.00001, 0.0001, 0.001, 0.01, 0.1]}  
  
sv\_best = GridSearchCV(svm,parameters)  
  
sv\_best.fit(X,y)

## GridSearchCV(cv=None, error\_score=nan,  
## estimator=SVC(C=1.0, break\_ties=False, cache\_size=200,  
## class\_weight=None, coef0=0.0,  
## decision\_function\_shape='ovr', degree=3,  
## gamma='scale', kernel='rbf', max\_iter=-1,  
## probability=True, random\_state=None, shrinking=True,  
## tol=0.001, verbose=False),  
## iid='deprecated', n\_jobs=None,  
## param\_grid={'gamma': [1e-05, 0.0001, 0.001, 0.01, 0.1]},  
## pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=False,  
## scoring=None, verbose=0)

print("Best CV params", sv\_best.best\_params\_)

## Best CV params {'gamma': 1e-05}

print("Test accuracy of best grid search hypers:",   
  
sv\_best.score(X,y))

## Test accuracy of best grid search hypers: 0.5454545454545454

sv\_best.predict\_proba(X)

It is also possible to directly compare and choose the best classifier, using SGDClassifier. As logistic regression and SVM differ basically on their loss functions, the SGDClassifier chooses the loss function and parameters that gives the best performance.

choice = SGDClassifier(random\_state=0)  
  
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1],   
 'loss':["hinge","log"], 'penalty':["l1","l2"]}  
searcher = GridSearchCV(choice, parameters, cv=10)  
  
searcher.fit(X\_train, y\_train)

## GridSearchCV(cv=10, error\_score=nan,  
## estimator=SGDClassifier(alpha=0.0001, average=False,  
## class\_weight=None, early\_stopping=False,  
## epsilon=0.1, eta0=0.0, fit\_intercept=True,  
## l1\_ratio=0.15, learning\_rate='optimal',  
## loss='hinge', max\_iter=1000,  
## n\_iter\_no\_change=5, n\_jobs=None,  
## penalty='l2', power\_t=0.5, random\_state=0,  
## shuffle=True, tol=0.001,  
## validation\_fraction=0.1, verbose=0,  
## warm\_start=False),  
## iid='deprecated', n\_jobs=None,  
## param\_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1],  
## 'loss': ['hinge', 'log'], 'penalty': ['l1', 'l2']},  
## pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score=False,  
## scoring=None, verbose=0)

print("Best CV params", searcher.best\_params\_)

## Best CV params {'alpha': 1e-05, 'loss': 'hinge', 'penalty': 'l1'}

print("Best CV accuracy", searcher.best\_score\_)

## Best CV accuracy 0.5719696969696969

print("Test accuracy of best grid search hypers:", searcher.score(X\_test, y\_test))

## Test accuracy of best grid search hypers: 0.5

Results:

Best CV params {‘alpha’: 0.001, ‘loss’: ‘hinge’, ‘penalty’: ‘l2’} Best CV accuracy 0.5643939393939392 Test accuracy of best grid search hypers: 0.5

The results point out that SVM (hinge loss) with C = 1000 (alpha = 0.001) and l2 regularization is the best option.

Although there is no clear difference on the accuracy of the two algorithms, the classification probabilities produced by SVM are more useful, as SVM responds better in the case of a small sample (as is our case here) when returning class probabilities. This caracteristic is useful if you are interested in building a dovishness/hawkishness index, for instance.

Maybe, some of the misclassifictions are due to the Central Bank of Brazil lack of credibility in part of the period under analisys. If we control the class probabilities by the credibility level of the CBB we expect to obtain (I still don’t know if this is the case) a probability closer to the real classification in our data. But it comes in the next article.