Machine Learning não é Data Mining

Integrantes:

- Dupla 1: Victor Rodrigues Matsuguma RA: 118893
  Tiago Lobato Gimenes RA: 118827
  Extração de dados referentes às buscas por sintomas de doenças pelo Google Trends. Utilização de algoritmos de machine learning para o aprendizado supervisionado e predição de doenças, logo, prevendo a demanda de medicamentos por região para um período de tempo.

- Dupla 2: Tamara Martinelli de Campos RA: 157324
  José Henrique Ferreira Pinto RA: 155976
  Desenvolvimento da interface gráfica e front-end e extração de dados referentes a medicamentos e sintomas.

- Dupla 3: Flavio Matheus Muniz Ribeiro da Silva RA:146098
  Thales Mateus Rodrigues Oliveira RA:148051
  Desenvolvimento da integração entre o banco de dados, interface gráfica e criação do banco de dados.

- Dupla 4: Humberto Politi de Oliveira RA: 119555
  Victor Cerqueira Leal RA: 046873
  Desenvolvimento de preditores neurais para as séries temporais de consumo de medicamentos e sintomas.

O Problema:

Nos últimos anos, mais e mais pessoas têm acesso à internet e a utilizam para procurar possíveis doenças a partir de seus sintomas. Isso gera uma quantidade enorme de dados que, quando comparada com ocorrências provenientes de dados oficiais de uma certa doença em uma região específica, pode nos dar um modelo de como buscas por sintomas na internet (Google) se relacionam com casos concretos de doenças, possibilitando a resolução de um dos grandes problemas da administração da saúde pública, que é a predição da demanda de medicamentos futura por estado do país.

Lista de Requisitos:

Deve-se poder responder:

- Quais sintomas uma doença apresenta;
- Quais medicamentos tratam um sintoma;
- Quais sintomas são buscados na internet (Google Trends);
- Quais estados do Brasil apresentam essas buscas;
- Quantas vezes um sintoma foi buscado;
- O período de tempo em que os dados da busca foram analisados.

Sites de extração de dados:

Considerando todas as duplas do cluster, os sites utilizados para extração de dados foram:

- **ANVISA** - Agência Nacional de Vigilância Sanitária.

Neste site estão as listas de medicamentos registrados no Brasil com os seus preços de venda.Utilizamos a versão XLS para download de dados, que contém mais de 25 mil medicamentos.

- **Site MinhaVida**

Nesse site encontramos os sintomas das doenças selecionadas para a predição de medicamentos. Os dados foram extraídos manualmente através da pesquisa pelo nome da doença e pelo acesso à seção ‘sintomas’.

- **Google Trends**

Reúne os dados de buscas realizadas no Google ao longo dos anos. É possível fazer consultas por termos específicos e a localização das buscas.

- **DataSus**

Contém as doenças que possuem dados oficiais divididas por estados do Brasil. Desse site foram retiradas quais doenças seriam usadas para na predição.

Modelos:

Modelo Relacional:

Doença(NomePopular)

Causa(doença, sintoma)
- ➔ doença: chave estrangeira para Doença
- ➔ sintoma: chave estrangeira para Sintoma

Sintoma(Nome)

Trata(medicamento, sintoma)
- ➔ medicamento: chave estrangeira para Medicamento
- ➔ sintoma: chave estrangeira para Sintoma

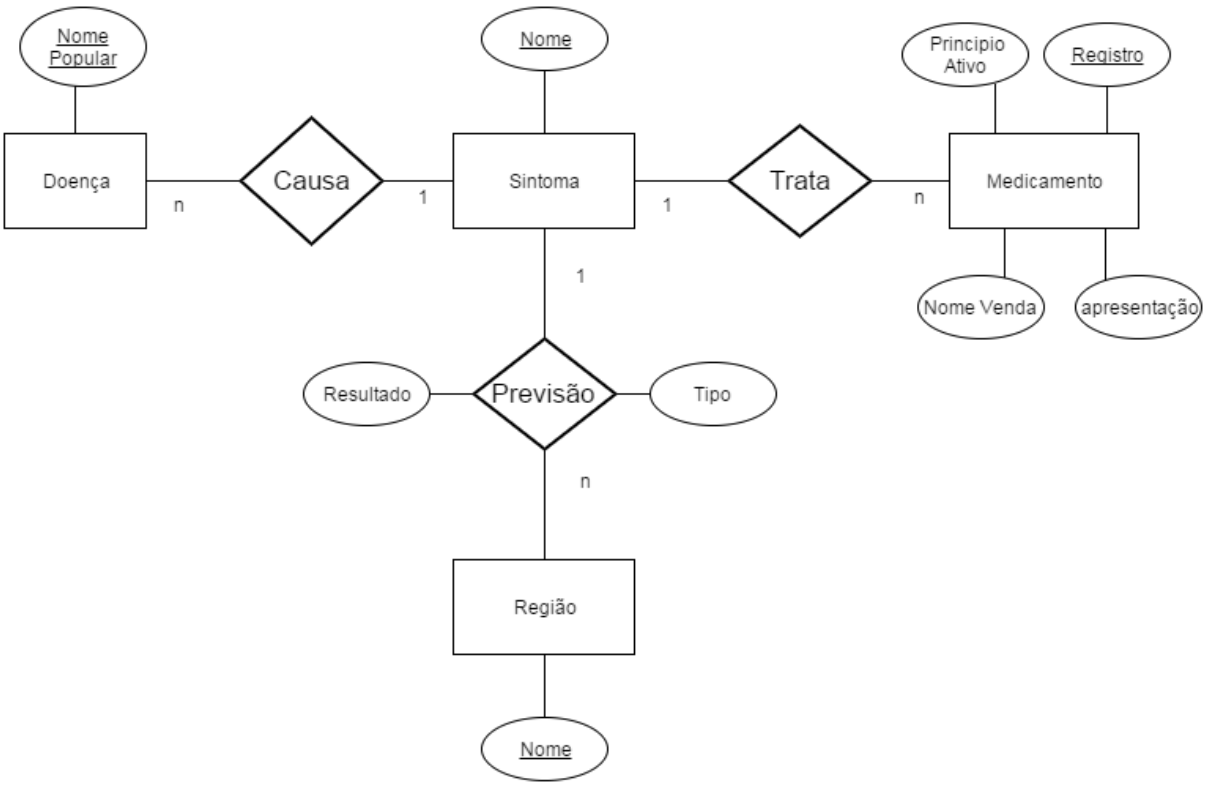
Medicamento(Registro, PrincipioAtivo, NomeVenda, Apresentação)

Previsão(sintoma, região, Resultado, Tipo)
- ➔ sintoma: chave estrangeira para Sintoma
- ➔ região: chave estrangeira para Região

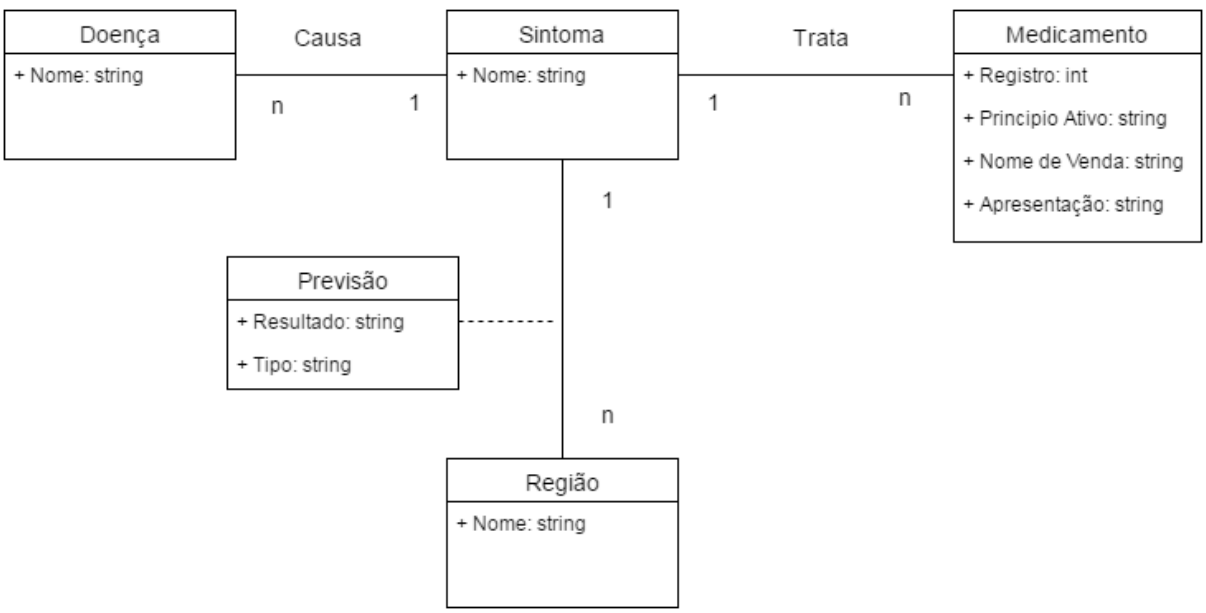
Região(Nome)

Modelo Conceitual:

Entidade Relacionamento:



UML:



Descrição da Proposta e Como Foi Realizada a Predição:

A proposta do projeto é realizar uma predição de estoque de medicamentos por estados do Brasil de acordo com os sintomas das doenças cadastradas no banco de dados.

Para isso cadastramos no nosso banco uma série de doenças e seus respectivos sintomas. A partir desses sintomas, podemos determinar quais medicamentos podem ser utilizados para o tratamento daquela doença. Para a predição do estoque utilizamos os dados do Google Trends, que indicam as quantidades de busca por um sintoma na internet.

Para a realização da predição começamos buscando os dados que seriam necessários para a execução de cada parte do projeto. De acordo com a seção de sites, obtemos do site da ANVISA a lista de medicamentos registrados no Brasil, do site do DataSus a lista de doenças que disponibilizaríamos na pesquisa, do site MinhaVida a lista de sintomas correspondente a cada doença e do Google Trends os dados de buscas por sintomas ao longo dos anos.

Com os dados em mãos, começamos refinando e limpando as tabelas de sintomas e medicamentos para que elas se ajustassem ao nosso problema.

Após o tratamento inicial dos dados, foi necessário fazer as predições. Para poder prever a frequência de demanda dos medicamentos é necessário prever o sinal para cada estado brasileiro.

As previsões foram feitas utilizando duas classes de algoritmos diferentes, a primeira, uma rede neural perceptron de múltiplas camadas, e a segunda, uma regressão não paramétrica baseada em kernel. Ambos algoritmos são não lineares, mas lidam com a tarefa de predição de forma bem diferente. A rede neural deveria ter predições melhores à curto prazo enquanto que a regressão deveria dar uma tendência e visão global da função de demanda.

Os dados das predição são exibidos em páginas html que foram montadas utilizando o framework Bootstrap.

Após receber os dados tratados e as previsões feitas o banco foi montado e levantado conforme o modelo previsto.

Após ter o banco pronto era necessário prover ao front end meios de acessá-lo. Através da utilização do framework django essa comunicação entre front end e dados é feita a partir de requests vindas de páginas web.

Com os dados integrados, montamos a pesquisa que gera uma lista de medicamentos que tratam os sintomas da doença pesquisada. Escolhendo um medicamento, exibimos os mapas para mostrar as predições realizadas. Cada mapa do Brasil corresponde à um tipo de previsão e as tonalidades das cores indicam os estoques previstos para cada estado. Há ainda um slider para escolher a semana de previsão desejada. Para essa exibição, utilizamos a biblioteca d3.js.

Descrição dos papéis:

❖ Dupla 1

A dupla é responsável por dois módulos essenciais do projeto. O primeiro é a criação de uma rotina de extração de dados do *Google Trends.* A segunda é a implementação e teste de algoritmos de machine learning supervisionados. Cada tarefa é especificada a seguir:

- **Extração de dados do Google Trends:** A rotina de extração de dados tem que ser capaz de retirar o maior número de dados possível do *Google Trends* dado um período específico, e, salvar na base de dados da aplicação Django.
  - Parâmetros: Período, termo de busca, país/região
  - Regras:
    - Caso haja uma entrada na base de dados da aplicação Django para o termo para aquele período, não atualizar dados
    - Caso não haja nenhum dado no banco de dados da aplicação Django, utilizar a API do *Google Trends* para sincronizar os dados e salvá-los no banco de dados local da aplicação.
  - Dificuldades e desafios:
    - Cada requisição somente comporta um período, um estado brasileiro e até cinco termos.
    - Para obter uma quantidade significativa de valores para cada termo de busca, é necessário realizar buscas com períodos de no máximo 36 meses. Logo, para obtermos os dados desde 2004, se fizeram necessárias 5 requisições por conjunto de termos por região.
    - No total, se faziam necessárias 55 requisições por estado. O que resultou em um total de 1485 requisições.
    - Há um limite não especificado de requisições por um determinado período de tempo. Por esse motivo, as requisições tiveram que ser espaçadas no tempo e, quando o limite era ultrapassado, novas requisições eram bloqueadas por tempo também indeterminado, variando de 15 minutos a 4 horas. Essa limitação foi um grande limitante deste trabalho, cortando em mais de 50% as requisições planejadas.

- **Algoritmo de machine learning não paramétrico:** Com o intuito de prever a demanda de medicamentos futura, é necessário prever o número de casos futuros. Assumimos que o número de casos tem uma forte correlação com buscas de sintomas de uma determinada doença no Google. Utilizando-se o algoritmo de regressão não paramétrico baseado em kernel para prever as tendências de busca de cada sintoma, será

possível prever o número de casos de cada sintoma, prevendo assim a frequência de demanda por região do país.

O algoritmo de regressão não paramétrico escolhido utiliza a seguinte função como função de fit.

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right),$$

onde K é a função denominada de kernel e joga um papel fundamental na qualidade da predição. O kernel escolhido é a soma de quatro funções, uma *Radial-basis function*, que tem o intuito de prever uma tendência a longo prazo, uma *Exp-sine-squared*, que explica a sazonalidade dos sintomas, uma *Rational-quadratic* que tem o intuito de gerar oscilações menores e de misturar as escalas, e por fim, um ruído branco.

Aplicando o método com essas funções, chegou-se à figura abaixo, que mostra a frequência de pesquisas por “febre” para o estado de São Paulo por tempo. A linha em azul que termina por volta do mês 150 são os dados para SP diretamente do Google Trends. A linha verde é a função estimadora aprendida.

Pode se verificar que a função exibe as características dos kernels mostradas anteriormente, e que ela explica parcialmente o sinal. Isso se deve ao fato das funções utilizadas no kernel serem geralmente suaves, o que não é observado nos dados reais do Trends. Para explicar a não suavidade do sinal, novas funções não comportadas deveriam ser adicionadas ao kernel, o que geraria instabilidade numérica, afetando até mesmo a factibilidade da solução

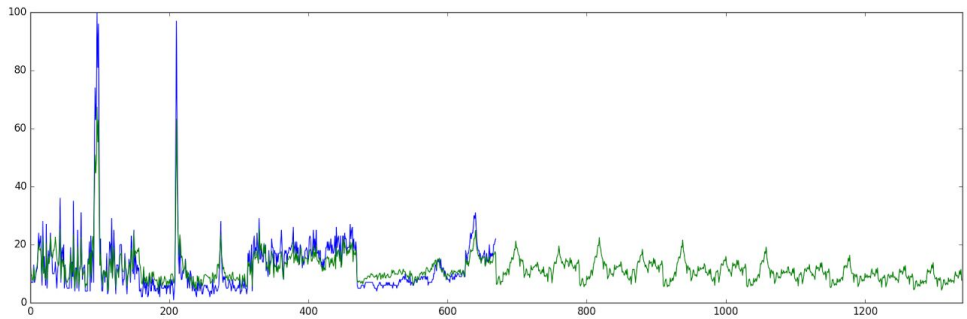


Figura: Frequência de pesquisas por “febre” por tempo para SP

❖ Dupla 2: Desenvolvimento da Interface Gráfica e Front-end e Extração de Dados Referentes a Medicamentos e Sintomas

O Trabalho realizado:

A extração de dados referentes a medicamentos foi realizada através do site da ANVISA - Agência Nacional de Vigilância Sanitária, que disponibiliza a lista de medicamentos registrados no Brasil para comercialização. Esses dados estão em formato CSV e contém mais de 25 mil registros.

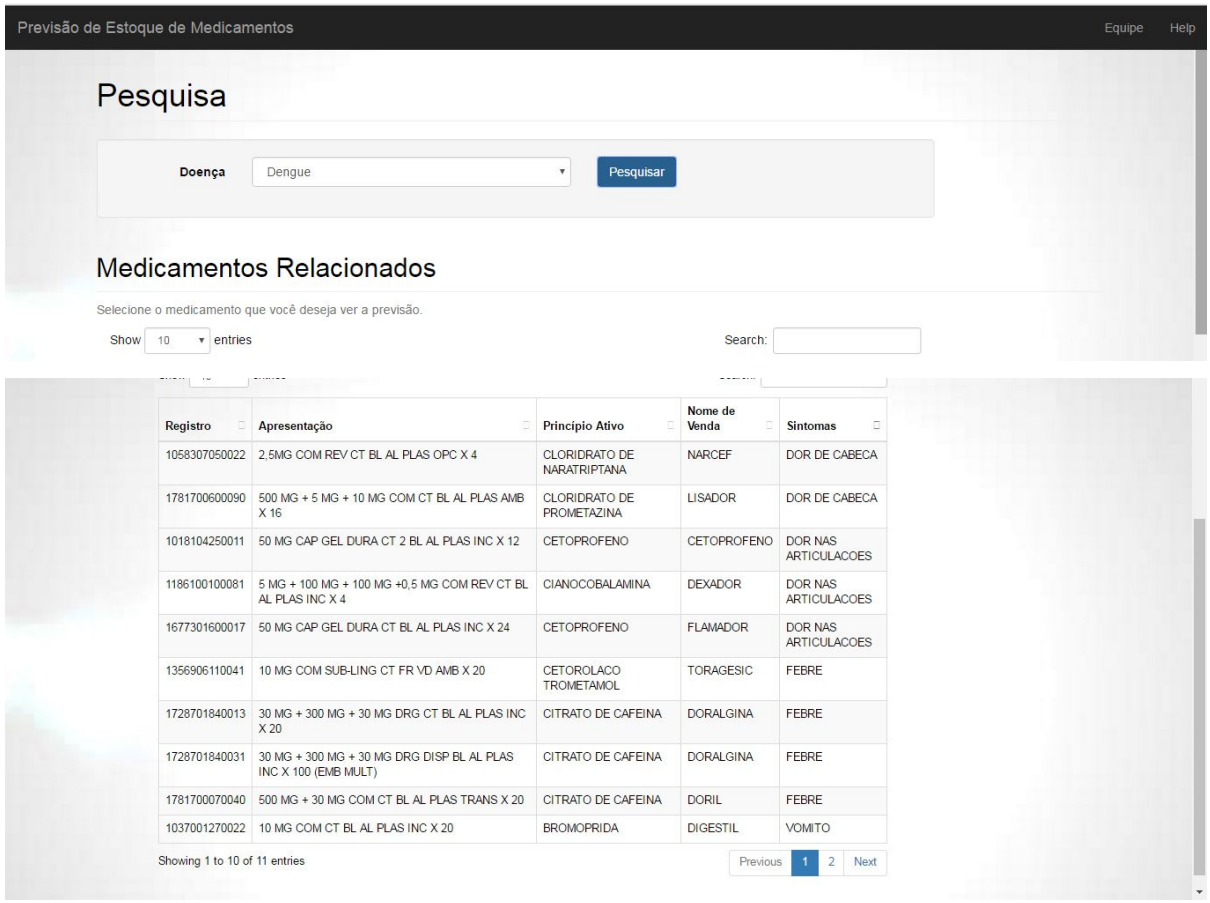
Para melhor manipulação dos dados, foi realizado um refinamento por 'classe terapêutica', que indica a área de atuação dos medicamentos, e realizada a substituição dessa classe pela classe 'sintomas', que contém o principal sintoma combatido pelo medicamento. Com esse refinamento, obtemos um registro CSV com mais de 5 mil registros.

Para a extração dos dados referentes aos sintomas foi realizada a extração manual de dados através do site 'Minha Vida', indicado na seção de sites. As doenças foram escolhidas de acordo com o site do datasus, também indicado na seção de sites. Uma vez que as doenças foram selecionadas, bastou realizar uma busca pelo nome da doença e acessar a parte de 'sintomas'. Com esses dados montamos um CSV que foi cadastrado no banco de dados para a predição de medicamentos.

Para o desenvolvimento da interface gráfica e do front-end, utilizamos um framework MVC e desenvolvemos a parte de View, que se liga ao resto do projeto pelo framework.

Utilizamos para isso html, JavaScript e a biblioteca **d3.js** para exibição dos dados usando um mapa do Brasil. Utilizamos ainda o framework Bootstrap para a criação das páginas html.

Na página inicial da interface gráfica temos um campo com o nome das doenças disponíveis no nosso banco de dados. O usuário escolhe uma dessas doenças e realiza uma pesquisa pelos medicamentos que tratam os sintomas dessa doença. Será exibida uma tabela com tais medicamentos e a partir dela é possível escolher um deles para conferir a previsão do estoque em todo o país.



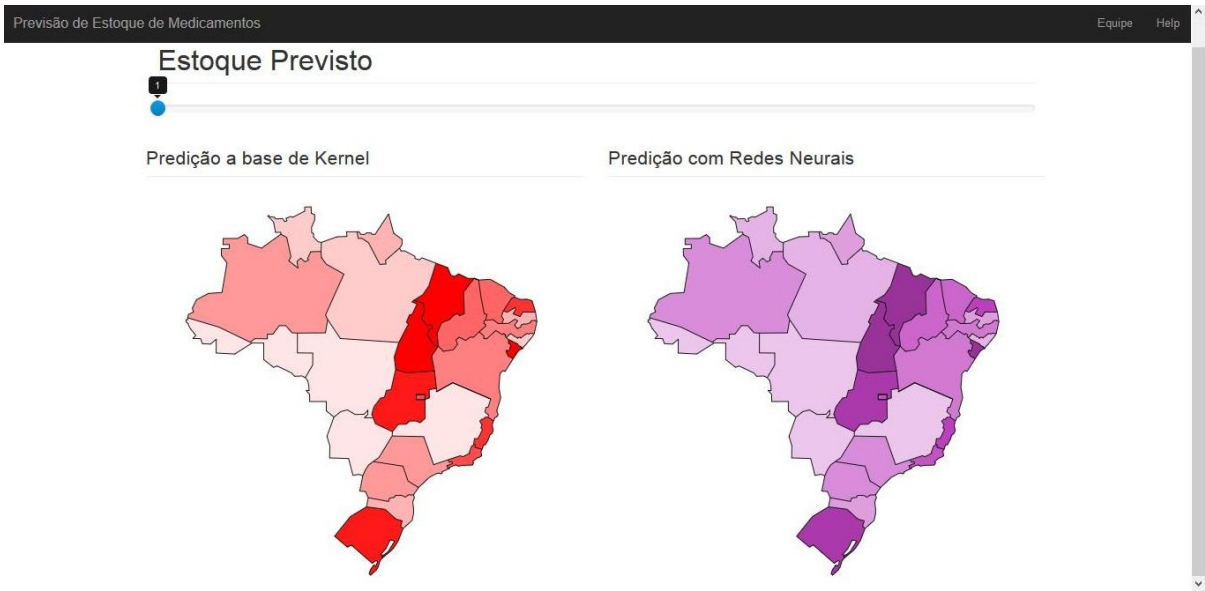
A tabela de medicamentos foi implementada utilizando o componente DataTables. Da dupla responsável pela integração, recebemos um vetor com os medicamentos que tratam os sintomas da doença escolhida e jogamos esses dados na tabela com a ajuda do JavaScript.

```
var dataSet = {{medicamentos|safe}};
    var table = $('#meds').DataTable( {

        data: dataSet,
        columns: [
            { title: "Registro" },
            { title: "Apresentação" },
            { title: "Princípio Ativo" },
            { title: "Nome de Venda" },
            { title: "Sintomaa" }
        ]
    } );


    $('#meds tbody').on('click', 'tr', function () {
        var data = table.row( this ).data();
        location.href = "{% url 'estoque:previsao' data[0] %}";
    } );

});
```

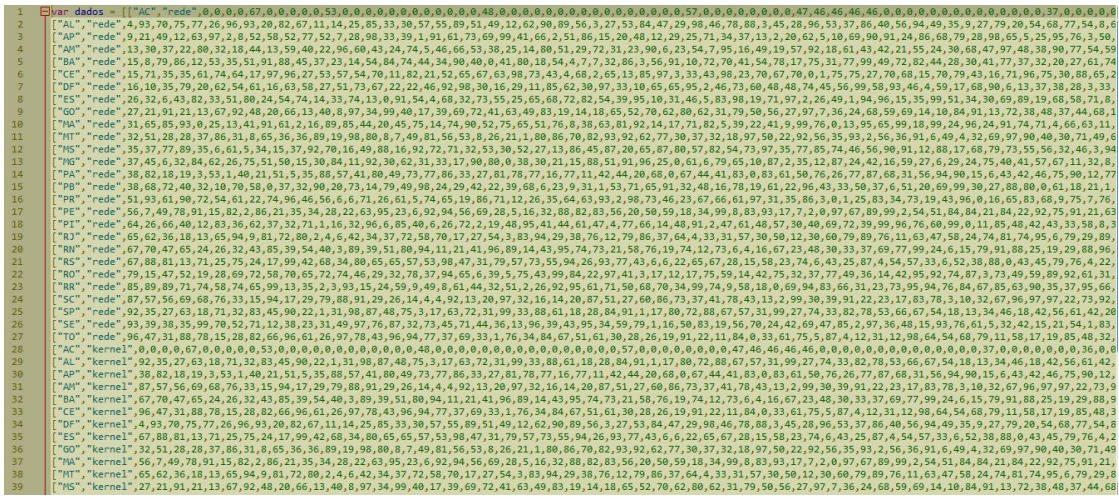
A partir dos campos da tabela, podemos escolher um medicamento para visualizar a sua predição por estados do Brasil. O clique em um dos campos redireciona para a página do mapa passando por parâmetro o id do medicamento que se deseja prever o estoque.

Para exibir o resultado da previsão, montamos um mapa do Brasil utilizando a biblioteca **d3.js** que chama o **topojson** no JavaScript utilizando um arquivo Json com as coordenadas dos estados dos país.

Com o mapa desenhado, separamos os estados pelo seu id no arquivo Json e fomos capazes de utilizar os valores provenientes das predições para colorir o mapa de acordo com a quantidade do medicamento selecionado por estado. A escolha do ano se dá em uma escala que vai até as últimas semanas previstas pelos algoritmos utilizados e a tonalidade dos estados muda de acordo com a semana selecionada. A página contém 2 mapas. O primeiro reflete os dados obtidos através da predição a base de Kernel e o segundo reflete os dados obtidos através da predição com Redes Neurais. Cada um possui uma cor com as suas tonalidades, permitindo com que a comparação entre as análises seja mais fácil.



Para alterar as cores do mapa, utilizamos os dados vindos do servidor. Os dados são compostos de um ID(sigla do estado), tipo de previsão e um vetor com a série de valores por semana a partir do dia **4/1/2004**. No total são 770 semanas.



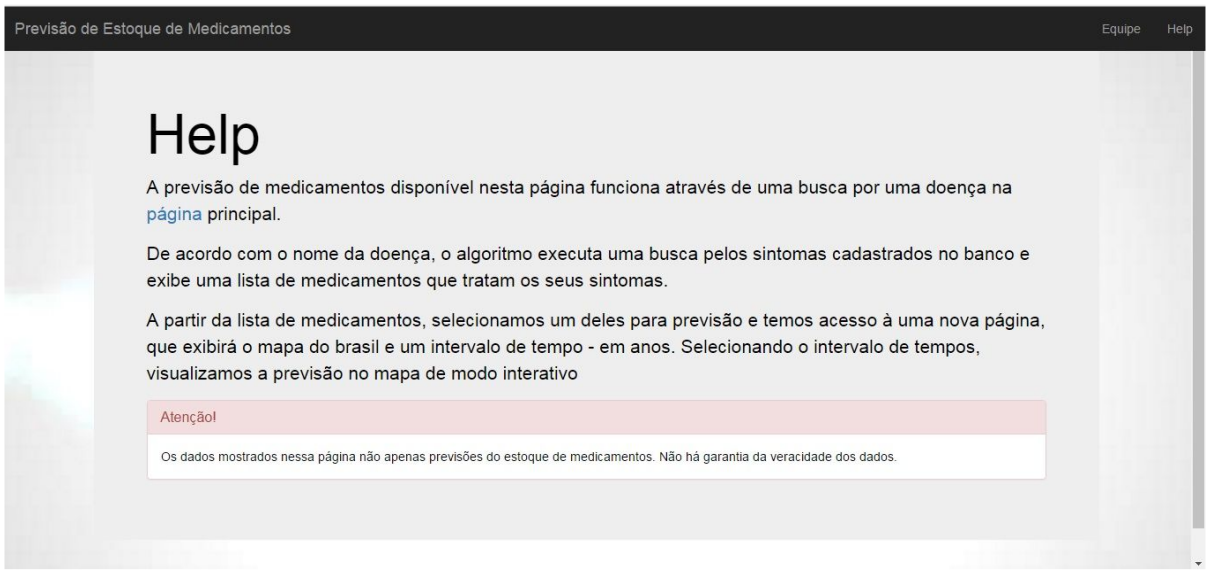
Para poder alterar o mapa pelo tempo, utilizamos um slider. Nele setamos a semana desejada, o que redesenha o mapa utilizando os valores já carregados

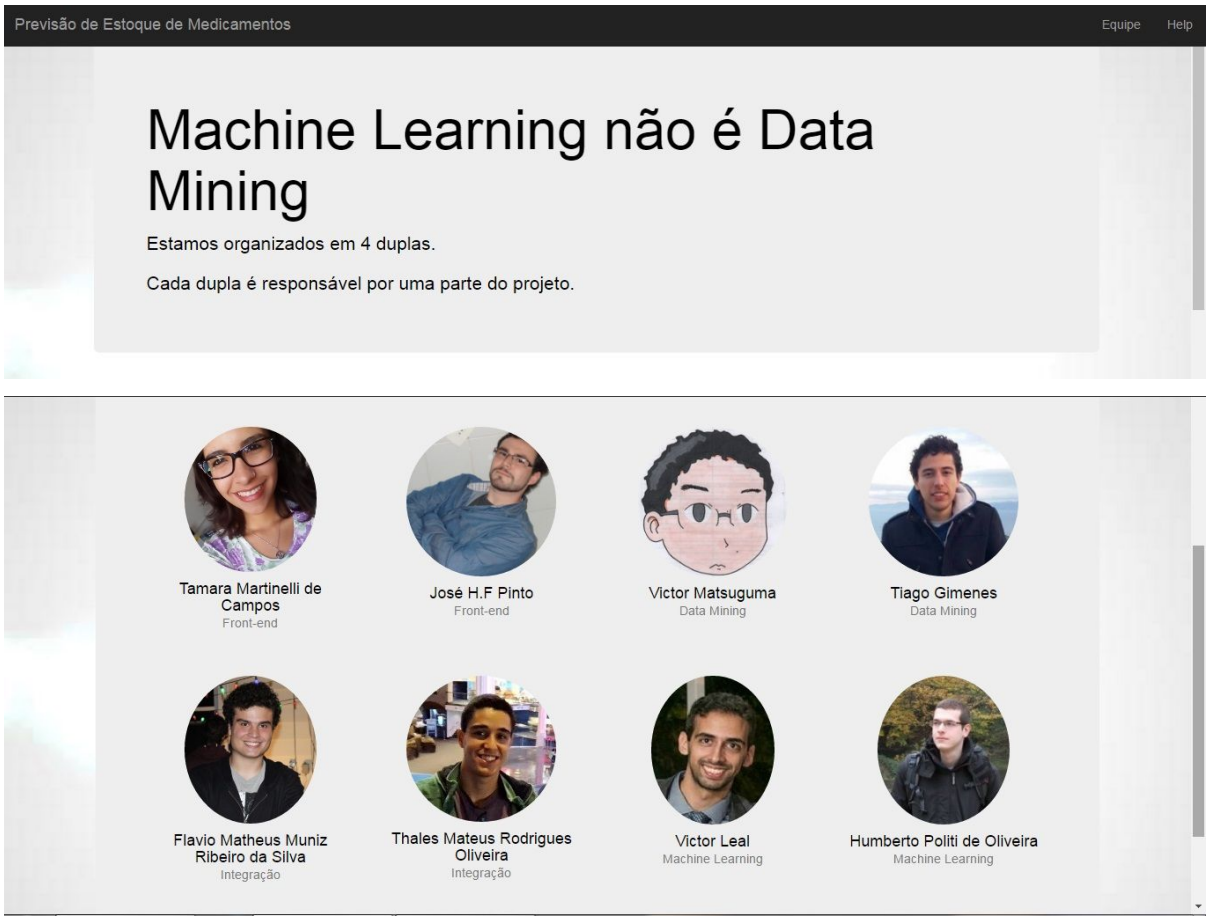
```
<script src="dados.js"></script>


<script>
var time = 1;

$("#slider").slider({
        min: 1,
        max: 770,
        value: 1,
        tooltip: 'always'});

$("#slider").on("slide", function(slideEvt) {
    $("#time").text(slideEvt.value);
    time = slideEvt.value;
    d3_queue.queue()
        .defer(d3.json, "files/br.json")
        //.defer(d3.tsv, "files/qtd.tsv")
        .await(ready);
});
```

Com os dados carregados, podemos comparar os valores apresentados com os limiares utilizados para colorir os estados. A função utilizada **d3.select.threshold()** cria um mapeamento entre um intervalo e uma cor dada utilizando dois vetores, range e domain. Por exemplo, se um determinado valor é 85, ele está entre 80 e 90 e será pintado com a oitava cor.

```
var width = 600;
var height = 500;

var svg1 = d3.select('div[id="svg1"]').append("svg")
        .attr("id", "svg1")
        .attr("width", "100%")
        .attr("height", 500)
        .style("position", "relative");

var svg2 = d3.select('div[id="svg2"]').append("svg")
        .attr("id", "svg2")
        .attr("width", "100%")
        .attr("height", 500)
        .style("position", "relative");


var g1 = svg1.append("g");
var g2 = svg2.append("g");

var color1 = d3.scale.threshold()
    .domain([20, 30, 40, 50, 60, 70, 80, 90, 100])
    .range(["#ffe6e6","#ffcccc","#ffb3b3","#ff9999","#ff8080","#ff6666","#ff4d4d","#ff3333","#ff1a1a","#ff0000"]);

var color2 = d3.scale.threshold()
    .domain([20, 30, 40, 50, 60, 70, 80, 90, 100])
    .range(["#ecc6ec","#e6b3e6","#df9fdf","#d98cd9","#d279d2","#cc66cc","#c653c6","#bf40bf","#ac39ac","#993399"]);
```

Como complemento e para criar uma página mais real que pudesse ser utilizadas por profissionais da saúde, criamos uma aba de *help* que explica como utilizar a ferramenta disponível na página e uma aba com o membros da *equipe* que desenvolveram a aplicação.





Dificuldades e Desafios:

- ➢ Tratar e realizar data cleaning dos dados referentes aos medicamentos. Devido à quantidade de dados que tínhamos disponíveis no csv obtido do site da ANVISA, o refinamento e data cleaning foi mais longo e trabalhoso do que imaginávamos. Inicialmente acreditamos que os dados estavam praticamente prontos para o nosso uso, mas no decorrer do projeto o cluster optou por mudar um pouco o foco do trabalho e isso acabou levando à necessidade de refinar os dados a fim de retirar tuplas que não seriam necessárias e mudar algumas colunas para atender aos nossos objetivos. Nesse processo, diminuímos a base de dados de 25 mil medicamentos para pouco mais de 5 mil.
- ➢ Impossibilidade de baixar dados de bulas. Apesar de todas as bulas de remédios cadastrados na ANVISA estarem disponíveis online, esses dados são fornecidos em formato PDF, e cada fabricante tem seu padrão de arquivo, impossibilitando a busca de dados automatizada. Por esse motivo, a ideia inicial de exibir as dosagens dos medicamentos teve que ser deixada de lado.

- ➢ Demora nos dados das predições: Os dados provenientes das predições demoraram um pouco do mais do que o previsto para ficarem prontos e com isso a exibição dos dados ficou comprometida quanto à qualidade do que havíamos imaginado. Ao final ficamos satisfeitos com o resultado obtido.
- ➢ Dificuldade em utilizar corretamente o framework de exibição de mapas **d3**. Apesar de ser muito utilizado atualmente, foi difícil encontrar tutoriais para leigos em javascript e no próprio framework. A montagem de ambiente foi muito dificultada por passos redundantes nos tutoriais, que seriam facilmente contornados por alguém mais experiente.
- ➢ Aprender a utilizar as classes do framework de estilos **bootstrap** foi um outro desafio, pois diferente das abordagens mais antigas, o framework não permite muita alteração de estilo hardcoded. ele possui suas próprias tags que geram na página todo o javascript e estilo necessário.
- ➢ Integrar o modelo de página ao framework MVC **Django**. Diferentemente de modelos mais naturais como o rest, a utilização do Django dificultou bastante a integração entre front e back end, apesar de facilitar o acesso ao banco de dados.
- ➢ Rodar o server localmente na configuração local da máquina. Como o Django deve ser instalado, e as máquinas do Instituto não permitem isso, foi necessário a configuração do servidor nas máquinas locais. O que nem sempre resultou em sucesso.

❖ Dupla 3: Desenvolvimento da integração do front end com o banco de dados:

Como o estamos utilizando o framework de MVC, ficamos responsáveis pelo desenvolvimento da parte de controle. Sendo assim utilizaremos a framework Django para facilitar a implementação da comunicação entre o modelo, que no caso é o banco de dados, e o front-end.

Esse controle será responsável por conferir se há mudanças no banco de dados e representar essas mudanças no front end de forma responsiva. Também será responsável por receber as requisições vindas dos usuários.

A dupla também ficou responsável pelo desenvolvimento da parte de modelo. A implementação dos modelos para o banco de dados foi realizada, assim como os scripts para a população do banco.

Para a implementação do modelo de banco de dados o framework Django foi utilizado. Django utiliza como gerenciador de banco de dados padrão o SQLite3, e transforma as entidades e relacionamentos em classes, facilitando a manipulação

dos dados. Segue a seguir a implementação dos modelos utilizando django:

```python
from __future__ import unicode_literals

from django.db import models

# Create your models here.

class Medicamento(models.Model):
    registro = models.CharField(max_length = 30,primary_key=True)
    apresentacao = models.CharField(max_length = 300)
    nome_venda = models.CharField(max_length = 20)
    classe_terapeutica = models.CharField(max_length = 20)
    principio_ativo = models.CharField(max_length = 20)
    #laboratorio = models.CharField(max_length = 20)
    #hospitalar = models.CharField(max_length = 20)

    def __str__(self):
        return self.nome_venda

class Doenca(models.Model):
    #nome_cientifico = models.CharField(max_length = 20, primary_key=True)
    nome_popular = models.CharField(max_length = 20, primary_key=True)

    def __str__(self):
        return self.nome_popular

class Sintoma(models.Model):
    nome = models.CharField(max_length= 50, primary_key=True)

    def __str__(self):
        return self.nome

class Regiao(models.Model):
    nome = models.CharField(max_length = 20, primary_key=True)

    def __str__(self):
        return self.nome
```

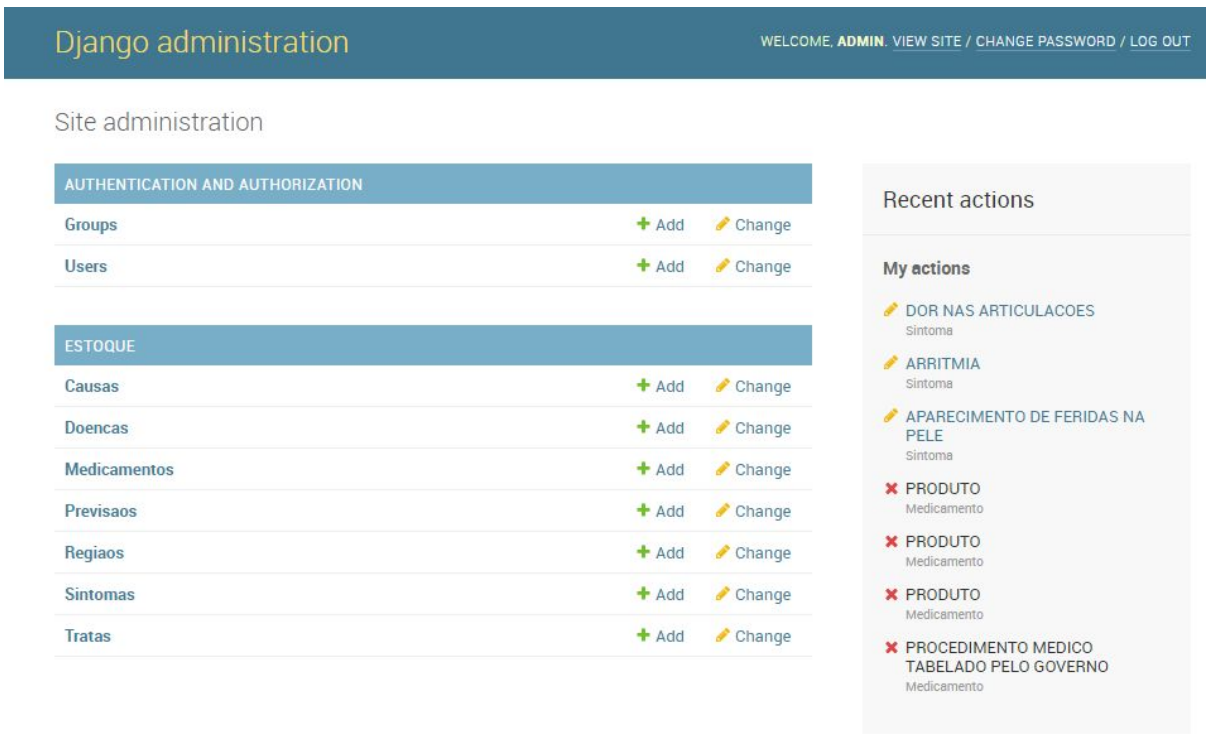
```python

class Previsao(models.Model):
    tipo = models.CharField(max_length=20)
    resultado = models.TextField()
    regiao = models.ForeignKey(Regiao, on_delete=models.CASCADE, default=None)
    sintoma = models.ForeignKey(Sintoma, on_delete=models.CASCADE, default=None)

    def __str__(self):
        return "resultado do tipo: " + self.tipo + " para o sintoma: " + self.sintoma.nome + " na regiao: " + self.regiao.nome

class Trata(models.Model):
    medicamento  = models.ForeignKey(Medicamento, on_delete=models.CASCADE)
    sintoma = models.ForeignKey(Sintoma, on_delete=models.CASCADE, default=None)

    def __str__(self):
        return self.medicamento.nome_venda + " trata " + self.sintoma.nome

class Causa(models.Model):
    doenca = models.ForeignKey(Doenca, on_delete=models.CASCADE)
    sintoma = models.ForeignKey(Sintoma, on_delete=models.CASCADE)

    def __str__(self):
        return self.doenca.nome_popular + " causa " + self.sintoma.nome

class Incide(models.Model):
    incidencia = models.IntegerField(default=0)
    periodo = models.DurationField()
    doenca = models.ForeignKey(Doenca, on_delete=models.CASCADE)
    regiao = models.ForeignKey(Regiao, on_delete=models.CASCADE)

```

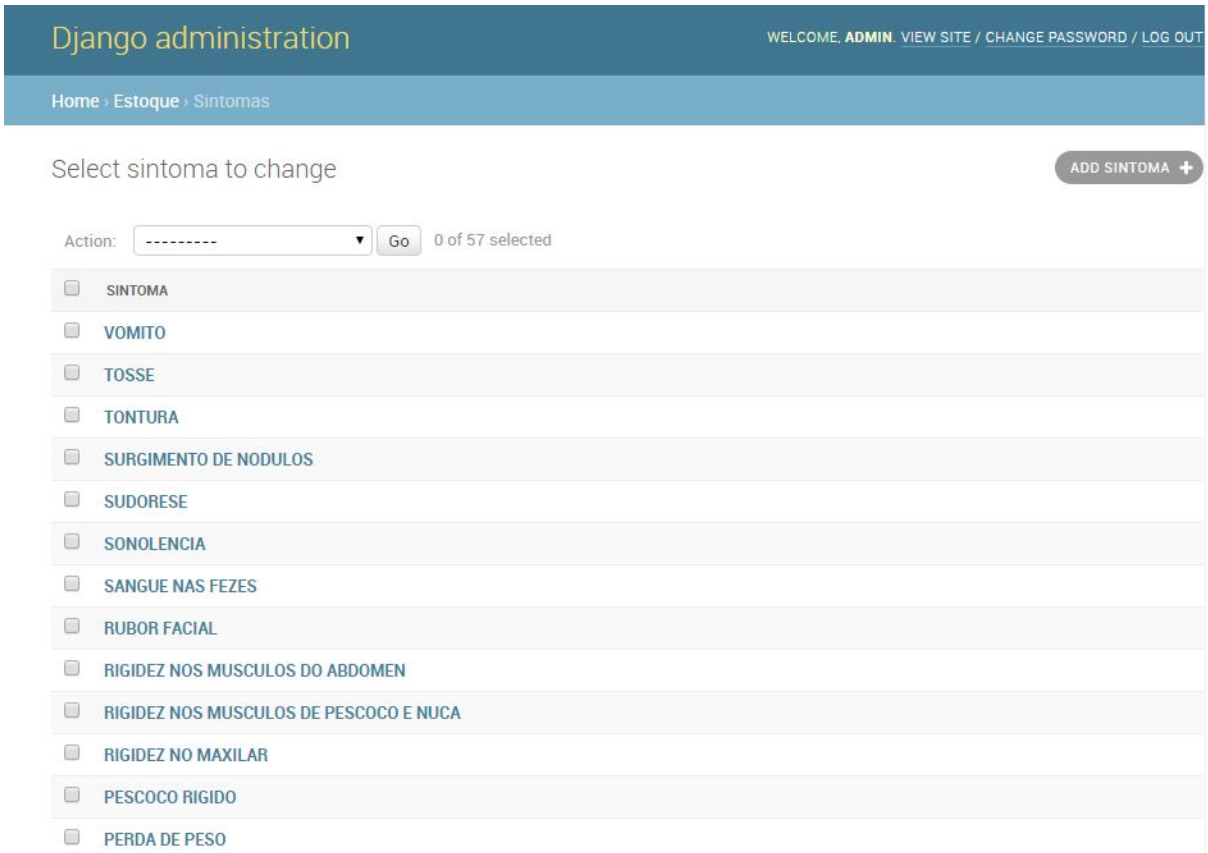
O framework também disponibiliza para uso um servidor simples que auxilia no desenvolvimento, pois podemos checar através de views criadas os dados, direto do navegador.

Servidor executando:

```
[28/Nov/2016 13:31:58] "GET /admin/ HTTP/1.1" 200 6927
[28/Nov/2016 13:32:03] "GET /admin/estoque/sintoma/ HTTP/1.1" 200 18183
[28/Nov/2016 13:32:03] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 13:32:15] "GET /admin/estoque/sintoma/APARECIMENTO%20DE%20%20FERIDAS%20NA%20PELE/change/ HTTP/1.1" 200 4196
[28/Nov/2016 13:32:15] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 13:32:38] "POST /admin/estoque/sintoma/APARECIMENTO%20DE%20%20FERIDAS%20NA%20PELE/change/ HTTP/1.1" 302 0
[28/Nov/2016 13:32:38] "GET /admin/estoque/sintoma/ HTTP/1.1" 200 18750
[28/Nov/2016 13:32:38] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 13:32:39] "GET /static/admin/img/icon-yes.svg HTTP/1.1" 200 436
^[[A[28/Nov/2016 13:32:57] "GET /admin/estoque/sintoma/APARECIMENTO%20DE%20%20FERIDAS%20NA%20PELE/change/ HTTP/1.1" 200 4196
[28/Nov/2016 13:33:01] "GET /admin/estoque/ HTTP/1.1" 200 4657
[28/Nov/2016 13:33:05] "GET /admin/estoque/previsao/ HTTP/1.1" 200 4581
[28/Nov/2016 13:33:05] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 13:33:07] "GET /admin/estoque/ HTTP/1.1" 200 4657
[28/Nov/2016 13:35:55] "GET /admin/estoque/sintoma/ HTTP/1.1" 200 18499
[28/Nov/2016 13:35:56] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 13:36:19] "GET /admin/estoque/sintoma/ARRITIMIA/change/ HTTP/1.1" 200 4085
[28/Nov/2016 13:36:19] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 13:36:34] "POST /admin/estoque/sintoma/ARRITIMIA/change/ HTTP/1.1" 302 0
[28/Nov/2016 13:36:34] "GET /admin/estoque/sintoma/ HTTP/1.1" 200 18933
[28/Nov/2016 13:36:34] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 14:14:39] "GET /admin/ HTTP/1.1" 200 7581
[28/Nov/2016 14:14:39] "GET /admin/ HTTP/1.1" 200 7581
[28/Nov/2016 14:14:43] "GET /admin/estoque/sintoma/ HTTP/1.1" 200 18738
[28/Nov/2016 14:14:43] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 14:18:07] "GET /admin/estoque/sintoma/DOR%20NAS%20ARTICUACOES/change/ HTTP/1.1" 200 4133
[28/Nov/2016 14:18:08] "GET /admin/jsi18n/ HTTP/1.1" 200 3217
[28/Nov/2016 14:18:14] "POST /admin/estoque/sintoma/DOR%20NAS%20ARTICUACOES/change/ HTTP/1.1" 302 0
[28/Nov/2016 14:18:15] "GET /admin/estoque/sintoma/ HTTP/1.1" 200 19238
```

Navegador:





O framework também permite com que dados sejam adicionados/modificados/excluídos direto no navegador, sem a necessidade de se recorrer a um script específico.

Para rodar o servidor, o comando “python manage.py runserver” é executado, dentro da pasta principal do aplicativo django implementado:

```
^Cthales@THALES:/mnt/c/Users/Thales/Documents/GitHub/ProjetoBD/MC536---Data-Mining-Nao-Machine-Learning/
PrevisaodeEstoque$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
November 28, 2016 - 16:41:14
Django version 1.10.3, using settings 'PrevisaodeEstoque.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Para popular o banco, os scripts são rodados a partir do shell python, também dentro do aplicativo django implementado:

```
thales@THALES:/mnt/c/Users/Thales/Documents/GitHub/ProjetoBD/MC536---Data-Mining-Nao-Machine-Learning/Pr
evisaodeEstoque$ python manage.py shell
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Os scripts para popular o banco, para as classes Medicamento, Sintoma, Doenca, Regiao, Causa, Trata e Previsao foram implementados e executados no shell.

A seguir um exemplo de implementação e execução de um script de população(Sintoma):

```
In [5]: import csv
   ...: from estoque.models import Sintoma
   ...: with open('sintomasSemAcentoInc.csv', 'rb') as f:
   ...:     reader = csv.reader(f)
   ...:     for row in reader:
   ...:         s = Sintoma(row[1])
   ...:         print s
   ...:         s.save()
   ...:
CANSACO
DOR ATRAS DOS OLHOS
DOR DE CABECA
DOR NAS ARTICUACOES
DOR MUSCULAR
DOR NOS OSSOS
FEBRE
MANCHAS VERMELHAS PELO CORPO
MOLEZA
NAUSEA
FALTA DE APETITE
TONTURA
VOMITO
AUMENTO DO TAMANHO DO BACO
AUMENTO DO TAMANHO DO FIGADO
DIARREIA
```

Os outros scripts de população se encontram na pasta dump.

Para a comunicação com o front end apenas o django foi necessário. Ele trabalha com um sistema de templates em html com uma linguagem própria de tags em que o próprio servidor consegue, a partir dos contextos enviados junto com a request substituir essas tags por qualquer elemento que um html poderia ter, sendo até possível com as tags usar diferentes instruções de forma que loops e condicionais são possíveis.

```
<!-- Bootstrap core CSS -->
<link href="{% static "estoque/bootstrap.min.css" %}"rel="stylesheet">

<!-- IE10 viewport hack for Surface/desktop Windows 8 bug -->
<link href="./files/ie10-viewport-bug-workaround.css" rel="stylesheet">

<!-- Custom styles for this template -->
<link href="{% static "estoque/dashboard.css" %}" rel="stylesheet">

<!-- Just for debugging purposes. Don't actually copy these 2 lines! -->
<!--[if lt IE 9]><script src="../../assets/js/ie8-responsive-file-warning.js"></script><![endif]-->
<script src="./files/ie-emulation-modes-warning.js.download"></script>
```

Como se pode ver nessa imagem a linguagem de tags consegue ser utilizada em conjunto com o html. Como se pode ver há a necessidade de substituir todas as referências a links do html por uma formatação própria do Django em ordem que ele possa redirecionar o fluxo das páginas de forma correta. Arquivos estáticos também devem ser posicionados

Outra parte que ficamos responsáveis foi a criação dos endpoints do servidor. Os endpoints serão usados para direcionar o fluxo mas também servirão para passar informações do servidor para o front end.

```
def filtro(request):
    return render(request, 'estoque/filtro.html')

def equipe(request):
    return render(request, 'estoque/equipe.html')

def help(request):
    return render(request, 'estoque/help.html')

def mapa(request):
    return render(request, 'estoque/mapa.html')


def medicamentos(request, nomeDoenca):
    doenca = Doenca.objects.get(nome_popular=nomeDoenca)
    causas = Causa.objects.filter(doenca=doenca)
    medicamentos = []

    for causa in causas:
        med = Trata.objects.filter(sintoma=causa.sintoma)
        for m in med:
            aux = [
                m.medicamento.registro,
                m.medicamento.apresentacao,
                m.medicamento.principio_ativo,
                m.medicamento.nome_venda,
                m.medicamento.classe_terapeutica
                ]
            medicamentos.append(aux)

    json_data = json.dumps(medicamentos)

    context = {'medicamentos': json_data}

    return render(request, 'estoque/medicamentos.html', context)
```

Cada função representa um endpoint que será chamado pelo front end.

Dificuldades e Desafios

- O maior desafio foi aprender a ferramenta como um todo devido ao fato de existir dentro dela duas linguagens diferentes(python e as tags de template).

- ➢ O próprio banco de dados nos deu certos problemas devido a necessidade de os dados entre os csv estarem na mesma formatação na hora de se popular o banco
- ➢ O git apesar de facilitar o compartilhamento dos dados nos deu vários problemas, entre eles a necessidade de re-clonar o repositório pois o banco de dados havia sido deletado entre um commit e outro pelo próprio git
- ➢ Passagem de dados entre front end e servidor se mostrou complicada pela dificuldade de adaptação a linguagem de tags

❖ Dupla 4: Desenvolvimento de preditores neurais para as séries temporais

Em posse dos dados, é necessário separar as séries temporais de cada dado que se deseja fazer as predições. Tendo feito isso, serão calculados os coeficientes de autocorrelação de cada série, para decidir quantos valores serão necessários utilizar para realizar uma predição futura.

Será então treinada uma rede neural artificial feedforward (Multilayer Perceptron, MLP) de três camadas para cada série temporal. A arquitetura de cada rede dependerá do número de entradas (obtido através do coeficiente de autocorrelação de cada série temporal), porém é possível definir à priori o tamanho da camada intermediária como sendo ⅔ do número de neurônios da camada de entrada. Esse número em geral permite uma boa capacidade de generalização para a rede neural, mas caso as predições não sejam boas, o número de neurônios na camada intermediária da rede deverá ser ajustado.

Para realizar predições temporais com maior precisão para diversas amostras futuras, será explorado o uso de redes neurais recorrentes (echo state network). Tais redes são conhecidas por serem fáceis de se treinar, pois sua arquitetura recorrente é baseada em reservoir computing, transformando o problema do treinamento da rede apenas no ajuste dos pesos da camada de saída, que pode inclusive ser composta de funções de ativação lineares.

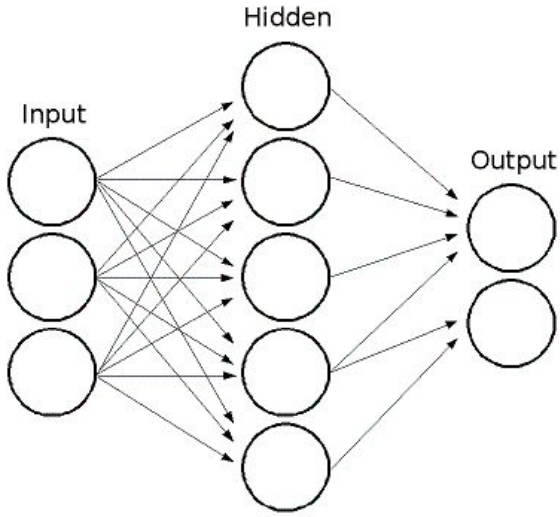
Objetivo

Esperamos obter boas previsões de curto prazo com as redes neurais MLP. O uso das redes recorrentes, entretanto, será o responsável por gerar predições de longo prazo. Com os dados previstos, será possível prever o consumo de medicamentos e incidência de sintomas de doenças nas regiões estudadas.

Resultados

Inicialmente, foi escrito um script em Matlab para treinamento de um ensemble de redes neurais MLP para cada sintoma a ser previsto. O script realiza a decomposição da série temporal de entrada em 10 folds. São então treinadas 10

redes neurais para cada série temporal, cada uma utilizando um dos folds como conjunto de validação e os outros 9 como conjunto de treinamento. Desta forma todos os dados participam tanto do treinamento como da validação de alguma rede neural. Uma porcentagem de 15% dos dados é separada para testes ao final do treinamento.

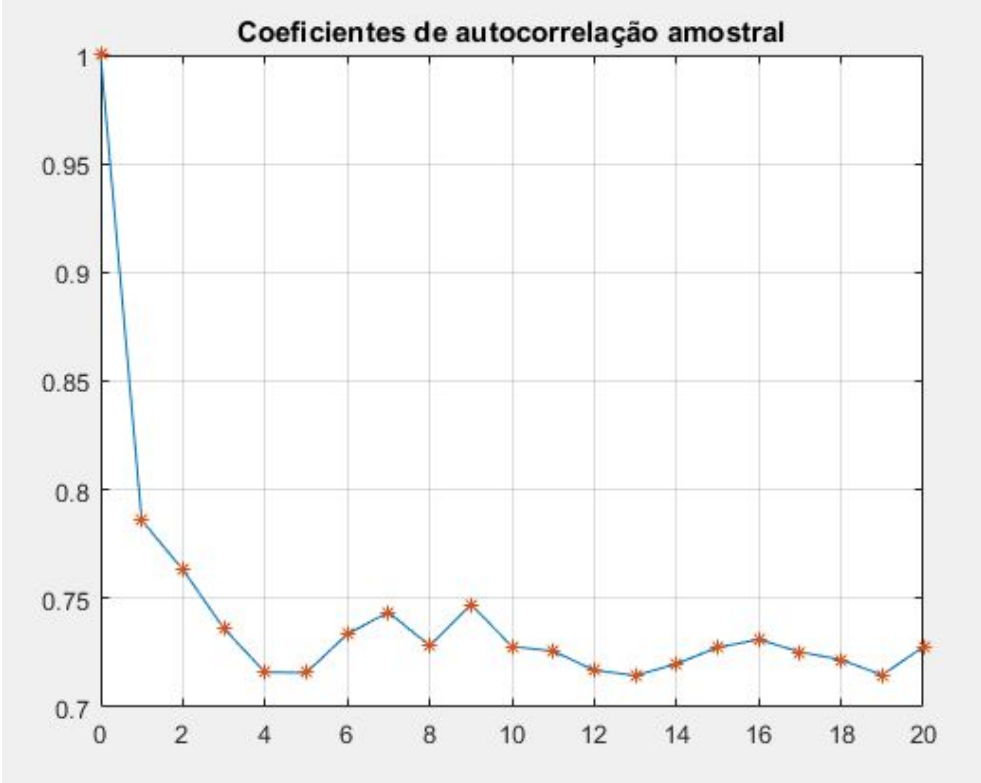


Rede neural Feed Forward MLP

Antes de se iniciar o treinamento, calculamos o coeficiente de autocorrelação de cada série temporal. Isso permite saber qual a correlação entre um elemento e seus elementos anteriores da série e, portanto, definir qual será o tamanho da camada de entrada da rede neural.

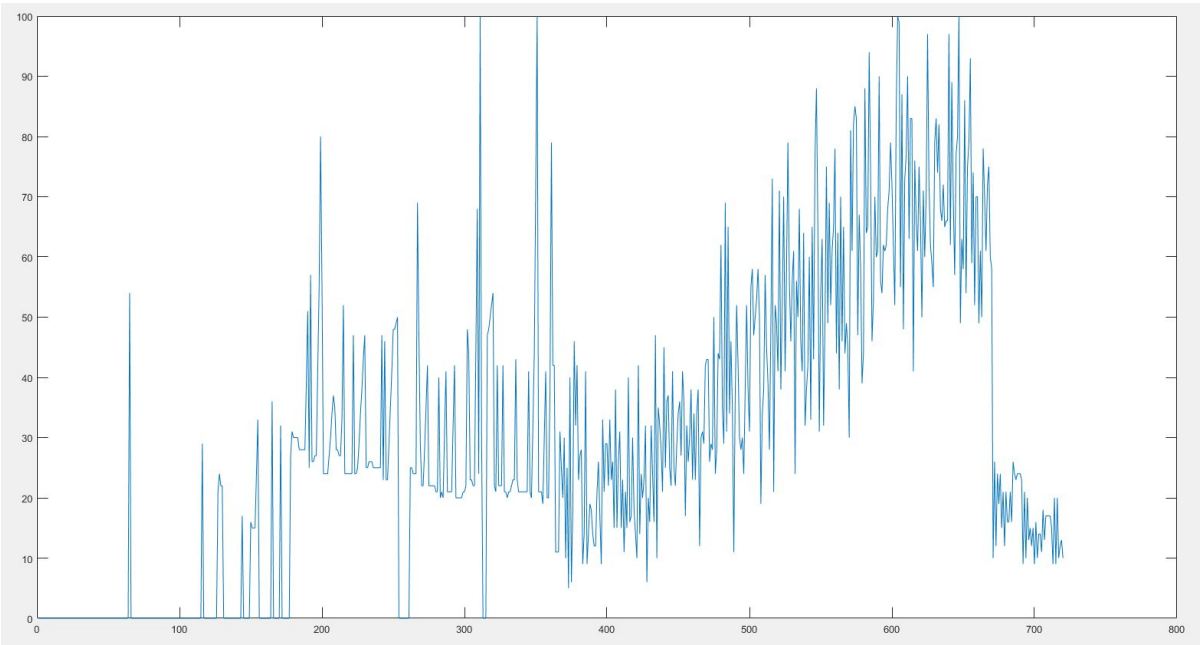
Ao final do treinamento das 10 redes neurais, é realizado em ensemble para predizer valores futuros para as séries. O ensemble é feito através da média das saídas das 10 redes e baseia-se no fato de que o ensemble terá uma média de erro quadrático médio menor que a média do erro quadrático médio de cada rede separada.

A figura abaixo mostra o coeficiente de autocorrelação para o sintoma de vômito no estado de Minas Gerais.

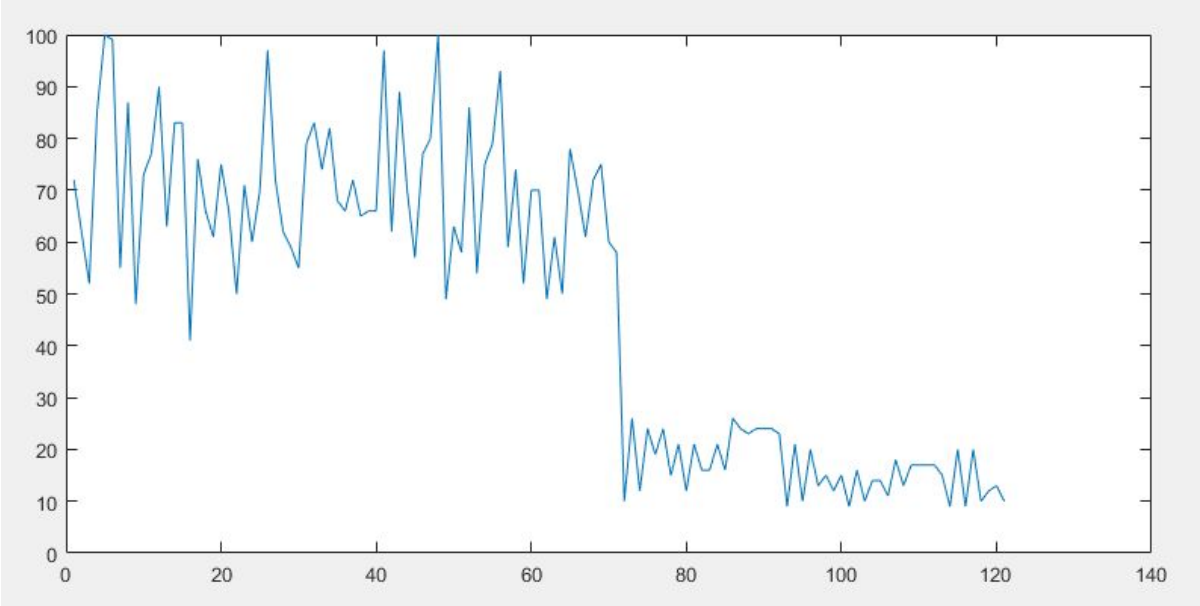


Coeficiente de Autocorrelação da Série Temporal de Vômito no Estado de Minas Gerais

Infelizmente, o uso de redes neurais MLP não se mostrou boa o suficiente para realizar predições em longo prazo. A figura abaixo mostra o resultado da predição da série temporal de buscas pelo termo vômito no estado de minas gerais. A predição começa a partir da semana de número 670.

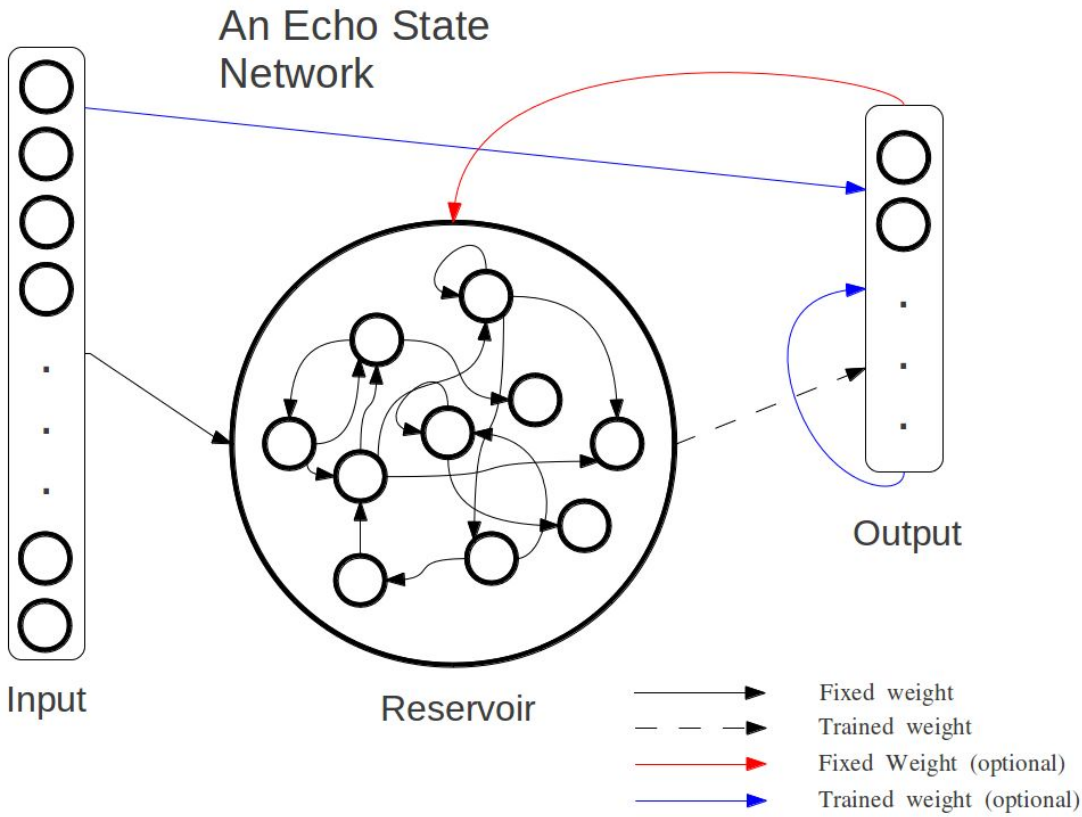


Um plot da série das semanas 600 até 720 é mostrado abaixo:



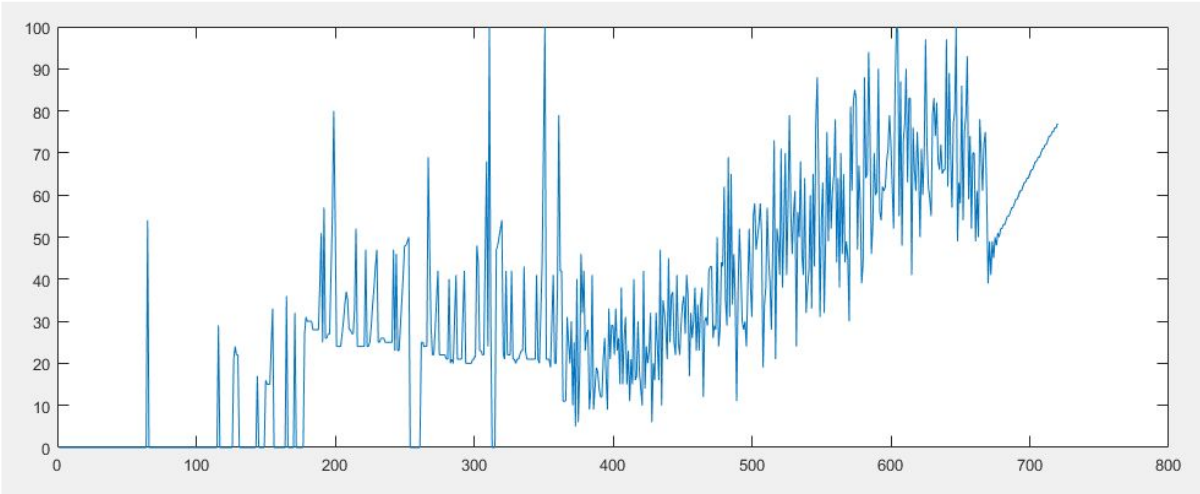
Com estes resultados, decidiu-se testar o uso de redes neurais recorrentes para realizar a predição. A vantagem do uso de redes neurais recorrentes é a realimentação de saídas ou estados internos da rede como entrada. Após uma breve análise das arquiteturas de redes disponíveis, optou-se pelo uso de redes de estado eco, por seu treinamento ser mais rápido e simples.

Redes de estado eco fazem parte de um conjunto de técnicas de machine learning chamada de reservoir computing. Nesta rede, existe um reservatório de estados composto de neurônios com ligações aleatórias entre si, formando uma camada recorrente. A ideia por trás do funcionamento é que o reservatório seja suficientemente grande para que existam dinâmicas entre neurônios suficientes para que seja possível aproximar a saída desejada apenas realizando uma combinação linear dessas dinâmicas. Em outras palavras, o número de dinâmicas guardadas no reservatório deve ser suficientemente grande para que existam algumas que possam ser usadas como solução. A figura abaixo mostra uma arquitetura genérica de uma rede de estados eco.



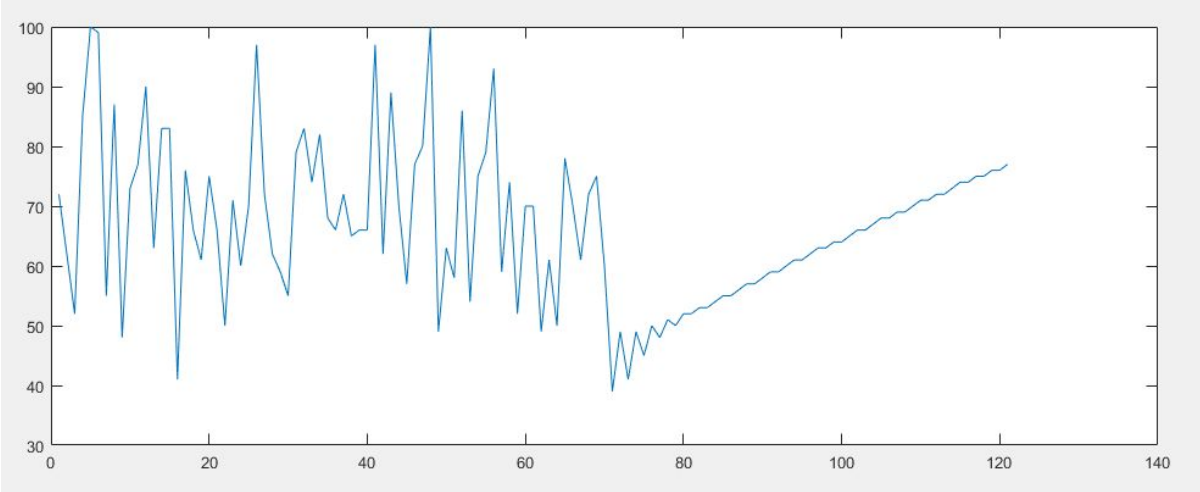
Rede Neural de Estados Eco

Nesta tentativa foram definidas como entradas da rede a saída anterior e o número daquela semana no ano. O resultado de uma predição de 50 valores futuros para a mesma série de vômito em Minas Gerais é mostrada na Figura abaixo:

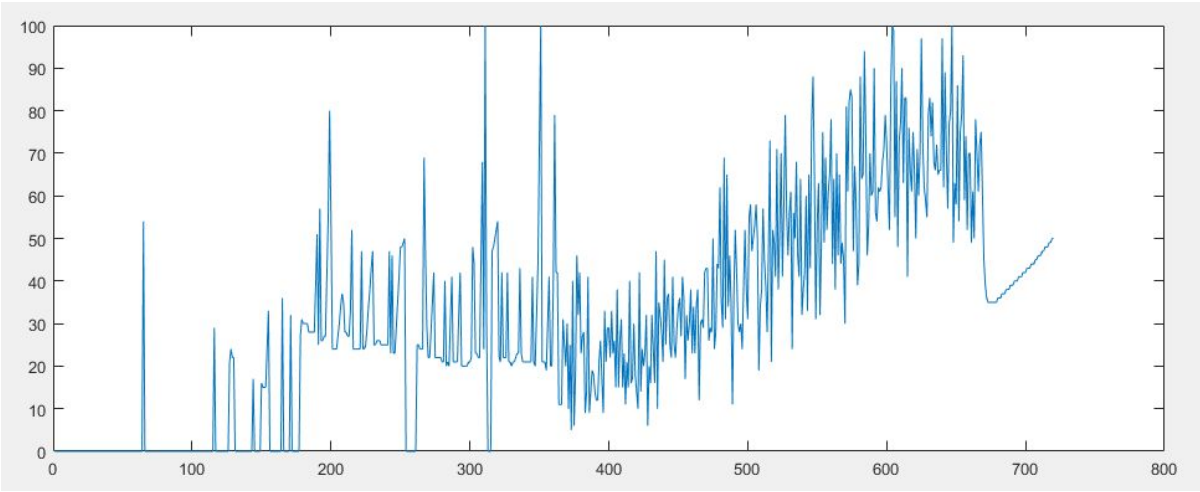


Ao observar a série no intervalo das semanas 600 a 720, é possível observar que para alguns valores preditos (após 70, na figura abaixo) ainda há um comportamento parecido com a série, porém este valor se estabiliza na entrada de

número da semana, tendendo para uma reta crescente.A figura abaixo mostra a série no intervalo das semanas 600-720.



Para tentar obter melhores resultados, alternamos a arquitetura da rede para rede de estado de eco invariante de distorção temporal (Time-warping invariant echo state network). Desta forma, espera-se melhores predições, uma vez que a rede pode lidar com variações nas velocidades entre amostras da série temporal. Infelizmente, o comportamento das predições não se modificou significativamente. A figura abaixo mostra o resultado da série e predição da série temporal de vômito em Minas Gerais para 50 valores futuros:



Visto que a utilização dos preditores neurais de estado eco não obteve muito sucesso, foi optado por usar as previsões realizadas pelos ensembles de redes MLP.