Universidade Federal de Pernambuco
Graduação em Ciência da Computação
Centro de Informática

Rafael Felipe Nascimento de Aguiar

Aprendizagem de Máquina para Previsão de Demanda em um Sistema de Compartilhamento de Bicicletas

Trabalho de Graduação

Recife, Julho de 2015

Rafael Felipe Nascimento de Aguiar

Aprendizagem de Máquina para Previsão de Demanda em um Sistema de Compartilhamento de Bicicletas

Trabalho de Graduação

Trabalho de Graduação apresentado à graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco para obtenção do grau de Bacharel em Ciência da Computação.

Orientador – Prof. Germano Crispim Vasconcelos <gcv@cin.ufpe.br>

Recife, Julho de 2015

Rafael Felipe Nascimento de Aguiar

Aprendizagem de Máquina para Previsão de Demanda em um Sistema de Compartilhamento de Bicicletas

Trabalho de Graduação

Trabalho de Graduação apresentado à graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco para obtenção do grau de Bacharel em Ciência da Computação.

Recife, ___ de julho de 2015

BANCA EXAMINADORA

_____

Prof. Germano Crispim Vasconcelos (Orientador)

_____

Prof.ª Patricia Cabral Azevedo Restelli Tedesco (Avaliadora)

Agradecimentos

Agradeço a minha família pelo apoio, aos meus professores pelos conhecimentos e a todos que de alguma forma me ensinaram a dar os pequenos passos que me trouxeram aqui e que me levarão adiante.

“ LIFE IS A MARATHON, NOT A SPRINT. ”
— PHILLIP C. MCGRAW

Resumo

O objetivo do presente trabalho é desenvolver um modelo de previsão de demanda de bicicletas em uma rede de compartilhamento, considerando dados das estações de compartilhamento (e.g, assinatura de tempo, número de registros) e da meteorologia da região (e.g., temperatura, umidade, velocidade do vento). A pergunta a qual o modelo deve responder é: dada uma data e hora do dia, qual o número total de bicicletas retiradas. Esta problemática é importante para garantir um balanceamento de carga entre as estações, conferindo estabilidade à rede de compartilhamento. Proposta por um popular site de competições na temática de aprendizagem de máquina chamado Kaggle, ela oferece uma oportunidade extraordinária para a utilização de técnicas de regressão (e.g., Redes Neurais, Ensembles, Random Forests, Gradient Tree Boosting) no suporte de uma Economia Compartilhada que visa melhorar a mobilidade urbana. Este estudo realiza um comparativo dessas técnicas (nos âmbitos de erro de validação, erros de teste e tempo de execução), obtendo melhores resultados através um ensemble entre uma Random Forest e uma Gradient Tree Boosting.

Palavras-chave: Aprendizagem de Máquina, Redes Neurais, Ensembles, Random Forests, Gradient Tree Boosting

Abstract

The goal of this work is to develop a model to forecast the demand for bicycles in a sharing network, considering data from the sharing stations (e.g., timestamps, number of records) and meteorology of the region (e.g., temperature, humidity, wind speed). The question which the model should answer is: given a date and time of day, what is the total number of withdrawn bicycles. This issue is important to ensure load balancing between stations, granting stability to the sharing network. Proposed by a popular website in machine learning competitions called Kaggle, it offers an extraordinary opportunity for the use of regression techniques (e.g., Redes Neurais, Ensembles, Random Forests, Gradient Tree Boosting) in support of a Sharing Economy aimed at improving urban mobility. This study conducts a comparison of these techniques (taking into account validation errors, test errors and running time), obtaining better results by ensembling a Random Forest and a Gradient Boosting Tree.

Keywords: Machine Learning, Neural Networks, Ensembles, Random Forests, Gradient Tree Boosting

Sumário

1.Introdução **9**

1.1.Contexto 9

1.2.Objetivos 10

1.3.Métodos de Implementação 10

2.Características do Conjunto de Dados **11**

2.1.Descrição das Variáveis 11

2.2.Análise Exploratória dos Dados 12

3.Pipeline de Aprendizagem de Máquina **16**

3.1.Diagramas 16

4.Engenharia de Variáveis **18**

4.1.Transformação log 19

4.2.Agrupamentos de dados 20

4.3.Redimensionando Variáveis 22

4.4.Simplificação de Variáveis 23

5.Estimadores **23**

5.1.Ensembles 23

5.1.1.Random Forest 24

5.1.2.Gradient Tree Boosting 25

5.2.Redes Neurais 26

5.2.1.Resilient Backpropagation com Backtracking 28

5.2.2.Backpropagation com Momentum 28

5.2.3.Parametrização 30

6.Resultados **30**

6.1.Análise Comparativa das Técnicas Propostas 31

7.Conclusão **32**

7.1.Contribuições 33

7.2.Dificuldades 33

7.3.Trabalhos Futuros 33

8.Referências **35**

Anexo A - Principais Componentes do Código de Aprendizagem **37**

1.Introdução

Este capítulo fornece uma contextualização do objeto de estudo, descreve o problema a ser resolvido e por último define os métodos que serão empregados na resolução.

1.1.Contexto

Em meados dos anos 2000 surge o termo “Economia Compartilhada”, nomeando um fenômeno que pretendia mudar a maneira como os seres humanos utilizavam recursos (Benkler 2004). O movimento de Economia Compartilhada questionava a relação quase que linear entre crescimento populacional e a produção de bens duráveis; onde um carro, por exemplo, era produzido com o fim de servir a um único consumidor.

Esse modelo de produção com finalidade individual acabara por prejudicar (e.g., congestionamentos, poluição) meios que são inerentemente compartilhados como as vias de transporte público.

Como exemplo bem sucedido de antagonismo a esse modelo individualista de produção, surgem as primeiras redes de bicicletas compartilhadas; oferecendo uma alternativa mais econômica, ecologicamente correta e benéfica à saúde que o ônibus ou o carro.

Uma rede de bicicletas compartilhadas é uma malha de estações conectadas por rodovias ou ciclovias, em que um usuário pode retirar um bicicleta em um ponto A e devolver em um ponto B (Susan Shaheen 2011). Esses trajetos costumam ser de curta duração e se dividem em duas categorias: lazer e trabalho.

O problema mais desafiador de se manter uma rede como essa é lidar com a distribuição irregular de bicicletas entre estações. Posto que bicicletas podem ser retiradas em uma estação e retornadas em qualquer outra, em um dado instante de tempo estações podem estar tanto vazias quanto cheias; e esses dois estados são problemáticos para rede pois impedem que um usuário inicie uma viagem por falta de bicicletas ou termine uma viagem por falta espaço para retornar uma bicicleta.

Por essa razão, bicicletas precisam ser movidas de estações com alta lotação para estações com baixa lotação e este trabalho aborda uma das formas de se apoiar um sistema de logística para este fim, através da previsão de demanda de bicicletas nas estações.

1.2.Objetivos

O objetivo do presente trabalho é desenvolver um modelo de previsão de demanda de bicicletas em uma rede de compartilhamento considerando dados das estações de compartilhamento (e.g, assinatura de tempo, número de registros) e da meteorologia da região (e.g., temperatura, umidade, velocidade do vento).

A pergunta a qual o modelo deve responder é: dada uma data e hora do dia, qual o número total de bicicletas retiradas.

Esta problemática foi proposta por um popular site de competições na temática de aprendizagem de máquina (Machine Learning - ML) chamado Kaggle[1]. Os dados são provenientes da plataforma americana de compartilhamento de bicicletas Capital Bikeshare[2] e são referentes as retiradas no período de 2011 a 2012 em Washington, DC.

1.3.Métodos de Implementação

Este trabalho se propõe a implementar tal modelo utilizando técnicas de aprendizagem de máquina.

A justificativa para tal escolha se dá através da análise dos seguintes fatores do problema: não existe uma função que dado um exemplo retorna o resultado exato; existem dados para serem explorados; existem padrões nesses dados.

A aprendizagem de máquina engloba uma série de técnicas que a princípio são divididas em supervisionadas e não supervisionadas. A principal diferença entre elas sendo que a primeira demanda rótulos (labels) nos dados e a segunda não.

Os problemas a serem resolvidos por tais técnicas também comumente se enquadram em duas categorias: classificação e regressão. O primeiro sendo um problema onde dado um item a pergunta a ser respondida é a qual classe esse item pertence (e.g., dado um paciente e seus sintomas, responder se ele é classificado como alto risco ou baixo risco), o segundo sendo um problema onde o objetivo é quantificar o valor de uma variável (dados atributos de uma casa, prever seu valor de mercado).

A previsão de demanda que é objeto deste trabalho se enquadra na categoria de problemas de regressão e o conjunto de dados fornecido possui exemplos com

[1] https://www.kaggle.com/c/bike-sharing-demand

[2] http://www.capitalbikeshare.com

labels para a variável de interesse (a ser medida), possibilitando a utilização de técnicas supervisionadas.

Mesmo depois dessa segmentação do problema ainda existem uma pletora de técnicas que podem ser empregadas; cada técnica explora diferentes conceitos matemáticos e por consequência tem melhores resultados para certos tipos de dados. Por isso, é comum que um problema seja abordado por diversas técnicas e que a solução final seja obtida através de uma extensiva análise comparativa de resultados.

O presente trabalho se reservará à análise do problema através de um subconjunto reduzido dessas técnicas: Redes Neurais, Random Forests, Gradient Tree Boosting e Ensembles.

2.Características do Conjunto de Dados

Neste capítulo serão apresentadas informações detalhadas sobre o conjunto de dados utilizado para a competição. Essa análise do dataset resultou em descobertas que motivaram importantes decisões e acabaram proporcionando os maiores ganhos (quando comparadas aos ganhos por escolha de um estimador específico ou otimização de parâmetros) de desempenho no modelo final. O conjunto de treinamento possui 10886 registros e o de testes 6493.

2.1.Descrição das Variáveis

As doze variáveis disponíveis no conjunto de dados (antes de qualquer modificação) estão representadas a seguir nas Tabelas 1 e 2.

Observações: nenhuma das variáveis possui valores ausentes (*missing values)*; ***count*** é a variável cuja previsão deverá ser submetida para a competição.

Tabela 1 - Variáveis 1 a 6

	Season	**Holiday**	**Working day**	**Weather**	**Temp**	**aTemp**
Tipo	Categórico	Booleano	Booleano	Categórico	Numérico	Numérico
Valores	1 a 4	0 ou 1	0 ou 1	1 a 4	0.8ºC a 41ºC	0.7ºC a 45ºC
Descrição	Estações do ano: inverno, primavera, verão, outono	Feriado ou não	Dia de semana ou não	Condições do Tempo: nublado, neblina, chuva/neve leve, chuva/ neve forte	Temperatura	Sensação térmica

Tabela 2 - Variáveis 7 a 12						
	Humidity	**Windspeed**	**Casual**	**Registered**	**Count**	**DateTime**
Tipo	Numérico	Numérico	Numérico	Numérico	Numérico	String
Valores	0% a 100%	0 a 57 km/h	0 ou 367	0 ou 886	1 a 997	2011-01-01 12:00:00 AM a 2012-11-02 12:00:00 AM
Descrição	Humidade	Velocidade do vento	# bicicletas retiradas por usuários casuais	# bicicletas retiradas por usuários registrados na plataforma	casual + registrado	Data e hora

2.2. Análise Exploratória dos Dados

Afim de detectar padrões nos dados, uma análise exploratória foi conduzida através de um software de análise visual de dados (visual analytics) chamado Tableau. Ao contrário das várias outras opções (e.g., Matplotlib) para realização dessa tarefa o Tableau possibilitou uma análise rápida, interativa, com rico teor visual e livre da necessidade de escrever código.

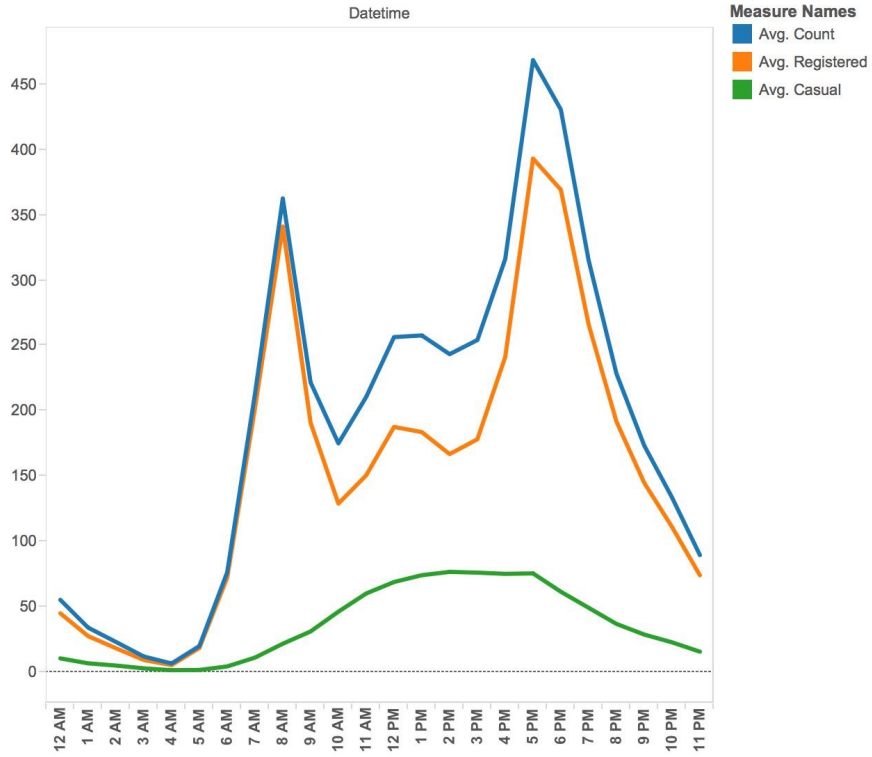

FIGURA 1. (count, registered, casual) por Hora

Na Figura 1 são observados o valor médio das variáveis *count*, *registered* e *casual* durante as 24 horas do dia. Nela existem dois fatos importantes a serem identificados: o comportamento da variável *registered* é substancialmente diferente da variável *casual;* o comportamento da variável *registered* é semelhante (considerando alguns deslocamentos) ao da variável de interesse.

Avaliando esses insights de acordo com a definição da variável *count* expressada na relação (I),

$$(I) \quad count = casual + registered$$

é possível antecipar que uma previsão direta para a variável *count* (variável de interesse para submissão na competição) irá apresentar baixo desempenho, pois ela é uma função composta por duas variáveis de comportamentos diferentes. Logo, a geração de dois modelos com variáveis de interesse diferentes (um utilizando *casual* e outro *registered*) e posterior agregação utilizando a relação (I) se faz mais efetiva (um erro cerca de 43% menor considerando como modelo uma Random Forest que será apresentada no capitulo 5).

Na Figura 2 são observados o somatório de cada uma das variáveis de interesse (*casual* e *registered)* durante os sete dias da semana. Outra justificativa para criação de modelos independentes para *casual* e *registered* é que o comportamento semanal dessas variáveis é explicitamente antagônico.

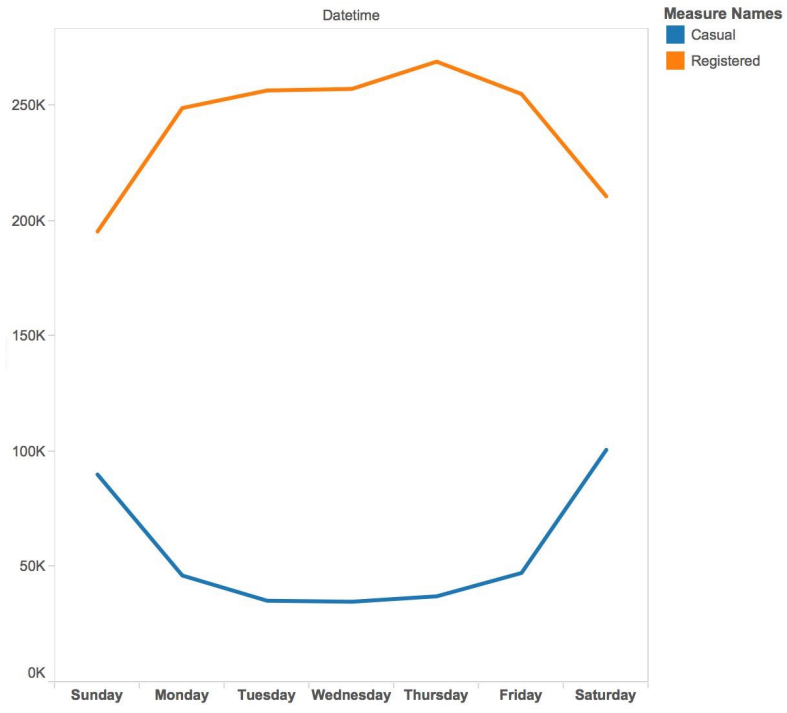

FIGURA 2. (casual, registered) por Dia da Semana

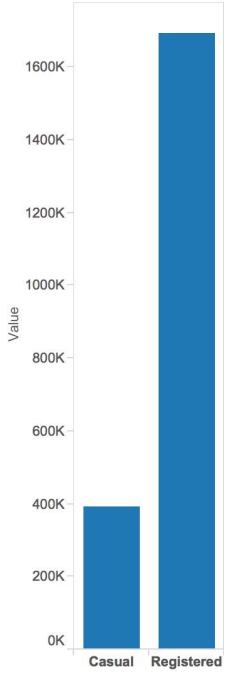

FIGURA 3. (# casual, # registered)

Na Figura 3 são observados os somatórios globais (em todo o conjunto de dados) para as variáveis *casual* e *registered*. Aqui fica evidente que o número de retiradas de bicicletas por usuários registrados é extremamente maior que por usuários casuais.

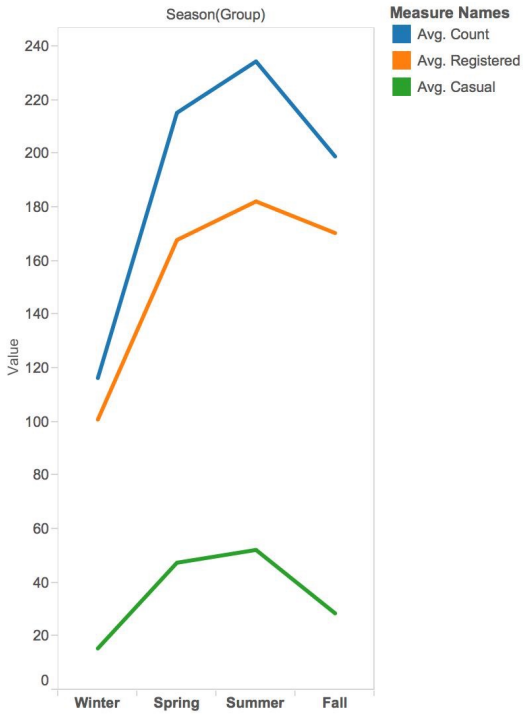

FIGURA 4. (count, casual, registered) por Estação do Ano

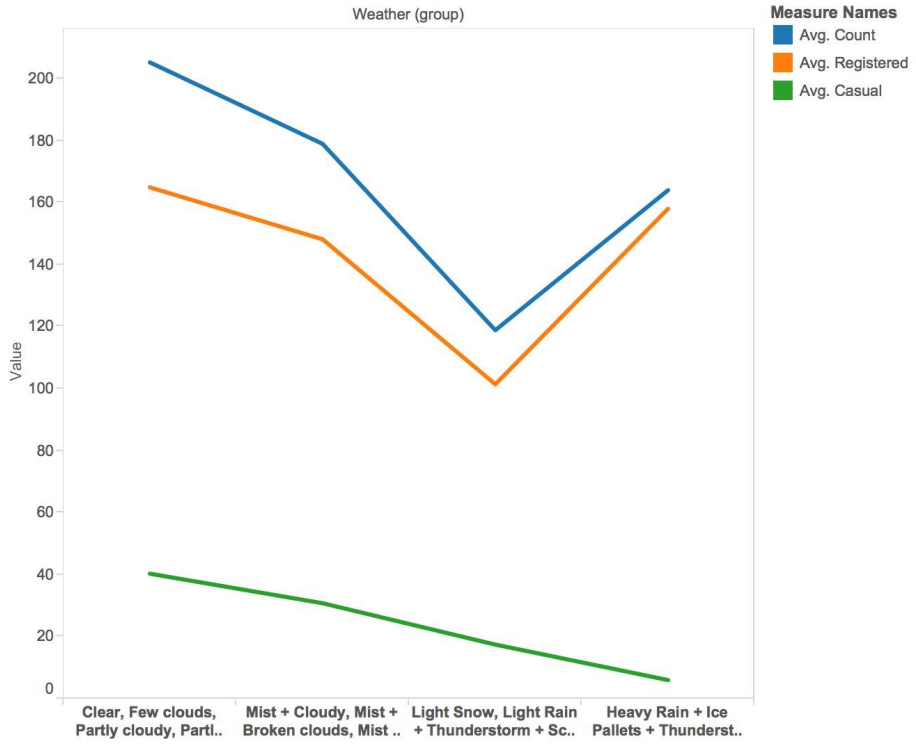

FIGURA 5. (count, casual, registered) por Condição do Tempo

Nas Figuras 4 e 5 são verificadas como as variáveis *count*, *casual* e *registered* se comportam quando dados são agrupados sob variáveis categóricas (estação do ano e condições de tempo).

Isto é, considere essa operação de agrupamento tal e qual uma consulta SQL da forma:

```
"SELECT * FROM Data GROUP BY season"
```

Ainda nas Figuras 4 e 5 percebemos uma certa semelhança no padrão das curvas (em 4, as três variáveis se assemelham; em 5, duas delas). Assim, duas tentativas de explorar esses padrões foram feitas: a primeira, um ensemble com um estimador para cada grupo possível (mais no capítulo 4 seção 2); a segunda, a criação de uma variável de agregação (mais no capítulo 4 seção 2).

Na Figura 6 estão ilustradas as médias da variável *count* para algumas horas do dia (em saltos de 4 horas), com uma curva para cada dia da semana. Aqui pode ser percebida a diferença entre o comportamento da variável *count* para dias de semana (Seg, Ter, Qua, Qui, Sex) e finais de semana (Sab, Dom).

Assim, é esperado que modelos que de alguma forma incorporem esses padrões (e.g., previsões independentes para *casual* e *registered*), apresentem

baixas taxas de erro. Parte da seção Engenharia de Variáveis (no capítulo 4) será destinada a mencionar quais desses insights trouxeram melhorias aos resultados.

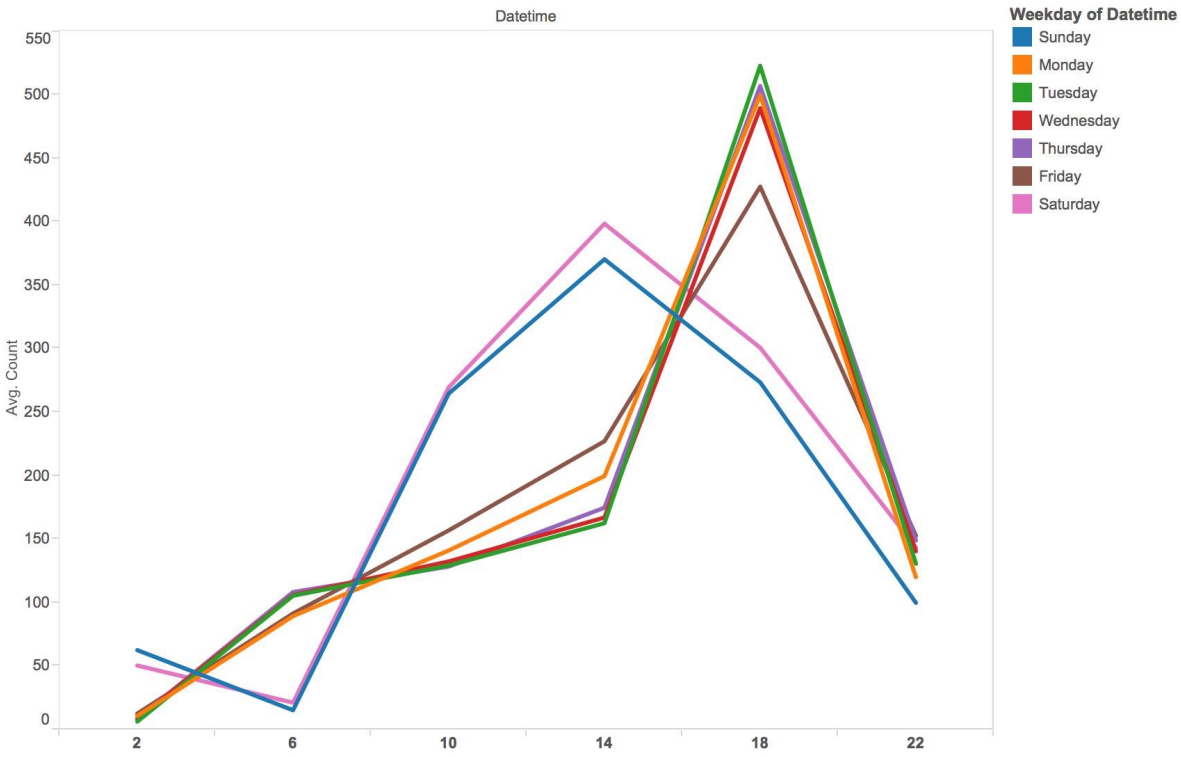

FIGURA 6. (count, weekday) por Hora

3.Pipeline de Aprendizagem de Máquina

Neste capítulo será apresentado o pipeline final utilizado para efeitos deste estudo e serão também mencionadas algumas alternativas descartadas durante o processo.

3.1.Diagramas

A Figura 7 ilustra a fase de preparação dos dados, envolvendo atividades de carregamento dos arquivos (treinamento e teste) e de pré-processamento. A última englobando: modificação de variáveis, criação de variáveis baseadas em agrupamento de dados, criação de variáveis simplificadas e por fim escrita dessas modificações em disco. A fase de pré-processamento será explicada em detalhes no capítulo 4.

A Figura 8 ilustra o processo de aprendizagem de máquina propriamente dito, onde quatro modelos de regressão são utilizados dentro desse estudo.

Os estimadores *Random Forest* e *Gradient Tree Boosting* são provenientes do módulo Scikit-Learn para a linguagem Python, já os estimadores *Rede Neural* (Rprop+) e *Rede Neural (Momentum)* são originários da linguagem R através dos pacotes NeuralNet e AMORE, respectivamente.

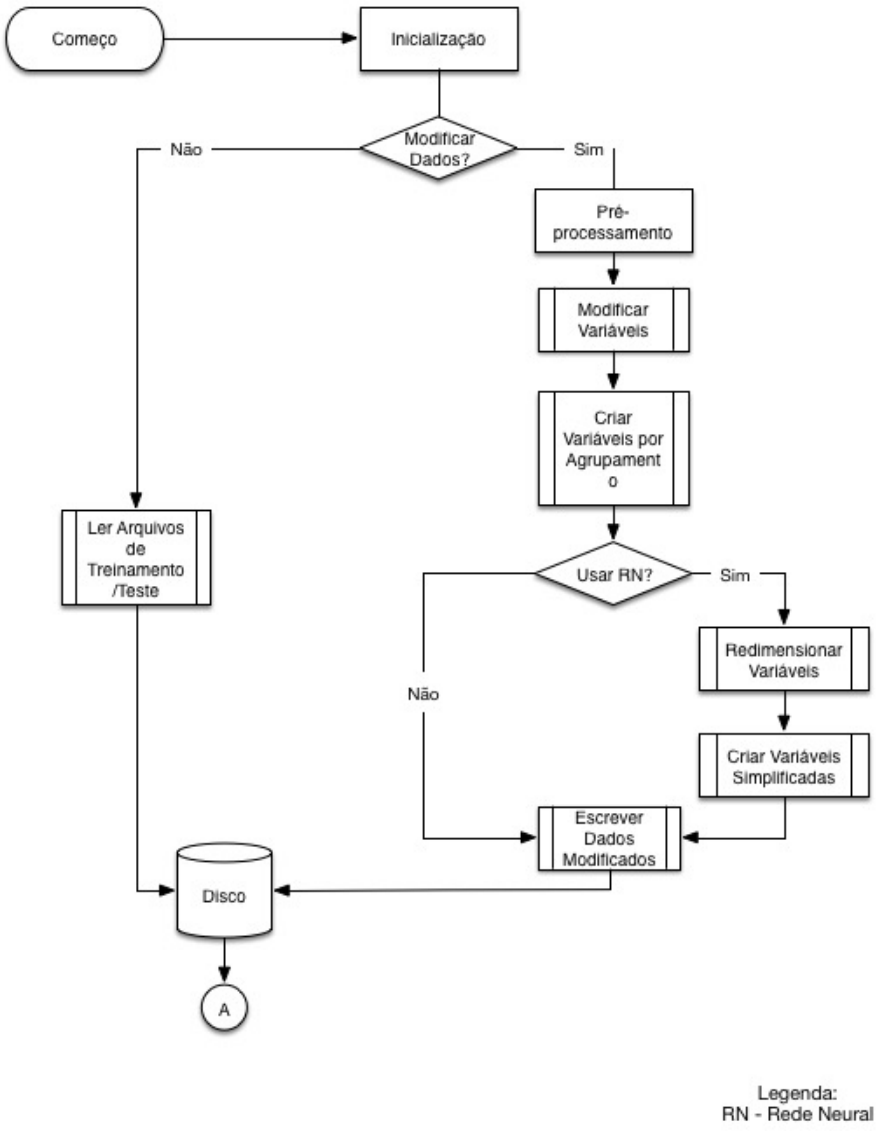

FIGURA 7. Diagrama de Preparação de Dados

Neste projeto foi dada preferência a linguagem Python para o desenvolvimento da codificação. A breve presença da linguagem R se dá: um, pela qualidade dos modelos e documentação oferecidos; dois, pela facilidade de construção de um pipeline integrado entre as duas linguagens (através de um módulo Python chamado Rpy2).

Os modelos mencionados acima tem seus parâmetros ajustados através de um processo de 5-fold cross-validation.

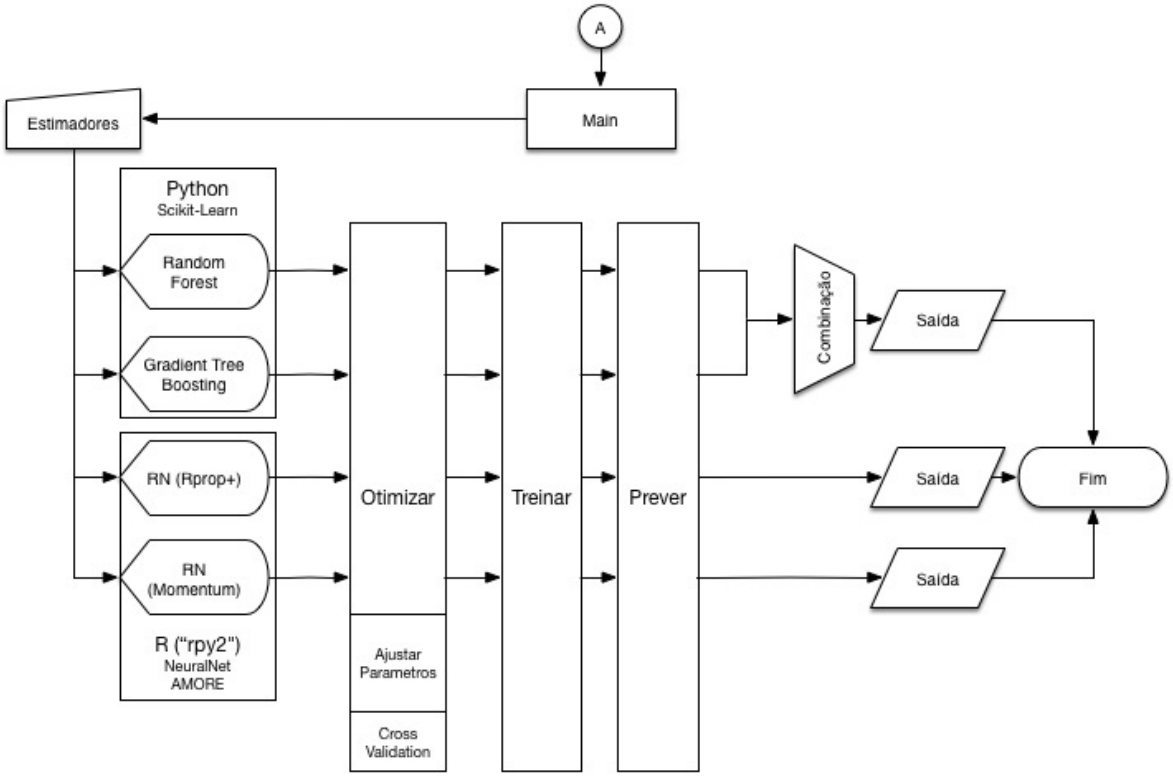

FIGURA 8. Diagrama de Aprendizagem de Máquina

Em seguida eles são treinados em todo o conjunto de treinamento (na fase Treinar) e utilizados para fazer previsões no conjunto de testes (na fase Prever).

Alguns modelos (Random Forest e Gradient Tree Boosting) são ainda combinados em um ensemble (capítulo 5.1) para só então produzirem um resultado e serem escritos em um arquivo de submissão para a plataforma Kaggle.

4.Engenharia de Variáveis

Este capítulo discorre acerca de modificações de variáveis (i.e,. features) fornecidas originalmente e da criação de novas variáveis para aumentar o desempenho de modelos. A motivação por traz dessa atividade é que apenas a otimização de parâmetros de estimadores não é grande garantia de bom desempenho; em verdade, os maiores ganhos desse projeto foram devidos a manipulação de variáveis.

4.1.Transformação log

A primeira transformação tem sua justificativa na equação de cálculo de erro utilizada para fins da competição. O Root Mean Squared Logarithmic Error (RMSLE) é definido pela equação (II) a seguir:

(II) $$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

- n é o número de horas no conjunto de testes
- p_i é o valor previsto para a variável *count*
- a_i é o valor verdadeiro da variável *count*
- $\log(x)$ é o logaritmo natural

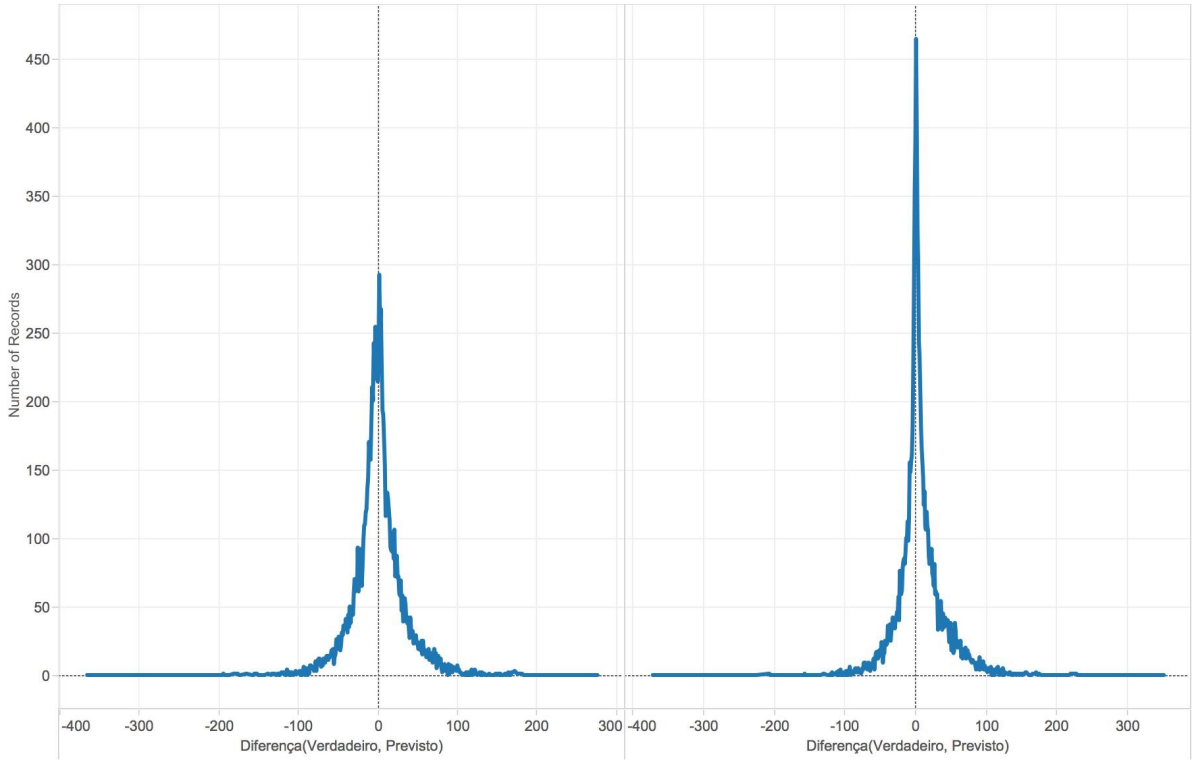

FIGURA 9. Histogramas da Diferença Entre o Valor Verdadeiro e o Valor Previsto (a) Sem a Transformação Log, (b) Com a Transformação Log.

Em geral, um bom motivo para se aplicar uma transformação em uma variável é obter-se resíduos da função de erro que são simetricamente distribuídos e aproximadamente zero (Stine 2001, Benoit 2011).

Na Figura 9 são observados dois histogramas para a diferença (valor atual, valor previsto). Ao aplicar a transformação (III) para as variáveis *casual* e *registered* e prever diretamente os valores envolvidos no calculo da função de erro (II) foi-se observada uma distribuição normal com mais exemplos com erro zero (Figura 9 (b)).

(III) $$feature = \log(feature + 1)$$

4.2. Agrupamentos De Dados

Em busca de explorar alguns dos padrões de agregação mencionados no capítulo 2, foram testados tanto a criação de ensembles quanto a criação de variáveis por agrupamento.

A estratégia baseada em ensembles de grupos específicos (e.g., um estimador para cada hora do dia) teve um desempenho muito inferior ou similar ao desempenho (erro de validação ~ 0.44) de um único grupo (treinamento de um único estimador no dataset inteiro). Isso pode ser justificado pela baixa quantidade de exemplos para treinar cada estimador, já que essas quantidades ficam limitadas de acordo com o tamanho médio do grupo mostrado na Tabela 3.

Para evitar esse problema com baixa quantidade de exemplos, foi dado inicio a abordagem de criação de variáveis por agrupamento (e.g., count group by season).

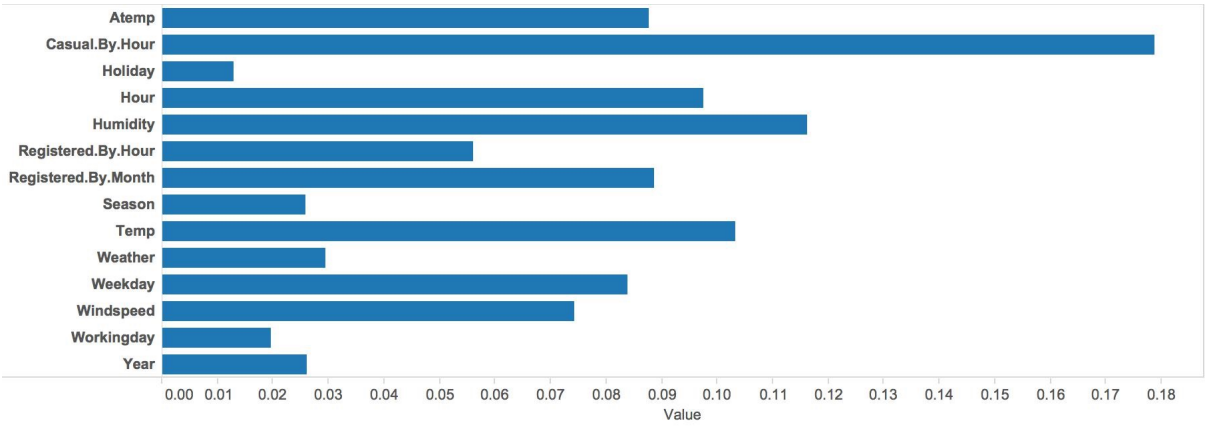

FIGURA 10. Gradient Boosting Tree (casual) - Importância de Variáveis

O erro de validação para cada par (variável de interesse, variável de agrupamento) considerando uma Random Forest (erros para outros estimadores apresentaram comportamento similar, isto é, erros de validação similarmente distribuídos) que inclui tais variáveis por agrupamento é reportado na Tabela 3. Esses dados foram obtidos através de cinco 5-fold cross-validation, cada um incluindo as variáveis base e apenas uma variável de agrupamento por vez.

Tabela 3 - Variáveis por Agrupamento

Variável / Grupo	Ano	Estação do Ano	Condição de Tempo	Mês	Hora
Tamanho médio do grupo	**5443**	**2722**	**2722**	**907**	**454**
count	0.46	0.44	0.45	0.44	0.44
casual	0.44	0.44	0.44	0.44	**0.43**
registered	0.44	0.44	0.44	**0.42**	**0.42**

Os resultados presentes na Tabela 3 serviram de embasamento para a adição de 3 variáveis ao dataset: *registered_by_month, registered_by_hour* e *casual_by_hour*.

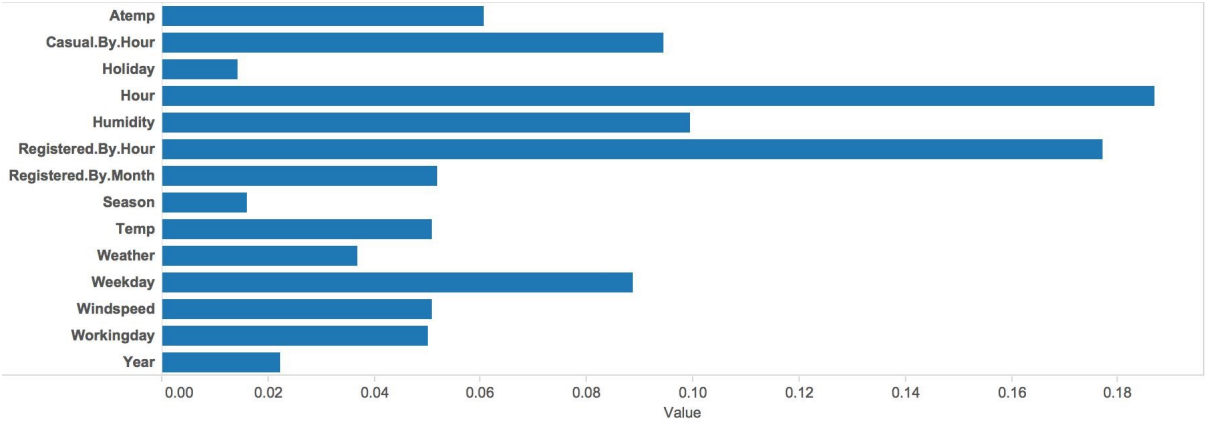

FIGURA 11. Gradient Boosting Tree (registered) - Importância de Variáveis

A importância dessas variáveis (considerando como variável dependente casual ou registered) é ainda ilustrada nas figuras 10, 11, 12 e 13; indo além da diminuição de erro demonstrada na Tabela 3 e mostrando a real importância de cada feature dentro de alguns modelos.

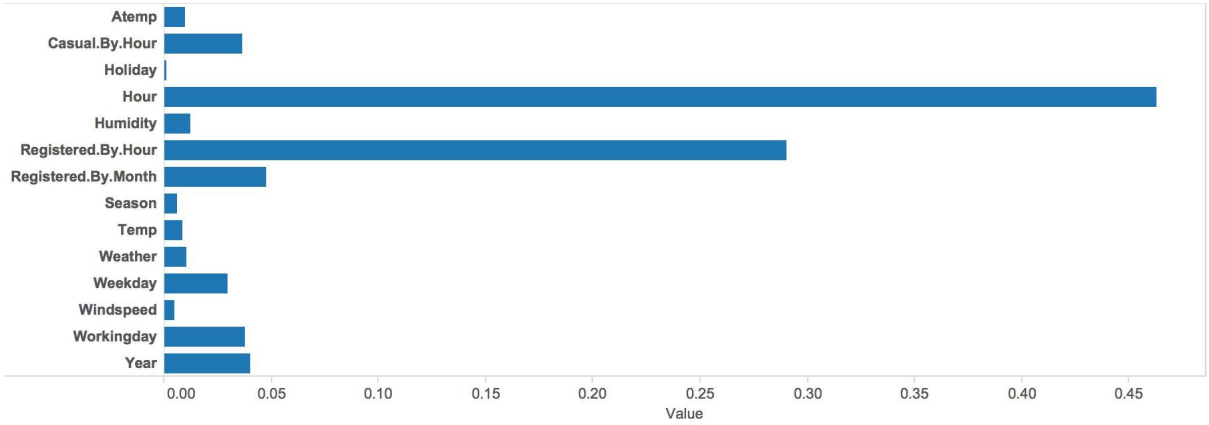

FIGURA 12. Random Forest (casual) - Importância de Variáveis

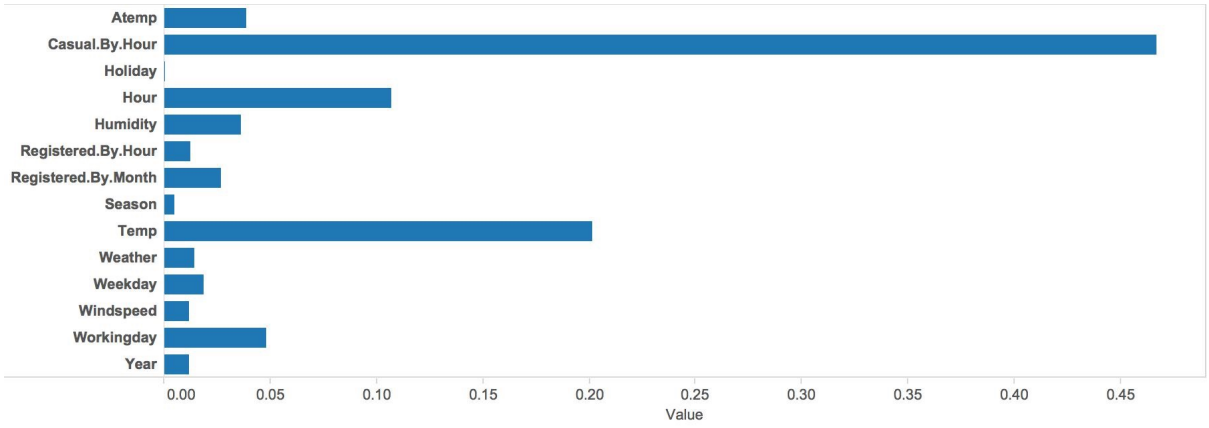

FIGURA 13. Random Forest (registered) - Importância de Variáveis

4.3.*Redimensionando Variáveis*

Para uso de redes neurais se faz necessário redimensionar todas as variáveis numéricas dentro de um intervalo [0,1]. O redimensionamento se dá através da classe MinMaxScaler definida no modulo de pré-processamento do Scikit-Learn de acordo com as equações abaixo:

(IV) `X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))`

(V) `X_scaled = X_std * (max - min) + min`

Todas as variáveis numéricas são transformadas na fase de pré-processamento e apenas as variáveis de interesse (*casual* e *registered*) sofrem a

transformação inversa (em momentos posteriores) para o calculo da RMSLE ou para criação do arquivo de submissão.

Tanto essa, quanto a transformação da seção seguinte (4.4) foram realizadas em cópias dos datasets originais, de forma que apenas os modelos que utilizam redes neurais fazem uso delas. Os demais modelos ignoram as transformações das seções 4.3 e 4.4.

4.4.Simplificação de Variáveis

Um último tratamento de variáveis para possibilitar a exploração desse dataset através de redes neurais envolve a criação de variáveis simplificadas (dummy variables).

Garavaglia, define dummy variables como:

> "Variáveis simplificadas são variáveis que assumem o valor de zero ou um. Assim como um manequim está para uma verdadeira pessoa, em uma análise quantitativa, uma variável simplificada é uma representação numérica para um fato categórico ou proposição lógica."

Nessa transformação, variáveis qualitativas serão transformadas em um conjunto de variáveis booleanas (Garavaglia 2004). Por exemplo, a variável *season* que possuía quatro valores categóricos (Winter, Spring, Summer, Fall) será transformada em três variáveis booleanas.

5.Estimadores

O presente capítulo é destinado a brevemente explicar o funcionamento dos estimadores utilizados (Random Forest, Gradient Tree Boosting, Redes Neurais com Resilient Backpropagation e Backtracking, e Redes Neurais com Momentum) e a apresentar algumas vantagens e desvantagens dos mesmos (verificadas durante o projeto).

5.1.Ensembles

Dois estimadores utilizados neste projeto (Random Forest e Gradient Tree Boosting) são baseados em um conceito chamado aprendizado em conjunto (ensemble learning).

Um ensemble consiste de um conjunto de classificadores (a mesma ideia se aplica para regressão) treinados individualmente (como redes neurais ou árvores de decisão) cujas previsões são combinadas para classificar novas instancias.

Pesquisas passadas indicam que um ensemble é frequentemente mais preciso que qualquer um dos classificadores no ensemble (Opitz and Maclin 1999). Bagging (Breiman 1996) e Boosting (Schapire and Freund 2012) são dois métodos relativamente novos, mas populares, para produzir ensembles homogêneos (compostos por uma combinação do mesmo estimador).

Em Bagging (acrônimo para "**b**ootstrap **agg**regat**ing**"), um conjunto de dados com N exemplos é amostrado randomicamente com repetição de forma a gerar uma quantidade pré-determinada (T) de estimadores, cada um treinado independentemente em um conjunto diferente (de tamanho N) (Breiman 1996).

Em Boosting, um conjunto de dados com N exemplos é amostrado para treinar um estimador T com base em T_{-1} , através do ajuste do peso de cada exemplo na distribuição (Freund and Schapire 1996). Exemplos mal classificados aumentam de peso e exemplos bem classificados diminuem de peso, de forma a forçar um classificador fraco a aprender seus erros. Ao contrário de Bagging, o treinamento não é feito de maneira independente; cada estimador é influenciado pelo erro dos estimadores treinados anteriormente.

5.1.1.Random Forest

Random Forests são uma combinação de árvores de decisão em que cada árvore depende dos valores de um vetor aleatório amostrado independentemente e com a mesma distribuição de todas as árvores da floresta (Breiman 2001).

De maneira mais formal, Breiman define uma Random Forest como:

> "Uma Random Forest é um classificador que consiste de uma coleção de classificadores {h(x, k), k = 1, . . .} em estrutura de árvore em que {k } são vetores randômicos independentes identicamente distribuídos e cada árvore contribui com um voto para a classe mais popular dado uma entrada x."

Random Forests para o propósito de regressão são formadas através do crescimento de árvores que dependem de um vetor aleatório tal qual uma árvore h (x,) assume em valores numéricos ao invés de rótulos de classe (class labels) (Breiman 2001).

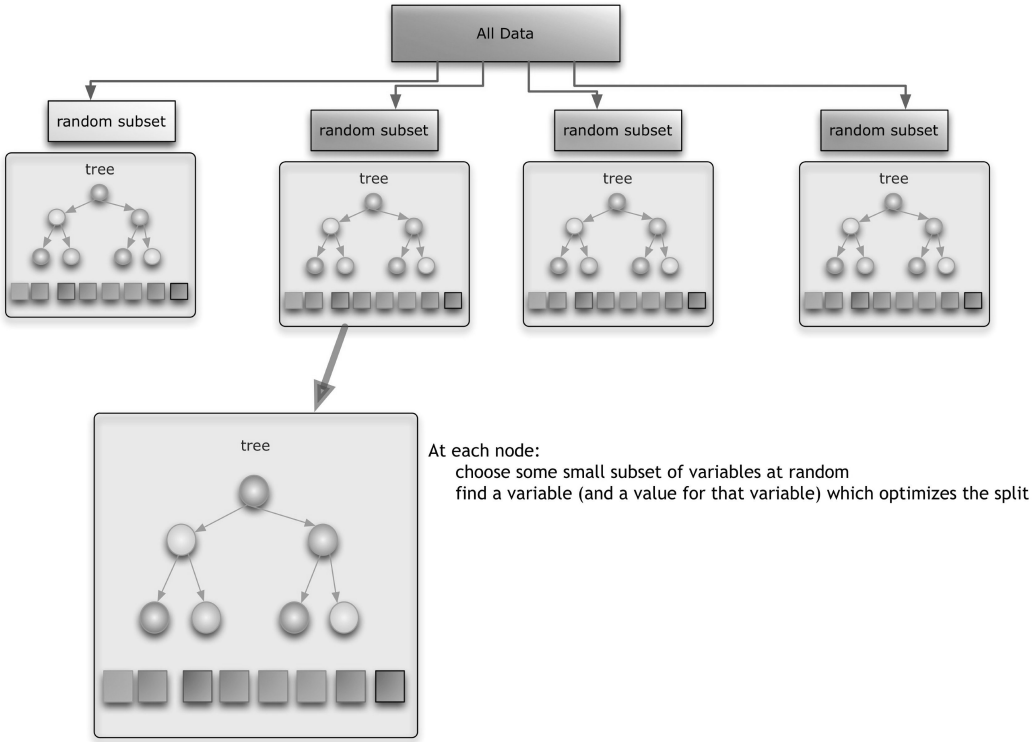

FIGURA 14. Construção de uma Random Forest

O processo de ensembling de árvores de decisão é demonstrado na Figura 14.

Entre as razões pelas quais esse estimador foi escolhido para modelagem estão: baixo número de parâmetros (essencialmente o número de árvores e o número mínimo de variáveis em um conjunto randômico em cada nó); robustez em relação a overfitting; tempo de execução costuma ser baixo (considerando baixo número de árvores) (Andy Liaw 2002).

5.1.2.Gradient Tree Boosting

Com o objetivo de investigar o desempenho de métodos baseados em Boosting, o método AdaBoost foi avaliado para utilização neste problema. Infelizmente não foram obtidos resultados satisfatórios com o AdaBoost para esse dataset. Dessa forma, um outro método baseado em Boosting foi empregado: o Gradient Tree Boosting.

Tomando o conceito de Boosting apresentado no capítulo 5.1, o conceito de Gradient Boosting pode ser entendido como uma generalização em que T é definido a partir de uma otimização de uma função de erro diferenciável em T_{-1} (Mason, Baxter et al. 1999). Isto é, ao contrário de uma Random Forest onde todos os estimadores dentro do ensemble possuem o mesmo peso (contribuem em igualdade

para a previsão final), uma Gradient Boosted Tree utiliza pesos diferentes para cada estimador. Estes pesos são aprendidos pela minimização de uma função de erro através de gradiente similarmente ao processo descrito para redes neurais no capítulo 5.2.

Gradient Boosting pode ser aplicado essencialmente a qualquer algoritmo de aprendizagem. Neste estudo Gradient Boosting é aplicado em árvores de decisão; um algoritmo genérico (para classificação) é apresentado na Figura 15 (Mason, Baxter et al. 1999).

Algorithm 1 : AnyBoost

Require :
- An inner product space $(\mathcal{X}, \langle , \rangle)$ containing functions mapping from X to some set Y.
- A class of base classifiers $\mathcal{F} \subseteq \mathcal{X}$.
- A differentiable cost functional $C\colon \text{lin}\,(\mathcal{F}) \to \mathbb{R}$.
- A weak learner $\mathcal{L}(F)$ that accepts $F \in \text{lin}\,(\mathcal{F})$ and returns $f \in \mathcal{F}$ with a large value of $-\langle \nabla C(F), f \rangle$.

Let $F_0(x) := 0$.
for $t := 0$ to T **do**
  Let $f_{t+1} := \mathcal{L}(F_t)$.
  if $-\langle \nabla C(F_t), f_{t+1} \rangle \leq 0$ **then**
    return F_t.
  end if
  Choose w_{t+1}.
  Let $F_{t+1} := F_t + w_{t+1} f_{t+1}$
end for
return F_{T+1}.

FIGURA 15. Algoritmo Genérico de Boosting para Classificadores

5.2.Redes Neurais

Redes Neurais ganharam popularidade por frequentemente apresentarem um desempenho competitivo em relação a outros métodos e principalmente por sua extensiva capacidade de expressar combinações não lineares entre variáveis. Como brevemente mencionado no capítulo 3, as redes neurais utilizadas nesse projeto são provenientes de dois pacotes da linguagem R (NeuralNet[3] e AMORE[4]) e são explicadas brevemente nos sub-capítulos a seguir.

Essas redes são MLP (Multi Layer Perceptron) do tipo feedforward com algumas modificações no algoritmo de backpropagation. Por isso, para que as

[3] http://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf

[4] http://cran.r-project.org/web/packages/AMORE/AMORE.pdf

diferenças se tornem claras, uma descrição de como o backpropagation padrão funciona é apresentada neste capítulo.

Uma rede MLP do tipo feedforward, ilustrada na Figura 16, é composta por neurônios divididos em camadas (e.i., entrada, escondidas, saída). A função de um neurônio é dado um estimulo de entrada, produzir uma propagação de saída de acordo com seu peso. O peso de um neurônio é determinado durante o treinamento da rede neural da seguinte forma: a rede é iniciada com pesos randômicos e um estimulo é propagado entre todas as camadas até a saída; uma função de erro (diferenciável) é utilizada para expor a diferença entre o valor real e o valor previsto pela rede; o erro é propagado através da rede por um método chamado backpropagation.

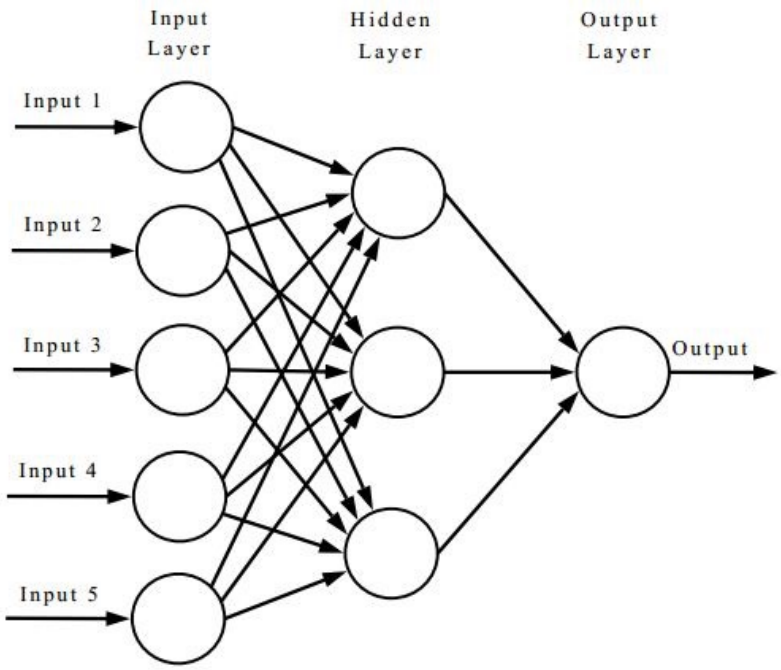

FIGURA 16. Topologia de uma Rede Neural

Backpropagation (backward propagation of errors) é um método para aprender pesos de uma rede neural e costuma ser utilizado em conjunto com um método de otimização (e.g., gradiente descendente). Após o cálculo do erro na saída da rede, o backpropagation propaga esse erro no sentido reverso (da saída para entrada) possibilitando o calculo do gradiente da função de erro com respeito a todos os pesos da rede. O papel do gradiente descendente passa então a ser a otimização dos pesos da rede em relação ao gradiente, objetivando uma minimização da função de erro. A equação VII (seção 5.2.2), com a omissão do termo precedido pelo momentum para o caso do backpropagation padrão, captura essa relação de atualização de pesos.

5.2.1.Resilient Backpropagation com Backtracking

Esta rede, baseada no pacote NeuralNet, é uma rede neural MLP do tipo feedforward usando resilient backpropagation with backtracking (rprop+). O uso de rprop+ se justifica em uma diminuição de 15% de erro em comparação ao backpropagation padrão (considerando o dataset em questão e uma rede com a mesma topologia e parâmetros).

Rprop é um esquema de aprendizagem adaptativa, realizando aprendizagem supervisionada em lotes (i.e., o ajuste dos pesos é calculado apenas depois de o gradiente de toda a rede ser computado) em redes MLP. O princípio básico do Rprop é eliminar a influência prejudicial do tamanho da derivada parcial no peso de cada passo do backpropagation. Como consequência, apenas o sinal da derivada é considerado para indicar a direção de ajuste do peso. Isto é, toda vez que a derivada parcial do peso correspondente muda de sinal (o que indica que o último ajuste foi muito grande e o algoritmo pulou um mínimo local) o peso é diminuído; no caso de a derivada manter o sinal o peso é levemente aumentado (Riedmiller 1994). Esse comportamento é capturado através da equação VI a seguir.

$$\text{(VI)}\quad \triangle_{ij}^{(t)} = \begin{cases} \eta^{+} * \triangle_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^{-} * \triangle_{ij}^{(t-1)} & , \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \triangle_{ij}^{(t-1)} & , \text{ else} \end{cases}$$

$$\text{where } 0 < \eta^{-} < 1 < \eta^{+}$$

O aumento e diminuição do peso é controlado pelos fatores η^{+} e η^{-}, que para evitar um aumento desnecessário do espaço de busca dos parâmetros de treinamento da rede foram definidos como um aumento de 20% e uma redução para a metade, respectivamente (Riedmiller 1994).

O Rprop+ adiciona backtracking ao Rprop mencionado acima. Isto é, em casos que o sinal da derivada muda, se o erro associado em $t-1$ era menor, então o peso deverá retornar ao seu valor em $t-1$.

5.2.2.Backpropagation com Momentum

Esta rede, baseada no pacote AMORE também é uma MLP do tipo feedforward (sem ciclos diretos entre nós) utilizando backpropagation com momentum.

Quando o mínimo da função de erro para uma dada tarefa de aprendizagem se encontra em um "vale" estreito, seguir a direção do gradiente pode ocasionar grandes oscilações no processo de busca (Rojas 1996).

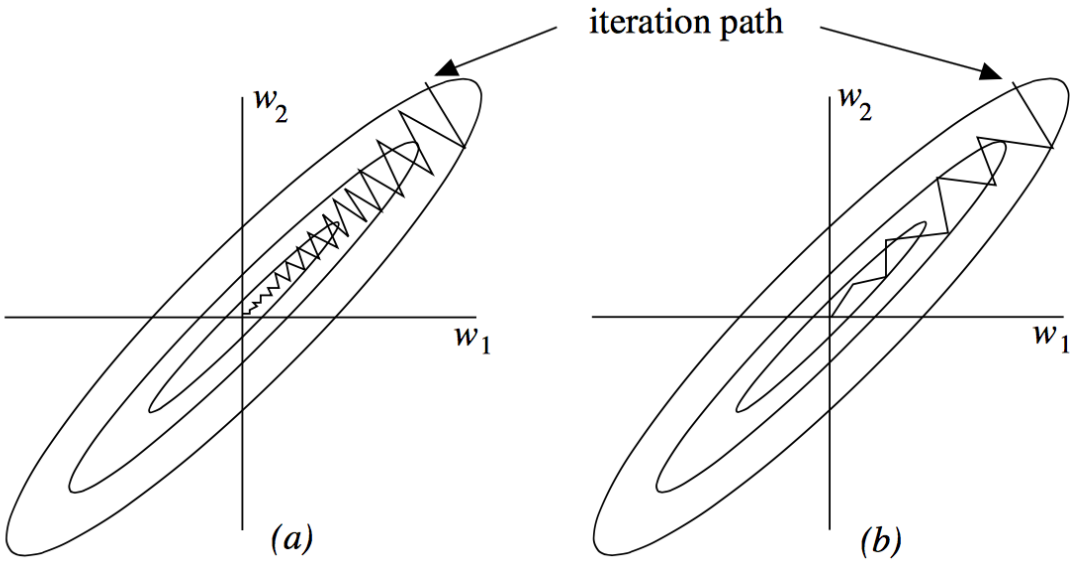

FIGURA 17. Backpropagation Sem (A) Ou Com (B) Momentum

A Figura 17 mostra um exemplo de uma rede com apenas dois pesos (w_1, w_2). A melhor estratégia nesse caso está direcionada para o centro do vale, mas a função de erro é tal que o gradiente não aponta nessa direção. Para solucionar esse problema, o termo *momentum* é introduzido.

O ajuste do peso atual é então calculado com base na média ponderada entre o gradiente atual e o ajuste do peso anterior de acordo com a equação (VII) (Rojas 1996).

(VII) $$\Delta w_k(i) = -\gamma \frac{\partial E}{\partial w_k} + \alpha \Delta w_k(i-1)$$

- γ é a taxa de aprendizagem
- α é a taxa de momentum

Teoricamente, essa abordagem deveria proporcionar um processo de busca com um tipo de inércia que poderia ajudar a evitar oscilações excessivas em vales estreitos da função de erro (Rojas 1996).

5.2.3.Parametrização

Diferentemente dos estimadores mencionados nas seções 5.1.1 e 5.1.2, a eficiência de redes neurais depende de um grande número de parâmetros (taxa de aprendizagem, critério de parada, função de ativação, taxa de momentum, etc); de maneira que encontrar a parametrização ótima para um certo conjunto de dados não é um problema trivial.

Redes neurais são normalmente treinadas utilizando métodos locais baseados em gradiente (e.g., backpropagation). Tais métodos frequentemente encontram soluções sub-ótimas quando presos em mínimos locais. Além disso, eles costumam ser lentos e experimentar várias combinações possíveis de parâmetros se torna ainda mais lento e as vezes impraticável (Rojas 1996).

Algoritmos genéticos podem também ser combinados com redes neurais (Artificial Neural Network and Genetic Algorithm - ANN/GA) para selecionar pesos e evitar mínimos locais (Duch and Korczak 1998).

Técnicas para otimizar a topologia (número de camadas e número de neurônios em cada camada) de uma rede neural são divididas em destrutivas (e.g., Pruning ou Optimal Brain Surgeon), construtivas (e.g., Cascade Correlation) e híbridas (Ragg, Braun et al. 1997).

A otimização da topologia e dos parâmetros de redes neurais tem então forte influência no desempenho da rede e dentro deste trabalho ela foi contemplada, ainda que de forma simplória.

A seguinte série de experimentos foi conduzida, variando o parâmetro de interesse em intervalos específicos e comparando os erros de validação (em um 5-fold cross validation):

1. Número de neurons na camada intermediária. Intervalo: [5, 6, 7, 8, 9, 10]
2. Taxa de Aprendizagem. Intervalo: [0.001, 0.01, 0.1]
3. Taxa de Momentum global. Intervalo: [0.001, 0.01, 0.1]

Cada experimento foi realizado com base no melhor valor observado para o parâmetro do experimento anterior. O experimento inicial utilizava taxa de aprendizagem e momentum iguais a 0.001. A Taxa de momentum foi utilizada apenas para redes que suportam esse parâmetro. A topologia da rede se resumiu a uma única camada intermediária, pois nenhum benefício foi notado ao aumentá-las manualmente.

6.Resultados

Neste capítulo será feito um comparativo entre os resultados (erro de teste, erro de validação, tempo de execução) dos estimadores mencionados no capítulo 5, e alguns Ensembles dos mesmos.

Por último será analisado o modelo final de melhor desempenho na competição.

6.1. Análise Comparativa das Técnicas Propostas

Na Tabela 4 podem ser observados os erros de validação, de teste (após submissão na competição da plataforma Kaggle) e o tempo de execução necessário para o treinamento de cada modelo.

Embora os scores pareçam próximos, é válido mencionar que mesmo as menores diferenças significaram grandes saltos (e.g., duzentas e oitenta e oito posições entre o menor e maior erro da Tabela 4) em posicionamento dentro da competição.

Um fator importante que pode passar despercebido nessa tabela é que os erros de validação são razoavelmente próximos dos erros de teste. A provável razão por trás disso é que o conjunto de testes deve ter uma distribuição próxima ao conjunto de treinamento. Isso possibilitou que modificações (feitas nos modelos de aprendizagem) baseadas em erro de validação tivessem um bom impacto no conjunto de testes.

Tabela 4 - Comparativo: Erros e Tempo de Execução

	Random Forest	**Gradient Tree Boosting**	**Ensemble (RF, GB)**	**Rede Neural (rprop+)**	**Rede Neural (Momentum)**
Erro de Validação	0.41852	0.40162	0.38787	0.39646	0.39008
Erro de Teste	0.38561	0.38219	**0.37106**	0.41244	0.39633
Tempo de Execução (Conjunto de Testes)	34s	6s	50s	2h 11min	1h 39 min
Parâmetros	#árvores =1000	#árvores =100	#árvores =100	1 camada intermediária com 10 nós; taxa de aprendizagem =0.001	1 camada intermediária com 5 nós; taxa de aprendizagem =0.001; taxa de momentum =0.001

Talvez mais relevante do que o resultado final de qual foi o melhor modelo, seja o comparativo entre os tempos de execução e os respectivos erros de teste. Random Forests e Gradient Tree Boosting proporcionaram resultados melhores que ambos os tipos de redes neurais em uma fração ínfima do tempo.

Todo o processo de construção desse trabalho foi sensivelmente penalizado pelo tempo de execução das redes neurais e no final (mesmo com extensiva, ainda que simplória, parametrização) os resultados não foram muito competitivos. É possível que melhor parametrização pudesse trazer melhores resultados, mas a lenta convergência das redes impediu que isso pudesse ser melhor explorado.

O modelo “vencedor” (erro igual à 0.37106, em negrito na Tabela 4), é um ensemble entre a Random Forest e o Gradient Tree Boosting. Este ensemble é uma combinação entre os dois modelos (através de uma Gradient Tree Boosting) da forma aproximada:

$$Y = 0.1 * RF + 0.9 * GB$$

- *Y* é o valor final previsto
- *RF* é o valor previsto pela Random Forest
- *GB* é o valor previsto pelo Gradient Tree Boosting

A decisão de se fazer um ensemble desses dois modelos (mesmo observando que seus erros eram muito próximos) se deu pela relativa distinção entre suas previsões (em inúmeros casos muito próximas, mas em alguns divergiam por uma quantidade significativa).

O modelo “vencedor" foi o octogésimo colocado entre três mil duzentos e cinquenta e dois participantes da competição.

No momento em que este documento é escrito o fechamento da competição ainda é recente e as melhores soluções ainda não foram tornadas públicas. No entanto, as redes neurais propostas neste documento apresentam resultados 30% superiores aos resultados de usuários que também utilizaram redes neurais e divulgaram seus resultados no fórum da competição.

7.Conclusão

Neste capítulo serão abordadas as contribuições deste estudo para a comunidade acadêmica, as dificuldades encontradas durante o desenvolvimento do projeto e os trabalhos que podem vir a ser desenvolvidos como continuação deste trabalho.

7.1.Contribuições

A principal contribuição deste trabalho é a detalhada investigação de técnicas de aprendizagem de máquina para resolução de um problema real (Previsão De Demanda Em Um Sistema De Compartilhamento De Bicicletas) através de: um estudo comparativo de diferentes técnicas de regressão como Random Forests, Gradient Tree Boosting, Redes Neurais, Redes Neurais Com Momentum e Ensembles; uma análise de seus desempenhos (erros de validação e teste), tempos de execução e particularidades de otimização; uma cuidadosa manipulação (criação e modificação) de variáveis.

Os códigos deste projeto estão disponíveis em repositórios abertos no CIn-UFPE[5] e GitHub[6], ambos sob licença MIT. Uma porção importante deles foi incluída no Anexo A, de forma à ilustrar o caráter construtivo além de analítico deste projeto. Os códigos foram cuidadosamente escritos para proporcionar alto desempenho e flexibilidade de utilização de diferentes estimadores (sejam eles escritos em Python ou R).

7.2.Dificuldades

Os experimentos envolvendo parametrização de redes neurais envolveram uma quantidade exorbitante de tempo (considerando cross validation). Isso poderia ter sido minimizado com uma implementação de um algoritmo de cross-validation em paralelo, mas isso provou-se uma tarefa difícil na medida em que estimadores escritos em linguagens diferentes precisavam ser suportados.

Várias técnicas (e.g., ensembles por agrupamento em uma variável categórica - capítulo 4.2) tomaram um bom tempo para serem implementadas e produziram resultados pobres.

7.3.Trabalhos Futuros

Em trabalhos futuros, uma melhor parametrização das redes neurais poderia ser empregada através de técnicas automatizadas (mencionadas no capítulo 5.2.3).

Implementações de redes neurais em GPUs (Cireşan, Meier et al. 2012) ou de outros algoritmos para cálculo do gradiente (e.g., Levenberg-Marquardt em detrimento do backpropagation) poderiam ser exploradas para dar mais velocidade ao processo de treinamento (Suratgar, Tavakoli et al. 2007).

---

[5] http://www.cin.ufpe.br/~rfna

[6] http://github.com/rafadaguiar/bike_sharing_demand

Dada a importância da variável temporal demonstrada nas figuras 10, 11, 12 e 13 do capítulo 4.2, uma análise sob o ponto de vista de séries temporais, em que uma variável dependente poderia ser determinada com base em uma função de variáveis retardadas no tempo (e.g., determinar *count* com base no valor de *casual* e *registered* das 3 horas anteriores), também poderia ser avaliada.

8.Referências

Andy Liaw, M. W. (2002). Classification and Regression by randomForest. R News. 2/3: 4.

Benkler, Y. (2004). "Sharing Nicely: On Shareable goods and the emergence of sharing as a modality of economic production." The Yale Law Journal 114.

Benoit, K. (2011). Linear Regression Models with Logarithmic Transformations, London School of Economics.

Breiman, L. (1996). "Bagging Predictors." Machine Learning 24.

Breiman, L. (2001). "Random Forests." Machine Learning Volume 45, Issue 1 , pp 5-32(Issue 1): 5.

Cireşan, D., et al. (2012). "Multi-column deep neural network for traffic sign classification." Neural Networks.

Duch, W. and J. Korczak (1998). "Optimization and global minimization methods suitable for neural networks." NEURAL COMPUTING SURVEYS.

Freund, Y. and R. E. Schapire (1996). "Experiments with a New Boosting Algorithm." Machine Learning.

Garavaglia, S. (2004). A Smart Guide toDummy Variables: Four Applications and a Macro.

Mason, L., et al. (1999). "Boosting Algorithms as Gradient Descent." Advances in Neural Information Processing Systems.

Opitz, D. and R. Maclin (1999). "Popular Ensemble Methods: An Empirical Study." Journal of Artificial Intelligence Research 11.

Ragg, T., et al. (1997). "A Comparative Study of Neural Network Optimization Techniques."

Riedmiller, M. (1994). Rprop - Description and Implementation Details, University of Karlsruhe.

Rojas, R. (1996). Fast Learning Algorithms, Springer-Verlag.

Schapire, R. E. and Y. Freund (2012). Boosting Foundations and Algorithms.

Stine, R. (2001). Logs Transformation in a Regression Equation, The Wharton School of the University of Pennsylvania.

Suratgar, A. A., et al. (2007). "Modified Levenberg-Marquardt Method for Neural Networks Training." International Journal of Computer, Electrical, Automation, Control and Information Engineering 1(7).

Susan Shaheen, S. G. (2011). Worldwide Bikesharing. Access Magazine.

Anexo A - Principais Componentes do Código de Aprendizagem

—> BEGIN README <—
This project is my first attempt in solving a real world problem using
machine learning. Here I made use of Random Forests, Gradient Tree Boosting,
Neural Networks and Ensembles to participate on the "Bikesharing Demand"
(http://www.kaggle.com/c/bike-sharing-demand) competition on Kaggle.

Author: Rafael Aguiar <rfna@cin.ufpe.br>
License: MIT

Project Source Tree:

```
.
├——— README
├——— __init__.py
├——— data_handling.py
├——— main.py
├——— ml.py
├——— optimize.py
├——— utils.py
```

—> END README <—

—> BEGIN ml.py <—

```
import copy

import numpy as np
import pandas as pd
from rpy2.robjects import pandas2ri
import rpy2.robjects as ro
import utils


class RegressionModel:
    """A regression model is composed by an estimator, its variables
    and a scaling transformation (if the dataset is scaled).
    Methods fit and predict are the same as usual, but extended to support
    multiple dependent variables, inverse scaling transformation and inverse
    log transformation.
    """
    def __init__(self, estimator, variables, scaler=None):
        self.estimator = estimator
        self.variables = variables
        self.scaler = scaler
```

```
        self.fitted_models = dict()

    def fit(self, df):
        for var in self.variables:
            estimator = copy.copy(self.estimator)
            model = estimator.fit(
                df[self.variables[var]],
                df[var]
            )
            self.fitted_models[var] = model

    def predict(self, df):
        count = 0
        for var in self.variables:
            model = self.fitted_models[var]
            prediction = model.predict(df[self.variables[var]])
            if self.scaler:
                prediction = self.scaler[var].inverse_transform(prediction)
            count += np.exp(prediction) - 1
        count = np.around(count)
        count[count < 0] = 0
        return count


class NNFromR:
    """This class acts as a wrapper handling fit and predict operations on Robjects
    (more specifically, neural networks).
    """
    class RReturn:
        """The NNFromR fit method returns an Robject, this class purpose
        is to avoid converting it to a pandas df before calling predict.
        In this way, R can get the data, fit, predit and return the result
        as a pandas df.
        """
        def __init__(self, Robject, algorithm):
            self.fitted_model = Robject
            self.algorithm = algorithm

        def predict(self, indep_vars):
            ro.globalenv['test'] = pandas2ri.py2ri(indep_vars)
            ro.globalenv['fit'] = self.fitted_model
            if self.algorithm == "rprop+":
                return pandas2ri.ri2py(
                    ro.r("compute(fit,test)$net.result")
                )
            elif self.algorithm == "ADAPTgdwm":
                return pandas2ri.ri2py(
                    ro.r("sim(fit$net, test)")
```

```
            )

    def __init__(self, **kwargs):
        self.param = kwargs
        pandas2ri.activate()
        ro.r("library(neuralnet)")
        ro.r("library(AMORE)")

    def fit(self, indep_vars, dep_var):
        ro.globalenv['train'] = pandas2ri.py2ri(indep_vars)
        ro.globalenv[dep_var.name] = pandas2ri.py2ri(dep_var)

        # Builds the parameters string
        param = utils.build_R_parameters(self.param)

        # In order to support neural networks from different packages it
        # was necessary to wrap their respective methods for the "fit" concept
        if self.param.get("algorithm") == "rprop+":
            formula = dep_var.name+"~"+"+".join(indep_vars.columns.tolist())
            ro.r("formula <- as.formula(%s)" % formula)
            return self.RReturn(
                ro.r("neuralnet(formula, data=train,%s)" % param),
                self.param.get("algorithm")
            )
        elif self.param.get("method") == "ADAPTgdwm":
            ro.r("fit <- newff(%s)" % param)
            return self.RReturn(
                ro.r(
                    "fit <- train(fit, train, %s, error.criterium='LMS',\
                    report=TRUE, show.step=1000, n.shows=100)" % dep_var.name
                ),
                self.param.get("method")
            )


class Ensemble:
    """This class provides two ways of combining the results of different
    models: averaging the response or training a new estimator (the combiner
    ) to learn the weights for each model.
    """
    def __init__(self, models, combiner):
        self.models = models
        self.combiner = combiner

    def fit(self, df, param=None):
        if len(self.models) > 1:
            predictions = pd.DataFrame()
            for key in self.models:
```

```
            self.models[key].fit(df)
            predictions[key] = self.models[key].predict(df)
        self.combiner = self.combiner.fit(predictions, df['count'])
    else:
        self.combiner = self.combiner.fit(df)

def predict(self, df):
    if len(self.models) > 1:
        predictions = pd.DataFrame()
        for key in self.models:
            prediction = np.around(self.models[key].predict(df))
            prediction[prediction < 0] = 0
            predictions[key] = prediction
    else:
        predictions = df
    return np.around(self.combiner.predict(predictions))

@staticmethod
def average_response(predictions):
    """This method provides a quick way of combining models by using the
    same weight for each.
    """
    prediction = np.around(sum(predictions)/len(predictions))
    prediction[prediction < 0] = 0
    return prediction
```

—> END ml.py <—

—> BEGIN optimize.py <—

```
from sklearn.metrics import mean_squared_error
from sklearn import cross_validation
import matplotlib.pyplot as plt
import numpy as np
import copy


def cv(data, folds, model):
    """Cross Validation using K-fold

    Parameters
    ----------
    data    data to be sliced on training and testing datasets
    k       number of folds on K-fold algorithm

    Returns
    -------
```

```
    errors  array of model errors
    models  dict including casual and registered models of lowest errors
    """
    def rmsle(predicted, actual):
        # Root Mean Squared Logarithmic Error
        return mean_squared_error(
            np.log(predicted+1),
            np.log(actual+1)
        ) ** 0.5

    errors = []
    print " Cross Validation in progress..."
    kf = cross_validation.KFold(n=len(data.index), n_folds=folds)
    for i, (train_index, validation_index) in enumerate(kf):
        print ' F%d.' % i
        train = data.iloc[train_index]
        validation = data.iloc[validation_index]

        model.fit(train)
        prediction = model.predict(validation)
        actual = data.iloc[validation_index]['count'].as_matrix()
        error = rmsle(prediction, actual)
        errors.append(error)
    return np.mean(errors)


def tune_parameters(model, param_name, param_range):
    errors = []
    min_error = float('Inf')
    min_param = None
    for param_value in param_range:
        print ">>> Param: %s <<<" % param_name
        if param_name == 'n_neurons':
            param_value = "c(ncol(train),%d,1)" % param_value
        m = copy.copy(model)
        m.estimator.param[param_name] = param_value
        error = cv(data=train_, folds=5, model=model)
        errors.append(error)
        if error < min_error:
            min_param = param_value
    return errors, min_param


def feature_worth(model, train):
    """Eases the process of testing if a new feature improves the model.
    """
    error = cv(data=train, folds=5, model=model)
    print error
```

```
    model.fit(train)
    for var in model.variables:
        print var
        print model.fitted_models[var].feature_importances_
```

—> END optimize.py <—