
Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

# Identificação de Páginas de Anúncio de Automóvel

Samuel Paz Mendes

Trabalho de Graduação

Recife
Novembro de 2017


Universidade Federal de Pernambuco
Centro de Informática

# Identificação de Páginas de Anúncio de Automóvel

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Luciano de Andrade Barbosa

Recife
Novembro de 2017

# Agradecimentos

Em primeiro lugar gostaria de agradecer a meus pais por terem superado as condições que tiveram em suas vidas para prover educação a meus irmãos e a mim. Vocês sempre foram minha fonte de inspiração maior.

Ao professor Luciano de Andrade Barbosa, por todo tempo e esforço dedicado a orientação neste trabalho.

Ao professor de matemática Ednaldo Ernesto, que despertou meu interesse pelas ciências exatas.

A todos docentes, que fazem do Centro de Informática um centro de excelência.

A todos funcionários do Centro de Informática, que criam os meios para as atividades acadêmicas.

Ao programa Ciência sem Fronteiras, que me proporcionou oportunidades acadêmicas e pessoais incríveis no exterior.

A toda comunidade da University of California Santa Barbara, que foi parte primordial na minha educação.

Ao projeto Maratona de Programação, que transformou minha forma de pensar.

Aos amigos que fiz durante a graduação.

Aos alunos do Centro de Informática, que servem de inspiração pela dedicação ao curso apesar de todas dificuldades.

A todos brasileiros, que sustentam o ensino público apesar de todas dificuldades econômicas enfrentadas no seu dia-a-dia.

*Are you prepared to face your destiny?*
—The Oracle, Tibia

# Resumo

O crescimento da internet modificou a maneira com que as pessoas compram e vendem automóveis. Muitos sites fornecem, em páginas web, especificações estruturadas – com preço, ano de fabricação, marca, entre outros – sobre automóveis que estão à venda. Esse tipo de dado é importante, pois pode ser usado em diversas aplicações (*Data Integration*, *Web Search*, entre outras). A tarefa de identificar computacionalmente especificações estruturadas de automóveis não é simples, pois, existem bilhões de páginas web em toda internet, e muitas delas não possuem esse tipo de informação. Este trabalho se propõe a estudar e aplicar os métodos de classificação de texto para construir uma solução que identifique especificações estruturadas de automóveis em um conjunto restrito de domínios web na Internet.

**Palavras-chave:** páginas web, especificação estruturada, identificação, classificação de texto.

# Abstract

The growth of internet modified the way people buy and sell vehicles. Many websites provide structured vehicles specifications - price, year of manufacture, brand and others – for vehicles that are on sale. This type of data can be used in many applications (Data Integration, Web Extraction and others). The task of identifying structured vehicles specifications is not straightforward because there are billions of webpages in the internet that does not match their description. This work proposes to study and apply text classification methods to construct a solution that identifies vehicles structured vehicles specifications in a restricted set of sites on the Internet.

**Keywords:** webpages, structured specifications, identification, text classification.

# Sumário

# Lista de figuras

# Lista de tabelas

*Capítulo 1*

# Introdução

A especificação de produtos em formato de dados estruturados é um tipo de informação bastante presente na Internet. Uma especificação de produto é um conjunto de par (atributo e valor) e também é conhecida como entidade [3]. Esse tipo de informação é grande importância, pois, a partir da identificação, extração e armazenamento desses dados é possível se construir soluções para diversos problemas como: *Demand Forecasting*, recomendação de produtos, *Web Search* [1, 3]. O processo de identificar e categorizar esses dados não é simples, pois, a *World Wide Web* é formada por um gigante repositório de informações com páginas de diversos formatos e temas. Esse repositório, para se ter uma ideia de sua dimensão, possuía cerca de 4,66 bilhões de páginas online em março de 2016 [1, 2].

As páginas de especificação de automóveis que buscamos estão contidas em domínios web de anúncios. Para identificar essas especificações será necessário realizar diversas etapas de classificação de texto. Nesse contexto, o processo de identificação de páginas web pode ser disposto nessas etapas: definição do tema do produto; definição dos domínios web; coleta e rotulação da base de dados; pré-processamento da base de dados; aplicação de métodos de processamento de *features* para criação de diversas configurações de bases de dados; escolha e treinamento de modelos de classificação; definição de métricas para escolha dos melhores modelos de classificação com posterior fase de avaliação.

*Capítulo 2*

# Fundamentos

Nesse capítulo será explicado os conceitos fundamentais utilizados durante o projeto. Esses conceitos são importantes pois constroem uma base de informações para o entendimento da solução proposta. Serão analisados, respectivamente, os seguintes fundamentos: aprendizado supervisionado, algoritmos de classificação, pré-processamento de dados, *feature selection*, *feature extraction,* métodos de ensemble e auto machine learning.

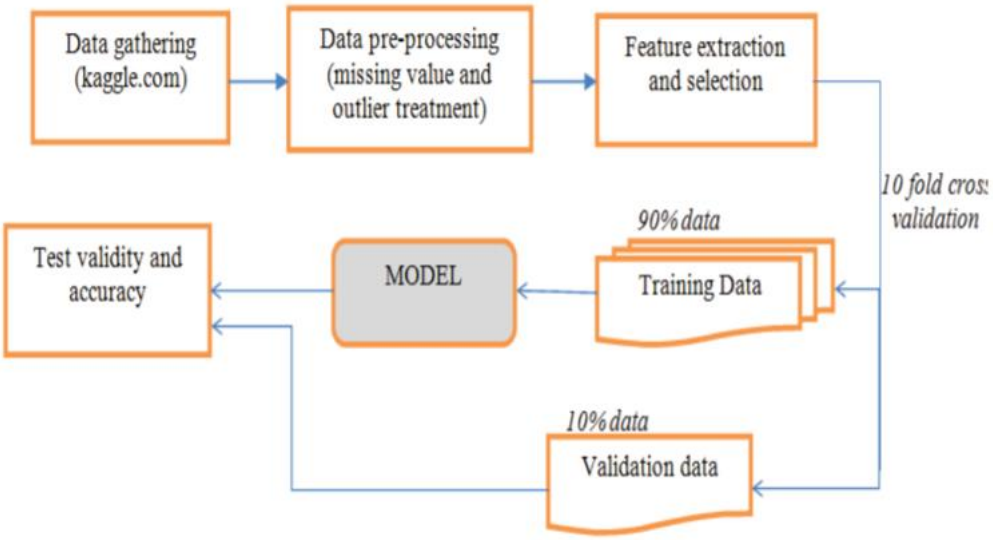
## 2.1 Aprendizado Supervisionado

Aprendizado supervisionado é uma área abrangente que, através de algoritmos, constrói padrões e hipóteses sobre instâncias de um problema a partir do aprendizado de um estimador sobre um conjunto anterior de instâncias [6]. Existem dois tipos principais de estimadores em aprendizado supervisionado: regressor e classificador. A regressão é fundamentada na estimativa de valores reais dado um padrão de instância [7]. A classificação é articulada em prever a *label* (ou classe) de uma instância não-rotulada a partir *features* (características) da mesma.

Ambos tipos de estimadores (classificador e regressor) precisam de uma base de dados de instâncias do problema, chamada de conjunto de treinamento, para aprender e criar uma função estimadora. Cada instância é composta de uma serie de *features*, que representam a característica da mesma. Em classificação, as instâncias de treinamento possuem *features* e sua respectiva *label*. Já em regressão, as instâncias de treinamento possuem *feature*s e o valor real de saída esperado para cada instância [7].

O resultado final do aprendizado supervisionado é o modelo de estimador. Para gerar esse modelo é necessário seguir uma série de etapas: coleta e rotulação da base de dados, pré-processamento de dados, *feature selection* e *feature extraction*, separação da base em conjunto de treinamento e conjunto de testes, treinamento do modelo de classificador, testes de validação do modelo [6]. A figura **Figura 1** mostra a série de passos em aprendizado supervisionado. Apesar desses passos representarem uma sequência, a aprendizagem supervisionada tem um aspecto cíclico pois cada passo poderá ser refeito com outros parâmetros e configurações para se alcançar um resultado melhor nos testes. É comum, por exemplo: aumentar e diversificar a base de dados, usar novas estratégias de *features*, escolher novas configurações de classificadores e etc.

**Figura 1:** Passos do aprendizado supervisionado




**Fonte:** [6]

O processo de aprendizagem supervisionada é dependente do problema que está sendo resolvido. Há tipos de problemas que são mais facilmente estimados que outros, isso depende muito da diversidade dos dados no mundo real do problema e da eficácia que o pesquisador deseja para seu estimador.

Nesse trabalho iremos focar em aprendizagem supervisionada para classificação de textos. Em classificação de texto, a instância do classificador pode ser chamada de documento. Essa área é bastante importante, pois a quantidade documentos de texto digitais aumentou muito nos últimos anos [1]. No entanto, existem alguns desafios principais para classificação de texto, podemos citar:

- Alta dimensionalidade dos dados, que implica em respostas menos deterministas do classificador. Ela ocorre quando o número de *features* na base de dados é maior de que o número de documentos [4].
- Escolha de *features* independentes e que forneçam informação adicional a base de dados.
- Escolha do melhor algoritmo. Essa escolha pode ser bastante onerosa pois é preciso testar diversos algoritmos e configurações para encontrar o mais adequado para uma solução.
- Coleta e rotulação manual de uma base de dados grande e diversificada. Esses processos podem requerer bastante tempo do projeto pois são etapas manuais e que precisam de muitas instâncias de documentos.

## 2.4 Algoritmos de Classificação

O algoritmo de classificação junto com o conjunto de treinamento é responsável por criar o modelo de classificação que categorizará novas instâncias de documentos. Escolher esse algoritmo é uma das etapas mais importantes em classificação de texto e demanda esforço e compreensão para definir as formas de teste e métricas que serão usadas para definir o melhor algoritmo para um dado problema.

Existem dezenas de classificadores, cada um deles pode aprender algum tipo de informação que outros talvez não consigam. Essa abundância é muito positiva, pois testar diversos algoritmos fortalece a definição de um algoritmo final que servirá como solução para o problema. É possível separar os algoritmos disponíveis em categorias [7]:

- Árvore de decisão. São árvores em que cada nó representa uma *feature*, e cada aresta representa um valor possível para a *feature* analisada na classificação. Um documento de texto começa sua classificação no nó-pai até chegar a uma folha, que dirá a categoria do documento.
- *Rule-based.* São formados por expressões booleanas em que cada variável representa a satisfação ou não de uma *feature*. A expressão total é analisada em cima do documento e a sua classe é o resultado da avaliação.
- *Statistical learning.* São baseados em modelos probabilísticos que fornecem a probabilidade de um documento pertencer a alguma classe. Essa probabilidade é então é comparada a algum limiar para o classificador definir a classe proposta para o documento.
- *Instance-based.* Esse tipo de algoritmo não procura aprender alguma generalização de padrão do conjunto de treinamento, ao invés disso ele compara as instâncias testadas com as instâncias já armazenadas no conjunto de treinamento. O documento testado é comparado através de métricas de distância entre documentos.
- *Support Vector Machines*. Cria uma representação em que cada documento é um ponto do espaço de *feature*s. A partir dessa representação o algoritmo tenta encontrar funções matemáticas que separem os conjuntos de pontos através de hiperplanos, cada subseção de conjunto separado faz parte de uma classe específica.

A tabela **Tabela 1** mostra uma relação entre a categoria de classificadores e algoritmos populares nessas categorias.

**Tabela 1:** Categoria e lista de algoritmos

| **Categoria de classificador** | **Algoritmos de exemplo** |
|---|---|
| Árvore de decisão | *Random Tree, Random Forest* (tipo de bagging em árvores de decisão), REPTree, *DecisionStump, Hoeffding Tree*, LMT, J48 |
| *Rule-based* | *DecisionTable* |
| *Statistical learning* | *Naive Bayes, Bayes Network, LogitBoost, SimpleLogistic* |
| *Instance-based* | *k-Nearest Neighbour* (KNN) |
| *Support vector machines* | SVM |

**Fonte**: Elaborada pelo autor.

## 2.2 Pré-processamento

Em aprendizado supervisionado, o pré-processamento de base de dados consiste na limpeza, normalização, integração, discretização e transformação de informações irrelevantes, corrompidas, imprecisas ou em formatos diferentes [8]. Esse processo precede os processos de *feature selection* e *feature extration [4]* e é fundamental para a construção do modelo de classificador, pois as inconsistências em dados não pré-processados geram problemas com a acurácia do estimador. O resultado final do pré-processamento de dados é composto do conjunto de treinamento e testes.

Existem diversos métodos de pré-processamento de dados. No caso de classificação de texto, podemos usar stopwords, stemming, técnicas para preencher valores faltantes, entre outras. Um exemplo típico para páginas web é remover as tags HTML.

## 2.3 Feature Selection e Feature Extraction

Em classificação de texto, um documento de texto pode ser interpretado como um conjunto de *features*. Essas *features* são características que, de alguma forma, fornecem informação para se descrever e categorizar algum documento. Ao se criar um classificador, precisamos dessas *features* para dizer sobre quais aspectos do documento o classificador deve analisar e raciocinar. As *features* podem ser contínuas, de categoria ou binárias [7].

*Feature extraction* e *feature selection* são processos que criam, manipulam e removem as *features* de uma base de documentos para tentar minimizar alguns problemas recorrentes em classificação de texto: (1) a alta dimensionalidade dos dados (número de *features* maior de que número de documentos), que deprecia a acurácia final do classificador; (2) *features* irrelevantes que estão presentes na base de dados sem que forneçam informação adicional suficiente em relação às outras *features* da base de dados.

*Feature extraction* é definida como o processo de gerar novas *features* a partir de *features* já presentes nos documentos. A ideia dessa técnica é que as novas *features* representem melhor os padrões das instâncias de documentos. Podemos realizar *feature extraction* através de métodos matemáticos ou a através da observação e raciocínio de um especialista sobre as características que podem ser derivadas a partir das características já presentes no conjunto de documentos. Alguns métodos matemáticos para *feature* extraction incluem:

- *Principal Component Analysis*. Determina os N principais componentes da base de dados para representar os documentos [4].
- *Latent Semantic Indexing*. Baseada na aplicação de decomposição de valores singulares para reduzir o número de dimensões da base [4].
- TF-IDF. Busca relativizar os valores das *features* através dos conceitos: quanto mais vezes um termo aparece no documento, maior sua importância; quanto mais vezes um termo aparece em documentos, menor sua importância. O TF-IDF é a mistura da frequência do termo (TF) e do inverso da frequência nos documentos (IDF) [11]. Existem diversas propostas de fórmulas para o TF-IDF, a figura **Figura 2** mostra uma das formas mais simples de ser implementada: o "w" representa a *feature*; o "tf" representa a frequência da *feature* no documento; o "N" representa o número total de documentos; o "n" representa o número de documentos com ocorrência da *feature*.

**Figura 2:** TF-IDF

$$w_{ij} = tf_{ij} \times idf_i = tf_{ij} \times \log\left(\frac{N}{n_i} + 0.01\right)$$

**Fonte:** [11]

*Feature selection* é definida como processo de selecionar parte das *features* do conjunto total de features dos documentos e descartar o restante. A ideia dessa técnica é encontrar as *features* que realmente são importantes para a base de dados. Essa seleção pode acontecer com o uso de métodos matemáticos ou através da observação de especialistas sobre as *features* disponíveis na base. Algumas técnicas matemáticas para *feature selection* são [4]:

- *Information Gain* (IG). É a medida de quanto quanta informação adicional uma *feature* provê em relação as outras *features* da base de dados. O ganho de IG é calculado através do cálculo da diferença de entropia entre o nó-pai e os nós-filhos de uma árvore, onde cada nó é um subconjunto de documentos da base e os nós-filhos são um uma divisão do nó-pai a partir de um critério. A ideia do IG é calcular seu valor através do ganho da diferença de entropia entre pai e filhos na árvore, isso funciona porque as *features* mais importantes vão aumentar essa diferença de entropia mais de que *features* menos relevantes. A partir do cálculo de IG, as *features* com ganho muito baixo ou nulo são removidas da base de documentos. A fórmula na figura **Figura 3** mostra um exemplo de fórmula de Information Gain que usa a probabilidade de presença de um termo em uma determinada classe e documento.

**Figura 3:** Information Gain

$$IG(t) = -\sum_{i=1}^{M} P(C_i) \log p(C_i) + p(t) + \sum_{i=1}^{M} p\left(\frac{C_i}{t}\right) \log p\left(\frac{C_i}{t}\right) + p(\bar{t}) \sum_{i=1}^{M} p(C_i/\bar{t}) log\, p(C_i/\bar{t}) \quad (1)$$

**Fonte:** [4]

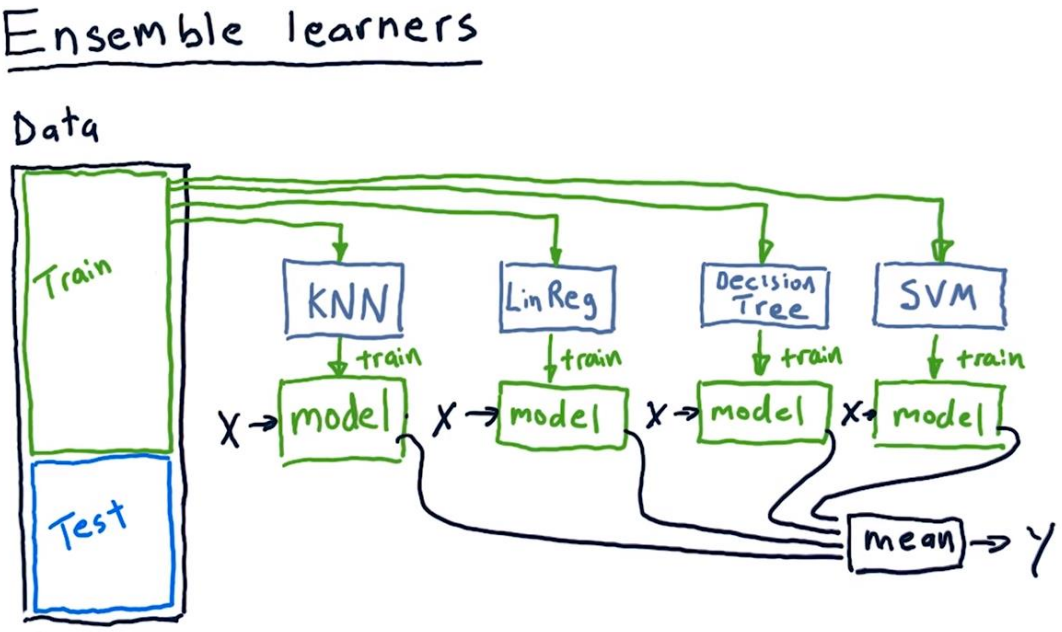
- Frequência do termo. Remove *features* que não aparecem na base de dados mais do que um limiar determinado. A ideia é que essas *features* não são importantes para o conjunto total de dados.
- Qui-quadrado. Usa o cálculo de independência entre entre eventos (onde um evento é a classe e o outro é o termo) para encontrar *features* importantes.

## 2.5 Ensemble

*Ensemble* é o processo de combinar diversos classificadores para gerar um método geral que usa as qualidades individuais de cada classificador. A figura **Figura 4** mostra o fluxo básico de

métodos de *ensemble*. Existem diversas técnicas de ensemble, iremos cobrir três delas: *bootstrap aggregation* (*bagging*), *boosting e stacked generalization* (*stacking*). A característica principal dos três (e de outros) é que cada um terá alguns classificadores-base que treinará sobre um subconjunto da base de treinamento, a resposta de cada classificador de base então será usada como conjunto de treinamento em um meta-classificador, responsável por unir e integrar as respostas para produzir uma resposta mais correta. Esse processo traz diversidade para o aprendizado supervisionado. Os classificadores-base e meta-classificadores podem ser instâncias de classificadores conhecidos (por exemplo: *KNN*, *NaiveBayes*, *Random Tree*...) ou até mesmo classificadores *ensemble*. É claro que deve haver moderação na escolha dos classificadores-base e meta-classificadores, pois problema como *overfiting* podem ocorrer e nem sempre "mais é melhor".

**Figura 4:** Exemplo de ensemble




**Fonte:** https://www.youtube.com/watch?v=Un9zObFjBH0

- *Bootstrap aggregation (*bagging*)* cria sub-bases randomicamente e as usa para treinar classificadores-base independentes. Cada classificador-base dirá a qual categoria o documento pertence, a resposta final será a categoria que foi mais foi dada como resposta. Essa é a definição conservadora de *bagging*, mas também é possível usar meta-classicadores mais complexos que esse sistema de voto, o próprio WEKA fornece essa opção.
- *Stacked generalization* (*stacking*) também cria sub-bases como o bagging, mas a ideia é que sejam criados N subconjuntos de base em que o complemento da união de quaisquer (N-1) subconjuntos seja o subconjunto faltante. São formados N classificadores-base, cada um treinado em uma base diferente em que seu conjunto de treinamento é formado pela união dos (N-1) subconjuntos. A resposta de cada classificador base é, então, analisada por um classificador-meta e a resposta final é gerada [9].
- O *boosting* cria n classificadores que serão treinados em um processo iterativo de escolha de bases de treinamento. Para o caso de 3 classificadores temos as seguintes bases respectivas: a base do primeiro classificador será todo conjunto de treinamento; a base do segundo classificador será uma junção de 50% dos documentos classificados corretamente e 50% dos documentos classificados incorretamente pelo primeiro

classificador; a base do último classificador é formada pelos documentos que os primeiros dois classificadores chegaram a categorias diferentes. O resultado dos três classificadores é processado por um sistema de votos para decidir o resultado final [9], mas é possível usar um meta-classificador mais complexo. O algoritmo de *boosting* é bastante eficiente e alguns métodos de ensemble, como o AdaBoost, são especializações dele.

Ensembles, como todos outros tipos de classificadores, possuem pontos positivos e negativos. As justificativas para usar um ensemble são variadas: (1) combinar respostas vários modelos diminui o risco de usar um classificador com resultados ruins para alguns padrões de documentos; (2) bases de documentos muito grandes em classificadores não-escaláveis geram tempo de treinamento exorbitante; (3) bases de dados pequenas podem gerar sub-bases randômicas para cada classificador-meta e projetar boas respostas; (4) dividir-e-conquistar é, geralmente, uma boa opção, pois classificadores normais podem não ser suficientes para solucionar um problema; (5) bases de documentos com features incompatíveis entre si podem ser usadas, cada meta-classificador cuidaria de uma base e a resposta final do meta-classificador seria a união de informação de todas bases [9]. Os pontos negativos de ensembles geralmente são: overfitting, tempo de treinamento grande em comparação a um classificador comum e crescente em relação ao número de classificadores-base, incerteza em relação a que combinação de classificadores-base e meta-classificadores usar.

## 2.6 Auto Machine Learning

Definir um modelo de classificação em aprendizagem supervisionada pode ser um processo penoso e demorado dependendo da dificuldade do problema a ser resolvido. Muitas vezes, é preciso testas diversos parâmetros, modelos e técnicas de feature selection e extraction para encontrar um modelo satisfatório. A ideia de *auto machine learning* (automl) é proporcionar uma ferramenta que execute automaticamente essas opções de configuração de aprendizado de máquina. Cada configuração é testada e métricas são usadas para definir as melhores. O processo de automl pode ser bem mais custoso (em relação aos recursos de tempo e poder computacional) de que o aprendizado de máquina convencional, mas ele pode gerar resultados muito eficientes e tem a vantagem de ser automatizado.

Existem diversas instâncias de automl, a maioria delas usa hiper-parametrização com seleção de modelos e algoritmos de features como base. Alguns exemplos são: Auto-Weka (escrito em Java), Auto-sklearn (escrito em Python), TPOT (wrapper do Auto-sklearn escrito em python), H2O AutoML (com interface em R, Python, Java).

*Capítulo 3*

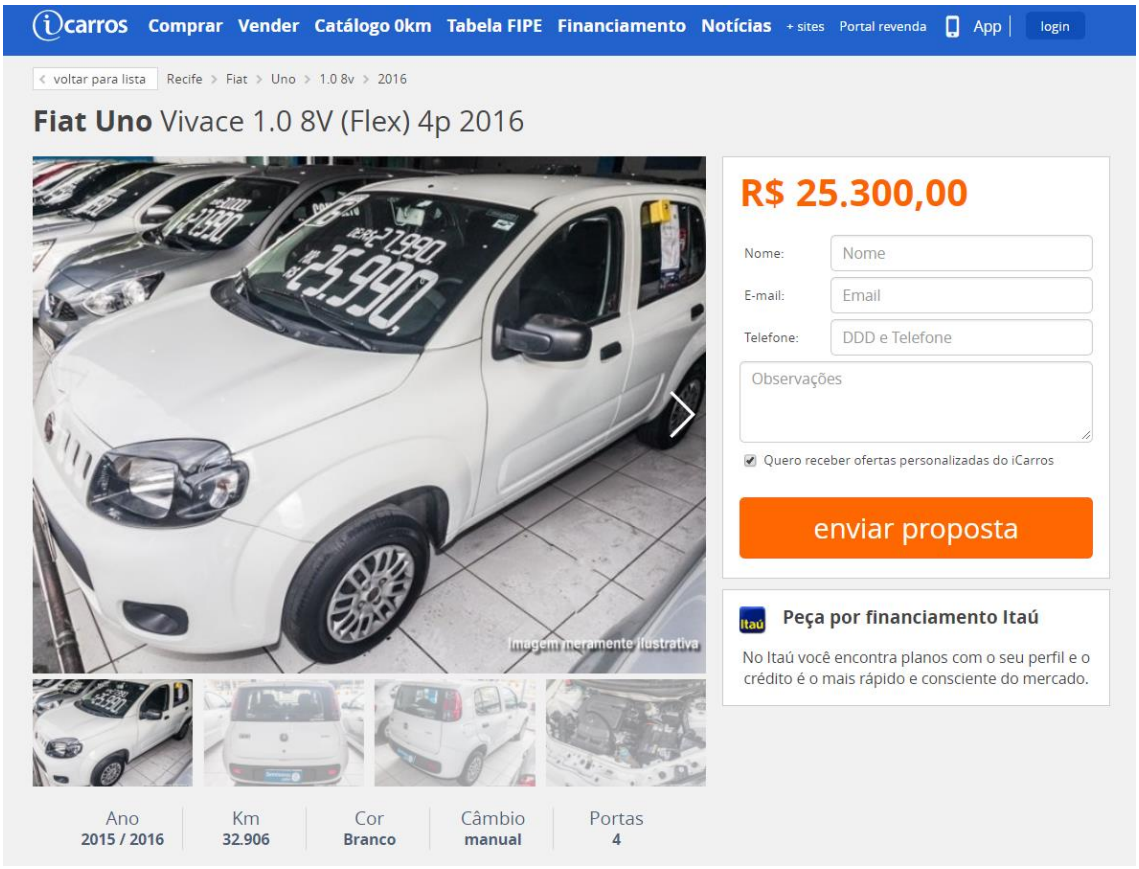
# Solução proposta

Nesse capítulo daremos o contexto geral da solução para detectar páginas de venda de automóveis juntamente com o resumo das principais bibliotecas usadas durante a construção da solução. As bibliotecas são: Jsoup, WEKA e Auto-WEKA.

## 3.1 Detecção de páginas de venda de automóveis

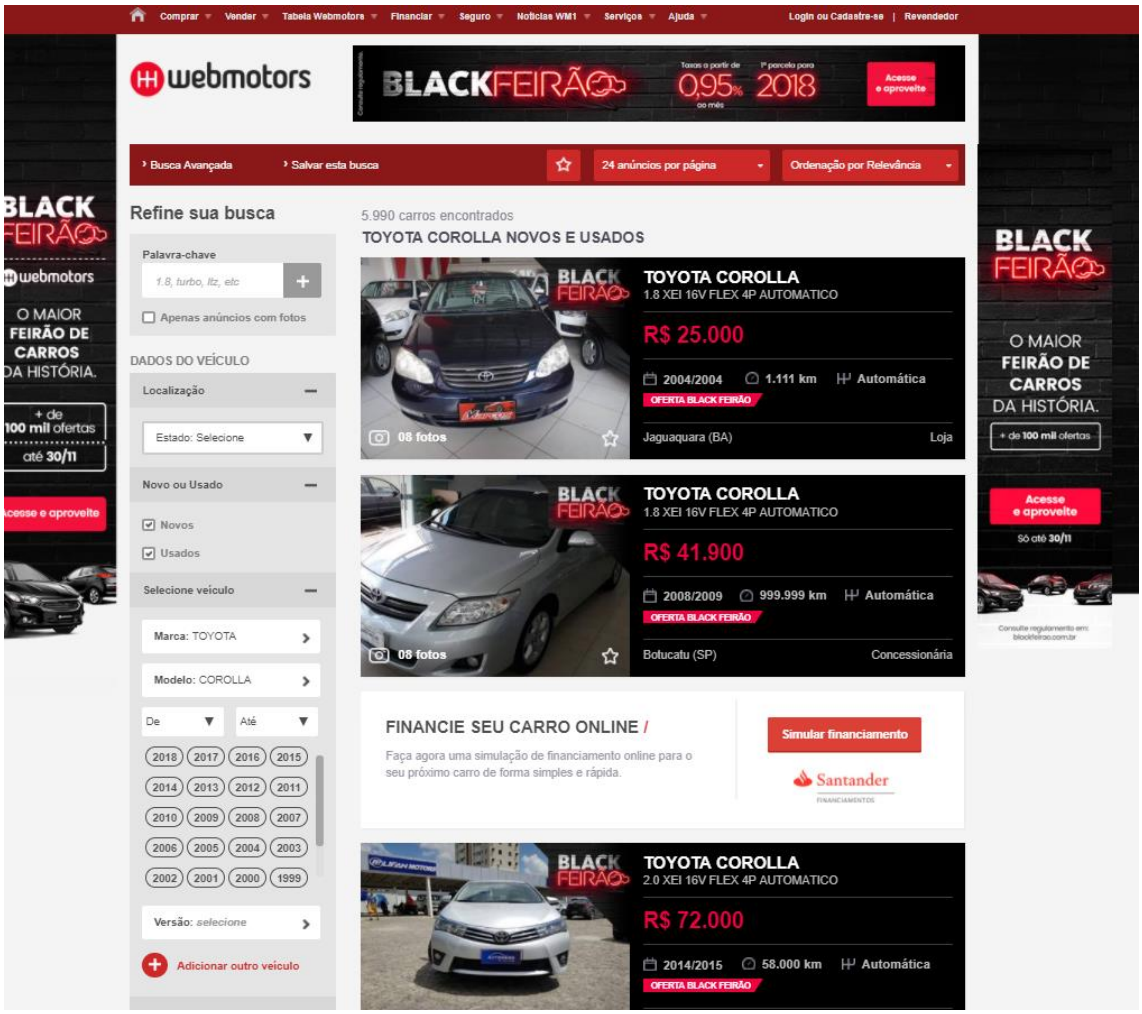
Páginas de vendas de automóveis tem uma aparência bem característica para um detector humano, nossa solução tem como objetivo usar aprendizado supervisionado para criar um detector artificial especializado nesse tipo de páginas. As páginas de vendas de automóveis são páginas de anúncio que mostram informação sobre a venda de apenas um automóvel, esse anuncio pode conter informações sobre o automóvel como ano, modelo, preço, quilometragem, entre outras. Separamos os tipos de páginas presentes na Internet em dois grupos: o primeiro, chamado de “positivo” representa as páginas de venda de automóveis; o segundo, chamado de “negativo” representa todos outros tipos de páginas. A figura **Figura 5** representa uma página positiva, e as figuras **Figura 6** e **Figura 7** representam páginas negativas.

**Figura 5:** Página positiva



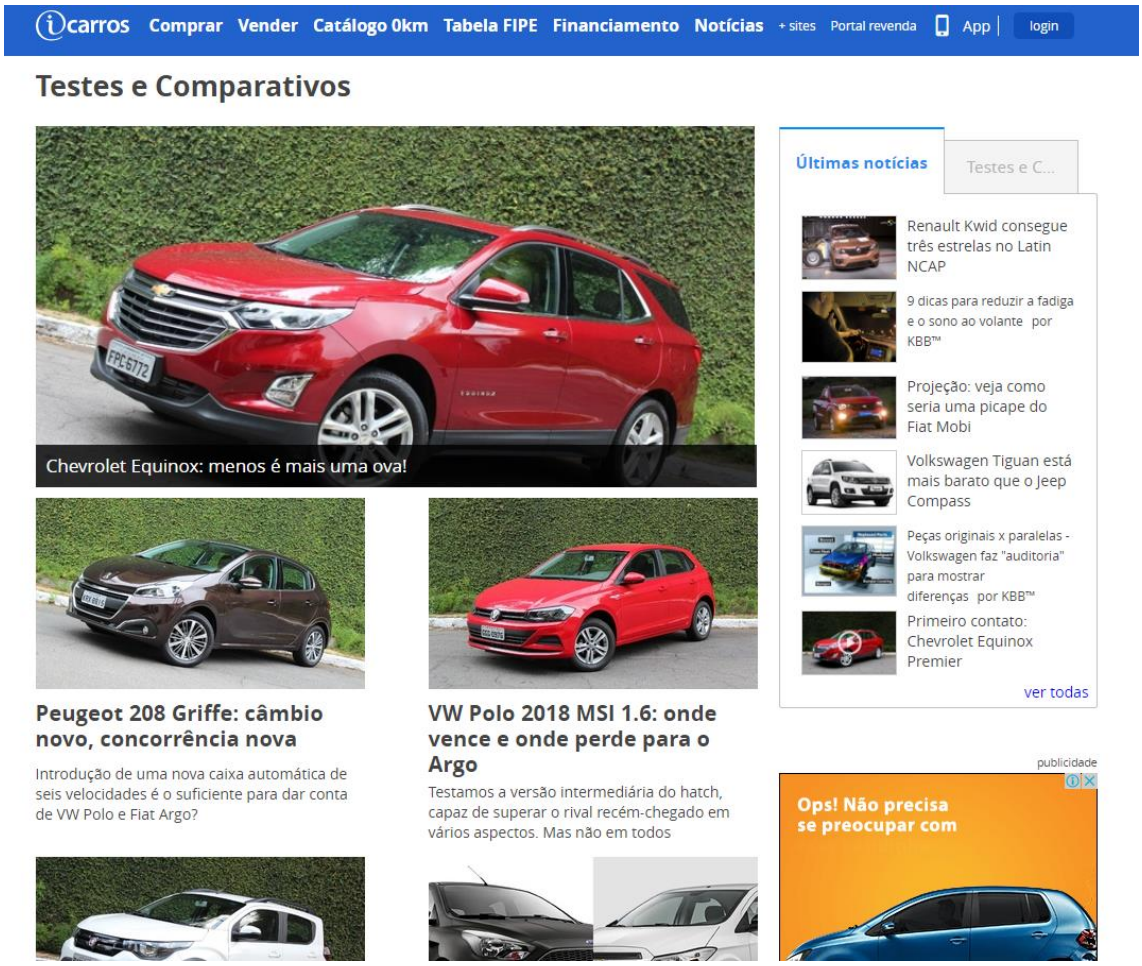

**Fonte:** icarros.com.br

**Figura 6:** Página negativa 1




**Fonte:** www.webmotors.com.br

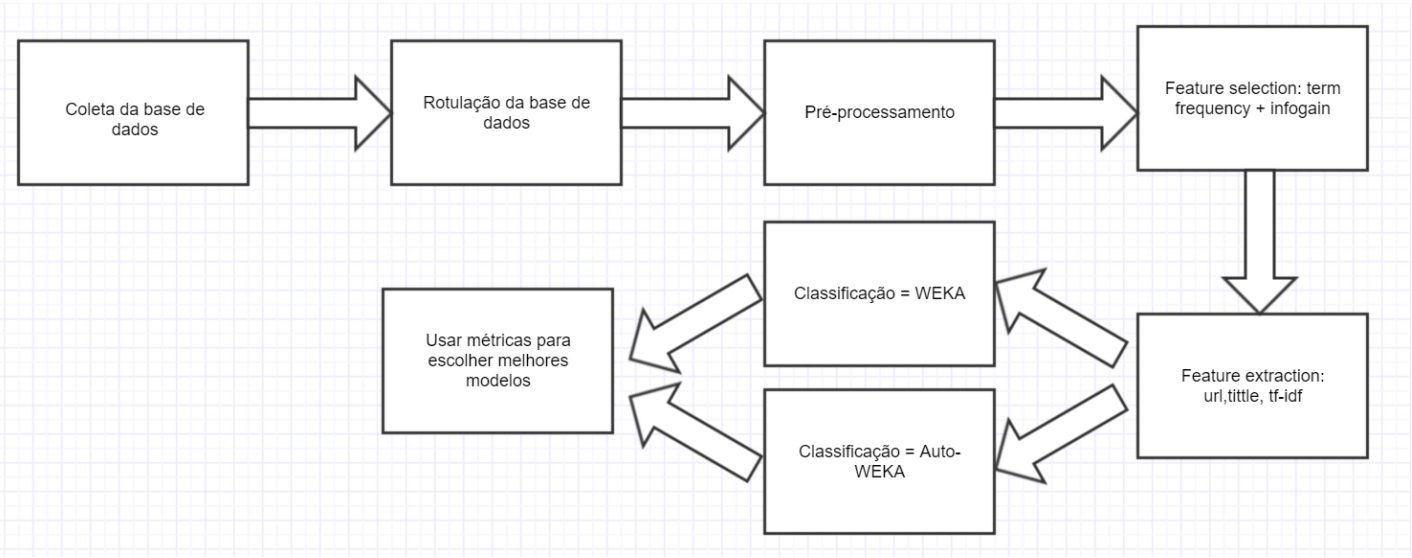
**Figura 7:** Página negativa 2




**Fonte:** icarros.com.br

O objetivo do trabalho é a criação de um modelo de classificação capaz de detectar distinguir páginas positivas (anúncio estruturado de veículos) de páginas negativas (outros tipos de páginas). A solução em si é construída nas seguintes etapas: coleta e rotulação da base de documentos, pré-processamento de dados, *feature selection* e *feature extraction*, escolha do algoritmo de classificação (com WEKA e Auto-WEKA) através da análise de métricas fornecidas pelo WEKA e Auto-WEKA. A figura **Figura 8** mostra o contexto geral da solução. Todas essas fases obedecem uma ordem pré-definida, mas cada uma delas foi eventualmente reformulada de forma que o processo representasse um ciclo de desenvolvimento.

**Figura 8:** Fluxo solução




**Fonte:** Elaborada pelo autor.

## 3.2 Jsoup

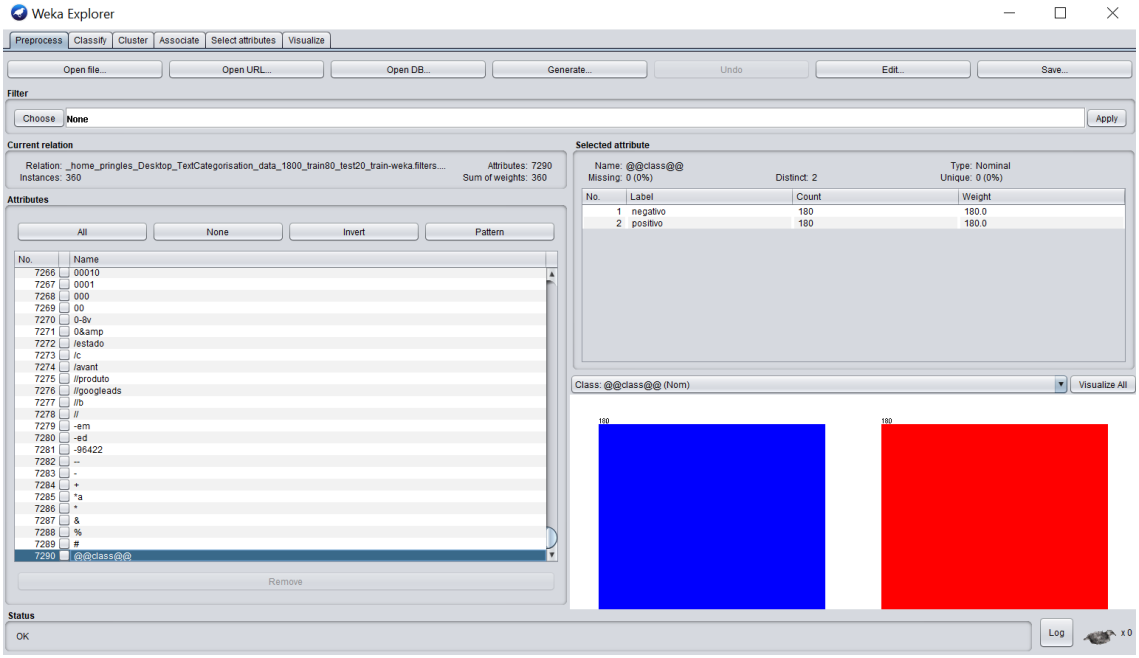
Jsoup é uma biblioteca escrita em Java capaz de baixar, manipular e extrair dados de documentos HTML. A biblioteca pode usar métodos que envolvem DOM, CSS, jquery e praticamente todas variedades de tecnologias que envolvem HTML. Jsoup é open-source e é distribuído através da licença MIT. A sua lista de operações possíveis é grande, alguns exemplos são:

1. *Parse* de documentos para *strings* e vice-versa.
2. Métodos DOM para navegar em documentos.
3. Seletores baseado em sintaxe para achar elementos.
4. Extração de atributos, textos e HTML de elementos.
5. Manipulação do valor de atributos e elementos.
6. Limpeza de documentos para prevenir *cross-site scripting*.

## 3.3 WEKA

O Waikato Environment for Knowledge Analysis (WEKA) é um pacote de ferramentas de machine learning escrito em Java. A disponibilidade de recursos é enorme: pré-processamento de dados, processamento de *features*, *clusters*, visualização de dados, modelos de classificação e suporte com interface GUI. Os acessos ao WEKA são feitos pela interface GUI (a figura **Figura 9** mostra a interface GUI do WEKA), pela linha de comando ou programaticamente.

**Figura 9:** Explorer da Interface GUI do WEKA




**Fonte:** Elaborada pelo autor

O conjunto de documentos no WEKA é representado no formato *Attribute Relation File-Format* (ARFF). Um ARFF é um arquivo de texto com um cabeçalho e uma lista de instância de documentos. O cabeçalho é composto de relação e uma lista de atributos; os atributos podem ser numéricos, nominais, *strings*, *datas*. Um ARFF pode ser importado de outro formato, criado de pastas com documentos (onde o nome da pasta é a classe do documento) ou formado programaticamente pelo usuário através da própria biblioteca WEKA.

O WEKA possui o recurso de filtros de manipulação de dados, cada filtro tem um propósito específico e são usados para representar, excluir e derivar informações da base. Os filtros são de extrema importância pois através deles conseguimos formatar a base de dados e até integrar ela com outras bases externas. Após manipular os dados é possível os salvar em um ARFF.

O WEKA possui dezenas de classificadores, a variedade de tipos de algoritmos é enorme: árvores de decisão, *rule-based* ,*perceptron-based* ,*statistical learning*, *instance-based*, *support vector machines*, *ensemble methods*...

O WEKA é capaz de testar modelos de classificação e fornece várias métricas para analisar os resultados, a figura **Figura 10** mostra um exemplo de métricas a partir de um resultado de classificação do WEKA.

**Figura 10:** Métricas dos resultados WEKA

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         320               88.8889 %
Incorrectly Classified Instances        40               11.1111 %
Kappa statistic                          0.7778
Mean absolute error                      0.1108
Root mean squared error                  0.332
Relative absolute error                 22.1641 %
Root relative squared error             66.394  %
Total Number of Instances              360

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,867    0,089    0,907      0,867    0,886      0,779    0,937     0,909     negativo
               0,911    0,133    0,872      0,911    0,891      0,779    0,922     0,875     positivo
Weighted Avg.  0,889    0,111    0,890      0,889    0,889      0,779    0,930     0,892

=== Confusion Matrix ===

   a   b   <-- classified as
 156  24 |   a = negativo
  16 164 |   b = positivo
```

**Fonte:** Elaborada pelo autor.

## 3.4 Auto-WEKA

O Auto-WEKA é uma ferramenta de auto machine learning usada para encontrar automaticamente as melhores configurações de classificadores fornecidos pelo WEKA. Ele usa hiper-parametrização com cross-validation sobre todos classificadores do WEKA: 27 classificadores base, 2 ensembles methods e 10 meta-methods [5], a figura **Figura 11** mostra esses classificadores. Hiper-parametrização é um método de tentativa e erro, e por isso pode levar bastante tempo para encontrar a melhor configuração possível. O próprio Auto-WEKA, em sua documentação, avisa que devem ser executadas milhares de configurações na hiper-parametriziação. A função importante do Auto-WEKA é que a hiper-parametrização é feita de maneira automática, com apenas uma configuração inicial feita pelo usuário. Essa configuração tem opções de diversos parâmetros do Auto-WEKA, dois parâmetros importantes são o tempo aproximado de execução e número de threads.

**Figura 11:** Classificadores do Auto-Weka: todos presentes no WEKA

**Table 1: Classifiers in Auto-WEKA: Classifiers marked with * are meta-methods, which in addition to their own parameters take one 'base' classifier and its parameters. Classifiers marked with + are ensemble methods that take as input up to 5 'base' classifiers and their parameters. *Categorical* and *Numeric* refer to the number of hyperparameters of each kind for each classifier.**

| Classifier | Categorical | Numeric |
|---|---|---|
| Bayes Net | 2 | 0 |
| Naive Bayes | 2 | 0 |
| Naive Bayes Multinomial | 0 | 0 |
| Gaussian Process | 3 | 6 |
| Linear Regression | 2 | 1 |
| Logistic Regression | 0 | 1 |
| Single-Layer Perceptron | 5 | 2 |
| Stochastic Gradient Descent | 3 | 2 |
| SVM | 4 | 6 |
| Simple Linear Regression | 0 | 0 |
| Simple Logistic Regression | 2 | 1 |
| Voted Perceptron | 1 | 2 |
| KNN | 4 | 1 |
| K-Star | 2 | 1 |
| Decision Table | 4 | 0 |
| RIPPER | 3 | 1 |
| M5 Rules | 3 | 1 |
| 1-R | 0 | 1 |
| PART | 2 | 2 |
| 0-R | 0 | 0 |
| Decision Stump | 0 | 0 |
| C4.5 Decision Tree | 6 | 2 |
| Logistic Model Tree | 5 | 2 |
| M5 Tree | 3 | 1 |
| Random Forest | 2 | 3 |
| Random Tree | 4 | 4 |
| REP Tree | 2 | 3 |
| Locally Weighted Learning* | 3 | 0 |
| AdaBoost M1* | 2 | 2 |
| Additive Regression* | 1 | 2 |
| Attribute Selected* | 2 | 0 |
| Bagging* | 1 | 2 |
| Classification via Regression* | 0 | 0 |
| LogitBoost* | 4 | 4 |
| MultiClass Classifier* | 3 | 0 |
| Random Committee* | 0 | 1 |
| Random Subspace* | 0 | 2 |
| Voting+ | 1 | 0 |
| Stacking+ | 0 | 0 |

**Fonte:** [5]

*Capítulo 4*

# Avaliação da Solução

Nesse capítulo iremos apresentar a formação da solução a partir da coleta da base de dados até os resultados apresentados pelos classificadores. O processo de coleta, rotulação, pré-processamento, processamento de *features* será dado inicialmente. O processo de criação de classificadores será dividido em 3 seções: classificadores do WEKA, Auto-WEKA e analise do melhor classificador a partir de um crawler automático.

Para formar a base de documentos usamos um coletor de páginas web, esse coletor foi alimentado com urls de páginas HTML. Foram coletadas 1736 páginas de 30 domínios web diferentes. Essa base de dados foi pré-processada de forma que as tags HTML fossem descartadas e em sequência foi separada em uma base de treinamento e uma base de testes. A base de treinamento é composta das páginas de 90% dos domínios web e a base de testes é composta das páginas de 10% dos domínios conforme indicado na tabela **Tabela 2**. Isso implica que os domínios usados no treinamento não estão sendo utilizados nos testes e vice-versa. A base de treinamento possui 1376 páginas (79% do total de páginas) e a de testes 360 (21% do total de páginas).

**Tabela 2:** Domínios de treinamento e teste

| Domínios na base de treinamento | Domínios na base de teste |
|---|---|
| www.autoline.com.br | www.meucarango.com.br |
| www.autoshow.com.br | www.meucarronovo.com.br |
| carro.mercadolivre.com.br | olx.com.br |
| www.carrosnaweb.com.br | primeiramao.band.com.br |
| www.compreauto.com.br | www.compreblindados.com.br |
| www.vrum.com.br | carros-saopaulo-zc.temusados.com.br |
| salaodocarro.com.br | |
| www.webclassicos.com.br | |
| www.webmotors.com.br | |
| www.vivalocal.com | |
| www.itavema.com.br | |
| www.icarros.com.br | |
| www.lugardecomprarcarro.com.br | |
| www.meucadillac.com | |
| seminovositaliana.com.br | |
| www.classificadoscb.com.br | |
| www.litoralcar.com.br | |
| www.autoshoppingcuritiba.com.br | |
| www.gncseminovos.com.br | |
| www.boulevardshoppingcar.com.br | |
| www.bariguiseminovos.com.br | |

| | |
|---|---|
| classificados.atarde.uol.com.br | |
| www.nacionalveiculos.com.br | |
| www.usadosbr.com | |

**Fonte:** Elaborada pelo autor.

Ao criar a base de teste e treinamento, foi observado que o conjunto de treinamento possuía 33599 *features*. Esse número era extremamente alto e gerava alta dimensionalidade na base de dados pois só tínhamos 1376 instâncias de treinamento. A alta dimensionalidade dos dados tornou necessário executar a próxima etapa de classificação de textos: *feature selection* e *feature extraction.* Essas técnicas foram aplicadas de maneira incremental e permitiram aumentar a consistência dos dados através da criação de 3 configurações diferentes de base. O objetivo de ter diversas configurações de base é diversificar a base de dados para que os algoritmos de classificação possam se adequar melhor com alguma das configurações disponíveis.

Abaixo listamos o processo incremental de aplicação de *feature selection.*

1. O primeiro método de *feature selection* usado foi o de frequência do termo. Definimos e usamos um limiar de frequência mínima 5 em todas configurações de base. Essa técnica reduziu o número de *features* de 33599 para 10365.
2. O algoritmo de *Information Gain* do WEKA foi usado para determinar *features* de ganho nulo. Essas *features* com ganho nulo foram identificadas em formato de lista e usamos o filtro Remove do WEKA para exclui-las da base de documentos. No total, 3075 *features* com ganho nulo e foram excluídas da base de dados. Essa mudança fez com que sobrassem apenas 7290 *features* das 10365 selecionadas pela frequência do termo.

Abaixo listamos o processo incremental de aplicação de *feature extraction:*

1. Através da observação das páginas de anúncio de automóveis podemos notar que a url e o título da página fornecem bastante informação. Incluir essa informação na base de dados através de novas *features* se tornou o primeiro passo de *feature extraction*. No caso do título da página, essa informação já estava presente dentro dos documentos como *features*, mas elas não eram identificadas diferentemente das *features* extraídas do texto das páginas. Isso tornava a informação de título menos relevante, pois um termo que aparecia no título poderia ser confundido (pelo classificador) com um termo do texto geral da página. No caso da URL da página, essa informação não estava presente nos documentos como *features*. Para tratar ambos casos, criamos as tags "#url" e "#tittle" para que todas palavras chaves da url ou título pudessem ser identificadas diferentemente nos documentos. Alguns exemplos de palavras-chave são "vendo", "carro", "reais". Com esse processo, o número de *features* aumentou para 7833.
2. As *features* presentes na base de dados apresentavam como valor apenas a frequência daquele termo na instância. Esse método de representação poderia se tornar um problema porque a diversidade de páginas de anúncios é muito grande, e uma páginas com frequências muito diferentes poderiam prejudicar a base. Era preciso usar algum método que assinalasse melhor o valor de uma *feature*. Para isso, implementamos a versão do tf-idf mostrado na figura **Figura 2**. O resultado da aplicação dessa técnica manteve o número de *features* e documentos na base, apenas mudando o valor de cada *feature* nas instâncias.

A **Tabela 3** mostra, em detalhes, como ficou cada configuração de base. Essas configurações de base serão usadas nos processos de criação de classificadores.

**Tabela 3:** Configuração de base e processos

| Configuração de OUTRACOISA | Processos | Features | Treinamento | Testes |
|---|---|---|---|---|
| Base 1 | Frequência do termo, Information Gain | 7290 | 1376 | 360 |
| Base 2 | Frequência do termo, Information Gain, URL e título da página | 7833 | 1376 | 360 |
| Base 3 | Frequência do termo, Information Gain, URL e título da página, tf-idf | 7833 | 1376 | 360 |

**Fonte:** Elaborada pelo autor

O próximo etapa da classificação de textos é a criação dos modelos de classificação, mas para analisar esses modelos é preciso definir os métodos de avaliação. Existem diversos métodos de separações da base de dados para avaliar a performance de classificadores, nesse estudo consideramos importantes duas delas:

- *Cross-validation*, onde se particiona os dados em subconjuntos disjuntos entre si e os separa entre conjuntos de treinamento e teste em diversas iterações
- *Train-test split*, onde o conjunto de treinamento é dividido em treinamento e teste sobre uma porcentagem pré-determinada (normalmente, a porcentagem de treinamento é maior) [7].

Além da definição dos métodos de separação da base, também foi preciso definir as principais métricas usadas no estudo. O WEKA e Auto-WEKA fornecem diversas métricas de teste (como citado no capítulo 2), mas as mais utilizadas nesse trabalho foram:

1. Precisão. É o número de verdadeiros positivos dividido pela soma de verdadeiros positivos mais falsos positivos. A figura **Figura 12** mostra sua fórmula.

**Figura 12:** Precisão

$$\text{Precision} = \frac{tp}{tp + fp}$$

**Fonte:** Wikipedia.com

2. Cobertura. É o número de verdadeiros positivos dividido pela soma de verdadeiros positivos mais falsos negativos. A figura **Figura 13** mostra sua fórmula.

**Figura 13:** Cobertura

$$\text{Recall} = \frac{tp}{tp + fn}$$

**Fonte:** Wikipedia.com

3. F-measure. É uma média harmônica da precisão e cobertura. É importante pois combina as duas medidas. A figura **Figura 14** mostra sua fórmula.

**Figura 14:** F-measure

$$F = 2 \cdot \frac{\mathrm{precis} \cdot \mathrm{revoc}}{\mathrm{precis} + \mathrm{revoc}}$$

**Fonte:** Wikipedia.com

O método de *train-test split* foi usado para analisar os 18 algoritmos de classificação treinados no WEKA, essa estratégia foi adotada porque isso nos permite separar os domínios das páginas web. O Auto-WEKA funciona, internamente, usando *cross-validation* para avaliar os classificadores treinados, mas a técnica de *train-test split* também foi usada para avaliar a eficiência dos melhores classificadores.

# Criação de modelos de classificação no WEKA

Nessa etapa da solução iremos aplicar classificadores nativos do WEKA sobre as configurações de bases definidas. Foram treinados 18 classificadores envolvendo Árvores de decisão, *Rule-based Methods, Statistical learning*, *Instance-based Methods, Support Vector Machines* e *Ensemble Methods.* Cada classificador foi executado com a configuração inicial recomendada pelo WEKA ou por variações manuais do autor. Os classificadores foram testados em todas 3 configurações de bases e o resultado das métricas será apresentado para apenas os 5 melhores classificadores de cada configuração. Os classificadores treinados estão presentes na tabela **Tabela 4.**

**Tabela 4:** Classificadores treinados no WEKA

| Classificador | Configurações |
|---|---|
| Naive Bayes | Default do WEKA |
| BayesNet | Default do WEKA |
| SVM | Default do WEKA |
| SimpleLogistic | Default do WEKA |
| AdaBoostM1 (ensemble) | Iterações: 20, 30, 40, 50, 200 |
| Random Forest | Default do WEKA |

| | |
|---|---|
| SGD | Default do WEKA |
| DecisionTable | Default do WEKA |
| LogitBoost | Default do WEKA |
| REPTree | Default do WEKA |
| J48 | Default do WEKA |
| Ibk (KNN) | Default do WEKA |
| Stacking (ensemble) | "meta-classificador = Naive Bayes, classificadores-base = Hoeffding Tree, J48, RandomTree, REPTree, LMT, DecisionStump, RandomForest" |
| Bagging (ensemble) | Iterações: 10, 20, 30 |
| Hoeffding Tree | Default do WEKA |
| Decision Stump | Default do WEKA |
| LMT | Default do WEKA |
| Random Tree | Default do WEKA |

**Fonte:** Elaborada pelo autor.

As tabelas **Tabela 5**, **Tabela 6** e **Tabela 7** detalham os resultados das avaliações para cada base de dados a partir do WEKA. Através de todos processos do ciclo de etapas em classificação de texto conseguimos encontrar o melhor modelo para resolver o problema de categorização de páginas web de anúncios de carros. O modelo escolhido foi gerado pela combinação da base 3 com o algoritmo AdaBoostM1 de 30 iterações, esse modelo atingiu aproximadamente precisão, cobertura e f-measure (respectivamente) de 91%, 91% e 91%.

**Tabela 5:** Resultado para base 1

| Algoritmo | Precisão | Cobertura | F-measure |
|---|---|---|---|
| BayesNet | 0,822 | 0,775 | 0,766 |
| ADABoostM1 | 0,853 | 0,811 | 0,805 |
| LogitBoost | 0,783 | 0,783 | 0,783 |
| DecisionStump | 0,804 | 0,708 | 0,683 |
| LMT | 0,709 | 0,675 | 0,661 |

**Fonte:** Elaborada pelo autor

**Tabela 6:** Resultado para base 2

| Algoritmo | Precisão | Cobertura | F-measure |
|---|---|---|---|

| | | | |
|---|---|---|---|
| BayesNet | 0,829 | 0,781 | 0,772 |
| ADABoostM1 | 0,845 | 0,819 | 0,816 |
| RandomForest | 0,803 | 0,747 | 0,735 |
| Stacking | 0,825 | 0,800 | 0,796 |
| LMT | 0,744 | 0,733 | 0,730 |

**Fonte:** Elaborada pelo autor

**Tabela 7:** Resultado para base 3

| **Algoritmo** | **Precisão** | **Cobertura** | **F-measure** |
|---|---|---|---|
| AdaBoostM1 | 0,909 | 0,908 | 0,908 |
| RandomForest | 0,858 | 0,808 | 0,801 |
| REPTree | 0,852 | 0,847 | 0,847 |
| Stacking | 0,897 | 0,862 | 0,863 |
| Bagging | 0,862 | 0,858 | 0,858 |

**Fonte:** Elaborada pelo autor

# Criação de modelos de classificação no Auto-WEKA

O Auto-WEKA é uma ferramenta que se propõe a testar milhares de configurações de modelos de classificação sobre uma base. Executar o Auto-WEKA é uma tarefa que exige muito tempo de execução e, por isso, foi necessário escolher uma configuração de base de dados para realizar os testes. No projeto, decidimos escolher a configuração de base 3 pois com ela que conseguimos, anteriormente, os melhores resultados no WEKA. Em primeiro lugar, comentaremos o processo de execução do Auto-WEKA e depois mostraremos os seus resultados.

O setup inicial do Auto-WEKA fornece a possibilidade de definir diversos parâmetros de execução, como por exemplo: tempo total de execução, forma de avaliação dos resultados, memória RAM usada no processo, número de threads, forçar treinamento em *batchs*, e etc [12]. Decidimos usar memória de 80 gigabytes, 5 threads e tempo de execuções de 1 dia, 5 dias, 20 dias. A princípio, os resultados obtidos com essas configurações não foram satisfatórios. Uma explicação para isso é o baixo número de configurações de modelos treinados pelo Auto-WEKA. A quantidade máxima de modelos treinados foi 367 em 20 dias de execução do Auto-WEKA, esse número é baixo, pois o próprio Auto-WEKA recomenda que sejam treinados milhares de modelos de classificação. Os resultados de cada execução do Auto-WEKA estão presentes na tabela **Tabela 8**, onde destacamos: os parâmetros de configuração, número de modelos treinados e melhor algoritmo de classificação encontrado com sua respectiva acurácia, cobertura e f-measure. Na tabela **Tabela 9** destacamos as configurações dos 3 melhores modelos encontrados

**Tabela 8:** Parâmetros de execução do Auto-WEKA

| Execução | Parâmetros da execução | Número de modelos | Melhor modelo | Acurácia | Cobertura | F-measure |
|---|---|---|---|---|---|---|
| 1 | 1 dias, 5 threads, 80 gibabytes RAM | 146 | Logistic | 0.733 | 0.708 | 0.700 |
| 2 | 5 dias, 5 threads, 80 gibabytes RAM | 280 | SMO | 0.634 | 0.633 | 0.633 |
| 3 | 20 dias, 5 threads, 80 gibabytes RAM | 337 | SMO | 0.634 | 0.633 | 0.633 |

**Fonte:** Elaborado pelo autor.

**Tabela 9:** Configuração dos melhores modelos no Auto-Weka

| Execução | Melhor modelo | Configuração do modelo |
|---|---|---|
| 1 | Logistic | {"-R", "3.4262002091923836"} |
| 2 | SMO | {"-U", "2", "-A", "weka.core.neighboursearch.LinearNNSearch", "-W", "weka.classifiers.functions.SMO", "--", "-C", "1.3181976994061202", "-N", "0", "-M", "-K", "weka.classifiers.functions.supportVector.PolyKernel -E 1.8939516765304818"} |
| 3 | SMO | {"-U", "2", "-A", "weka.core.neighboursearch.LinearNNSearch", "-W", "weka.classifiers.functions.SMO", "--", "-C", "1.3181976994061202", "-N", "0", "-M", "-K", "weka.classifiers.functions.supportVector.PolyKernel -E 1.8939516765304818"} |

**Fonte:** Elaborado pelo autor.

O resultado da segunda e terceira execução foi igual, enquanto o da primeira foi o melhor apesar de ser a execução do Auto-WEKA com menor tempo de execução. Isso pode ser

explicado pelo Auto-WEKA não ser uma ferramenta determinística e tentar configurações de classificador diferentes para os parâmetros de configuração testados. Os resultados em si foram bem abaixo dos resultados da execução manual do WEKA, isso talvez se dê pelo fato de que no WEKA treinamos diversos classificadores complexos (como, por exemplo, Stacking ou AdaBoostM1) que talvez não tenham sido treinados durante o tempo de execução do Auto-WEKA. A reflexão é que seria necessário bem mais tempo de execução (talvez, vários meses) para encontrar um modelo adequado para esse problema usando o Auto-WEKA.

*Capítulo 5*

# Conclusão

Neste trabalho apresentamos o ciclo de desenvolvimento aplicado para se criar uma solução eficiente para o problema de classificação páginas web de anúncio de vendas de carros na internet. A solução gerada nesse processo foi possível graças ao acúmulo de técnicas de classificação de texto, foram usados métodos de pré-processamento, *feature extraction*, *feature selection*, modelos de classificação e de avaliação de testes. O resultado final do trabalho então se consagra como o processo de escolha de cada método em classificação de texto, e o modelo final escolhido como o melhor para resolver o problema de classificação de páginas web com anúncio de automóvel. O modelo de classificação final apresentou 91% de acurácia de acordo com a base de testes, e, portanto, é eficaz na solução do problema apresentado no trabalho.

Os resultados encontrados pelo trabalho foram eficientes e o processo de criação garantiu muito aprendizado para o autor, foram necessárias muitas horas estudo para entender as adversidades em classificação de texto e as técnicas para amenizar esses problemas. Os trabalhos futuros nessa área poderiam usar este como base, podendo ter um aumento da diversidade da base usada e mais tempo disponível para aplicar técnicas de *auto machine learning* para que resultados melhores fossem encontrados. Outro aspecto considerado nessa conclusão é o poder dos métodos *ensemble* em classificação de texto, o uso dessa técnica favoreceu muito o resultado final do projeto e deve ser aplicado em outros problemas similares. Como palavra final sobre o projeto, posso afirmar que a experiência de aprendizado foi muito valorosa e que será levada em toda minha vida.

# Referências


[1] M. H. Rao, D. R. Sashikumar, "A Survey on Automated Web Data Extraction Techniques for Product Specification from E-commerce Web Sites," in International Journal of Advanced Research in Computer Science and Software Engineering, vol 6, August 2016.

[2] S. Pappas, "How Big Is the Internet, Really?" 2016. Disponível em https://www.livescience.com/54094-how-big-is-the-internet.html, acesso em: 05 Setembro 2017.

[3] D. Qiu, L. Barbosa, X. L. Dong, Y. Shen, and D. Srivastava, "Dexter: Large-scale discovery and extraction of product specifications on the web," in Proceedings VLDB Endowment, vol. 8, pp. 2194– 2205, September 2015.

[4] Foram P. Shah, Vibha Patel, "A Review on Feature Selection and Feature Extraction for Text Classification," in 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), (Chennai, India), March 2016.

[5] Chris Thornton, Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms," in Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, (Chicago, Illinois, USA), pp. 847-855, August 2013.

[6] Amanpreet Singh, Narina Thakur, Aakanksha Sharma ,"A Review of Supervised Machine Learning Algorithms, " in Computing for Sustainable Global Development (INDIACom), March, 2016.

[7] S. B. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," in Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies (Slovenia), vol 31, pp. 3-24, 2017.

[8] S. B. Kotsiantis, D. Kanellopoulos, P. E. Pintelas, "Data Preprocessing for Supervised Leaning" in International Journal of Computer Science, vol 1, pp. 111-117, January 2006.

[9] R. Polikar, "Ensemble based systems in decision making," in IEEE Circuits and Systems Magazine, Vol 6, pp. 21-45, September 2006

[10] Alex Smola, S.V.N. Vishwanathan, "Introduction to Machine Learning," in Press Sindicate of The University of Cambridge, October 2010

[11] Aizhang Guo, Tao Yang, "Research and Improvement of feature words weight based on TFIDF Algorithm," in Information Technology, Networking, Electronic and Automation Control Conference (Chongqing, China), May 2016.

[12] Lars Kotthoff, Chris Thornton, Frank Hutter, "User Guide for Auto-WEKA version 2.6," in http://www.cs.ubc.ca/labs/beta/Projects/autoweka/, July 2017.