Universidade Federal do Rio de Janeiro

Inteligência Computacional

Felipe Schreiber Fernandes - 116206990
Matheus Guimarães de Moura - 118106706

22 de Dezembro de 2019

Conteúdo

1 INTRODUÇÃO

1.1 DESCRIÇÃO DO PROBLEMA

Possuir a capacidade de predição do custo de um imóvel é um problema recorrente no mercado mobiliário e é de grande interesse tanto para os compradores quanto para os vendedores. O problema em questão é um problema de classificação que tem por objetivo classificar, por meio de suas características, se um determinado imóvel tem um custo caro ou barato.

2 DATASET E TECNOLOGIA

2.1 DESCRIÇÃO DOS DADOS

Os dados foram disponibilizados no site Kaggle, com informações sobre casas na cidade de Ames que fica no estado de Iowa, nos Estados Unidos. São 80 variáveis, dividas em 44 categóricas e 36 numéricas, com 2919 registros.

Para objetivo da disciplina, foi sugerido trabalhar apenas com as variáveis numéricas, portanto, o dataset terá apenas 36 variáveis. A Tabela 1 mostra o nome das variáveis e o que elas representam.

Variável	**Descrição**
MSSubClass	A classe de construção
LotFrontage	Tamanho da calçada em frente ao imóvel
LotArea	Tamanho do terreno
OverallQual	Qualidade geral do imóvel
OverallCond	Condições gerais do imóvel
YearBuilt	Ano de construção
YearRemodAdd	Ano de remodelagem
MasVnrArea	Tamanho do folheado de alvenaria
BsmtFinSF1	Tamanho do acabamento do primeiro porão
BsmtFinSF2	Tamanho do acabamento do segundo porão
BsmtUnfSF	Tamanho inacabado do porão
TotalBsmtSF	Tamanho total do porão
1stFlrSF	Tamanho do primeiro andar
2ndFlrSF	Tamanho do segundo andar
LowQualFinSF	Tamanho do acabamento de todos os andares
GrLivArea	Área construída
BsmtFullBath	Quantidade de banheiros no porão
BsmtHalfBath	Quantidade de lavabos no porão
FullBath	Quantidade de banheiros
HalfBath	Quantidade de lavabos
BedroomAbvGr	Quantidade de quartos
KitchenAbvGr	Quantidade de cozinhas
TotRmsAbvGrd	Quantidade de aposentos excluindo os banheiros
Fireplaces	Quantidade de lareiras
GarageYrBlt	Ano de construção da garagem
GarageCars	Quantidade de carros que cabem na garagem
GarageArea	Tamanho da garagem
WoodDeckSF	Área do deck
OpenPorchSF	Área aberta na varanda
EnclosedPorch	Área fechada na varanda
3SsnPorch	Área da varanda de três estações
ScreenPorch	Área da varanda com tela

PoolArea	Tamanho da piscina
MiscVal	Valor das demais variaveis não consideradas
YrSold	Ano de venda
SalePrice	Valor da venda

Tabela 1: Descrição das variáveis utilizadas

2.2 APRESENTAÇÃO DA TECNOLOGIA

Para o desenvolvimento do trabalho, foi utilizado o Jupyter Notebook com a linguagem Python na versão Python 3, para analisar e visualizar de dados e modelar e avaliar os modelos. Essa tecnologia apresenta diversas vantagens. É uma linguagem de uso livre, de fácil implementação e com muitas bibliotecas prontas para essa área de machine learning.

Apesar da existência de algumas bibliotecas na linguagem, foram escolhidos outras para um melhor resultado no projeto. Sendo elas, Pandas, Matplotlib, Numpy, Seaborn, Scikitlearn e Cvxopt.

2.2.1 PANDAS

Biblioteca para leitura e manipulação de dados. Oferece estruturas de dados e operações para manipular tabelas numéricas e séries temporais.

2.2.2 MATPLOTLIB

Biblioteca para plotagem e visualização de dados.

2.2.3 NUMPY

Biblioteca que suporta arrays e matrizes multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com estas estruturas.

2.2.4 SEABORN

Biblioteca de visualização de dados baseado no matplotlib. Ele fornece uma interface de alto nível para desenhar gráficos estatísticos atraentes e informativos.

2.2.5 SCIKITLEARN

Biblioteca de aprendizado de máquina de código aberto que inclui vários algoritmos de classificação, regressão e agrupamento, dentre outros.

2.2.6 CVXOPT

Biblioteca de otimização convexa que usaremos para obter os parâmetros da rede RBF a partir de um problema de programação quadrática.

3 METODOLOGIA

3.1 APRESENTAÇÃO DA SOLUÇÃO DO PROBLEMA PROPOSTO

Cada modelo escolhido será treinado e validado pelo método de validação cruzada k-folds, com k=10. Este método consiste em dividir a base de dados em k partes, onde k-1 partes são utilizadas para o treinamento e uma serve como base de testes. O processo é repetido k vezes, de forma que cada parte seja usada uma vez como conjunto de testes e o restante como conjunto de treino. Ao final, a correção total é calculada pela média dos resultados obtidos em cada etapa, bem como a sua variância, obtendo-se assim uma estimativa da qualidade do modelo de conhecimento gerado e permitindo análises estatísticas.

Dessa forma, resolvemos o problema de sobre-ajuste que ocorre quando um modelo se ajusta muito bem ao

conjunto de dados anteriormente observado, mas se mostra ineficaz para prever novos resultados.

A partir da construção de cada modelo é necessário verificar sua adequação ao problema. Portanto, para cada modelo utilizado neste trabalho, analisaremos os resultados da execução através das métricas e matriz de confusão. Sendo essas métricas a Acurácia, Precisão, Recall, F1-score e o AUC.

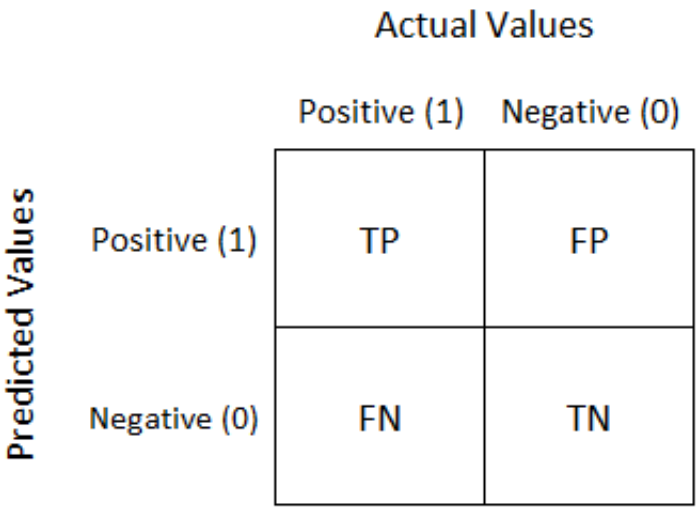


Figura 1: Matriz de Confusão

3.1.1 ACURÁCIA

A acurácia indica uma performance geral do modelo. É basicamente o número de acertos dividido pelo número total de exemplos. Ela deve ser usada em datasets com a mesma proporção de exemplos para cada classe, e quando as penalidades de acerto e erro para cada classe forem as mesmas.

Em problemas com classes desproporcionais, ela causa uma falsa impressão de bom desempenho. Por exemplo, num dataset em que 80% dos exemplos pertençam a uma classe, só de classificar todos os exemplos naquela classe já se atinge uma precisão de 80%, mesmo que todos os exemplos da outra classe estejam classificados incorretamente.

$$Acuracia = \frac{VP+VN}{Total}$$

3.1.2 PRECISÃO

A precisão é a probabilidade de, dado que o modelo classificou como determinada classe, quantos de fato pertencem a ela. O cálculo é feito pela razão entre a quantidade de exemplos classificados como pertencentes a uma classe que realmente são daquela classe (verdadeiros positivos - VP - considerando a classe em questão como como positiva e as demais negativas), dividido pela soma entre este número (VP), e o número de exemplos classificados nesta classe, mas que pertencem a outras (falsos positivos - FP).
Em resumo:

$$Precisao = \frac{VP}{VP + FP} \tag{1}$$

3.1.3 RECALL

O recall é a probabilidade de, dado os registros que são realmente daquela classe, quantos o modelo classificou corretamente. O cálculo é feito dividindo a quantidade de registros classificados corretamente (VP) pela quantidade total de exemplos que pertencem a esta classe, mesmo que classificados pelo modelo em outra (falso negativos - FN).
Em resumo

$$Recall = \frac{VP}{VP + FN} \tag{2}$$

3.1.4 F1-SCORE

O f1-score é uma média harmônica entre precisão e recall. Ela é muito boa quando você possui um dataset com classes desproporcionais, e o seu modelo não emite probabilidades. Em geral, quanto maior o F1 score, melhor.

$$F1 = \frac{2 * Precisao * Recall}{Precisao + Recall} \tag{3}$$

3.1.5 AUC – AREA UNDER THE ROC CURVE

O AUC é uma métrica muito útil para tarefas com classes desbalanceadas. Nela, mede-se a área sob uma curva formada pelo gráfico entre a taxa de exemplos positivos (recall da classe positiva) e a taxa de falsos positivos (calculado pela razão entre os registros classificados como positivos mas que são negativos, pela quantidade total pertencente a classe negativa).

Uma das vantagens em relação ao f1-score, é que ela mede o desempenho do modelo em vários pontos de corte.

3.2 DESCRIÇÃO TEÓRICA DOS MODELOS UTILIZADOS

3.2.1 REGRESSÃO LOGÍSTICA

Diferentemente da regressão linear, que é usada para fazer uma previsão da resposta numérica, a regressão logística é usada para resolver problemas de classificação. A regressão logística utiliza a função sigmoide (4) que nos ajuda a encolher as entradas contínuas com valor real em um intervalo de (0,1) que é muito útil ao lidar com probabilidades.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

Dessa forma, no caso da variável Y assumir apenas dois possíveis valores (0 ou 1 (barato ou caro)) e haver um conjunto de n variáveis X_1, X_2, ..., X_n, podemos escrever o modelo de regressão logística da seguinte maneira:

$$P(Y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + ... + \beta_n x_n)}} \tag{5}$$

onde $P(Y = 1)$ é a probabilidade de ser da classe 1, β_0 é o bias e β_1 até β_n são os coeficientes de cada variável de entrada, que será aprendida pelo dataset de treino.

O ajuste, dos coeficientes β_0, β_1 ... β_n, ocorre pela estimativa a partir do dataset, através do método da máxima verossimilhança (7). Esse método é calculado pelo logaritmo da função de verossimilhança (6) e consiste em encontrar uma combinação de coeficientes que maximiza a probabilidade da amostra ter sido observada. Considerando $h(x_i, \beta)$ como a equação (5), temos que:

$$l(\beta) = \prod_{i=1}^{N} P(y_i|x_i) = \prod_{i=1}^{N} h(x_i, \beta)^{y_i} h(x_i, \beta)^{1-y_i} \tag{6}$$

$$L(\beta) = \log(l(\beta)) = \sum_{i=1}^{N} y_i \log(h(x_i, \beta)) + (1 - y_i) \log(1 - h(x_i, \beta)) \tag{7}$$

Por fim, multiplicando-se L(β) por -1, o ajuste dos parâmetros pode ser obtido através do método do gradiente, dado que a função objetivo L passa a ser de minimização. Considerando ainda a regularização de Tikhonov com um parâmetro de regularização C, temos que a função de custo fica:

$$L_{regularized}(\beta) = C * ||\beta||^2 - L(\beta) = C * \sum_{j=1}^{p} \beta_j^2 - \sum_{i=1}^{N} y_i \log(h(x_i, \beta)) + (1 - y_i) \log(1 - h(x_i, \beta)) \tag{8}$$

Por fim, caso a independência estatística entre as variáveis seja observada e os parâmetros estimados num conjunto de treinamento suficientemente grande, a regressão logística terá o mesmo comportamento do classificador Bayesiano Simples, abordado a seguir.

3.2.2 NAIVE BAYES

Naive Bayes é um classificador de aprendizado de máquina simples, eficaz e muito utilizado. É um classificador probabilístico que faz classificações usando a regra de decisão Máximo A Posteriori (MAP) em um cenário bayesiano.

O objetivo de um classificador probabilístico é, com as variáveis $x_0...x_n$ e as classes $c_0...c_k$, determinar a probabilidade das variáveis que ocorrem em cada classe e retornar a classe mais provável. Portanto, para cada classe, queremos poder calcular $P(c_i|x_0, ..., x_n)$. Para isso, utilizamos a regra de Bayes:

$$P(c_i|x_0, ..., x_n) = \frac{P(x_0, ..., x_n|c_i)P(c_i)}{P(x_0, ..., x_n)} \tag{9}$$

Para facilitar os cálculos, foi feito uma suposição de que $x_0...x_n$ são independentes entre si. Com isso temos a regra de Bayes da seguinte forma:

$$P(c_i|x_0, ..., x_n) = \frac{P(x_0|c_i)...P(x_n|c_i)P(c_i)}{P(x_0)...P(x_n)} \tag{10}$$

Como o denominador fica constante para qualquer entrada, podemos dizer que:

$$P(c_i|x_0, ..., x_n) \propto P(x_0|c_i)...P(x_n|c_i)P(c_i) \tag{11}$$

onde $\propto$ significa 'é proporcional a'.

Essa suposição não é sempre verdadeira, mas o classificador apresenta bom desempenho na maioria das situações. Dessa forma, a representação final da probabilidade de classe é:

$$P(c_i|x_0, ..., x_n) \propto P(c_i) \prod_{j=1}^{n} P(x_j|c_i) \tag{12}$$

O cálculo dos termos individuais de $P(x_j|c_i)$ dependerá da distribuição das variáveis. Portanto, como neste trabalho tem apenas variáveis numéricas foi decidido utilizar a distribuição Gaussiana, onde $P(x_j|c_i) \approx N(\mu_{ji}, \sigma^2_{ji})$, sendo μ_{ji} e σ^2_{ji}, respectivamente, a média e a variância da distribuição gaussiana da variável x_j na classe c_i estimados a partir do conjunto de dados.

Agora que temos uma maneira de estimar a probabilidade de uma determinada variável ocorrer em uma determinada classe, falta apenas utiliza-la para produzir as classificações. Para isso, vamos utilizar uma regra de decisão chamada Máximo A Posteriori (13), também chamada pela sigla MAP, que simplesmente escolhe o c_i que tem a maior probabilidade, dado as variáveis.

$$y = \arg\max_{c_i}(P(c_i) \prod_{j=1}^{n} P(x_j|c_i)) \tag{13}$$

3.2.3 LINEAR DISCRIMINANT ANALYSIS

Um dos principais problemas do classificador bayesiano simples (naive bayes) se dá pelo fato de que ele não leva em consideração a correlação entre as variáveis dentro de uma mesma classe, já que a estimativa da probabilidade condicional é feita considerando a independência das mesmas, segundo a equação 11. O Bayesiano Linear (LDA) faz uma estimativa para a matriz de covariâncias das classes a partir da matriz de covariâncias dos dados, ou seja: $\hat{\Sigma}_i = \hat{\Sigma}, \forall i \in 1, 2, ..., m$(sendo "m"o número de classes). Então, o resultado da função discriminante para uma classe C_i e registro x(t) pode ser calculada como:

$g_i(x(t)) = ln(P(C_i|x(t))) = ln(p(x(t)|C_i)) + ln(P(C_i))$,

onde $p(x(t)|C_i) = N(\hat{\mu_i}, \hat{\Sigma}_i)$ é a distribuição normal multivariada (diferente do bayesiano simples que usa a monovariada para a condicional de cada variável e pela suposição de independência multiplica os resultados) com médias $\hat{\mu_i}$ e covariâncias $\hat{\Sigma}_i$ para a classe C_i. Substituindo a fórmula da distribuição normal multivariada na equação acima temos:

$$g_i(x(t)) = -\frac{1}{2}(x(t) - \hat{\mu_i})\hat{\Sigma_i}^{-1}(x(t) - \hat{\mu_i})^T - \frac{ln(2\pi)}{2} - \frac{ln(|\hat{\Sigma}_i|)}{2} + ln(P(C_i)) \tag{14}$$

$$g_i(x(t)) = -\frac{1}{2}(x(t)\hat{\Sigma_i}^{-1}x(t)^T) + (x(t)\hat{\Sigma_i}^{-1}\hat{\mu_i}^T) - \frac{1}{2}(\hat{\mu_i}\hat{\Sigma_i}^{-1}\hat{\mu_i}^T) - \frac{ln(2\pi)}{2} - \frac{ln(|\hat{\Sigma_i}|)}{2} + ln(P(C_i)) \quad (15)$$

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

$\lvert\Sigma\rvert^{\frac{1}{2}}$	is a real number
$(x-\mu)^T$	is [m*n] or [n*m] dimensional
$(x-\mu)$	is [m*n] or [n*m] dimensional
Σ^{-1}	is [n*n] dimensional

Figura 2: Distribuição Normal Multivariada

Analisando 15 vemos que o primeiro termo $-\frac{1}{2}(x(t)\hat{\Sigma_i}^{-1}x(t)^T)$, o quarto $-\frac{ln(2\pi)}{2}$ e o quinto $-\frac{ln(|\hat{\Sigma_i}|)}{2}$ serão iguais para todas as classes devido à forma como o LDA estima a matriz de covariâncias de cada classe como sendo a matriz de covariância de todos os dados. Portanto, para o Bayesiano Linear, teremos a função discriminante como sendo:

$$g_i(x(t)) = (x(t)\hat{\Sigma_i}^{-1}\hat{\mu_i}^T) - \frac{1}{2}(\hat{\mu_i}\hat{\Sigma_i}^{-1}\hat{\mu_i}^T) + ln(P(C_i)) \quad (16)$$

Uma vez calculado $g_i(x(t))$ para todas as "m"classes, classifica-se o registro pertencente à classe que obteve o maior valor para a função discriminante. Note que quando temos um equilíbrio na quantidade de classes, a probabilidade a priori $(P(C_i))$ será a mesma para todas as classes, e então o último termo desaparece.

3.2.4 QUADRATIC DISCRIMINANT ANALYSIS

É similar ao LDA, contudo calcula a matriz de covariâncias de forma exata para cada classe. Com isso, apenas o termo constante da equação 15 poderá ser retirado e os demais irão variar conforme a classe. Assim, a função discriminante fica:

$$g_i(x(t)) = -\frac{1}{2}(x(t)\hat{\Sigma_i}^{-1}x(t)^T) + (x(t)\hat{\Sigma_i}^{-1}\hat{\mu_i}^T) - \frac{1}{2}(\hat{\mu_i}\hat{\Sigma_i}^{-1}\hat{\mu_i}^T) - \frac{ln(|\hat{\Sigma_i}|)}{2} + ln(P(C_i)) \quad (17)$$

Portanto, no caso do classificador Bayesiano Quadrático, teremos uma quantidade significativamente maior de parâmetros a serem obtidos, se comparado aos modelos bayesianos anteriores. A consequência disso é que a quantidade de registros pode não ser suficiente, levando ao sobre-ajuste do modelo.

3.2.5 RANDOM FOREST

O Random Forest é um algoritmo de aprendizado supervisionado que pode ser utilizado tanto em problemas de regressão quanto nos problemas de classificação. Ele consiste em um grande número de árvores de decisões individuais que operam como um ensemble learning, onde cada árvore individual na random forest prevê uma classe e a classe que foi prevista com maior quantidade será o resultado do modelo. A razão do modelo random forest funcionar de uma forma satisfatória é que um grande número de modelos relativamente não correlacionados que operarem juntos superará qualquer um dos modelos que operarem individualmente.

Primeiramente vamos compreender melhor as árvores de decisão. Dado um conjunto de dados, a cada nível o algoritmo efetua uma partição desse conjunto escolhendo-se qual dentre as variáveis restantes (que não foram escolhidas ainda) é aquela que melhor separa o conjunto nas classes objetivo, obtendo maior pureza - já que muito dificilmente um nó folha (correspondendo a classificação do algoritmo) possuirá 100% de registros de uma classe, portanto é dito impuro. Logo, afim de determinarmos qual separação é a melhor, precisamos de uma métrica para o grau de impureza da separação. Dada uma separação, para cada nó podemos calcular a impureza de Gini, que é dada por:

$G = \sum_{i=1}^{m} p(i) * (1 - p(i))$, sendo "m"o número de classes do problema e p(i) a probabilidade de pegar um registro pertencente a classe "i". Vamos considerar um exemplo para 2 classes:

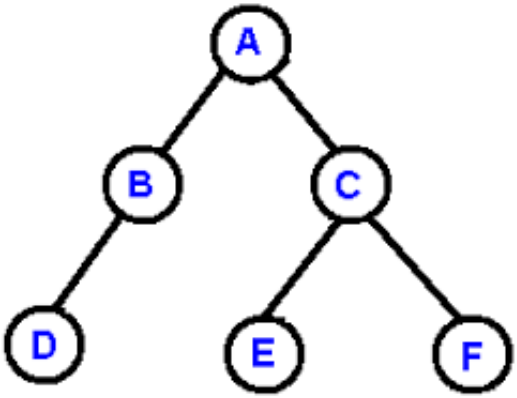


Figura 3: Decision Tree

Suponha que estamos no nó raiz A e queremos escolher, dentre as variáveis do conjunto de dados, aquela que melhor separa em classe positiva e negativa. Para fazer tal escolha calculamos primeiramente a impureza de Gini para o nó A (denotado por G_A)e, para cada variável utilizada na separação, o valor da métrica para os nós B (G_B) e C (G_C). Como não necessariamente essa separação é feita de forma simétrica - pode haver mais registros num ramo da divisão - faremos uma ponderação das impurezas para avaliar a separação. Suponha que 40% dos registros tenham ido para o nó B e 60% para o C. Então teremos que a impureza após a divisão é de:

G_{Total} = 0.4*G_B + 0.6*G_C.

Por fim, calculamos o ganho de Gini da seguinte forma:

$Gini_{Gain}$ = G_A - G_{Total} Com isso, escolhemos dentre as variáveis aquela com a qual obtemos o maior valor de ganho e processo é repetido recursivamente até que não seja possível obter mais nenhum ganho de informação.

Percebemos, contudo, que tal procedimento leva facilmente ao sobre-ajuste desse modelo com o conjunto de treino. Dissemos, anteriormente, que a Random Forest é um ensemble de árvores de decisão com pouca ou nenhuma correlação entre si. Como isso é obtido? Por meio de duas técnicas: Bagging e Aleatoriedade das Features (Features Randomness).

Bagging

Essa técnica consiste em, dado um conjunto de treino com N amostras, cada modelo constituinte do ensemble será ajustado segundo uma amostra de mesmo tamanho N, contudo com registros repetidos. Suponha que o dataset seja composto por 7 registros $[reg_1, reg_2, reg_3, reg_4, reg_5, reg_6, reg_7]$. Então no caso da Random Forest cada árvore receberá um conjunto de 7 amostras, por exemplo $[reg_1, reg_1, reg_3, reg_4, reg_4, reg_6, reg_7]$, $[reg_6, reg_1, reg_3, reg_6, reg_4, reg_6, reg_7]$... e assim sucessivamente, de forma que cada uma tenha um conjunto de treinamento distinto.

Feature Randomness

Dissemos que anteriormente que a cada nó o algoritmo da árvore de decisão escolhe uma variável que melhor separa o conjunto de dados. O que aconteceria se tivéssemos um conjunto de árvores treinadas com o mesmo conjunto de dados? Provavelmente todas seriam iguais, já que todas escolheriam categoricamente as mesmas variáveis para realizar a divisão. Uma forma de impedir isso é utilizando a técnica de Bagging, na qual cada árvore recebe um conjunto de treino distinto. A outra é usando a Feature Randomness. Nessa última, a intuição consiste em limitar o conjunto de variáveis dentre as quais uma árvore de decisão pode escolher para efetuar a divisão do conjunto de dados. Suponha que tenhamos 3 variáveis: área da piscina, área do quintal e quantidade de quartos. Então cada árvore só poderia escolher efetuar uma repartição escolhendo-se duas das três variáveis, de forma aleatória. Por exemplo, a primeira árvore só poderia escolher entre área da piscina e área do quintal, enquanto a segunda área do quintal e quantidade de quartos. O raciocínio pode ser empregado para um conjunto maior de variáveis.

3.2.6 SVM LINEAR

Partindo da suposição que os registros são linearmente separáveis, temos de forma geral um conjunto de entrada no modelo com variáveis $x_i, i \in 1, 2, ..., p$ e uma variável de saída y. A saída do modelo é um conjunto de pesos w_i, um para cada variável, cuja combinação linear fornece a predição do valor de y. O objetivo do classificador é encontrar a maior margem possível de forma a reduzir a quantidade de pesos w que são diferentes de zero, garantindo maior generalização do modelo. Ou seja, queremos encontrar um hiperplano que separa os dados nas suas respectivas classes com erro zero (no caso linearmente separável, condição essa que posteriormente será relaxada). Seja v(t) uma variável indicadora e a função discriminante $g(x) = xw^T + b$ tal que:

$$g(x(t)) \begin{cases} \geq 0, \text{então o registro será classificado como pertencente a classe positiva} \\ < 0, \text{o registro será classificado como pertencente a classe negativa} \end{cases} \tag{18}$$

Tomemos o exemplo 2d (podemos generalizar pra mais dimensões):

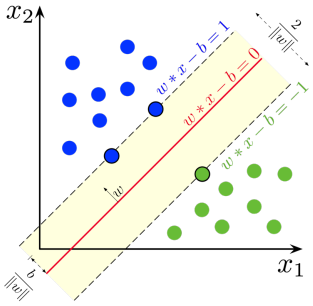


Figura 4: svm como problema de maximização de margem

Queremos maximizar a distância de um ponto (x1,x2), pertencente ao conjunto de suporte (pontos extremos de uma classe, mais próximos do hiperplano que separa as classes), até a função discriminante (que nesse caso é uma reta). Temos que a distância é $d = |w_1X_1 + w_2X_2 + b|/\sqrt{w_1^2 + w_2^2} = |w_1X_1 + w_2X_2 + b|/||w||$. Como o ponto pertence ao conjunto de suporte, $d = |1|/||w||$ se o ponto pertencer a classe positiva ou $d = |-1|/||w||$ se pertencer a classe negativa. Em ambos os casos, $d = 1/||w||$. Como queremos maximizar uma função que decai conforme $||w||$ aumenta, é equivalente a encontrarmos o mínimo de $||w||$. Para modelar como um problema de programação quadrática e dado que $min||w||$ é equivalente a $min||w||^2$ e a multiplicação por uma constante também não altera a solução do problema, podemos formular da seguinte forma:

$$\begin{gathered} \min_w \frac{1}{2}||w||^2 \\ \text{suj. } v(t)g(x(t)) \geq 1, \forall t \in 1, 2, ..., N \end{gathered}$$

As restrições do problema de otimização garantem que o erro de classificação, no caso linearmente separável, será nulo (já que quando $v(t) = -1$, $g(x(t)) < 0$ e quando $v(t) = 1$, $g(x(t)) \geq 0$), enquanto que a minimização da norma quadrática de W garante que teremos a maior margem de seperação e a maior quantidade possível de pesos w_i iguais a zero. Podemos ainda escrever o Lagrangeano, transformando assim em um problema sem restrições:

$$L(\alpha, w, b) = \frac{1}{2}||w||^2 - \sum_{t=1}^{N} \alpha(t)(v(t)g(x(t)) - 1) = \frac{1}{2}||w||^2 - \sum_{t=1}^{N} [\alpha(t)v(t)x(t)w^T + \alpha(t)v(t)b) - \alpha(t)] \tag{19}$$

Assim temos:

$$\begin{gathered} \min L(\alpha, w, b), \\ \text{suj. } \alpha_i \geq 0, \forall i \in 1, 2, ..., N \end{gathered}$$

De forma geral, se temos o seguinte problema de otimização não-linear (PNL) e o Lagrangeano da mesma:

$$(PNL) \begin{cases} min_x f(x), \\ suj. g_i(x) \leq 0, \forall i \in 1, 2, ..., m \\ h_j(x) = 0, \forall j \in 1, 2, ..., l \end{cases} \tag{20}$$

$$L(x, \mu, \lambda) = f(x) + \sum_{i=1}^{m} \mu_i g_i(x) + \sum_{j=1}^{l} \lambda_j h_j(x) \tag{21}$$

As seguintes propriedades são satisfeitas se x* for um minimizador local:

$$\nabla f(x*) + \sum_{i=1}^{m} \mu_i \nabla g_i(x*) + \sum_{j=1}^{l} \lambda_j \nabla h_j(x*) = 0 \tag{22}$$

$$\mu_i g_i(x*) = 0, \forall i \in 1, 2, ..., m \tag{23}$$

Assim, como no nosso problema queremos achar w* que fornece $min_w \frac{1}{2}||w||^2$, temos (comparando-se 22 e 19):

$$\frac{\partial L(\alpha, w, b)}{\partial w} = w* - \sum_{t=1}^{N} \alpha(t) v(t) x(t) = 0 \Rightarrow w* = \sum_{t=1}^{N} \alpha(t) v(t) x(t) \tag{24}$$

$$\frac{\partial L(\alpha, w, b)}{\partial b} = \sum_{t=1}^{N} \alpha(t) v(t) = 0 \tag{25}$$

Por 23 e 19 temos que somente quando $g(x(t)) = -1$ e $v(t) = -1$ ou $g(x(t)) = 1$ e $v(t) = 1$ é que as restrições de desigualdade são ativas (iguais a zero) e portanto teremos multiplicadores de Lagrange diferente de zero. Nesses pontos, que possuem desigualdades ativas e serão usados para o cálculo dos parâmetros w_i, são denominados vetores de suporte. Finalmente, pode-se substituir as expressões 24 e 25 em 19 e então resolvemos o problema dual como maximização em relação a α:

$$\max_{\alpha} L_d(\alpha) = \sum_{t=1}^{N} \alpha(t) - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha(i)\alpha(j)v(i)v(j)\langle x(i), x(j)\rangle \tag{26}$$

$$\begin{gathered} \text{suj. } \sum_{t=1}^{N} \alpha(t) v(t) = 0 \\ \alpha(t) \geq 0, \ \forall t \in 1, 2, ..., N \end{gathered}$$

Resolvendo 26 obtemos o vetor de α^*, indicando os registros que compõem os vetores de suporte ($\alpha(t) \neq 0$). Podemos assim obter W* substituindo α^* em 24. Com isso, a função discriminante pode ser reescrita:

$$g(x) = \sum_{t=1}^{N} \alpha^*(t) v(t) \langle x, x(t)\rangle + b \tag{27}$$

3.2.7 SVM NÃO LINEAR

No SVM não linear basta trocarmos o produto interno $\langle x, x(t)\rangle$ por uma função de núcleo não linear k(x,x(t)). Tomando $\theta = (\alpha(1)v(1), \alpha(2)v(2), ..., \alpha(N)v(N))$ e sabendo que

$$\max_{x} f(x) = \min_{x} -f(x) \tag{28}$$

Sendo K a matriz de núcleo e V = [v(1),v(2),...,v(N)] o vetor de variáveis indicadoras, podemos reescrever 26 da seguinte forma:

$$\min_{\theta} \frac{1}{2} \theta K \theta^T - \theta V^T \tag{29}$$

$$\begin{gathered} \text{suj. } \sum_{i=1}^{N} \theta_i = 0 \\ \text{v(t)}\theta(t) \geq 0 \end{gathered}$$

Obtendo-se a solução ótima θ^* pelo problema acima, a função discriminante no caso não linear fica:

$$\text{g(x)} = \sum_{t=1}^{N} \theta^*(t) k(x, x(t)) + b$$

Até agora só tratamos problemas separáveis, tanto lineares quanto não lineares. Contudo podemos generalizar o SVM para problemas não separáveis acrescentando uma variável de folga $\xi(t)$ que representa o erro da classificação de um registro x(t):

$$\xi(t) = \begin{cases} \max(0, v(t) - g(x(t))), & \text{se v(t) = 1} \\ \max(0, g(x(t)) - v(t)), & \text{se v(t) = -1} \end{cases} \tag{30}$$

Assim, sendo k uma constante qualquer (normalmente 1 ou 2) e C também arbitrário, o problema de otimização fica:

$$\min_w \frac{1}{2}||w||^2 + C\sum_{t=1}^{N} \xi^k(t) suj. v(t)g(x(t)) \geq 1 - \xi(t)\xi(t) \geq 0, \forall t \in 1, 2, ..., N \tag{31}$$

Conforme definido acima, o problema de otimização representa um tradeoff entre encontrar a máxima margem de separação e reduzir o custo da classificação incorreta. Quando $C \to 0$ a primeira é favorecida. Por outro lado quando $C \to \infty$ o risco de classificação incorreta ganha maior importância.

3.2.8 RBF

Redes de Base Radial são geralmente usadas para aproximação de funções a partir de um conjunto de pontos, mas também pode ser utilizada para classificação. conforme veremos a seguir. De forma geral, queremos obter um conjunto de parâmetros tal que, multiplicando as funções de base previamente escolhidas, aproxima determinada função:

$$\hat{y}(t) = \sum_{i=1}^{n} \phi_i(x(t))\theta_i \tag{32}$$

sendo ϕ_i uma função de base de um total de "n"funções que leva do $R^p : R$ (com p sendo o número de variáveis do problema), θ_i o respectivo parâmetro multiplicativo e $\hat{y}(t)$ a aproximação feita pela rede RBF de uma função y(t). Podemos ainda acrescentar um bias para um melhor ajuste do modelo:

$$\hat{y}(t) = \theta_0 + \sum_{i=1}^{n} \phi_i(x(t))\theta_i = \hat{u}(t)\theta^T \tag{33}$$

Uma característica importante das funções de base radial é que elas são simétricas em relação a um centro w_i, podendo ser escritas como $\phi_i(x) = h(x - w_i)$. Nesse trabalho usaremos a função gaussiana centrada em w e com desvio padrão σ definida por:

$$h(||x - w||) = exp(-\frac{||x - w||^2}{2\sigma^2}) \tag{34}$$

Da equação acima notamos que a medida que o registro se afasta do centro, o valor da função radial decai exponencialmente. O que acontece se um registro está muito afastado de todos os centros? Todas as funções radiais darão valores próximos a 0. Por isso dizemos que RBF é um modelo local. No ajuste de parâmetros, uma vez determinada a quantidade de funções de base radial "n", a escolha dos centros foi feita a partir da execução do algoritmo k-médias para o agrupamento dos dados em "n"clusters. Já o desvio padrão de uma função ϕ_i com centro em w_i calculamos pegando a partir da distância até o centro mais próximo w_k dentre todos aqueles obtidos no passo anterior com o k-médias. Então o desvio padrão σ_i é calculado fazendo-se $\sigma_i = \frac{||w_i - w_k||}{2}$.

Para encontrarmos os valores dos parâmetros precisamos definir primeiramente uma função de custo que queremos minimizar. No método de Mínimos Quadrados, essa função é o erro médio quadrático, definida por:

$$l(\theta) = \sum_{t=1}^{N} (y(t) - \hat{y}(x(t), \theta))^2 \tag{35}$$

Seja $U^{NX(n+1)}$ a matriz de interpolação definida por:

$$U = \begin{bmatrix} 1 & \phi_1(x_1)\dots & \phi_n(x_1) \\ \vdots & \ddots & \vdots \\ 1 & \phi_1(x_N)\dots & \phi_n(x_N) \end{bmatrix} \tag{36}$$

,sendo ϕ_i a função de base radial centrada em w_i, com $i \in 1, 2, ..., n$ e x_j um ponto $\in R^p$ (p o número de variáveis), com $j \in 1, 2, ..., N$. Seja também y(j) a variável de saída correspondente ao registro x_j, então:

$$Y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(N) \end{bmatrix} \tag{37}$$

Então podemos calcular os parâmetros θ_i que minimizam 35 a partir de um problema de programação quadrática definida por:

$$\begin{gathered} \min_\theta \frac{1}{2}\theta H\theta^T - \theta C^T \\ \text{suj. } \theta_j \leq \sqrt{n} \\ -\theta_j \leq \sqrt{n}, \forall j \in 0, 1, ...n. \end{gathered}$$

onde $H = U^T U$ e $C^T = U^T Y$. Podemos reescrever na forma padrão desconsiderando as restrições de igualdade:

$$\begin{gathered} \min_\theta \frac{1}{2}\theta^T P\theta + q^T\theta \\ \text{suj. } G\theta \leq h \\ A\theta = b \\ \text{com } P = H,\ q = -C^T \text{ e} \end{gathered}$$

$$G = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \dots & 1 \\ -1 & 0 & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & & \dots & -1 \end{bmatrix} \tag{38}$$

$$h = \begin{bmatrix} \sqrt{n} \\ \sqrt{n} \\ \vdots \\ \sqrt{n} \end{bmatrix} \tag{39}$$

Para predizer a variável de saída de um novo conjunto de dados X' basta calcularmos a matriz de interpolação U' correspondente e então: $\hat{Y}' = U'x\theta$. No problema de classificação podemos utilizar uma variável indicadora de saída $\hat{v}(t)$ de forma que:

$$\hat{v}(t) = \begin{cases} -1, & \text{caso não pertença a classe positiva} \\ 1, & \text{caso contrário} \end{cases} \tag{40}$$

Por fim, uma vez que a função discriminante possui valor positivo quando trata-se da classe positiva, basta verificarmos, para um determinado registro x_i, se $\hat{Y_i}' > 0$ para classificá-lo como tal.

3.2.9 REDE NEURAL MULTI LAYER PERCEPTRON (MLP)

A Rede Neural Multi Layer Perceptron, também conhecida como rede feed-forward, é um modelo não linear que mapeia as entradas $x_0...x_n$ com as saídas $y_0...y_m$, utilizando conexões com múltiplos pesos. A estrutura básica do MLP consiste em uma camada de entrada, uma camada intermediaria e uma camada de saída. A camada de entrada consiste em n neurônios, sendo n a quantidade de variáveis de entrada. A camada de saída consiste em m neurônios, sendo m a quantidade de classes do problema. A camada intermediária tem a quantidade de neurônios determinado de acordo com cada problema e cada neurônio recebe a soma dos valores

dos neurônios da primeira camada com seus respectivos pesos. Em seguida, é utilizado uma função de ativação em cada neurônio para gerar a saída de cada neurônio da camada intermediaria. Essa saída, juntamente com seus respectivos pesos, são as entradas da camada de saída, que por sua vez, a soma de todas as entradas também será submetida a uma função de ativação que nos dará o resultado final. Essa é a mesma ideia para qualquer quantidade de camadas intermediarias. A formulação matemática de cada neurônio desse modelo pode ser representado pela seguinte equação:

$$u_j^{(k)} = b_j^{(k)} + \sum_{i=1}^{q} w_{ij} h_i^{(k-1)} \tag{41}$$

$$h_j^{(k)} = g(u_j^{(k)}) \tag{42}$$

onde q é a quantidade de neurônios da camada anterior (camada $k-1$), w_{ij} é o peso da conexão que liga o neurônio i da camada anterior com o neurônio j da camada atual, $u_j^{(k)}$ é o resultado do neurônio j da camada atual (camada k), que é a soma do bias da camada atual com os valores vindos da camada anterior multiplicado por seus respectivos pesos e g é uma função de ativação que será utilizado em cima desse resultado para produzir a saída $h_j^{(k)}$ do neurônio j.

O aprendizado no MLP ocorre pela mudança de cada peso de conexão depois que cada dado é processado, baseado no erro da saída comparado ao valor esperado. Para isso, utilizamos o método do gradiente descendente (43) que consiste em atualizar os parâmetros (no nosso caso, os pesos), a cada interação, calculados pela derivada da função de avaliação.

$$w_{ij} \Leftarrow w_{ij} - \eta \frac{\partial J(y_i, \hat{y_i})}{\partial w_{ij}} \tag{43}$$

η é a taxa de aprendizagem que determina o tamanho do deslocamento na direção contrario a do gradiente e $\frac{\partial J(y_i, \hat{y_i})}{\partial w_{ij}}$ é calculado pelo algoritmo de backpropagation, que consiste em aplicar a regra da cadeia com a propagação do erro da ultima camada em direção ao inicio. Aplicando a regra da cadeia, temos que:

$$\frac{\partial J(y_i, \hat{y_i})}{\partial w_{ij}} = \frac{\partial J(y_i, \hat{y_i})}{\partial \hat{y_i}} \frac{\partial \hat{y_i}}{\partial u_j^{(k)}} \frac{\partial u_j^{(k)}}{\partial w_{ij}} \tag{44}$$

Sabendo que $u_j^{(k)} = b_j^{(k)} + \sum_{i=1}^{q} w_{ij} h_i^{(k-1)}$, temos que:

$$\frac{\partial u_j^{(k)}}{\partial w_{ij}} = h_i^{(k-1)} \tag{45}$$

Logo, temos que a atualização do nosso parâmetro peso fica:

$$w_{ij} \Leftarrow w_{ij} - \eta \frac{\partial J(y_i, \hat{y_i})}{\partial \hat{y_i}} \frac{\partial \hat{y_i}}{\partial u_j^{(k)}} h_i^{(k-1)} \tag{46}$$

4 RESULTADOS

4.1 VISUALIZAÇÃO E CARACTERIZAÇÃO DOS DADOS

4.1.1 EXISTÊNCIA DE VALORES AUSENTES

Inicialmente, foi realizada a verificação e análise da existência de valores ausentes no dataset. A Figura 5 mostra a porcentagem e a quantidade de valores ausentes das variáveis que tem pelo menos um valor ausente. Com isso, foi decidido retirar as variáveis com mais de 5% de valores ausentes, para não atrapalhar na previsão do modelo. Já com as variáveis que continham menos de 5% de valores ausentes, foi decidido retirar a linha do registro.

Dessa forma, foram retiradas as variáveis "LotFrontage"e "GarageYrBlt"e 26 linhas de registros, resultando em um dataset com 34 variáveis e 2893 registros.

		Missing Ratio	Missing Number
0	LotFrontage	16.65	486
1	MasVnrArea	0.79	23
2	BsmtFinSF1	0.03	1
3	BsmtFinSF2	0.03	1
4	BsmtUnfSF	0.03	1
5	TotalBsmtSF	0.03	1
6	BsmtFullBath	0.07	2
7	BsmtHalfBath	0.07	2
8	GarageYrBlt	5.45	159
9	GarageCars	0.03	1
10	GarageArea	0.03	1

Figura 5: Porcentagem e quantidade de valores ausentes

4.1.2 ESTATÍSTICA BÁSICA

Apos a retirada de valores ausente, podemos verificar na Figura 6 a estatística básica de cada variável do dataset. O "count"é a quantidade de linhas de registros, o "mean"é a média aritmética, o "std"é o desvio padrão, o "min"é o menor valor, "25%"é o primeiro quartil, "50%"é o segundo quartil (ou mediana), "75%"é o terceiro quartil e o "max"é o valor máximo.

	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF
count	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00
mean	57.23	10150.74	6.08	5.57	1971.09	1984.11	102.31	441.37	49.91	559.77
std	42.61	7876.96	1.41	1.12	30.27	20.89	179.40	455.28	169.81	439.11
min	20.00	1300.00	1.00	1.00	1872.00	1950.00	0.00	0.00	0.00	0.00
25%	20.00	7449.00	5.00	5.00	1953.00	1965.00	0.00	0.00	0.00	220.00
50%	50.00	9452.00	6.00	5.00	1973.00	1993.00	0.00	368.00	0.00	467.00
75%	70.00	11553.00	7.00	6.00	2000.00	2004.00	164.00	733.00	0.00	801.00
max	190.00	215245.00	10.00	9.00	2010.00	2010.00	1600.00	5644.00	1526.00	2336.00

	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr
count	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00
mean	1051.05	1158.21	335.83	4.74	1498.78	0.43	0.06	1.56	0.38	2.86
std	440.45	388.87	428.42	46.60	505.11	0.52	0.25	0.55	0.50	0.82
min	0.00	334.00	0.00	0.00	334.00	0.00	0.00	0.00	0.00	0.00
25%	792.00	876.00	0.00	0.00	1124.00	0.00	0.00	1.00	0.00	2.00
50%	989.00	1082.00	0.00	0.00	1442.00	0.00	0.00	2.00	0.00	3.00
75%	1300.00	1383.00	703.00	0.00	1742.00	1.00	0.00	2.00	1.00	3.00
max	6110.00	5095.00	2065.00	1064.00	5642.00	3.00	2.00	4.00	2.00	8.00

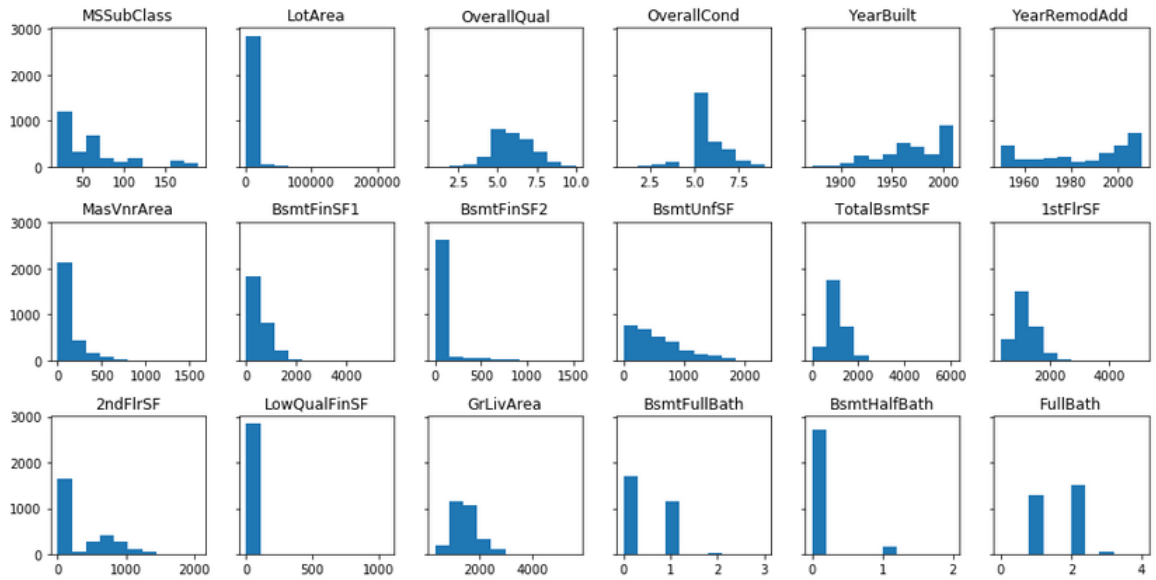
	KitchenAbvGr	TotRmsAbvGrd	Fireplaces	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch
count	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00	2893.00
mean	1.04	6.45	0.60	1.76	472.34	93.97	47.10	23.09	2.63	16.21
std	0.21	1.57	0.65	0.76	215.60	126.79	67.13	64.24	25.30	56.42
min	0.00	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	1.00	5.00	0.00	1.00	319.00	0.00	0.00	0.00	0.00	0.00
50%	1.00	6.00	1.00	2.00	478.00	0.00	26.00	0.00	0.00	0.00
75%	1.00	7.00	1.00	2.00	576.00	168.00	69.00	0.00	0.00	0.00
max	3.00	15.00	4.00	5.00	1488.00	1424.00	742.00	1012.00	508.00	576.00

	PoolArea	MiscVal	YrSold	SalePrice
count	2893.00	2893.00	2893.00	2893.00
mean	2.27	51.28	2007.79	179862.69
std	35.82	569.93	1.32	57338.05
min	0.00	0.00	2006.00	34900.00
25%	0.00	0.00	2007.00	154300.00
50%	0.00	0.00	2008.00	176515.50
75%	0.00	0.00	2009.00	191745.39
max	800.00	17000.00	2010.00	755000.00

Figura 6: Estatística básica das variáveis

4.1.3 HISTOGRAMAS

Histograma é um gráfico de frequência que tem como objetivo ilustrar como uma determinada amostra ou população de dados está distribuída. Abaixo os histogramas das variáveis do dataset:



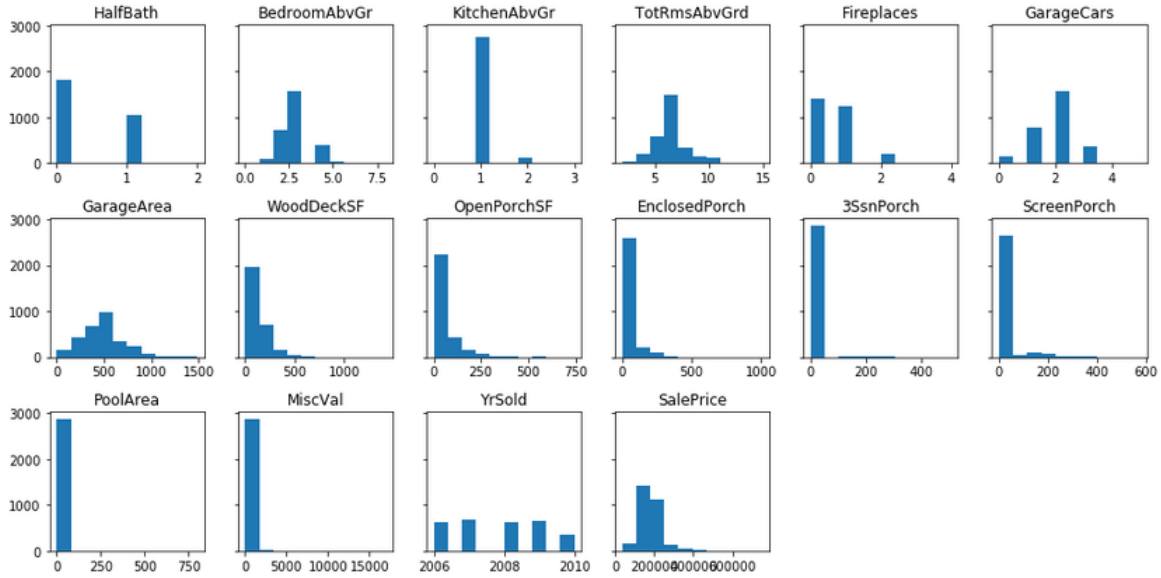
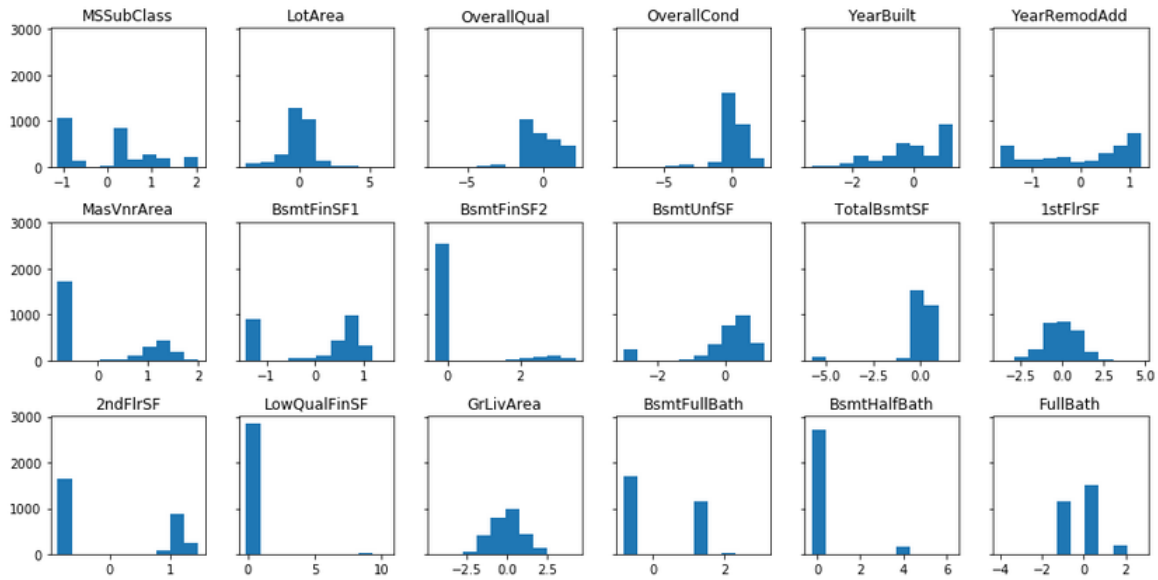


Figura 7: Histogramas das Variáveis

Através da análise do histograma, nota-se que as variáveis não obedecem uma distribuição normal. Além disso, a distribuição da maioria das variáveis estão deslocadas para esquerda ou direita, caracterizando uma assimetria. Também pode-se observar o quanto varia a ordem de grandeza entre as variaveis. Temos caso que vai de 0 a 10, como o "OverallQual"e temos caso que vai de 0 a 755000, como o "SalePrice". Dessa forma, para resolver esses problemas, foi necessário normalizar e padronizar nossos dados utilizando o z-score e o log.

Como nos nossos dados haviam variáveis com valor 0 no domínio, antes de aplicar o log somamos $1+min(x)$, a cada valor de uma variável x. Por exemplo: Se para a variável tamanho do estacionamento o menor valor dentre todos os registros era 13, então no registro 1 que tinha valor 15 para essa variável agora terá valor 1 + 13 + 15 = 28 , o registro 2 que tinha valor 16 terá valor 1 + 13 + 16 = 30, e assim sucessivamente, para todos os registros. Dessa forma garantimos que todos valores são sempre maiores ou iguais a 1. Dado $v(t)$ o valor da variável x para um registro "t"após essa transformação $(v(t) = 1 + min(x) + x(t))$, aplicamos o log e o z-score (subtraímos da média e dividimos pelo desvio padrão para cada registro "t"): $v(t)' = log(v(t))$ e $x(t)' = \frac{v(t)' - \bar{V}'}{\hat{\sigma}}$, onde $\hat{\sigma} = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(v'(t) - \bar{V}')^2}$ e $\bar{V}' = \frac{1}{N}\sum_{t=1}^{N} v(t)'$ Após essas transformações, podemos visualizar abaixo como ficou o novo histograma:



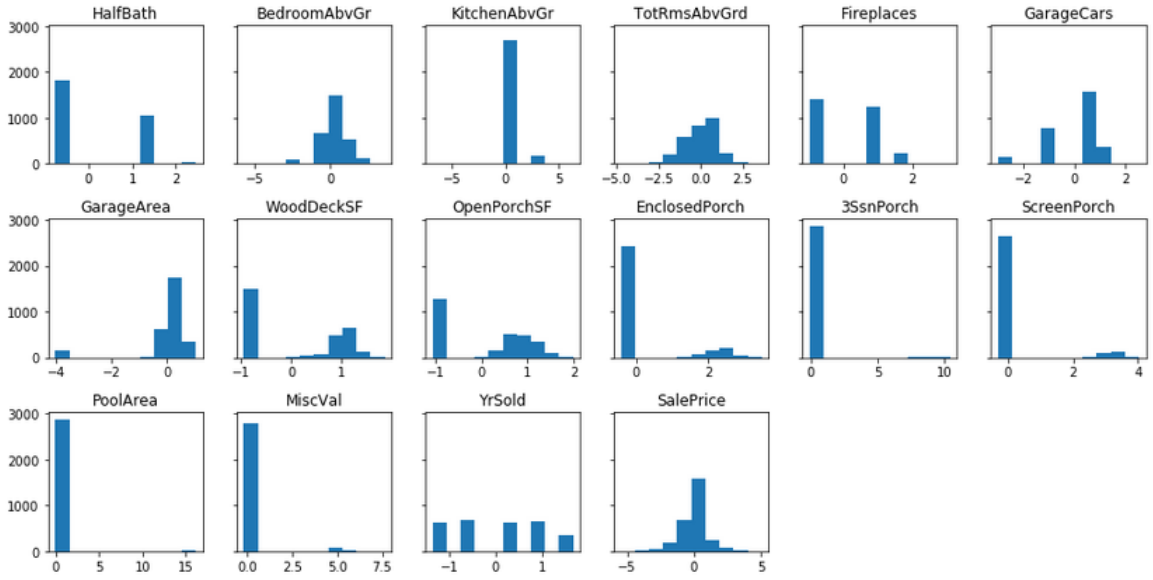


Figura 8: Histogramas das Variáveis após as transformações

4.1.4 EXISTÊNCIA DE OUTLIERS

Verificar a existência de possíveis outliers é um procedimento importante do pré-processamento, uma vez que estes valores podem prejudicar o desempenho dos modelos. Outlier é uma observação que apresenta um grande afastamento das demais da série ou que é inconsistente.

Dessa forma, para identificar os outliers, foi calculado uma matriz distancia, onde o número de linhas e o de colunas é dado pela quantidade de registros no dataset. Assim, trata-se de uma matriz quadrada, pertencente ao $\mathbb{R}^{N\times N}$ sendo N o número de registros, e simétrica pois dist(u,v) = dist(v,u). Essa matriz distancia (Figura 9) foi ordenada de forma crescente de acordo com a o valor da variável "SalePrice", para termos uma melhor visualização dos outliers. Uma vez que as variáveis foram normalizadas, o cálculo da distância entre os registros u e v foi feito da seguinte forma:

$$dist(u,v) = \sqrt{\sum_{i=1}^{p}(X_{ui} - X_{vi})^2} \tag{47}$$

onde p é o número de variáveis e X_u denota o vetor com as variáveis do registro "u". Idem para X_v.

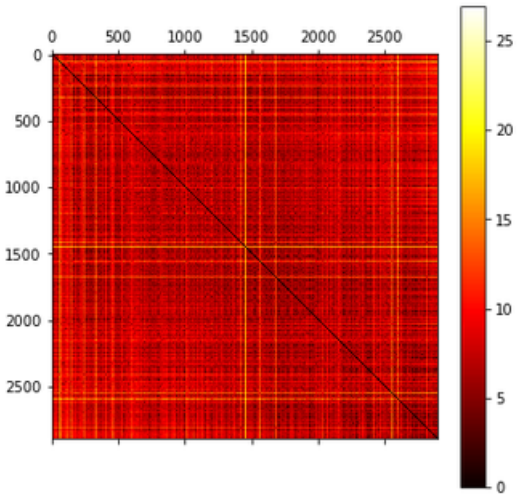


Figura 9: Matriz distancia ordenada com outlier

Como pode ser visualizado na figura acima, podemos verificar alguns outliers, sinalizados por linhas mais amareladas. Dessa forma, determinamos os outliers sendo os 10% registros cuja distância média são a maiores. Apos a retirada dos outliers, podemos visualizar na Figura 10 como ficou a matriz distância.

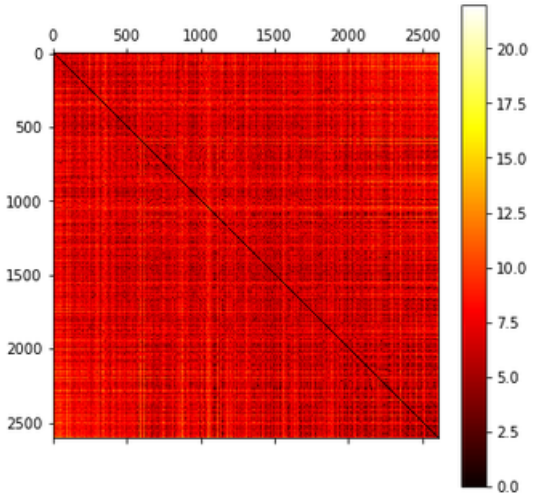
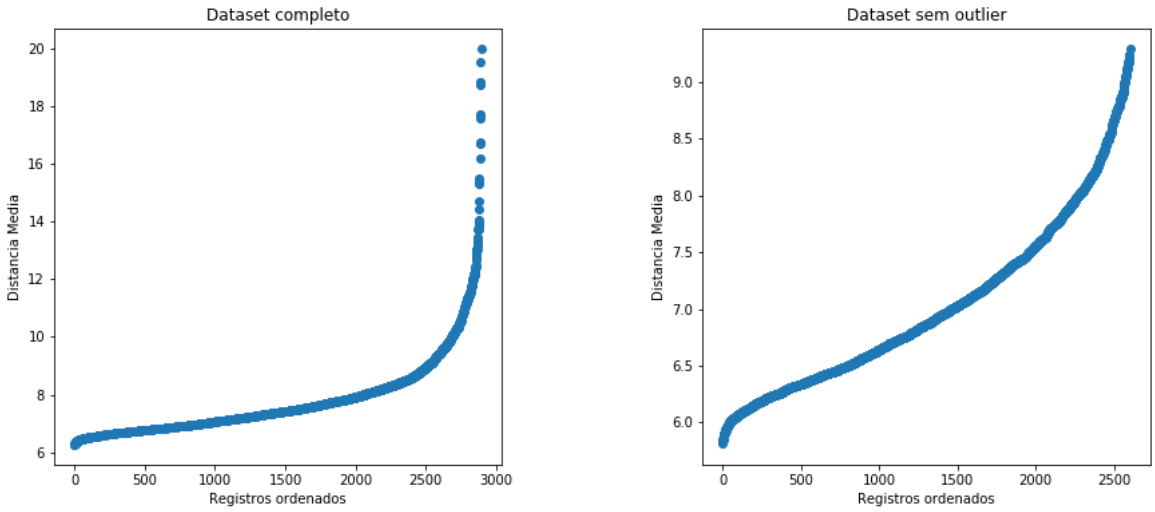


Figura 10: Matriz distancia ordenada sem outlier

Escolhemos retirar 10% já que, conforme pode-se observar da figura abaixo, a partir do registro 2600 a distância média dos registros ordenados começa aumentar de forma exponencial. Como após a retirada de alguns registros com dados faltantes tínhamos 2893 registros no total, então fazendo-se: $\frac{2600}{2893} \approx 89,87\%$. Arredondando esse valor para cima, concluímos que devemos manter cerca de 90% da quantidade original de registros.



(a) Com outliers

(b) Sem outliers

Figura 11: Distancia entre os registros

4.1.5 PCA

Uma das dificuldades mais frequentes tanto em problemas de classificação quanto de regressão é a quantidade de variáveis. Conforme esse número aumenta, mais os dados tendem a ficar afastados entre si nesse espaço. Assim, para manter uma representação mais acurada do espaço, é necessário uma quantidade de registros

maior. Outra situação em que reduzir a dimensionalidade é relevante é o cenário no qual há variáveis que pouco influenciam na predição da saída, aumentando ruído nos dados e o custo computacional (maior número de parâmetros no modelo, memória, etc.).

Há várias formas de mitigar esse problema. Uma delas é fazer uma seleção prévia de variáveis mais correlacionadas com a variável de saída. Outra, que será abordada a seguir, é efetuar uma projeção dos dados num espaço de menor dimensão, utilizando o PCA. Nesse novo espaço as variáveis não são correlacionadas entre si, logo a matriz de transformação deve ser ortogonal. Essa projeção tem como consequência perda de informações do dado original, contudo essa perda é controlada a partir da quantidade de vetores (dimensões) que formam a base desse novo espaço. Nesse trabalho, escolhemos os autovetores de forma que a variância total explicada é de 90%.

Seja $\hat{\Sigma} = \frac{X^T X}{N-1}$, com $X \in R^{Nxp}$. De forma geral, queremos encontrar uma matriz de transformação ortogonal $P \in R^{pxp}$ tal que Λ seja uma matriz diagonal:

$$\hat{\Sigma} P = P\Lambda \Rightarrow \hat{\Sigma} = P\Lambda P^T \tag{48}$$

, onde cada coluna de P representa um autovetor e um valor λ_i da diagonal de Λ um autovalor.

O cálculo da matriz P pode ser feito da seguinte maneira, vez obtida a decomposição espectral $X = USV^T$: $X^T X = (USV^T)^T (USV^T) = VS^T U^T USV^T$.

Como U e V são matrizes ortogonais, $U^T U = I$. Portanto:

$$X^T X = VS^T SV^T = VS^2 V^T \tag{49}$$

Comparando-se 49 e 48 temos que:

$$\text{V} = \text{P e } s_i = \sqrt{(N-1)\lambda_i}$$

Por fim basta considerarmos as "n"colunas de V (autovetores) correspondentes aos maiores "n"autovalores para compor a matriz de transformação.

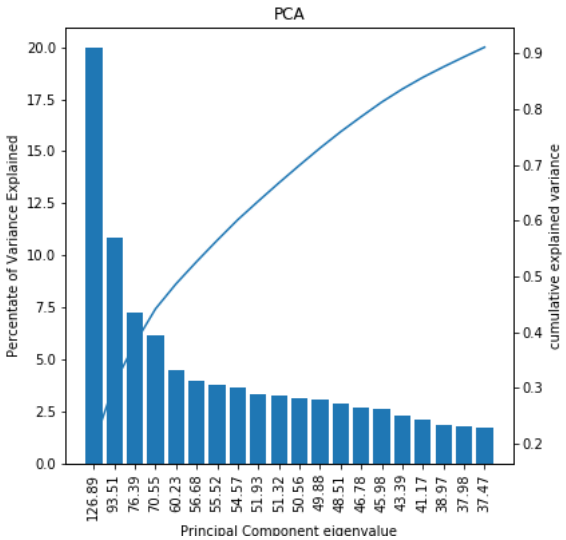


Figura 12: PCA considerando 90% da variância

4.1.6 MATRIZ DE CORRELAÇÃO

A Matriz de Correlação possibilita a análise simultânea da associação entre as variáveis. Através dela, é possível visualizar o grau de dependência linear entre duas variáveis no intervalo de (-1,1). Quanto mais próximo de -1 ou +1, mais forte a correlação entre as variáveis, sendo uma relação linear inversa ou direta, respectivamente.

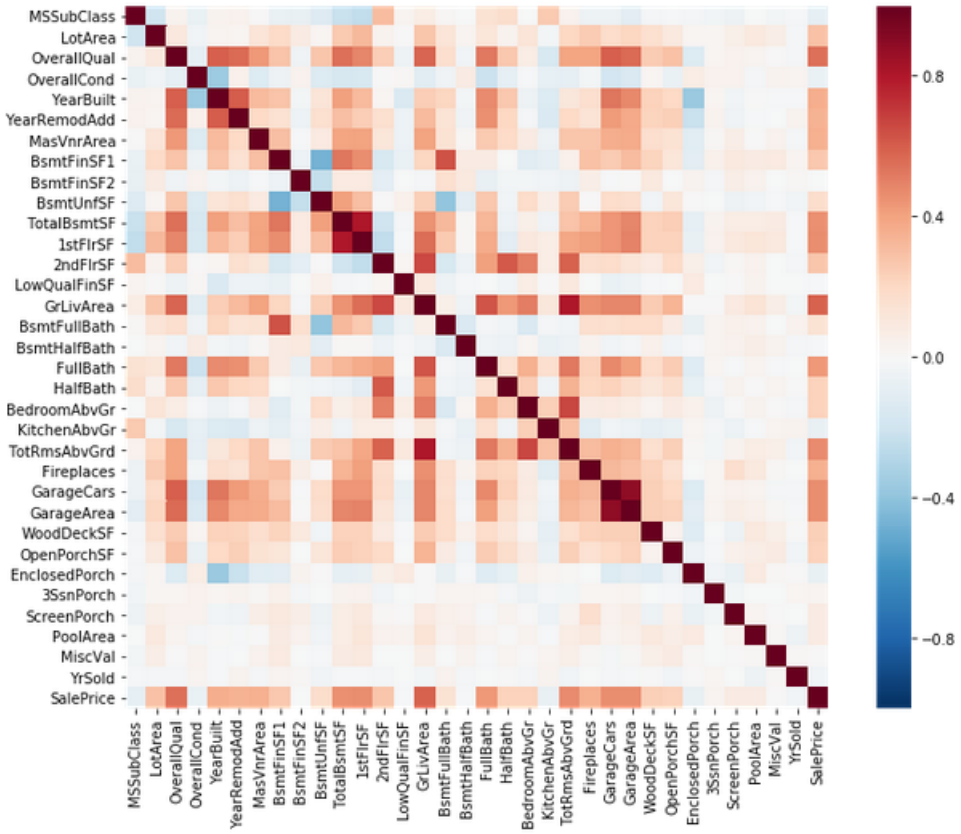


Figura 13: Matriz de Correlação

Na Figura 13 é possível ver que quanto mais vermelho, maior é a relação linear direta e quanto mais azul, maior é a relação linear inversa. Já as correlações próximas a cor branca, significa que as duas variáveis não tem nenhuma relação linear entre elas. Observa-se também que a diagonal principal da matriz está toda vermelha, ou seja, com valor +1, pois uma variável possui correlação máxima com ela mesma.

Por fim, é possível observar que nossa matriz de correlação possui muitas redundâncias, que podem prejudicar o resultado dos modelos. Portanto aplicaremos o ACP para reduzir não apenas a dimensionalidade mas também a correlação entre as variáveis e faremos uma comparação dos resultados dos classificadores com e sem ACP.

Porque as redundâncias e colinearidades prejudicam os modelos?

Esse problema pode ser descrito em termos de instabilidade dos menores autovalores e no efeito disso na matriz inversa de covariâncias, calculadas pelos algoritmos bayesiano linear e bayesiano quadrático. Além disso, quando temos um número muito maior de dimensões do que registros, o número de condição da matriz de covariância estimada, calculado como $\frac{\lambda_{max}}{\lambda_{min}}$ onde λ_{max} é o maior autovalor e λ_{min} o menor, tende a ir para infinito, de acordo com o livro Analysis of Large and Complex Data (referencia).

Treinando o LDA com o dataset completo obtemos o seguinte número de condição: -7.875.127.006.462.795. Já com a aplicação prévia do ACP nos dados, obtemos: 7,768. Ou seja, esses resultados sugerem que a matriz de covariância está mal condicionada no primeiro caso, indicando que pode haver variáveis colineares ou ainda que o número de registros é insuficiente para a quantidade de variáveis. Projetando os dados com o PCA não apenas eliminamos a colinearidade das variáveis mas também reduzimos significativamente o número de variáveis, o que pode explicar o porque do número de condição ser bem menor no segundo caso.

Numa análise mais a fundo, obtemos os seguintes autovalores máximo e mínimo para cada caso:

Dados	**Menor autovalor**	**Maior autovalor**
Sem ACP	-4.651e-16	3.663
Com ACP	0.552	4.292

Tabela 2: Autovalores da matriz de covariâncias estimada pelo LDA

Conforme pode ser observado da tabela acima, não houve grande mudança no maior autovalor considerando ambos os casos com e sem ACP. Contudo, o mesmo não vale para o menor autovalor: sem ACP obtivemos -4.651e-16, o que indica a existência de variáveis colineares, já com ACP 0.552.

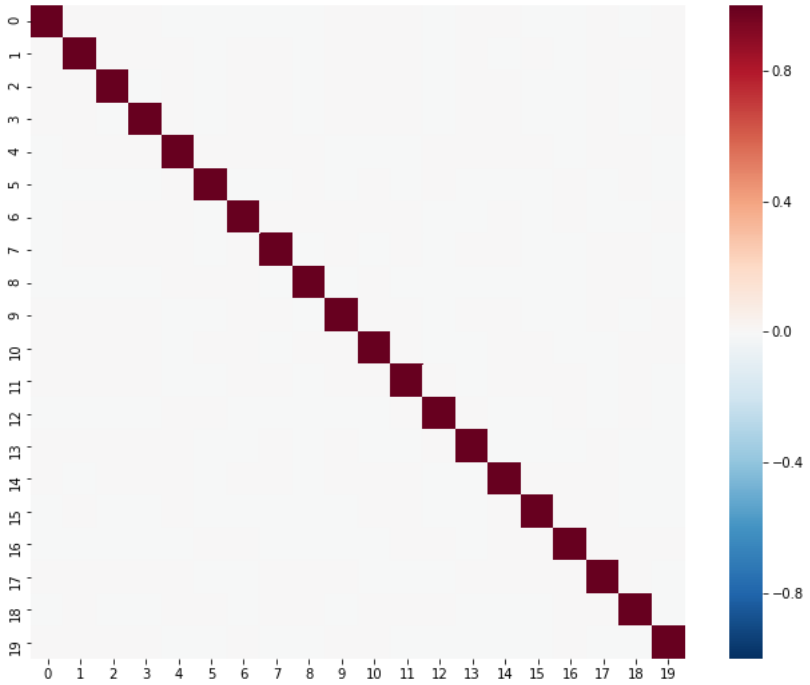


Figura 14: Matriz de Correlação das variáveis de entrada após ACP

Na Figura 14 é possível ver em primeiro lugar que a ACP efetuou uma redução de 33 para 20 variáveis, considerando uma variância de 90%. Essa redução significativa na quantidade de variáveis pode ser justificada pela grande redundância entre as variáveis, conforme analisamos observando a figura 13. Em segundo lugar, não há correlação entre as 20 variáveis do espaço projetado, conforme era de se esperar com a projeção feita pelo ACP.

4.1.7 CLASSES

Esse dataset vem preparado para um problema de regressão, porém como este problema é de classificação foi necessário acrescentar uma nova variável, sendo ela a variável de saída. Ela foi criada a partir da variável "SalePrice"e para ser balanceado, foi necessário dividir o dataset em partes iguais. Portanto, pegamos o dataset ordenado pelo "SalePrice"e decidimos colocar do menor valor até a mediana como barato e da mediana até o maior valor como caro.

4.2 RESULTADOS DOS MODELOS LINEARES

4.2.1 REGRESSÃO LOGÍSTICA

Neste modelo, foi utilizado o algorítimo LogisticRegression da biblioteca do scikit-learn. Como esse modelo possui um hiperparâmetro, sendo ele o parâmetro de regularização C, optamos por varia-lo de forma que possamos encontrar o melhor valor para ele. Com isso, foi decidido varia-lo utilizando os valores 0.001, 0.01,

0.1, 1, 10, 100 e 1000. Para isso, utilizamos o algoritmo GridSearchCV da biblioteca scikit-learn, em que para cada opção de parâmetro ele faz uma validação cruzada com o número de ciclos definido (no nosso caso, 10 ciclos) e no final, ele retorna qual o parâmetro que teve o melhor resultado. Após rodar esse algoritmo, o melhor resultado foi o de C igual a 1, ou seja, o valor default do algorítimo. Dessa forma, treinamos com esse valor de parâmetro e utilizando a validação cruzada. Após o treinamento, obtivemos os seguintes resultados de métricas:

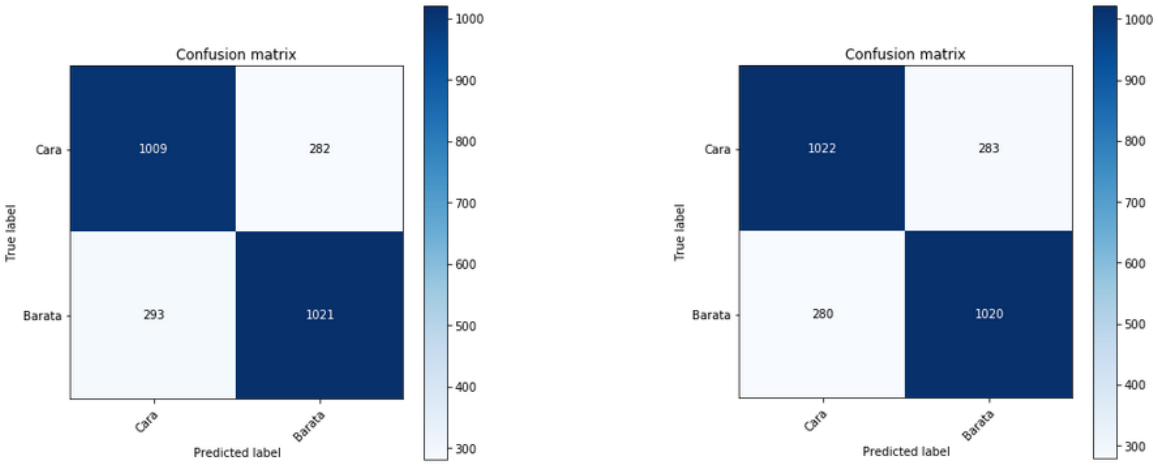
Métrica	**Valor Médio**	**Variância**
Acurácia	0.779	0.000468
Precisão	0.781	0.000303
Recall	0.775	0.001365
F1-Score	0.778	0.000598
Roc_Auc	0.779	0.000466

Tabela 3: Métricas da Regressão Logística sem ACP

Métrica	**Valor Médio**	**Variância**
Acurácia	0.784	0.000621
Precisão	0.783	0.000405
Recall	0.785	0.001722
F1-Score	0.784	0.000783
Roc_Auc	0.784	0.000619

Tabela 4: Métricas da Regressão Logística com ACP

Como pode ser visto na tabela acima (Tabela 3) obtivemos um resultado da média da acurácia de 0.779, ou seja, nosso modelo em média acertou 77.9% das duas classes juntas no total. Já na tabela do dataset com ACP (Tabela 4), temos que as métricas tiveram uma leve melhorada, porém a variância delas aumentaram um pouquinho. Além disso, podemos visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 15: Matriz Confusão da Regressão Logística

Através dela, podemos visualizar na Matriz 15a (Matriz 15b) que 38.7% (39.2%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 39.2% (39.2%) foi verdadeiro negativo, ou seja, previu barata quando a classe esperada era barata; 10.8% (10.9%) foi falso positivo, ou seja, previu cara quando a classe esperada era barata; e 11.3% (10.7%) foi falso negativo, ou seja, previu barata quando a classe esperada

era cara. Com isso, podemos perceber que com a aplicação da ACP não houve uma grande mudança, apenas um pequeno aumento na quantidade de verdadeiros positivos.

4.2.2 NAIVE BAYES

Neste modelo, foi utilizado o algorítimo GaussianNB da biblioteca do scikit-learn. Como esse modelo não possui hiperparâmetros, o modelo não pode ser variado, sendo apenas treinado dentro da validação cruzada de 10 ciclos. Após o treinamento, obtivemos os seguintes resultados de métricas:

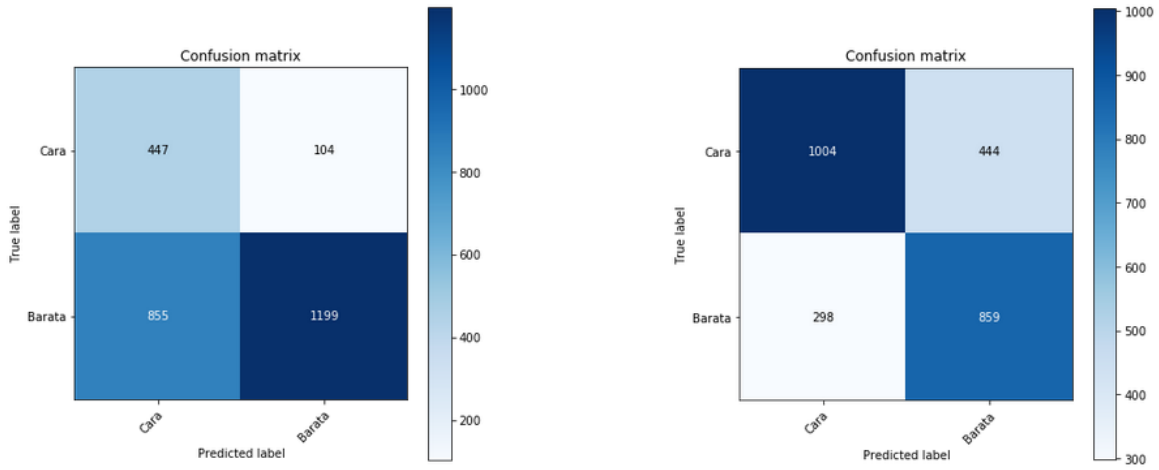
Métrica	Valor Médio	Variância
Acurácia	0.632	0.001812
Precisão	0.917	0.011702
Recall	0.344	0.054156
F1-Score	0.446	0.025894
Roc_Auc	0.632	0.001772

Tabela 5: Métricas do Naive Bayes sem ACP

Métrica	Valor Médio	Variância
Acurácia	0.715	0.001665
Precisão	0.697	0.002096
Recall	0.771	0.005299
F1-Score	0.729	0.001745
Roc_Auc	0.715	0.001664

Tabela 6: Métricas do Naive Bayes com ACP

Como pode ser visto na tabela acima (Tabela 5) obtivemos um resultado da média da acurácia de 0.632, ou seja, nosso modelo em média acertou 63.2% das duas classes juntas no total. Já na tabela do dataset com ACP (Tabela 6), temos que além das métricas terem melhorado bastante, também houve uma redução na variância de todas as métricas. Além disso, podemos visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 16: Matriz Confusão do Naive Bayes

Através dela, podemos visualizar na Matriz 16a (Matriz 16b) que 17.2% (38.5%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 46.0% (33.0%) foi verdadeiro negativo, ou seja, previu

barata quando a classe esperada era barata; 4.0% (17.1%) foi falso positivo, ou seja, previu cara quando a classe esperada era barata; e 32.8% (11.4%) foi falso negativo, ou seja, previu barata quando a classe esperada era cara. Com isso, podemos perceber que, com a utilizando da ACP, houve uma boa melhora na quantidade de verdadeiro positivos (mesmo perdendo um pouco na quantidade de verdadeiro negativo).

O fato dessa melhora ser tão significativa pode ser analisada a partir do modo como esse modelo efetua o ajuste dos parâmetros: considerando as variáveis de entrada independentes. Contudo, conforme vimos na matriz de correlação, essa suposição não é verdadeira. Com a aplicação prévia do PCA, a independência estatística é constatada no espaço projetado, a suposição do algoritmo é verdadeira, levando a melhora do desempenho.

4.2.3 LINEAR DISCRIMINANT ANALYSIS

Neste modelo, foi utilizado o algorítimo LinearDiscriminantAnalysis da biblioteca do scikit-learn. Assim como o naive bayes, esse modelo não possui hiperparâmetros, e portanto não pode ser variado, sendo apenas treinado dentro da validação cruzada de 10 ciclos. Após o treinamento, obtivemos os seguintes resultados de métricas:

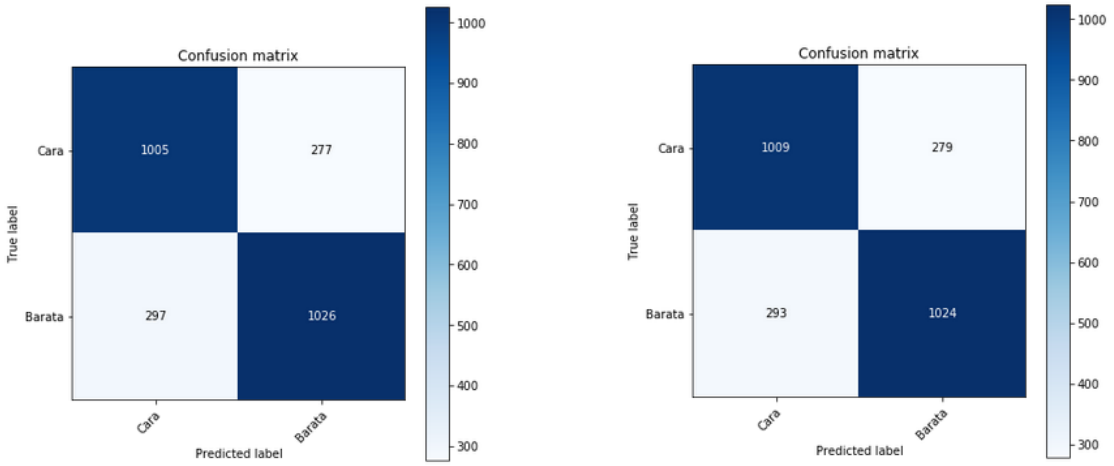
Métrica	**Valor Médio**	**Variância**
Acurácia	0.780	0.000563
Precisão	0.784	0.000249
Recall	0.772	0.002070
F1-Score	0.777	0.000790
Roc_Auc	0.780	0.000560

Tabela 7: Métricas do LDA sem ACP

Métrica	**Valor Médio**	**Variância**
Acurácia	0.780	0.000826
Precisão	0.783	0.000492
Recall	0.775	0.002193
F1-Score	0.779	0.001048
Roc_Auc	0.780	0.000824

Tabela 8: Métricas do LDA com ACP

Como pode ser visto na tabela acima (Tabela 7) obtivemos um resultado da média da acurácia de 0.780, ou seja, nosso modelo em média acertou 78.0% das duas classes juntas no total. Já na tabela do dataset com ACP (Tabela 8), temos que as métricas não mudaram muito em relação as métricas do dataset sem ACP. Já nas variâncias, tivemos um leve aumento. Além disso, podemos visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 17: Matriz Confusão do LDA

Através dela, podemos visualizar na Matriz 17a (Matriz 17b) que 38.6% (38.7%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 39.4% (39.3%) foi verdadeiro negativo, ou seja, previu barata quando a classe esperada era barata; 10.6% (10.7%) foi falso positivo, ou seja, previu cara quando a classe esperada era barata; e 11.4% (11.3%) foi falso negativo, ou seja, previu barata quando a classe esperada era cara. Com isso, podemos perceber que, com a utilização da ACP, não houve uma melhora muito significativa.

4.2.4 QUADRATIC DISCRIMINANT ANALYSIS

Neste modelo, foi utilizado o algorítimo QuadraticDiscriminantAnalysis da biblioteca do scikit-learn. Assim como o naive bayes e o LDA, esse modelo não possui hiperparâmetros, e portanto não pode ser variado, sendo apenas treinado dentro da validação cruzada de 10 ciclos. Após o treinamento, obtivemos os seguintes resultados de métricas:

Métrica	**Valor Médio**	**Variância**
Acurácia	0.553	0.002044
Precisão	0.547	0.001923
Recall	0.687	0.047091
F1-Score	0.589	0.010363
Roc_Auc	0.552	0.002081

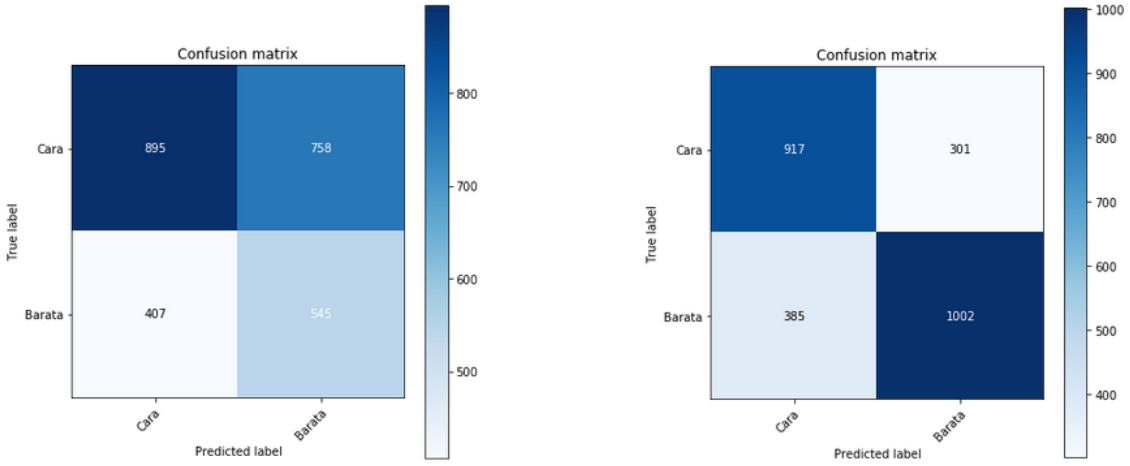
Tabela 9: Métricas do QDA sem ACP

Métrica	**Valor Médio**	**Variância**
Acurácia	0.737	0.002515
Precisão	0.776	0.006460
Recall	0.704	0.017254
F1-Score	0.723	0.004134
Roc_Auc	0.737	0.002470

Tabela 10: Métricas do QDA com ACP

Como pode ser visto na tabela acima (Tabela 9) obtivemos um resultado da média da acurácia de 0.553, ou seja, nosso modelo em média acertou 55.3% das duas classes juntas no total. Já na tabela do dataset com ACP (Tabela 10), temos que as métricas melhoraram bastante em comparação com o dataset sem ACP e as variâncias aumentarem de leve em algumas métricas e diminuíram de leve em outras. Além disso, podemos

visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 18: Matriz Confusão do QDA

Através dela, podemos visualizar na Matriz 18a (Matriz 18b) que 34.4% (35.2%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 20.9% (38.5%) foi verdadeiro negativo, ou seja, previu barata quando a classe esperada era barata; 29.1% (11.5%) foi falso positivo, ou seja, previu cara quando a classe esperada era barata; e 15.6% (14.8%) foi falso negativo, ou seja, previu barata quando a classe esperada era cara. Com isso, podemos perceber que, com a utilização da ACP, houve uma boa melhora na quantidade de verdadeiro negativos e uma leve melhora na quantidade de verdadeiro positivos.

Conforme abordamos na descrição teórica dos modelos, o QDA possui uma enorme quantidade de parâmetros a serem ajustados. Isso porque ele estima a matriz de covariância para cada classe. Como os dados são insuficientes para o ajuste adequado, ele obteve péssimo resultado. No entanto ao efetuarmos a projeção com ACP, não apenas reduzimos o número de variáveis a serem estimadas, como também a matriz de covariâncias fica diagonal, facilitando o cálculo.

4.2.5 RANDOM FOREST

Neste modelo, foi utilizado o algorítimo RandomForestClassifier da biblioteca do scikit-learn. Como esse modelo possui um hiperparâmetro, sendo ele o número de arvores, optamos por varia-lo de forma que possamos encontrar o melhor valor para ele. Com isso, foi decidido varia-lo de 1 árvore até 100 árvores. Para isso, utilizamos o algoritmo GridSearchCV da biblioteca scikit-learn, em que para cada opção de parâmetro ele faz uma validação cruzada com o número de ciclos definido (no nosso caso, 10 ciclos) e no final, ele retorna qual o parâmetro que teve o melhor resultado. Após rodar esse algoritmo, o melhor resultado foi o de 59 árvores. Dessa forma, treinamos com esse valor de parâmetro e utilizando a validação cruzada. Após o treinamento, obtivemos os seguintes resultados de métricas:

Métrica	**Valor Médio**	**Variância**
Acurácia	0.786	0.001009
Precisão	0.797	0.001488
Recall	0.769	0.002639
F1-Score	0.782	0.001145
Roc_Auc	0.786	0.001007

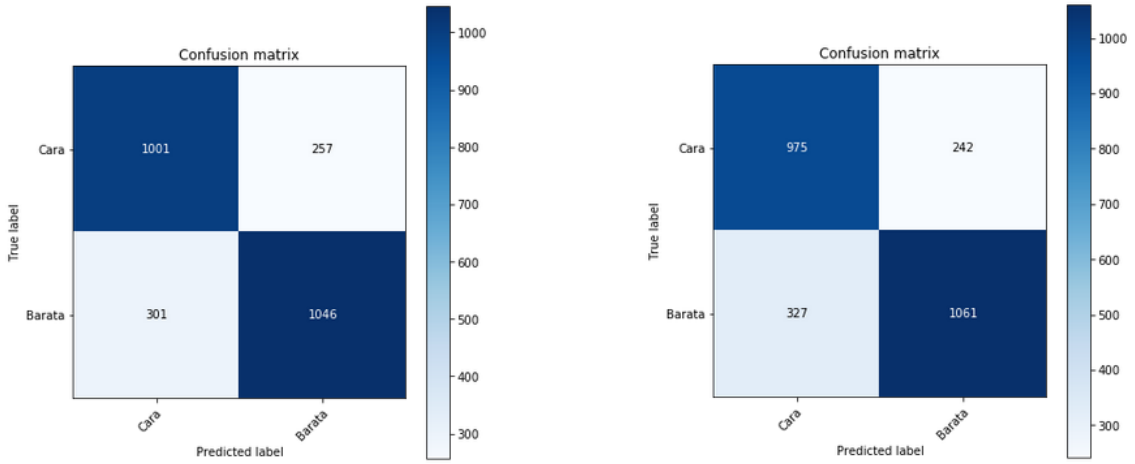
Tabela 11: Métricas do Random Forest sem ACP

Métrica	**Valor Médio**	**Variância**

Acurácia	0.782	0.001214
Precisão	0.800	0.000698
Recall	0.749	0.002766
F1-Score	0.773	0.001608
Roc_Auc	0.782	0.001209

Tabela 12: Métricas do Random Forest com ACP

Como pode ser visto na tabela acima (Tabela 11) obtivemos um resultado da média da acurácia de 0.786, ou seja, nosso modelo em média acertou 78.6% das duas classes juntas no total. Já na tabela do dataset com ACP (Tabela 12), temos que as métricas não mudaram muito tanto nas métricas quanto nas variância. Além disso, podemos visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 19: Matriz Confusão do Random Forest

Através dela, podemos visualizar na Matriz 19a (Matriz 19b) que 38.4% (37.4%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 40.2% (40.7%) foi verdadeiro negativo, ou seja, previu barata quando a classe esperada era barata; 9.9% (9.3%) foi falso positivo, ou seja, previu cara quando a classe esperada era barata; e 11.5% (12.6%) foi falso negativo, ou seja, previu barata quando a classe esperada era cara. Com isso, podemos perceber que, utilizando a ACP, não houve uma grande mudança nos resultados.

4.2.6 SVM LINEAR

Neste modelo, foi utilizado o algorítimo SVC da biblioteca do scikit-learn. Como esse modelo possui o parâmetro de regularização C e o tipo de kernel como hiperparâmetros, optamos por varia-los de forma que possamos encontrar o melhor conjunto de parâmetros. Com isso, foi decidido variar o parâmetro de regularização utilizando os valores 0.001, 0.01, 0.1, 1, 10, 100 e 1000 e como estamos trabalhando com o SVM Linear, iremos usar o kernel linear. Para isso, utilizamos o algoritmo GridSearchCV da biblioteca scikit-learn, em que para cada conjunto de parâmetros ele faz uma validação cruzada com o número de ciclos definido (no nosso caso, 10 ciclos) e no final, ele retorna qual o conjunto de parâmetros que teve o melhor resultado. Após rodar esse algoritmo, o melhor resultado foi o de C igual a 0.1. Dessa forma, treinamos com esse conjunto de parâmetros e utilizando a validação cruzada. Após o treinamento, obtivemos os seguintes resultados de métricas:

Métrica	**Valor Médio**	**Variância**
Acurácia	0.787	0.000696
Precisão	0.792	0.000354
Recall	0.777	0.001959

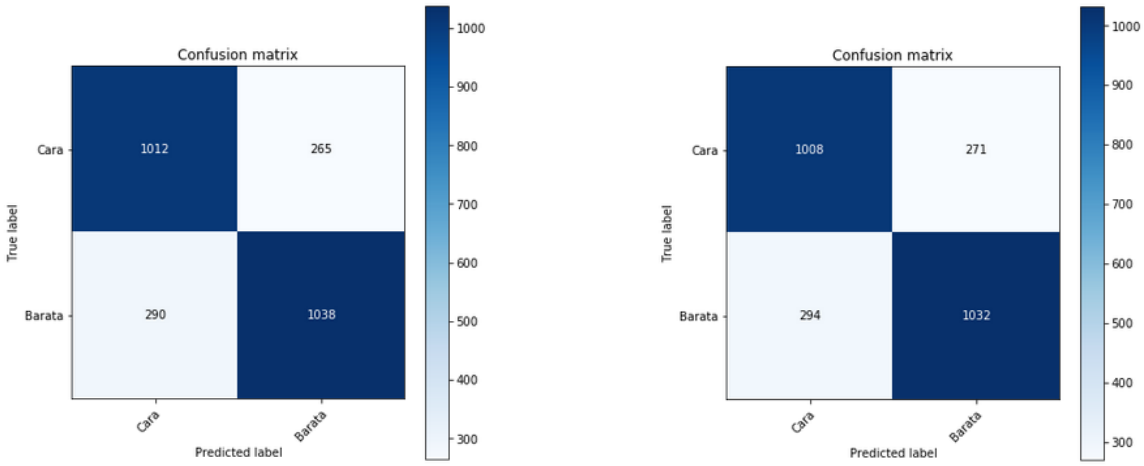
F1-Score	0.784	0.000889
Roc_Auc	0.787	0.000695

Tabela 13: Métricas do SVM Linear sem ACP

Métrica	**Valor Médio**	**Variância**
Acurácia	0.783	0.000882
Precisão	0.788	0.000571
Recall	0.774	0.001943
F1-Score	0.781	0.001070
Roc_Auc	0.783	0.000880

Tabela 14: Métricas do SVM Linear com ACP

Como pode ser visto na tabela acima (Tabela 13) obtivemos um resultado da média da acurácia de 0.787, ou seja, nosso modelo em média acertou 78.7% das duas classes juntas no total. Já na tabela do dataset com ACP (Tabela 14), temos que as métricas não mudaram muito tanto nas métricas quanto nas variância. Além disso, podemos visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 20: Matriz Confusão do SVM Linear

Através dela, podemos visualizar na Matriz 20a (Matriz 20b) que 38.9% (38.7%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 39.8% (39.6%) foi verdadeiro negativo, ou seja, previu barata quando a classe esperada era barata; 10.2% (10.4%) foi falso positivo, ou seja, previu cara quando a classe esperada era barata; e 11.1% (11.3%) foi falso negativo, ou seja, previu barata quando a classe esperada era cara. Com isso, podemos perceber que, utilizando a ACP, não houve uma grande mudança nos resultados.

4.3 RESULTADOS DOS MODELOS NÃO LINEARES

4.3.1 RBF

Este modelo foi implementado por nós mesmos, a partir das classes bases do classificadores do scikit-learn BaseEstimator e ClassifierMixin. Além disso, internamente o RBF utiliza o algoritmo k-médias para encontrar os centros das funções de base. O modelo possui apenas um único hiperparâmetro que é a quantidade de funções de base. Variando esse valor de 1 a 50, encontramos que 21 era a quantidade ótima de funções de base a partir do GridSearch. Após o treinamento com esse valor, usando a validação cruzada, obtivemos os seguintes resultados para as métricas:

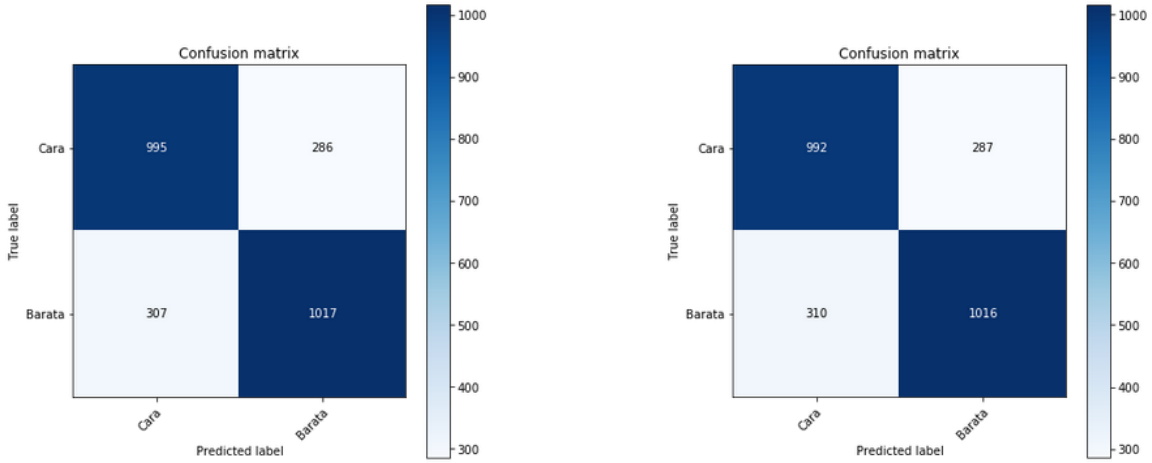
Métrica	Valor Médio	Variância
Acurácia	0.772	0.000844
Precisão	0.776	0.000439
Recall	0.764	0.002601
F1-Score	0.770	0.001171
Roc_Auc	0.772	0.000840

Tabela 15: Métricas do RBF sem ACP

Métrica	Valor Médio	Variância
Acurácia	0.771	0.000907
Precisão	0.775	0.000542
Recall	0.762	0.002916
F1-Score	0.768	0.001267
Roc_Auc	0.770	0.000904

Tabela 16: Métricas do RBF com ACP

Da tabela acima (Tabela 15) percebemos que todas as métricas possuem baixa variância e com o RBF conseguimos obter 77.2% de acurácia. Já na tabela do dataset com ACP (Tabela 16), temos que as métricas não mudaram muito, tanto nos valores das métricas quantos nas variâncias. Além disso, podemos visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 21: Matriz Confusão do RBF

Através dela, podemos visualizar na Matriz 21a (Matriz 21b) que 38.2% (38.1%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 39.0% (39.0%) foi verdadeiro negativo, ou seja, previu barata quando a classe esperada era barata; 11.0% (11.0%) foi falso positivo, ou seja, previu cara quando a classe esperada era barata; e 11.8% (11.9%) foi falso negativo, ou seja, previu barata quando a classe esperada era cara. Com isso, podemos perceber que, ao utilizar a ACP, não houve uma grande mudança nos resultados.

4.3.2 SVM NÂO LINEAR

Neste modelo, foi utilizado o mesmo algoritimo do SVM Linear, o algorítimo SVC da biblioteca do scikit-learn. A diferença é que no não linear a gente varia os tipos de kernels. Como isso, optamos por varia o parâmetro de regularização C e o tipo de kernel de forma que possamos encontrar o melhor conjunto de parâmetros. Com

isso, foi decidido variar o parâmetro de regularização utilizando os valores 0.001, 0.01, 0.1, 1, 10, 100 e 1000 e o tipo de kernel utilizando rbf e sigmoide. Para isso, utilizamos o algoritmo GridSearchCV da biblioteca scikit-learn, em que para cada conjunto de parâmetros ele faz uma validação cruzada com o número de ciclos definido (no nosso caso, 10 ciclos) e no final, ele retorna qual o conjunto de parâmetros que teve o melhor resultado. Após rodar esse algoritmo, o melhor resultado foi o de C igual a 0.1 juntamente com o kernel rbf. Dessa forma, treinamos com esse conjunto de parâmetros e utilizando a validação cruzada. Após o treinamento, obtivemos os seguintes resultados de métricas:

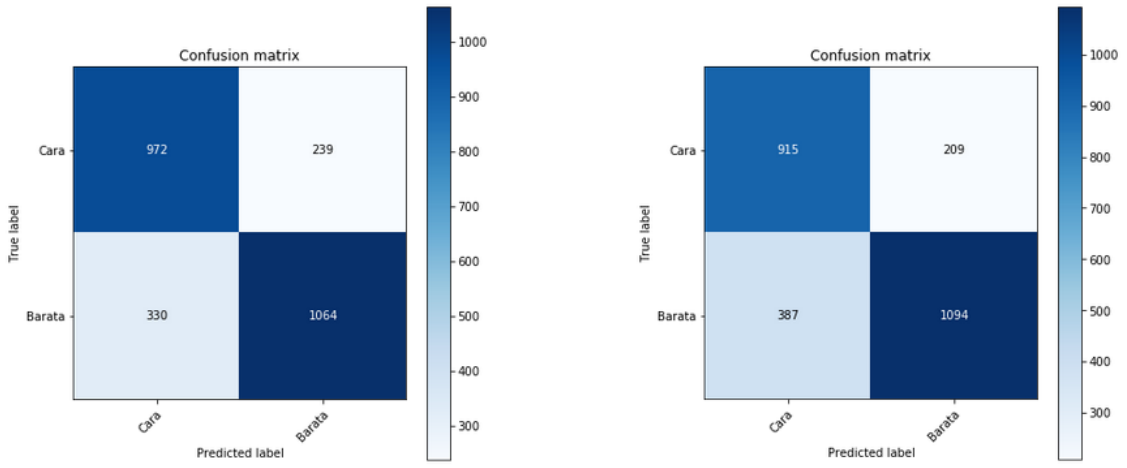
Métrica	**Valor Médio**	**Variância**
Acurácia	0.781	0.000599
Precisão	0.802	0.000271
Recall	0.747	0.001984
F1-Score	0.773	0.000869
Roc_Auc	0.782	0.000595

Tabela 17: Métricas do SVM Não Linear sem ACP

Métrica	**Valor Médio**	**Variância**
Acurácia	0.771	0.000943
Precisão	0.813	0.000619
Recall	0.702	0.002529
F1-Score	0.753	0.001397
Roc_Auc	0.771	0.000935

Tabela 18: Métricas do SVM Não Linear com ACP

Como pode ser visto na tabela acima (Tabela 17) obtivemos um resultado da média da acurácia de 0.781, ou seja, nosso modelo em média acertou 78.1% das duas classes juntas no total. Já na tabela do dataset com ACP (Tabela 18), temos que as métricas mudaram muito pouco e a variância aumentou um pouquinho também. Além disso, podemos visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 22: Matriz Confusão do SVM Não Linear

Através dela, podemos visualizar na Matriz 22a (Matriz 22b) que 37.3% (35.1%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 40.8% (42.0%) foi verdadeiro negativo, ou seja, previu barata quando a classe esperada era barata; 9.2% (8.0%) foi falso positivo, ou seja, previu cara quando a classe

esperada era barata; e 12.7% (14.9%) foi falso negativo, ou seja, previu barata quando a classe esperada era cara. Com isso, podemos perceber que, utilizando a ACP, não houve uma grande mudança nos resultados. Melhorou um pouquinho em uns e piorou um pouquinho em outros.

4.3.3 REDE NEURAL MULTI LAYER PERCEPTRON (MLP)

Neste modelo, foi utilizado o algorítimo MLPClassifier da biblioteca do scikit-learn. Como esse modelo possui a quantidade de camadas e neuronios (em cada camada) como hiperparâmetros, optamos por varia-los de forma que possamos encontrar o melhor conjunto de parâmetros. Como a quantidade de grupos que podemos forma é muito grande, ficaria muito custoso deixar todas as possibilidades. Portanto, foi decidido utilizar 34 neurônios na primeira camada, 1 neurônio na ultima camada e apenas uma camada intermediaria e variar a quantidade de neurônios de 1 até 50. Para isso, utilizamos o algoritmo GridSearchCV da biblioteca scikit-learn, em que para cada conjunto de parâmetros ele faz uma validação cruzada com o número de ciclos definido (no nosso caso, 10 ciclos) e no final, ele retorna qual o conjunto de parâmetros que teve o melhor resultado. Após rodar esse algoritmo, o melhor resultado foi o de 30 neurônios na camada intermediaria. Dessa forma, treinamos com esse conjunto de parâmetros e utilizando a validação cruzada. Após o treinamento, obtivemos os seguintes resultados de métricas:

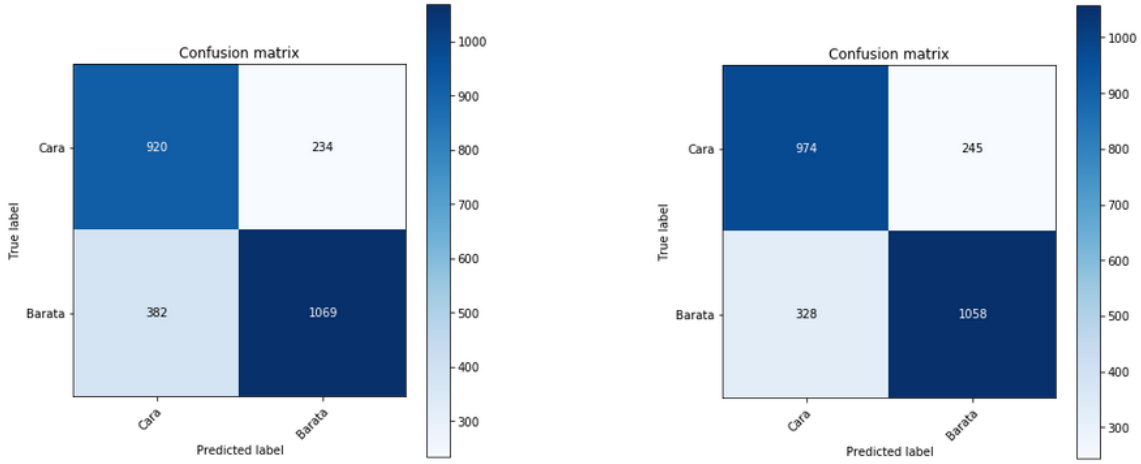
Métrica	**Valor Médio**	**Variância**
Acurácia	0.763	0.000879
Precisão	0.800	0.002065
Recall	0.707	0.001412
F1-Score	0.749	0.000805
Roc_Auc	0.763	0.000876

Tabela 19: Métricas do MLP sem ACP

Métrica	**Valor Médio**	**Variância**
Acurácia	0.780	0.000409
Precisão	0.800	0.000472
Recall	0.748	0.001925
F1-Score	0.772	0.000621
Roc_Auc	0.780	0.000407

Tabela 20: Métricas do MLP com ACP

Como pode ser visto na tabela acima (Tabela 19) obtivemos um resultado da média da acurácia de 0.763, ou seja, nosso modelo em média acertou 76.3% das duas classes juntas no total. Já na tabela do dataset com ACP (Tabela 20), temos que as métricas não mudaram muito, porém houve uma diminuição na variância na maioria das métricas. Além disso, podemos visualizar a Matriz de Confusão composto pelo resultado de cada ciclo da validação cruzada.



(a) Sem ACP

(b) Com ACP

Figura 23: Matriz Confusão do MLP

Através dela, podemos visualizar na Matriz 23a (Matriz 23b) que 35.3% (37.4%) foi verdadeiro positivo, ou seja, previu cara quando a classe esperada era cara; 41.0% (40.6%) foi verdadeiro negativo, ou seja, previu barata quando a classe esperada era barata; 9.0% (9.4%) foi falso positivo, ou seja, previu cara quando a classe esperada era barata; e 14.7% (12.6%) foi falso negativo, ou seja, previu barata quando a classe esperada era cara. Com isso, podemos perceber que, utilizando a ACP, não houve uma grande mudança, apesar do leve aumento de verdadeiros positivos e uma leva diminuição de verdadeiros negativos.

4.4 COMPARAÇÃO DOS RESULTADOS

Nos boxplots a seguir, vemos que, com a exceção do QDA, todos os modelos possuem baixa variância. A alta variância do QDA deve-se ao fato de que o mesmo possui uma quantidade de parâmetros muito grande a serem ajustados em relação à quantidade de dados. O Naive Bayes, por outro lado, não obteve resultados tão satisfatórios. Isso porque as variáveis possuem correlação e o classificador em questão considera a independência estatística entre elas.

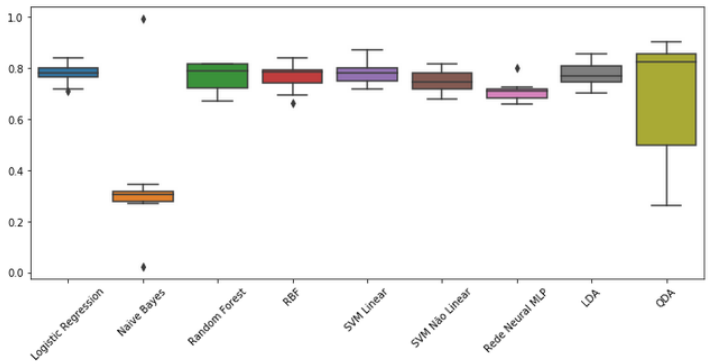


Figura 24: Boxplot para comparação da métrica Recall dos modelos utilizados sem ACP

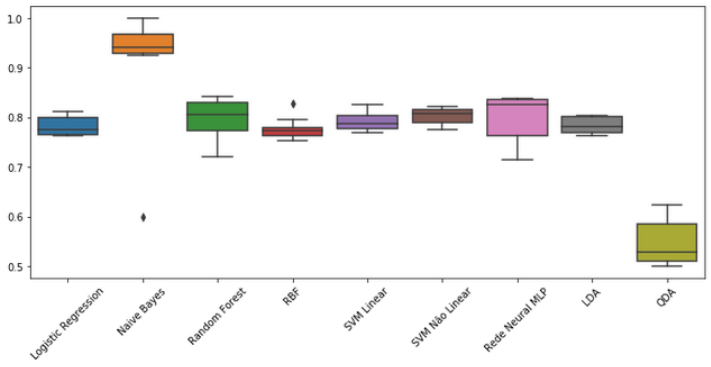


Figura 25: Boxplot para comparação da métrica precisão dos modelos utilizados sem ACP

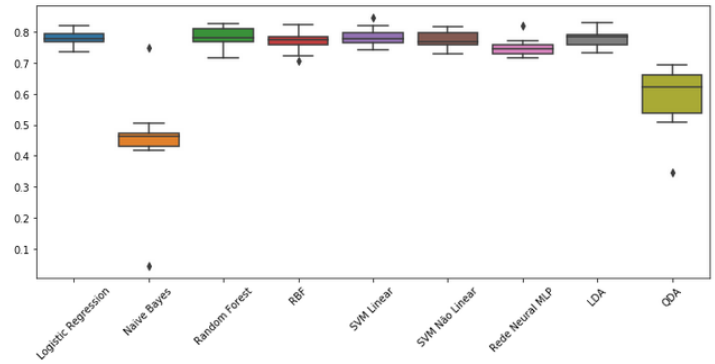


Figura 26: Boxplot para comparação da métrica f1-score dos modelos utilizados sem ACP

Com isso, sem a utilização da ACP, podemos dizer que para a métrica de recall os melhores modelos foram Regressão Logística e SVM Linear. Já para a métrica precisão, foram Naive Bayes e SVM Não Linear. E para a métrica f1-score, foram a Regressão Logística ,Random Forest e o SVM Linear.

É possivel observar, nos boxplots a seguir, que ao efetuar o ACP para a redução de dimensionalidade obtivemos resultados melhores com o Naive Bayes e pro QDA. O primeiro faz a suposição de variáveis independentes, enquanto o segundo calcula a matriz de covariância para cada classe. Com essa transformação feita pela ACP, elimina-se a correlação entre variáveis no novo espaço de projeção e reduz a quantidade de parâmetros a serem calculados, podendo ser essa a causa da melhora de desempenho por esses algoritmos nesse segundo cenário.

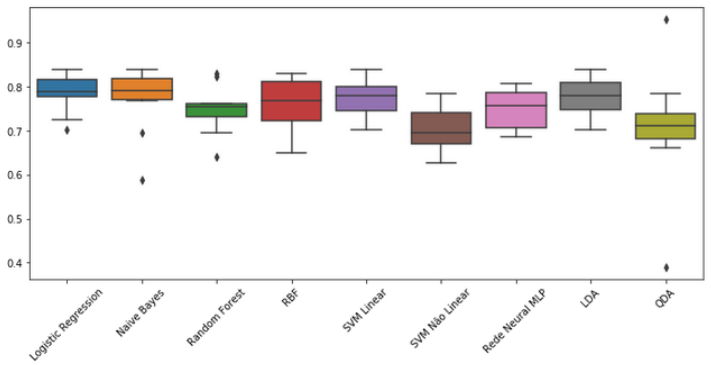


Figura 27: Boxplot para comparação da métrica Recall dos modelos utilizados com ACP

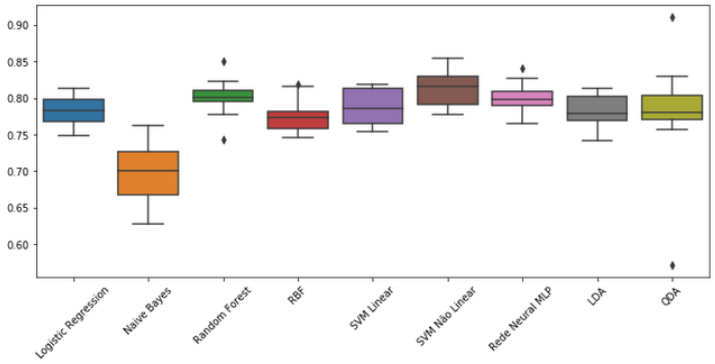


Figura 28: Boxplot para comparação da métrica precisão dos modelos utilizados com ACP

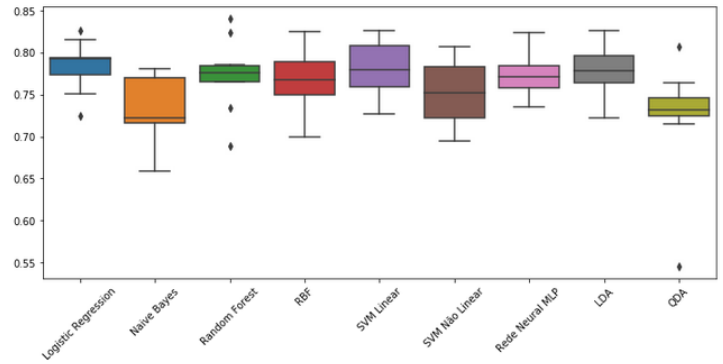


Figura 29: Boxplot para comparação da métrica f1-score dos modelos utilizados com ACP

Já com a utilização da ACP, podemos dizer que para a métrica de recall os melhores modelos foram Regressão Logística e LDA. Para métrica de precisão, foram Random Forest e SVM Não Linear. E para a métrica f1-score, foram Regressão Logística e SVM Linear.

	Name	Logistic Regression	Naive Bayes	Random Forest	RBF	SVM Linear	SVM Não Linear	Rede Neural MLP	LDA	QDA
0	acuracia	0.77929	0.63187	0.78582	0.77239	0.78698	0.78159	0.76354	0.77968	0.55285
1	precisão	0.78154	0.91715	0.79751	0.77637	0.79219	0.80237	0.80013	0.78376	0.54755
2	recall	0.77506	0.34359	0.76894	0.76434	0.77736	0.74664	0.70666	0.77199	0.68719
3	f1_score	0.77796	0.44565	0.78166	0.76977	0.78434	0.77304	0.74927	0.77733	0.58879
4	roc_auc	0.77930	0.63206	0.78584	0.77244	0.78698	0.78161	0.76353	0.77968	0.55265

Figura 30: Tabela de comparação das métricas dos modelos utilizados sem ACP

	Name	Logistic Regression	Naive Bayes	Random Forest	RBF	SVM Linear	SVM Não Linear	Rede Neural MLP	LDA	QDA
0	acuracia	0.78390	0.71520	0.78161	0.77085	0.78314	0.77123	0.78006	0.78046	0.73672
1	precisão	0.78307	0.69688	0.80030	0.77543	0.78778	0.81354	0.79973	0.78312	0.77654
2	recall	0.78506	0.77132	0.74897	0.76202	0.77429	0.70287	0.74817	0.77507	0.70456
3	f1_score	0.78369	0.72932	0.77347	0.76795	0.78071	0.75364	0.77222	0.77867	0.72355
4	roc_auc	0.78393	0.71533	0.78162	0.77091	0.78315	0.77121	0.78011	0.78048	0.73693

Figura 31: Tabela de comparação das métricas dos modelos utilizados com ACP

5 CONCLUSÕES

Depois de avaliar todos os modelos com e sem ACP, vemos que nosso problema é linearmente separável, dado que obtivemos resultados satisfatórios utilizando os classificadores LDA, SVM Linear e Regressão Logística.

De forma geral e considerando a simplicidade do modelo, podemos dizer que a Regressão Logística com ACP foi a que obteve o melhor desempenho, na maioria das métricas e principalmente na f1-score, que foi escolhido para medir o desempenho do modelo, por poder buscar um equilíbrio entre recall e precisão. Apesar do f1-score do SVM Linear sem ACP ter dado um f1-score levemente maior, além do f1-score, foi considerado também o custo do modelo e, portanto, como a regressão linear é um modelo mais simples que o SVM linear, ele foi escolhido como o melhor modelo. Por outro lado, se analisarmos individualmente as métricas recall e precisão veremos que os classificadores com melhores resultados continua sendo Regressão Logística com ACP para o recall e Naive Bayes sem ACP para a precisão.

Um estudo realizado neste trabalho é a comparação entre os modelos com e sem ACP. Os resultados mostram que para a maioria dos modelos, os resultados não tiveram uma grande diferença. Porém, outros, como o Naive Bayes e o QDA, tiverem uma melhora significativa, devida eliminação da correlação entre variáveis no novo espaço de projeção pela ACP.

6 REFERÊNCIAS

http://mariofilho.com/as-metricas-mais-populares-para-avaliar-modelos-de-machine-learning/

https://medium.com/@paulo_sampaio/entendendo-k-means-agrupando-dados-e-tirando-camisas-e90ae3157c17

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

Analysis of Large and Complex Data - editado por Adalbert F.X. Wilhelm, Hans A. Kestler