Gabriel Stankevix Follow
Aug 29, 2019 · 17 min read

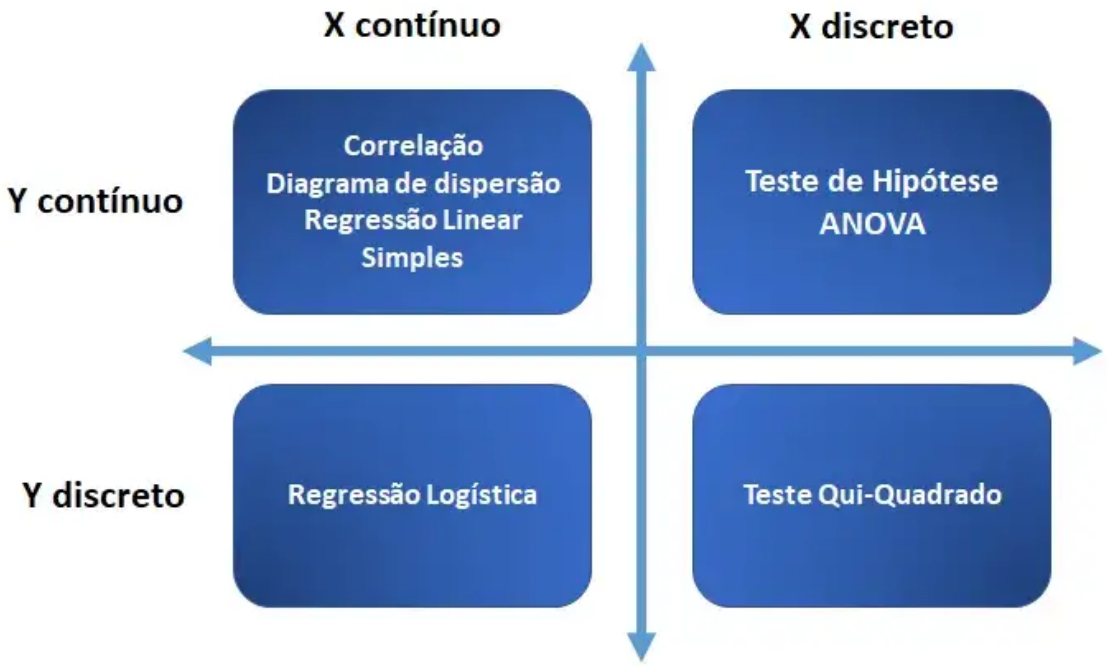
Regressão Logística em R e Python (PyTools)

Seguindo com os trabalhos em R, em mais uma demanda da área de negocio me proporcionou, foi um cenário para aplicação de Regressão Logística.

Uma das primeiras opções que busquei foi a correlação linear (Pearson, Spearman e Kendall), mas os dados de saídas demonstravam uma combinação binaria. Logo a correlação não se mostrou como o método de trabalho ideal.

Logo, antes de sair buscando o melhor método ou ferramenta que utilizaremos em uma

Construindo a figura abaixo, representamos um mapa de análise estatística onde é possível escolher a ferramenta quantitativa mais indicada para cada par entrada/saída de dados que trabalharemos.



A depender da amostra que trabalhamos as variáveis de entrada X podem ser continuas e com uma variável de saída Y discreta (*target*).

Agora voltando para o meu papel como cientista de dados de uma industria farmacêutica. Uma das tarefas determinadas era achar o valor real dos serviços prestados, criando um indicador para o corpo executivo entender a relação entre os pacientes e serviços. Meu trabalho era basicamente identificar se o serviço era consumido, ou não, e o quanto isso impactava na empresa. A minha amostra de dados era bem ampla com dados de mercado de mais de 10 anos.

Avaliando e estudando, compreendi que a melhor solução seria utilizar o aprendizado de máquina para resolver o problema. Claramente, o algoritmo de regressão linear não

Portanto, em vez de tentar prever exatamente se os pacientes consumiram um serviço ou não, podemos calcular a probabilidade ou a possibilidade de um paciente aderir a este serviço. Desta forma, podemos definir uma probabilidade entre 0 e 1, para encontrar pacientes com maior a possibilidade de consumo.

Revisando Regressão Logística

É muito importante antes de falarmos sobre regressão logística analisarmos sobre as suas diferenças com a regressão linear.

Não abordaremos a questão do linear ou não linear mas imagina que você tem uma regressão linear e que trace um gráfico de dispersão onde apresente uma variável dependente e outra independente, criando assim uma "nuvem" de pontos. Em uma regressão linear, trassaremos uma reta que tente a alcançar a maior parte dos pontos enquanto na regressão logística será trassado uma curva em S.

A segunda diferença esta relacionado a variável dependente, na regressão linear a variável dependente é uma variável continua portanto é incontável (pode assumir quaisquer valores) enquanto na regressão logística será uma variável categoria podendo assumir um conjunto de dados limitados de possibilidades([sim, não], [1,0],[ótimo, regular, ruim]).

A interpretação da regressão linear é bastante direta pois o Y previsto é Y = m1 * x1 +...+ mn * xn + c enquanto na regressão logística transformamos esse Y em uma especie de Y estrela sendo igual a ln (p/1-p). Essa variável dependente é também chamada de *logit*.

	Linear	Logística
Reta	Reta	Curva - S
Variável Dependente	Continua	Categórica
Interpretação	$\hat{y}$	$\ln\left(\frac{p}{1+p}\right)$

Qual a relação entre a Probabilidade e *Logit*? Tendo em vista que a nossa variável dependente em regressão logística por definição é *Logit*.

A primeira consideração é que a probabilidade sempre varia de 0 a 1 pois não existem probabilidades negativas e maiores que 1. Enquanto que o espaço *logit* vai de mais infinito ate menos infinito (quase sempre apresentado como -5 ate 5 ou -6 ate 6). Desta forma, podemos afirmar que quando x =0 a probabilidade é de 50 %. Em outras palavras, se o *logit* for maior que 0 então a probabilidade é maior do que 50 % e se o *logit* for menor do que 0 então a probabilidade determinada do evento é menor que 50 %.

A Regressão Logística tenta:

- **Modelar** a probabilidade de um evento ocorrer dependendo dos valores das variáveis independentes
- **Estimar** a probabilidade de que um evento ocorra versus a probabilidade de que o evento não ocorra
- **Prever** o efeito de uma série de variáveis em uma variável de resposta binária
- **Classificar** as observações estimando a probabilidade de uma observação estar em uma categoria específica ou não.

Uma **matriz confusão** ou **matriz de classificação** compara os resultados reais com os resultados previstos. As linhas são rotuladas com resultados reais, enquanto as colunas são rotuladas com resultados previstos.

Variável Dependente

Ao lado esquerdo teremos o *logit*, em regressão linear definimos um erro atrelado anexando o coeficiente de erro no lado direito, enquanto que na regressão logística colocamos o erro tudo junto e não separado.

$$\ln\left(\frac{p}{1+p}\right) = a + bX$$

Para transformar o *logit* em probabilidade , que eventualmente é o que queremos, o lado esquerdo da formula deverá passar por algumas manipulações.

Movemos o Log como expoente

$$\left(\frac{p}{1+p}\right) = e^{a+bX}$$

Desta forma retiramos a fração,

$$p = \left(\frac{e^{a+bX}}{1+e^{a+bX}}\right)$$

$$p = \left(\frac{1}{1 + e^{-a+bX}}\right)$$

Assim começamos com *logit* e terminamos com uma probabilidade.

Performance de um Modelo de Regressão Logística

A matriz de confusão é uma tabela que é frequentemente usada para descrever o desempenho dos modelos de classificação em um conjunto de dados de teste para os quais os valores verdadeiros são conhecidos.

n = 165	**Predict Não**	**Predict Sim**
Actual Não	50	10
Actual Sim	5	100

- Existem duas classes *predicted* possíveis: **"Sim"** e**"Não"**. Por exemplo, se nos estamos tentando prever o numero de alunos desistentes de um cursor, **"Sim"** significaria que eles desistirão, e **"Não"** significa que eles permanecerão no curso.
- O classificador fez ao total 165 predições.
- Classificados como *predicted* **"Sim"** 110 vezes e **"Não"** 55 vezes.

- **True positives (TP) ou Verdadeiro Positivos (VP) :** Estes são casos em que previmos que **"Sim"** (alunos desistiram o curso) e os alunos realmente desistiram(100).
- **True negatives (TN) ou Verdadeiro Negativos(VN) :** No prevemos que **"Não"** (os alunos não desistiram do curso) e ele não desistiram (50).
- **False positives (FP) ou Falso Positivos (FP) :** Previmos que os alunos **"Sim"** desistirão mas os alunos **"Não"** desistirão (10)
- **False negatives (FN) ou Falso Negativos (FN):** Previmos que os alunos **"Não"** desistirão, mas **"Sim"** os alunos desistirão do curso(5)

Acurácia (Accuracy) : (TP+TN)/Total. Descreve em geral, com que frequência o classificador esta correto (100+50/165).

Medindo a Performance dos Modelos

Sensibilidade e especificidade são medidas estatísticas de performance de uma classificação binaria:

- **Accuracy (Acurácia)**= TP + TN / TP + FP + FN + TN. Esta é uma das medidas mais intuitivas sendo a razão das predições mais corretas do total de observações. Pode-se pensar que se tivermos uma alta acurácia, o nosso modelo será ótimo. Sim, a acurácia alta é uma ótima medida, mas apenas para um conjunto de dados são simétricos em que TP e TN são quase iguais.
- **Sensitivity/Recall** = TP/(TP + FN). Quando a classificação é realmente **"Sim"**, e o quanto frequente ocorreu o **"Sim"** (100/(100+5)).
- **Specificity** = TN/(TN + FP) .Quando a classificação é realmente **"Não"**, e o quanto frequente isso ocorreu (50/(50+10)).
- **Precision** = TP/Predicted "**Sim**". Quando é predito "Sim",e o quanto frequente isso aconteceu ((100/(10+100))
- **F Score** = (2 * (Recall * Precision)/(Recall + Precision). É a media ponderada da

Accuracy (Acurácia) possui melhor performance preditiva quando os falso positivos e falso negativos tem um custo similar. Se o custo dos falso positivos e falso negativos são muito diferentes então é melhor focar a analise em Precision e Recall.

Regressão Logística no R

Primeira etapa de desenvolvimento deste projeto foi focado na linguagem R onde trabalhei com uma amostra reduzida, entendendo desta forma o comportamento do algoritmo e como trabalhar com os modelos desenvolvidos

Base de dados e Compartilhamento

Como este trabalho foi desenvolvido para um segmento privado os dados foram alterados e a divulgação de algumas imagens são meramente ilustrativas. Portanto não haverá distribuição dados, me limitarei na explicação e interpretação dos resultados e os códigos R compartilhados já são de domínio publico. Os nomes originais de variáveis também foram modificados.

Desenvolvimento em R

```
#CARREGAR AS BIBLIOTECAS
library(glm2)
library(xlsx)
library(dplyr)

# CARREGAR A BASE DE DADOS
sales <- read.xlsx("DataSetLab.xlsx",
                   sheetIndex=1,
                   as.data.frame=T,
                   header = T)
```

Analisando a “qualidade” do dataset

```
$ VarDependente: num  1 1 1 1 1 1 1 1 1 1 ...
$ VarIndependente3: Factor w/ 4 levels "###","#####",..: 2 2 4 2 4 2 2
2 2 2 ...
```

Podemos avaliar que nós temos 4 variáveis neste dataset, sendo 3 variáveis independentes e uma variável dependente.

Criando ando o modelo

Sabemos que a variável dependente 1 com valor TRUE (1) é mais comum que a variável dependente 1 com valor FALSE (0). Portanto neste caso, nós criamos um modelo preditivo para variável dependente 1 com valor TRUE (1). Para tornar isso preciso, pegaremos 705/1257 das nossas observações corretas com uma acurácia de 0,64. Então o baseline do nosso modelo é definido com uma acuraria de 64%. Portanto, esta será a nossa a ser alcançada no modelo de regressão logística.

```
> table(sales$VarDependente)
0    1
705 1257
> 1257/(1257+705)
[1] 0.64
```

Teste e Treinamento (Splitting Training & Testing Data)

Agora que nós temos a nossa base de dados (dataset), nós queremos aleatoriamente dividir (split) a nossa base de dados entre uma base de treinamento e uma base de teste. A base de teste é essencial para validarmos o nosso resultado. Para realizar o split da base utilizamos o pacote *caTools*. Utilizando a função *sample.split* dividiremos a nossa base em raio de 0.64. Isso significa que nos colocaremos 64% da nossa base de treinamento, que utilizaremos para construir o modelo de treino, e 36% da base colocaremos na base de teste utilizaremos para construir o modelo de teste.

```
> set.seed(123)

> split <- sample.split(sales$VarDependente,SplitRatio = 0.64)

> split

[1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
TRUE  TRUE FALSE  TRUE  TRUE

[29]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE

[57]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE

[85]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE

[113]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE

[141] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE

[169]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE FALSE  TRUE

[197] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE FALSE

[225]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE FALSE FALSE  TRUE

[253]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE  TRUE FALSE  TRUE

...

[ reached getOption("max.print") -- omitted 962 entries ]
```

Definindo a base de treinamento

Foi utilizada a função *subset* para criar as bases. A base de treinamento foi nomeada como **train** e a base de teste foi nomeada como **test.**

```
> train <- subset(sales, split == "TRUE")
> test <- subset(sales, split == "FALSE")
>
> nrow(train)
[1] 1255
> nrow(test)
[1] 705
```

Existindo 1085 amostras para a base de treino e 69 amostras para a base de teste.

Definindo modelo de Regressão Logística

Agora estamos prontos para construir o nosso modelo de regressão logística utilizando a Variável independente 1. O modelo foi nomeado como model e utilizaremos a função glm (*generalized linear model*) para construir o nosso modelo de regressão logística.

```
> model <- glm(VarDependente~VarIndep1+VarIndep0,family = binomial(link
= "logit"),data = train)
> summary(model)
Call:
glm(formula = VarDependente~ VarIndep1+ VarIndep0, family =
binomial(link = "logit"),
data = train)
Deviance Residuals:
Min       1Q   Median       3Q      Max
```


```
Estimate Std. Error z value Pr(>|z|)
(Intercept)                  1.7831     0.8841   2.017 0.043715 *
VarIndep1                   -0.2928     0.8095  -0.362 0.717580
VarIndep0Origem1   1.3027     0.3563   3.656 0.000256 ***
VarIndep0Origem2      -0.3272     0.6041  -0.542 0.588129
VarIndep0Origem3       -3.8457     0.3632 -10.589  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 1639.16  on 1254  degrees of freedom
Residual deviance:  665.26  on 1250  degrees of freedom
AIC: 675.26

Number of Fisher Scoring iterations: 5
```

- A tabela de Coeficientes nos mostra uma estimativa para cada coeficiente, or os betas, para o nosso modelo de regressão logística. Podemos notar que o coeficiente para a Variavel Independente 1 é negativo e o *VarIndep0Origem1* é positivo, o que significa que para valores altos neste agrupamento de variáveis indicam uma maior tendência de acontecer a variavel dependente true (1).

```
> (1/(1+exp(-(1.7831+(-0.2928+1.3027)*1))))*100
[1] 94.22964
> (1/(1+exp(-(1.7831+(-0.2928+1.3027)*1))))*100
[1] 94.22964
>
> (1/(1+exp(-(1.7831+(-0.2928+1.3027)*2))))*100
[1] 97.81807
>
```

```
>
> (1/(1+exp(-(1.7831+(-0.2928+1.3027)*4))))*100
[1] 99.70491
```

- As estrelas indicam o quanto as variáveis são significantes para o nosso modelo.
- AIC, esta é uma medida que avalia a qualidade do modelo (ajuste do r^2) que contabiliza o número de variáveis usadas em comparação com o número de observações. Valores baixos para AIC são desejáveis para comparação entre modelos.

Desenvolvendo predição para base de treinamento

Nomeamos o modelo de treinamento predictTrain e utilizaremos a função predict para fazer as predições usando o modelo model. Utilizaremos o argumento type = response que retorna a probabilidade.

```
> predictTrain = predict(model,type="response")
> summary(predictTrain)
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.06610 0.08664 0.94230 0.64060 0.94230 0.94230
> tapply(predictTrain, train$VarDependente, mean)
0 1
0.1949008 0.8906713
```

A função tapply calcula uma media preditiva cada saída verdadeira da variável independente (realmente 0 ou realmente 1).

Desta forma, encontramos a probabilidade para toda saída verdadeira para variável dependente TRUE 0.89 e também a probabilidade para todas as saídas verdadeiras para FALSE variável dependente 0.19.


limite threshold t, então nos predizemos que existe influencia. Mas se a probabilidade de primeira compra for menor que o valor limite de threshold t então predizemos que não existe influencia.

Como selecionar o Valor de t

O valor de T é frequentemente selecionado com base na menor incidência de erros. Isso implica que o melhor valor de t implica em um modelo sem erros mas isso é raro em modelos preditivos.

Existem dois tipos erros comuns em modelos preditivos:

1. Quando o modelo preditivo indica 1, ou primeira compra, mas o valor real é 0.
2. Quando o modelo preditivo indica 0, ou não primeira compra, mas o valor real é 1.

Matriz de Confusão

Para tornar a nossa analise um pouco mais qualitativa podemos utilizar a matriz de confusão ou matriz de classificação para entender o nosso modelo.

```
> #Confusion matrix for threshold of 0.2
> table(train$VarDependente,predictTrain>0.2)
FALSE TRUE
0 391 60
1 37 767
> #Sensibilidade
> 767/(37+767)
[1] 0.9539801
>
> #Especificidade
> 391/(391+60)
[1] 0.8669623
```


```
> table(train$VarDependente,predictTrain>0.5)
FALSE TRUE
0 391 60
1 37 767
> #Sensibilidade
> 767/(767+37)
[1] 0.9539801
> #Especificidade
> 391/(391+60)
[1] 0.8669623
>
> #Confusion matrix for threshold of 0.7
> table(train$VarDependente,predictTrain>0.7)
FALSE TRUE
0 391 60
1 37 767
> #Sensibilidade
> 767/(767+37)
[1] 0.9539801
> #Especificidade
> 391/(391+60)
[1] 0.8669623
>
> #Confusion matrix for threshold of 0.9
> table(train$VarDependente,predictTrain>0.9)
FALSE TRUE
0 408 43
1 106 698
```


```
> #Especificidade
> 408/(408+43)
[1] 0.9046563
```

Como podemos notar se aumentar o meu valor de Threshold *t* para valores acima de 0.9 a sensibilidade do modelo decresce e a especificidade aumenta enquanto se diminuirmos o valor de *t* teremos o cenário reverso. Portanto, escolher de *t* passa a ser um dilema. Utilizar a curva ROC nos auxiliara a escolher o melhor valor de *t*.

Pacote ROCR

Lembrando que fizemos as previsões em nosso conjunto baseado no conjunto de dados de treinamento e o denominamos como train. Usaremos estas previsões para criar a nossa curva ROC

```
ROCRpred <- prediction(predictTrain,train$VarDependente)
```

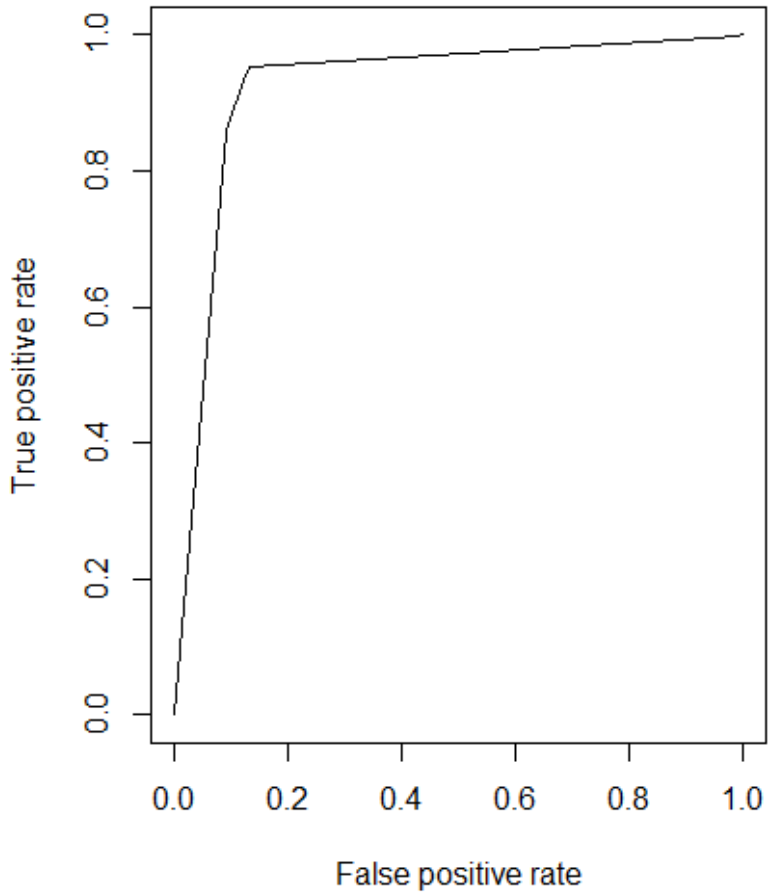
O primeiro argumento são as previsões que fizemos no nosso modelo (train) e o segundo argumento todos os valores verdadeiros da nossa amostra.

Agora utilizamos uma função de desempenho para definir os valores que gostaria de plotar nos eixos x e y da nossa curva ROC.

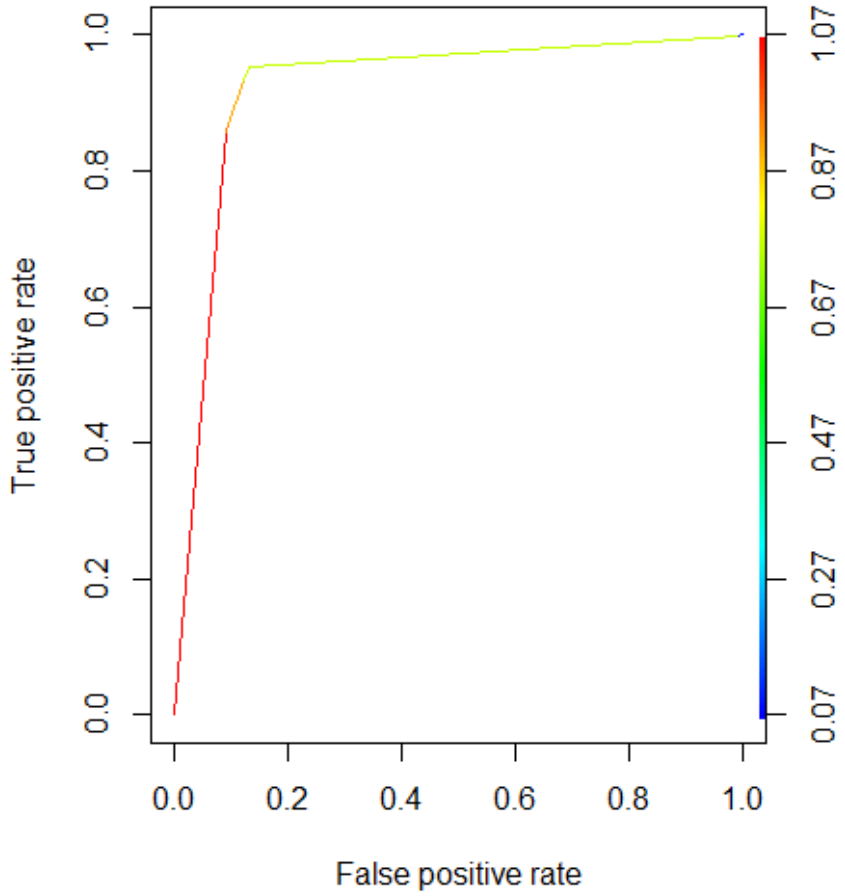
```
ROCRperf <- performance(ROCRpred, "tpr","fpr")
```

Agora definirmos as funções para plotar o gráfico com os resultados

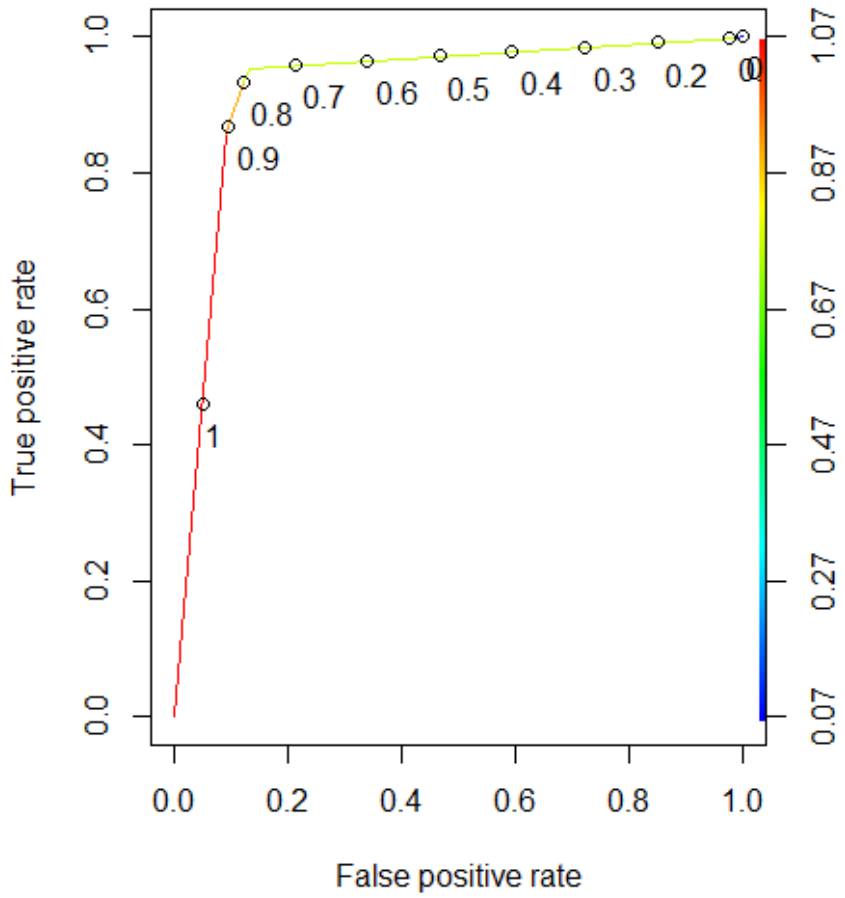
```
> plot(ROCRperf)
```


```
> plot(ROCRperf, colorize=TRUE)
```




```
> plot(ROCRperf, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1),
+ text.adj=c(-0.2,1.7))
```



A sensibilidade, ou a taxa de True Positives do modelo, é mostrada no eixo y. Enquanto a taxa de False Positive, ou — 1 a especificidade, é dada no eixo x. A linha mostra como essas duas medidas variam com diferentes valores no limite.

A curva ROC sempre começa no ponto (0, 0), ou seja, Threshold com valor 1. Isso significa que nesse valor de Threshold não capturaremos nenhum caso de para variavel dependente TRUE (sensibilidade de 0), mas rotularemos corretamente todos os casos que variável dependente FALSE (FP = 0)

A curva ROC sempre termina no ponto (1,1), ou seja, Threshold com valor 0. Isso significa que nesse valor de Threshold capturaremos todos os casos de variavel dependente TRUE (sensibilidade de 1), mas rotularemos incorretamente todos os casos de variavel dependente FALSE como casos de variavel dependente TRUE (FP = 1)

lado, no momento (0,6, 0,9), estamos rotulando corretamente cerca de 90% dos casos de variavel dependente TRUE, mas temos uma taxa de falsos positivos de 60%. No meio, em torno de (0,3, 0,8), rotulamos corretamente cerca de 80% dos casos de variavel dependente TRUE, com uma taxa de falso positivo de 30%.

A curva ROC captura todos os valores de Threshold simultaneamente. Quanto maior o Threshold, ou mais próximo de (0, 0), maior a especificidade e menor a sensibilidade. Quanto menor o valor de Threshold, ou mais próximo de (1,1), maior a sensibilidade e diminua consequentemente a especificidade.

Qual melhor valor de Threshold eu devo escolher?

A escolha vai de acordo com o melhor trade-off que desejamos fazer. Se desejamos ter uma alta especificidade ou baixo índice de falsos positivos, a escolha um valor de Threshold que maximiza os valores de verdadeiros positivos mantendo a taxa de falsos positivos baixa. Para a nossa analise um valor de Threshold (0.2,0.9) para a Curva ROC aparenta ser uma ótima escolha.

Predição na base de Teste

Para esta amostra em especifico, utilizamos um threshold de valor 0.9 e 0.2 e compramos os resultados.

```
table(test$VarDependente,predictTest >= 0.90)
FALSE TRUE

0 239 15

1 55 398

> #Accuracy

> (239+398)/(239+15+55+398)

[1] 0.9009901

> #Sensibility/Recall – TP/(TP + FN)

> 398/(398+55)
```


```
[1] 0.9409449
> table(test$VarDependente,predictTest >= 0.20)
FALSE TRUE
0 234 20
1 12 441
> #Accuracy
> (426+235)/(235+19+27+426)
[1] 0.9349364
> #Sensibility/Recall – TP/(TP + FN)
> 441/(441+12)
[1] 0.9735099
> #Specificity – TN/(TN + FP)
> 234/(234+20)
[1] 0.9212598
```

Para Threshold > 0.9, existem um total de 707 casos nesta amostra de teste, sendo que 453 são realmente casos de VarDependente TRUE e 254 são casos de VarDependente FALSE.

	Predicted FALSE	Predicted TRUE
Actually FALSE	239	15
Actually TRUE	55	398

Para Threshold > 0.2, existem um total de 707 casos nesta amostra de teste, sendo que 453 são realmente casos de VarDependente TRUE e 254 são casos de VarDependente FALSE.

Actually TRUE	12	441

Conclusão para a amostra em R

A acurácia do modelo indica uma probabilidade maior que 90% em ambos os casos, portanto podemos facilmente identificar que este valor é maior que o nosso baseline de 64%.

Portanto na pratica, o valor retornado pela regressão logistica pode ser utilizado na pratica. A escolha do valor de Threshold, neste caso, será para definir um modelo ótimo de acordo com as preferencias por índices altos e baixos de verdadeiros/falsos positivos.

Implementação Regressão Logística — PyTools Qlik Sense

Uma das *Extensions* disponíveis no Qlik Sense é o PyTools, que disponibiliza uma serie de algoritmos de aprendizados de maquina de maneira mais "simplificada" utilizando o pacote scikit learn do Python.

Desenvolvimento no Qlik Sense

Todo o desenvolvimento no Qlik Sense pode ser feito no Load onde é possível fazer toda a modelagem relacional e por meio do comando Extension utilizar o SSE PyTools, que contem os algoritmos de aprendizados de maquina não supervisionados em Python, disponíveis para integração como o Qlik Sense.

Após definir as Features, classificar todas as métricas (*target*, *feature*, ignored), identificar o tipo de variável e estrategias de ação para cada uma delas, definimos o *Estimator* como Logistic Regression indicado que utilizaremos o algoritmo de regressão logística.

Existem uma serie de funções que podem ser usada neste projeto, por exemplo, para

- sklearn_Setup (model_name, estimator_args, scaler_args, execution_args)
- sklearn_Set_Features(model_name, feature_name, variable_type, data_type, feature_strategy, strategy_args)
- sklearn_Fit (model_name, n_features)
- sklearn_Get_Metrics(model_name)
- sk_learn_Get_Confusion_Matrix(model_name)
- sklearn_Explain_Importances (model_name)

Cada uma destas funções podem ser melhor exploradas na documentação oficial do PyTools.

Melhor Modelo

Criando uma nova pasta de resultado em uma aplicação no Qlik Sense podemos avaliar as variáveis de saída da nossa regressão logística.

O PyTools oferece algumas variáveis como F-score, Precision, Recall, Grau de Importância, Acurácia e o Score.

Score

model_name	score_result	score
CPV-Portador-LR	LogisticRegression model has a score of 0.997 against the test data.	0,997477578475

Feature Importance

feature_name	Avg(importance)	Accuracy
Totais	**1**	**0,99747758**
Variavel Independente	1	0,99747758

Score

model_name	score_result	score
CPV-Portador-LR	LogisticRegression model has a score of 0.997 against the test data.	0,99747757847534

Metrics

model_na...	Valores	class	F-score		Precision		Recall	
			0.00000	1.00000	0.00000	1.00000	0.00000	1.000
CPV-Portador-LR			0,18	1,00	0,33	1,00	0,13	

Confusion Matrix

model_name | true_label | pred_label

	0	1
CPV-Portador-LR	3,00	3.56
0	1,00	
1	2,00	3.55

interpretando os resultados obtidos e definindo a sua resposta para cada caso.

O uso do Pytools pode ser utilizado como um complemento mais completo quando ja entendemos melhor sobre o modelo. Esta é uma excelente ferramente caso ja possua o Qlik Sense pois se torna uma ferramenta mais automatizada para o uso deste tipo de ferramenta estatística.

Graficamente falando, o ganho para área de negocio pode ser mais interessante a aplicação do PyTools. De toda forma, utilizar esta ferramenta independe se for em R ou Python, desde que possamos obter as respostas ideais para as nossas modelagens.