Regressão linear múltipla

Introdução

Até agora vimos dois modelos adequados para a previsão de dados contínuos, como o preço de um imóvel. No entanto, vamos pensar nesse outro conjunto de dados, também do Kaggle. Esse conjunto apresenta dados sobre os passageiros do Titanic que sobrevireram.

Vamos pensar então num modelo que preveja, baseado nos atributos de uma determinada entrada de dados, se uma pessoa sobreviveu ou não. Vejam que temos algumas variáveis independentes, como classe do ticket, idade da pessoa e número de pais e crianças a bordo. No entanto, a nossa variável dependente, sobrevivência, é categórica, ou seja, só temos como valores possíveis `0` (não sobreviveu) ou `1` (sobreviveu).

Apesar de ser atraente utilizar os modelos que já vimos nas notas de aula anteriores para resolver esse problema, a solução não é a ideal, justamente por não estar preparada para apresentar valores categóricos. Afinal de contas, os modelos de regressão linear simples e múltipla são modelos de previsão, e não modelos de classificação.

O modelo de regressão logística lida com uma variável dependente Y que é categórica, podendo ter dois e apenas dois valores. Ele mede a relação entre a variável dependente e uma ou mais variáveis independentes, estimando a probabilidade de ocorrência utilizando uma função logística.

Uma função logística (ou sigmoide) é uma curva em formato de S, cuja equação é dada por:

f(x) = L / (1 + e^(-k * (x - x0)))

Em que:

e -> número de Euler x0 -> valor de x no ponto médio da curva L -> valor máximo da curva k -> declividade da curva

Simplificando os valores k e L para 1, e de x0 para 0, temos:

f(x) = 1 / (1 + e^(-x))

Portanto, o nosso problema será tal que temos:

y = f(x * beta) + erro

Tendo múltiplas variáveis, nosso problema passa a ser:

y = f(beta0 + x1 * beta1 + x2 * beta2 + ... + xn * betan) + erro

Não vamos entrar muito mais no mérito da matemática pro trás do modelo. Vamos aplicar esse modelo diretamente em cima dos dados que discutimos.

Import e análise dos dados

```
%matplotlib inline
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics, preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split


titanic = pd.read_csv("titanic.csv")
titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71
				Heikkinen						

Aqui vemos as diferentes variáveis incluídas no nosso conjunto de dados. Algumas são auto-explicativas, mas algumas precisam de maiores explicações:

- PassengerId: A identificação do passageiro;
- Survived: O nosso alvo, diz se a pessoa sobreviveu (1) ou não (0);
- Pclass: A classe que o passageiro estava no navio (1ª, 2ª ou 3ª);
- SibSp: Número de irmãos e cônjuges da pessoa dentro do navio;
- Parch: Número de pais e filhos da pessoa dentro do navio;
- Fare: O valor pago pela passagem;
- Embarked: O porto que a pessoa embarcou (C = Cherbourg, Q = Queenstown, S = Southhampton)

Temos aqui algumas colunas que não serão necessárias para a nossa análise. ID do passageiro, nome, ticket e cabine são dados que, em teoria, são irrelevantes para o alvo que estamos procurando.

```
# axis define se os rótulos serão removidos dos índices (0) ou das colunas (1)
# inplace define se a função retorna uma cópia (False) ou se a operação é feita
# em cima do próprio DataFrame
titanic.drop(["PassengerId", "Name", "Ticket", "Cabin"], axis=1, inplace=True)
titanic.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

Agora vamos criar valores categóricos para a coluna `Embarked`, porque o modelo não suporta strings.

```
ports = pd.get_dummies(titanic.Embarked, prefix="Embarked")
ports.head()
```

	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1

```
titanic = titanic.join(ports)
titanic.drop(["Embarked"], axis=1, inplace=True)
titanic.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Emba
0	0	3	male	22.0	1	0	7.2500	0	0	
1	1	1	female	38.0	1	0	71.2833	1	0	
2	1	3	female	26.0	0	0	7.9250	0	0	
3	1	1	female	35.0	1	0	53.1000	0	0	
4	0	3	male	35.0	0	0	8.0500	0	0	

Na mesma linha, vamos alterar os valores da coluna "Sex" para valores binários.

```
titanic.Sex = titanic.Sex.map({"male": 0, "female": 1})
```

Por último nessa etapa de análise dos dados, vamos substituir os valores ausentes. Vamos fazer o que já fizemos no outro modelo e substituir os valores ausentes pela média dos dados.

```
# Primeiro vamos ver os dados nulos
```

```
titanic[pd.isnull(titanic).any(axis=1)]
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Emba
5	0	3	0	NaN	0	0	8.4583	0	1	
17	1	2	0	NaN	0	0	13.0000	0	0	
19	1	3	1	NaN	0	0	7.2250	1	0	
26	0	3	0	NaN	0	0	7.2250	1	0	
28	1	3	1	NaN	0	0	7.8792	0	1	
...	...	...	...	...	...	...	...	...	...	
859	0	3	0	NaN	0	0	7.2292	1	0	
863	0	3	1	NaN	8	2	69.5500	0	0	
868	0	3	0	NaN	0	0	9.5000	0	0	
878	0	3	0	NaN	0	0	7.8958	0	0	
888	0	3	1	NaN	1	2	23.4500	0	0	

177 rows × 10 columns

```
# Aparentemente apenas a coluna Age possui dados vazios
titanic.Age.fillna(titanic.Age.mean(), inplace=True)

# Vamos repetir a linha anterior para confirmar que tudo foi acertado
titanic[pd.isnull(titanic).any(axis=1)]
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_C	Embarked_Q	Embarked_S

Desenvolver o modelo

Agora que pre-processamos os dados, vamos extrair as variáveis independentes e a variável dependente, e montar os conjuntos de treino e de teste.

+ Código + Texto

```
surv = titanic.Survived.copy()
# Vejam que aqui não passamos inplace=True
data = titanic.drop(["Survived"], axis=1)

data_train, data_test, surv_train, surv_test = train_test_split(data, surv, test_size=0.2, random_state=0
```

Agora vamos usar a classe `LogisticRegression` do sklearn para ajustar o nosso modelo.

Um cenário comum em alguns conjuntos de dados é dos valores extrapolarem os limites disponíveis no computador, como por exemplo no uso de memória ou número de iterações no processo de ajuste do modelo.

Para isso, podemos utilizar um recurso de normalizar os nossos dados, para que o conjunto tenha média 0 e desvio padrão 1. Isso permite que trabalhemos com valores de grande

variabilidade.

```
scaler = preprocessing.StandardScaler().fit(data_train)
data_scaled = scaler.transform(data_train)
```

Em seguida, vamos ajustar nosso modelo.

```
model = LogisticRegression()
model.fit(data_scaled, surv_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Apesar de termos falado que o modelo de regressão logística é adequado para variáveis dependentes com duas e apenas duas categorias, existem variações deste modelo que se comportam muito bem com dados categóricos que possuam mais de duas categorias.

No caso da scikit-learn, podemos adotar a mesma classe `LogisticRegression`, porém precisamos definir o parâmetro `multi_class` para `ovg` ou `multinomial`, dependendo do algoritmo desejado. Veja a documentação oficial para mais informações.

Testando o modelo

Agora que treinamos nosso modelo, vamos fazer as previsões em cima do nosso conjunto de teste, lembrando de transformá-lo antes.

```
surv_pred = pd.Series(model.predict(scaler.transform(data_test)))

# Vamos resetar os índices do conjunto de teste
surv_test = surv_test.reset_index(drop=True)
surv_comparison = pd.concat([surv_test, surv_pred], axis=1)
surv_comparison.columns = ["Real", "Prediction"]
surv_comparison.head()
```

	Real	Prediction
0	0	0
1	0	0
2	0	0
3	1	1
4	1	1

Vimos que, dos cinco primeiros dados, nosso modelo conseguiu prever os 5 corretamente! Vamos usar agora o módulo `metrics` para medir o conjunto de testes como um todo.

```
print("Accuracy:", metrics.accuracy_score(surv_test, surv_pred))
print("Precision:", metrics.precision_score(surv_test, surv_pred))
print("Recall:", metrics.recall_score(surv_test, surv_pred))
```

```
Accuracy: 0.7988826815642458
Precision: 0.7391304347826086
Recall: 0.7391304347826086
```

Nosso modelo retornou quase 80% de acurácia, um resultado muito bom, considerando a simplicidade do modelo!

Certamente poderíamos melhorar o nosso modelo se analisássemos a relação das variáveis independentes, suas correlações e descartássemos possíveis variáveis que possam estar atrapalhando o nosso modelo, mas para uma primeira execução esse já é um ótimo resultado.

Analisando os resultados

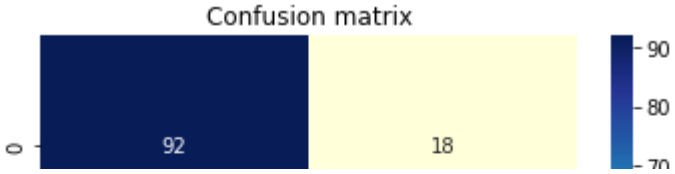
Uma boa forma de visualizar os resultados é montando uma matriz de confusão, como já fizemos na última aula. Aqui, vamos montar um gráfico de calor para exibir essa matriz.

```
cnf_matrix = metrics.confusion_matrix(surv_test, surv_pred)

print(cnf_matrix)
```

```
[[92 18]
 [18 51]]
```

```
labels = [0, 1]

plt.subplots()
tick_marks = np.arange(len(labels))

plt.xticks(tick_marks, labels)
plt.yticks(tick_marks, labels)

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu")

plt.title("Confusion matrix", y=1.1)
plt.ylabel("Real")
plt.xlabel("Predicted")
```

```
Text(0.5, 15.0, 'Predicted')
```



Na matriz de confusão, vemos que 92 das pessoas que não sobreviveram ao naufrágio foram corretamente previstas como não tendo sobrevivido, 18 pessoas que não sobreviveram foram incorretamente previstas como tendo sobrevivido.

18 pessoas que sobreviveram foram previstas como não tendo sobrevivido, e 51 pessoas sobreviveram e previmos corretamente.