

# ELT579 - Tópicos Especiais em Inteligência Artificial – Semana 1 - Titanic: ‘Machine Learning from Disaster’

Michel Batistin Fiório – Matrícula: 48376  
Departamento de Engenharia Elétrica,  
Universidade Federal de Viçosa, Viçosa – MG, E-mail: mbfiorio@gmail.com

## I. INTRODUÇÃO

Em 15 de abril de 1912, durante sua viagem inaugural, o amplamente considerado “inafundável” RMS Titanic afundou após colidir com um iceberg. Infelizmente, não havia botes salva-vidas suficientes para todos a bordo, resultando na morte de 1.502 dos 2.224 passageiros e tripulantes.

Embora houvesse algum elemento de sorte envolvido na sobrevivência, parece que alguns grupos de pessoas eram mais propensos a sobreviver do que outros.

Durante as aulas da primeira semana da disciplina ELT579 foi demonstrado a criação de um modelo de aprendizado de máquina para responder a seguinte questão: que tipos de pessoas tem mais chance de sobreviver?

Foi utilizado na modelagem um banco de dados com informações dos passageiros, tais como: nome, idade, sexo, classe socioeconômica, etc.

Os dados foram obtidos do site ‘kaggle.com/competitions/titanic’. São disponibilizados um arquivo com o nome ‘train.csv’ e outro intitulado ‘test.csv’. O primeiro arquivo contém detalhes de 891 passageiros e revela quais sobreviveram ou não. O segundo arquivo contém os mesmos detalhes de outros 418 passageiros, mas sem a informação de sobrevivência.

Este segundo arquivo é utilizado para testar a precisão de predição do modelo criado. Submete-se para o site Kaggle o arquivo com as predições do nosso modelo e o site disponibiliza o resultado indicando a percentagem de acertos obtida pelo modelo.

## II. OBJETIVOS

Nessa atividade o objetivo é realizar modificações no script criado em aula pelo professor, na tentativa de se obter uma melhora no resultado de predição do modelo.

Como referência, o melhor resultado obtido foi o modelo 5 criado em aula, que alcançou uma predição com taxa de acerto de 77,511%.

A partir desse modelo referência, iremos demonstrar nessa atividade as implementações realizadas no script, com uma breve explicação de cada alteração. Ao final, apresentaremos o melhor resultado obtido.

## III. MATERIAIS E MÉTODOS

A algoritmo de aprendizado de máquina será desenvolvido na linguagem de programação Python e executado no ambiente de desenvolvimento ‘Jupyter Notebook’.

Como modelo de referência obtido das aulas utilizamos o código disponibilizado pelo professor, cujo título do arquivo é ‘Script\_semana1.py’.

Faremos uso dos bancos de dados disponíveis no site da Kaggle, intitulados ‘train.csv’ e ‘test.csv’. Estes arquivos contém uma série de informações dos passageiros, conforme listado na Tabela I.

TABELA I

Variável	Definição	Valores
Survival	Sobrevivente	0 = não, 1 = sim
Pclass	Status socioeconômico	1 = alta, 2 = média, 3 = baixa
Sex	Sexo	male, female
Age	Idade	Numérico
SibSp	Número de irmãos ou cônjuges embarcados	Numérico
Parch	Número de pais ou filhos embarcados	Numérico
Ticket	Número da passagem	Texto
Fare	Valor da passagem	Numérico
Cabin	Número da cabine	Texto
Embarked	Porto de embarque	C = Cherbourg, Q = Queenstown, S = Southampton

## IV. RESULTADOS E DISCUSSÕES

A primeira alteração realizada no algoritmo foi na função criada para realizar o ajuste das características (‘features’) a serem extraídas dos bancos de dados a fim de ser utilizado para treinamento do modelo.

Na ‘Figura 1’ pode ser vista essa função, chamada de ‘features()’. Dentro da função são realizados 07 comandos para ajuste de algumas ‘features’ e criação de novas.

No passo (1) está sendo realizado o ajuste dos dados da coluna ‘Sex’ dos arquivos, transformando de texto para numérico. As células com valor ‘male’ são substituídos pelo

número 0, enquanto as células com valor 'female' para número 1.

```
def features(X):
    # 1 - Ajuste da coluna Sex para numérico
    subs = {'female':1, 'male':0}
    X['Sex'] = X['Sex'].replace(subs)

    # 2 - Preenchendo dados vazios em 'Age'
    X['Age'] = X['Age'].fillna(method='ffill')

    # 3 - Preenchendo dados vazios em 'Fare'
    X['Fare'] = X['Fare'].fillna(method='ffill')

    # 4 - Possui pai/mãe/filho/filha (0-não, 1-sim)
    X['Parch'] = X['Parch'].where(X['Parch']<1,1)

    # 5 - Crianças (0-menor 16 anos, 1-entre 16 e 55, 2-maior que 55 anos)
    X['idade'] = np.where(X['Age']<16,0,1)
    X['idade'] = X['idade'].where(X['Age']<55,2)

    # 6 - Adultos (maior ou igual 16 anos): (0-sem filhos, 1-com filhos)
    X['filhos'] = np.where((X['Age']>16) & (X['Parch']==1),1,0)

    # 7 - Crianças com ou sem pais (0-sem pais, 1-com pais)
    X['sempais'] = np.where((X['Age']<16) & (X['Parch']==0),0,1)

    return X
```

Fig. 1. Script da função 'features'.

No passo (2) é executado um comando para preencher as células vazias da coluna 'Age', adicionando nessas células o mesmo valor da célula imediatamente anterior.

O passo (3) realiza o mesmo comando, mas agora para a coluna 'Fare' da planilha.

O passo (4) realiza uma transformação nos dados numéricos da coluna 'Parch'. A informação dessa coluna se refere a quantidade de pais ou filhos que cada passageiro tinha a bordo do Titanic. O menor valor era '0' e o maior valor era '6'. Criamos uma lógica para transformar essa informação em um dado categórico, com dois grupos distintos, as de pessoas sem pais ou filhos a bordo e as de pessoas com pais ou filhos a bordo. Portanto, para células com valor 0 (sem pais ou filhos) nenhuma alteração era realizada. E para células onde o valor era diferente de 0 (com pais ou filhos) o valor era preenchido com o número 1.

No passo (5) foi realizado uma categorização semelhante, agora nos dados da coluna 'Age'. Optou-se por criar uma nova coluna, denominada 'idade'. Se a idade da pessoa era menor que 16 anos ('Age'<16), a célula correspondente da coluna 'idade' era preenchida com o valor 0. Para idades entre 16 e 55 anos, a célula era preenchida com valor 1. Para idades acima de 55 anos era preenchido o valor 2.

O passo (6) trata-se de uma lógica que cria a feature chamada 'filhos'. Essa 'feature' possui valor 1 para passageiros maiores de 16 anos ('Age'>16) e com valores da coluna 'Parch' diferente de 0. Para todas as outras condições, a célula é preenchida com valor 0.

O último passo (7) da função 'features' cria uma outra nova 'feature', denominada 'sem pais'. Nessa feature, são preenchidas com valor 0 os casos onde a pessoa tem menos de 16 anos ('Age'<16) e não possui pais a bordo ('Parch' = 0). Para os casos onde o valor de 'Parch' é diferente de zero, a respectiva célula na coluna 'sem pais' é preenchida com o valor 1.

Com a função 'features' desenvolvida, nós aplicamos essa função aos dois arquivos de dados, 'train.csv' e 'test.csv'. O

resultado representa arquivos que possuíam as seguintes colunas conforme visto na 'Figura 2'.

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked', 'idade', 'filhos',
       'sempais'],
      dtype='object')
```

Fig. 2. Lista das 'features' disponíveis.

A partir desse resultado, foram então selecionadas para treinamento do modelo as 'features' listadas na Tabela II a seguir.

TABELA II

Variável	Definição	Valores
Pclass	Status socioeconômico	1 = alta, 2 = média, 3 = baixa
Sex	Sexo	0 = homem, 1 = mulher
Parch	Pais ou filhos embarcados	0 = sem pais ou filhos, 1 = com pais ou filhos
Fare	Valor da passagem	Númerico
idade	Categoria da idade do passageiro	0 = até 16 anos, 1 = de 17 até 54 anos, 2 = acima de 54 anos
sempais	Crianças ou jovens de até 16 anos sem pais a bordo	0 = sem pais a bordo, 1 = com pais a bordo

Nos dados selecionados foi aplicada a normalização (média 0 e desvio padrão 1) através da função 'StandardScaler'. Nesse ponto os dados estão prontos para treinamento dos modelos.

Foram selecionados quatro modelos distintos para treinamento: 'Logistic Regression', Máquina de vetor de suporte (SVM), K vizinhos mais próximos (KNN) e 'Random Forest'.

Através da função 'gp-minimize' da biblioteca scikit-optimize foi desenvolvido um algoritmo para executar a otimização dos hiperparâmetros de todos esses quatro modelos. Nas 'Figura 2', 'Figura 3', 'Figura 4' e 'Figura 5' podem ser consultados os scripts para os modelos Random Forest, SVM, Logistic Regression e KNN, respectivamente.

Ao final de cada otimização os parâmetros ótimos são coletados e aplicados no modelo final. Na Tabela III temos a lista dos parâmetros ótimos obtidos para cada modelo.

TABELA III

Modelo	Parâmetros ótimos
Random Forest	Criterion = 'gini', n_estimators = 720, max_depth = 30, min_samples_split = 4, min_samples_leaf = 3
SVM	Kernel = 'rbf', C = 3.4549, gamma = 0.087
Logistic Regression	C = 1.782
KNN	n_neighbors = 4, weights = 'uniform', p = 1

```

# Grupo de Parâmetros para otimização do modelo Random Forest
parametros = [('entropy', 'gini', 'log_loss'),
              (100, 1000),
              (3,30),
              (2,12),
              (1,10)]

# Definindo a função de treino do modelo
def treinar_modelo_rf(parametros):
    modelo_rf = RandomForestClassifier(criterion = parametros[0],
                                      n_estimators = parametros[1],
                                      max_depth = parametros[2],
                                      min_samples_split = parametros[3],
                                      min_samples_leaf = parametros[4],
                                      random_state = 0)

    score = cross_val_score(modelo_rf, X_treino_sc, Y_treino, cv = 10)
    print(np.mean(score))

    return -np.mean(score)

# Rodando a otimização do modelo
otimos = gp_minimize(treinar_modelo_rf, parametros, random_state = 0,
                    verbose = True, n_calls = 40, n_random_starts = 10)

print(otimos.x)

# Ajustando hiperparâmetros do modelo ótimo obtido
modelo_rf = RandomForestClassifier(criterion = otimos.x[0],
                                  n_estimators = otimos.x[1],
                                  max_depth = otimos.x[2],
                                  min_samples_split = otimos.x[3],
                                  min_samples_leaf = otimos.x[4],
                                  random_state = 0)

```

Fig. 2. Script de otimização dos hiperparâmetros do modelo Random Forest.

```

# Grupo de Parâmetros para otimização do modelo SVM
parametros = [('rbf', 'poly', 'sigmoid'),
              (0.5, 4),
              (0.01, 0.1)]

# Definindo função
def treinar_modelo_svc(parametros):
    modelo_svc = SVC(kernel = parametros[0], C = parametros[1],
                    gamma = parametros[2], degree=4)

    score = cross_val_score(modelo_svc, X_treino_sc, Y_treino,
                            cv = 10)

    print(np.mean(score))

    return -np.mean(score)

# Rodando a otimização
otimos = gp_minimize(treinar_modelo_svc, parametros, random_state = 0,
                    verbose = True, n_calls = 50,
                    n_random_starts = 10)

print(otimos.x)

# Ajustando hiperparâmetros do modelo ótimo
modelo_svc = SVC(kernel = otimos.x[0], C = otimos.x[1], gamma = otimos.x[2])

```

Fig. 3. Script de otimização dos hiperparâmetros do modelo SVM.

Com os quatro modelos ajustados com os parâmetros ótimos, realizamos uma combinação dos mesmos em um modelo único de classificação de votos, utilizando a função ‘VotingClassifier’. Nessa lógica o resultado final de predição do modelo agrupado é o resultado que obteve a maioria entre as predições dos modelos individuais.

```

# Grupo de Parâmetros para otimização do modelo Logistic Regression
parametros = [(0.01, 3.0)]

# Definindo função
def treinar_modelo_lr(parametros):
    modelo_lr = LogisticRegression(C=parametros[0], random_state=0)

    score = cross_val_score(modelo_lr, X_treino_sc, Y_treino, cv = 10)

    print(np.mean(score))

    return -np.mean(score)

# Rodando a otimização
otimos = gp_minimize(treinar_modelo_lr, parametros, verbose = True,
                    n_calls = 50, n_random_starts = 10, random_state=0)

print(otimos.x)

# Ajustando hiperparâmetros do modelo ótimo
modelo_lr = LogisticRegression(C=otimos.x[0], random_state= 0)

```

Fig. 4. Script de otimização dos hiperparâmetros do modelo Logistic Regression.

```

# Grupo de Parâmetros para otimização do modelo KNN
parametros = [(1, 15),
              ('uniform', 'distance'),
              (1, 8)]

# Definindo função
def treinar_modelo_knn(parametros):
    modelo_knn = KNeighborsClassifier(n_neighbors=parametros[0],
                                    weights=parametros[1],
                                    p=parametros[2])

    score = cross_val_score(modelo_knn, X_treino_sc, Y_treino,
                            cv = 10)

    print(np.mean(score))

    return -np.mean(score)

# Rodando a otimização
otimos = gp_minimize(treinar_modelo_knn, parametros, random_state = 0,
                    verbose = True, n_calls = 50, n_random_starts = 10)

print(otimos.x)

# Ajustando hiperparâmetros do modelo ótimo
modelo_knn = KNeighborsClassifier(n_neighbors=otimos.x[0],
                                weights=otimos.x[1], p=otimos.x[2])

```

Fig. 5. Script de otimização dos hiperparâmetros do modelo KNN

## V. CONSIDERAÇÕES FINAIS

No modelo agrupado por classificação de votos, fizemos a avaliação do desempenho através da técnica de validação cruzada nos dados de treinamento. Observamos pelos resultados obtidos que o modelo conjugado obtinha um desempenho melhor quando se retirava o modelo de Logistic Regression.

Dessa forma, o modelo agrupado final, constituído dos modelos Random Forest, SVM e KNN, obteve um desempenho de 83,62% através da validação cruzada. Veja ‘Figura 6’.

```
# Criando classificador por voto
modelo_voto = VotingClassifier([('SVC', modelo_svc), ('KNN', modelo_knn),
                                ('RF', modelo_rf)])

# Realizando ajuste
modelo_voto.fit(X_treino_sc, Y_treino)

score = cross_val_score(modelo_voto, X_treino_sc, Y_treino, cv = 10)

print(np.mean(score))
```

0.8361922596754058

Fig. 6. Desempenho do modelo na validação cruzada dos dados de treino

Realizamos então a predição do nosso modelo nos dados de teste e salvamos no arquivo 'submission3.csv'. Este arquivo foi submetido ao site Kaggle e o resultado obtido foi uma precisão de predição de 78,708%. Veja 'Figura 7'.






1792	Michel Florio	0.78708
 Your Best Entry! Your submission scored 0.77511, which is not an improvement of your previous score. Keep trying!		
	<b>submission7.csv</b> Complete · 3d ago · Submissão 7	0.77511
	<b>submission6.csv</b> Complete · 3d ago · Submissão 6	0.78468
	<b>submission5.csv</b> Complete · 3d ago · Submissão 5	0.78468
	<b>submission3.csv</b> Complete · 3d ago · Submissão 3	0.78708

Fig. 7. Desempenho do modelo nos dados de teste (site Kaggle)