Feature Selection a Bala de Prata

[RODRIGO SANTANA](http://minerandodados.com.br/index.php/author/rodrigodb28/) MAIO 21, 2018 [CAFÉ COM CÓDIGO](http://minerandodados.com.br/index.php/category/cafe-com-codigo/), [MACHINE LEARNING](http://minerandodados.com.br/index.php/category/machine-learning/)

A capacidade de aprendizado dos algoritmos está totalmente relacionada com os **dados disponíveis**.

Eu sei que isso é obvio, mas as vezes a gente esqueci disso e ficamos gastando tempo demais em tunning do algoritmo ou até mesmo testando diversos algoritmos.

A fase de **tunning** do algoritmo deve existir, não estou dizendo que você não deve conhecer bem a sua ferramenta de trabalho.

O que estou dizendo é que na maioria das vezes, o que vai fazer diferença mesmo é o trabalho gasto nos dados.

Quando falo isso me vem a cabeça: Já falei sobre a importância de fazer pré-processamento nos dados? ;P

Chega de prolixidade.

Hoje quero falar sobre algo que é muito importante e muitas vezes não damos a devida atenção.

Vejo também que muitos livros não gastam esforços em dedicar um capitulo para falar sobre tal.

Você já deve ter adivinhado, estou falando sobre a arte de fazer **feature selection**. Na tradução livre: ‘Seleção de features’.

Seleção de Features



As features que usamos para treinar o algoritmo de Machine Learning têm alta relevância na performance do modelo.

Features irrelevantes impactam negativamente no modelo além de ser um custo computacional desnecessário.

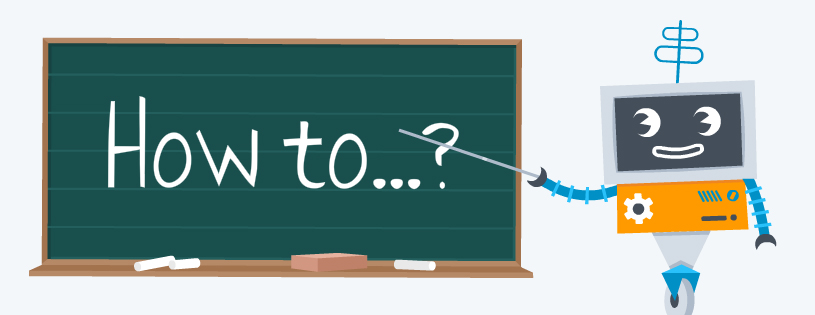
Imagina uma base de dados com mais **1000 features**, o custo computacional de treinamento do algoritmo é algo a se preocupar.

Os benefícios de ter uma boa seleção de features são:

* ***Evitar Overfitting***: Com um modelo com as features bem selecionadas evitamos oferfitting dado que temos modelos menos complexos, ou ainda, modelos menos ruidosos.
* ***Melhora da Performance***: Em algumas situações temos uma melhora na performance do algoritmo, devido a um modelo se tornar mais confiável.
* ***Redução do tempo de treinamento***: Modelos com muitas features tendem a ter um custo de treinamento muito alto, levando a um tempo muito grande para essa etapa. Isso pode ser um problema para alguns projetos.

Existem vários outros benefícios de fazer uma boa seleção de features, agora vamos para a parte mais interessante.

Como fazer feature selection?



Existem diversas formas de fazer a seleção de features.

Vou mostrar como podemos fazer isso usando um recurso da Scikit-learn.

Vamos ver a técnica que usa o teste estatístico [F-test](https://en.wikipedia.org/wiki/F-test).

O **F-Test** é usado para comparar modelos com configurações de features diferentes.

Isso é feito através da comparação erros de modelos distintos, de forma que as os erros de predição são usados para comparação entre modelos.

O F-test é utilizado para selecionar features, pois, a cada teste de modelo é dada um valor de importância para cada feature conforme cada feature melhora ou piora o modelo.

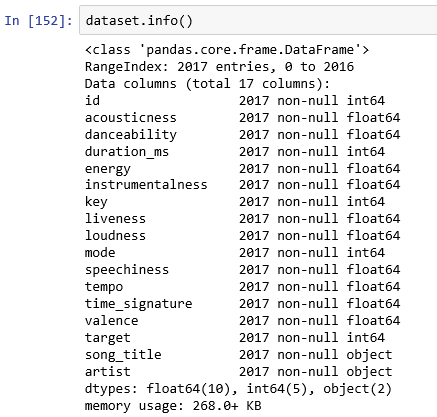
Vamos ver um teste de utilização desse recurso em uma base real de músicas do Spotify.

**Obs.:** Se você não viu o artigo onde aplicamos um SVM para classificar músicas do Spotify leia: [Classificando Músicas do Spotify com SVM (Com Códigos Python)](http://minerandodados.com.br/index.php/2018/04/04/spotify-svm-python/)

Vamos dar uma olhada na base de dados:

dataset = pd.read\_csv(‘data.csv’, sep=’,’)

dataset.info()



Veja que nossa base de dados possui 17 colunas.

Vamos excluir colunas **‘id’** e ‘**song\_title**’, pois, estas não tem relevância para o modelo.

Também iremos separa a coluna **target**, que são as nossas classes.

Dessa forma ficariamos com **14** colunas que seriam 14 features potenciais.

No artigo de SVM mostro que chegamos em uma acurácia por volta de **67%.**

Agora iremos aplicar o método de seleção de features e ver se isso melhora a performance do modelo.

Aplicando o F-test

Para utilizar o F-Test vamos importar o módulo feature\_seclection da scikit-learn:

from sklearn import feature\_selection

Em seguida irei criar um objeto que fará a seleção dos atributos, veja:

fs = feature\_selection.SelectPercentile(feature\_selection.f\_classif, percentile=50)

No exemplo acima usamos o método **SelectPercentile** para seleção das features.

O parâmetro percentile=50 diz que iremos testas o modelo com 50% das features, ou seja, mantém apenas 50% das features com maior pontuação de importância para o modelo.

O argumento **feature\_selection.f\_classif** significa que queremos usar o F-Test.

Essa técnica é utilizada para problemas de classificação, como no nosso caso.

Caso você queira testar em um problema de regressão use o f\_regression.

Com o objeto “fs” criado vamos treinar o algoritmo para efetivamente gerar um modelo com a seleção de features.

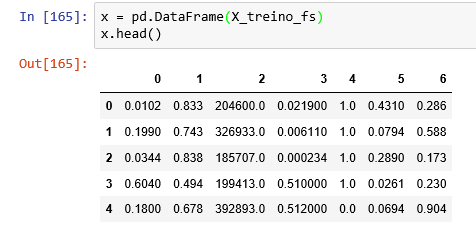
X\_treino\_fs = fs.fit\_transform(dataset, classes)

A variável X\_treino\_fs recebe agora um conjunto de dados com as 50% das instâncias com melhor importância para o modelo.

Veja o conteúdo de X\_treino\_fs:

x = pd.DataFrame(X\_treino\_fs)

x.head()

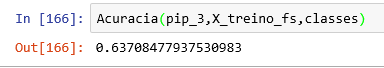


X\_treino\_fs é um array numpy, para melhor visualização gerei um Dataframe do pandas.

Veja que temos agora apenas **7** colunas. Estas seriam as colunas com maior importância para o modelo.

Vamos ver a performance do modelo com apenas 50% das features (7 features):

Acuracia(pip\_3,X\_treino\_fs,classes)



O modelo ficou **pior** que com todas as features.

Talvez o modelo ficou muito simples e não conseguimos uma acurácia boa, 63% com as 7 features.

Vamos aumentar o número de features, vamos dizer que aceitamos um modelo com **80%** do número de features originais.

Recriando o objeto fs:

fs = feature\_selection.SelectPercentile(feature\_selection.f\_classif, percentile=80)

Agora vamos treinar o modelo:

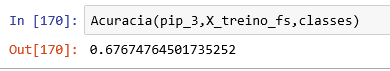
X\_treino\_fs = fs.fit\_transform(dataset, classes)

x = pd.DataFrame(X\_treino\_fs)

len(x.columns)

Agora nosso modelo tem **11** features, vamos testar o seu desempenho:

Acuracia(pip\_3,X\_treino\_fs,classes)



**67%**, já melhorou diante do nosso primeiro teste.

Vimos que com 11 features, conseguimos a mesma performance do modelo normal !

Vamos fazer mais um teste, mas agora com **90%** do número de features:

Recriando o objeto fs:

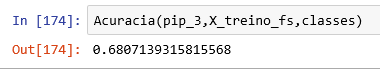
fs = feature\_selection.SelectPercentile(feature\_selection.f\_classif, percentile=90)

Treinando o modelo:

**X\_treino\_fs = fs.fit\_transform(dataset, classes)  
x = pd.DataFrame(X\_treino\_fs)  
len(x.columns)**

Agora temos **12 features**, vamos ver o desempenho do classificador:

Acuracia(pip\_3,X\_treino\_fs,classes)



**68%**de acurácia**,**tivemos uma pequena melhora.

Com menos features, tivemos 1% de melhora na acurácia.

Isso mostra que o modelo melhorou com menos duas features.

Conclusão

Conforme vimos acima, a seleção de features é uma tarefa muito importante em projeto de Data Science.

Com features ineficientes os algoritmos de Machine Learning não tem um bom desempenho.

No exemplo que mostrei acima, apliquei uma única técnica, porém existem várias e cada uma é adequada para um tipo de problema.

Vimos que 90% das features do modelo original tivemos uma melhora no desempenho do classificador.

Para uma base muito pequena isso talvez não faça muita diferença, mas para uma base com centenas de features seria de extrema valia esse tipo de recurso.

Se você tem alguma dúvida ou sugestão basta deixar um comentário, vou gostar de interagir com você.

Forte Abraço!