Machine Learning Engineer Nanodegree

Projeto Final

Gleber A. Baptistella
Fevereiro de 2018

I. Definição

Visão geral do projeto

A área de Reconhecimento de Atividades Humanas (RAH) está em franca expansão, com dispositivos e sensores cada vez mais fazendo parte do dia-a-dia das pessoas.

Suas pesquisas podem ser aplicadas em diversas áreas, entre elas da saúde (melhorando a detecção e diagnósticos de doenças) e de práticas esportivas, visando a melhoria do desempenho do indivíduo.

Descrição do problema

Este projeto tem como objetivo identificar uma determinada postura de um indivíduo após a coleta de dados através de sensores que foram previamente dispostos no corpo da pessoa. O embasamento deste projeto encontra-se em **http://groupware.les.inf.puc-rio.br/har**, bem como o dataset utilizado.

O dataset disponibilizado está em formato CSV e está bem estruturado, facilitando a leitura.

A saída do processamento é um valor categórico que representa a posição do indivíduo e suas possibilidades são: sitting, sittingdown, standing, standingup, walking. Em português seria algo como: sentado, sentando, de pé, levantando e andando.

Dessa forma, trata-se de um problema de aprendizado supervisionado de classificação.

Irei avaliar outros algoritmos de classificação diferentes do usado no artigo da PUC Rio, buscando um desempenho superior ao obtido.

Métricas

As métricas para avaliação dos modelos serão:

1. Recall
2. Precision
3. F1
4. ROC AUC

Todas as métricas são referentes aos problemas de classificação.

É comum utilizar a métrica *Accuracy* nos problemas de classificação, porém em alguns casos ela não é uma boa métrica. Tomemos como exemplo um sistema de classificação de fraudes em transações de cartão de crédito. A grande maioria das transações, digamos 99%, são legítimas. Apenas cerca de 1% das transações são fraudulentas. Se fizermos um modelo de predição onde considero que todas as transações são legítimas, minha taxa de acerto será de 99%, ou seja, a métrica acurácia será de 99%. Contudo este não é um modelo confiável apesar da alta taxa de acerto, já que todas as transações fraudulentas serão classificadas como legítimas. Para situações como esta temos as métricas referidas: Recall, Precision, F1 e ROC AUC.

As métricas levam em consideração as taxas de falsos positivos, verdadeiros positivos, verdadeiros negativos e falsos negativos para analisar a qualidade do modelo. Abaixo uma *confusion matrix* representando os resultados possíveis num problema de classificação:

		Valor Verdadeiro (confirmado por análise)	
		positivos	negativos
Valor Previsto (predito pelo teste)	positivos	**VP** Verdadeiro Positivo	**FP** Falso Positivo
	negativos	**FN** Falso Negativo	**VN** Verdadeiro Negativo

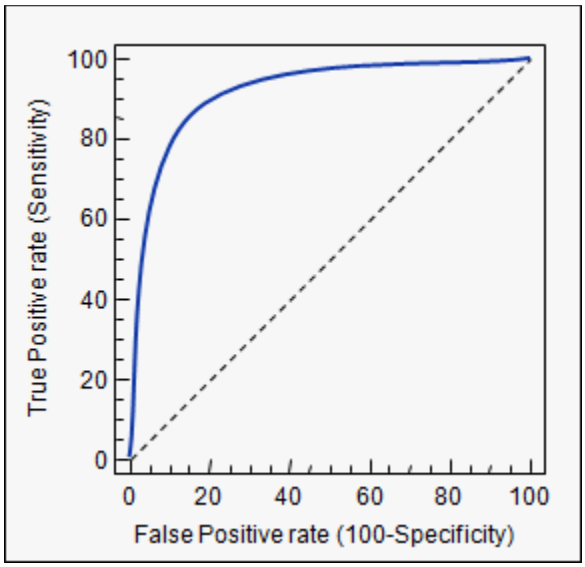
Dessa forma definimos as métricas da seguinte forma aplicando no exemplo do cartão de crédito:

Recall: é a capacidade do modelo predizer corretamente as operações legítimas. Seguindo pela tabela a fórmula seria: `VP/(VP+FN)`

Precision: é a capacidade do modelo predizer corretamente as operações fraudulentas. Seguindo pela tabela a fórmula seria: `VN/(VN+FP)`

F1: é a média harmônica entre Recall e Precision. A fórmula é: `2 * (precision * recall)/(precision + recall)`. O resultado é um número entre 0 e 1, onde 0 é o pior modelo e 1 é o modelo mais preciso.

ROC AUC: a curva ROC (*Receiver Operating Characteristics*) é um gráfico bidimensional que utiliza a taxa de verdadeiros positivos no eixo Y e a taxa de falsos positivos no eixo X.



A linha tracejada no meio do gráfico seria um modelo aleatório. A métrica AUC (*area under curve*) é a área abaixo da curva ROC (curva em azul) e pode variar de 0 a 1, sendo 0 o pior valor e 1 o melhor valor.
Como veremos na sessão "**II. Análise**", o dataset para o problema proposto também encontra-se desbalanceado. Dessa forma as métricas propostas são as mais adequadas para verificarmos a qualidade do modelo.

II. Análise

Exploração de dados

O conjunto de dados selecionado encontra-se disponível em http://groupware.les.inf.puc-rio.br/static/har/dataset-har-PUC-Rio-ugulino.zip e seus atributos estão descritos abaixo:

Dados Pessoais:

- user: nome da pessoa
- gender: sexo da pessoa
- how_tall_in_meters: altura da pessoa
- weight: peso da pessoa
- body_mass_index: Índice de massa corpórea da pessoa

Dados dos sensores:

São colocados 4 sensores no corpo do indivíduo.

Para cada sensor são definidos os eixos x, y e z e no dataset temos para o sensor 1 os dados, x1, y1 e z1. Para o sensor 2, os dados x2, y2 e z2, e assim por diante. A posição de cada sensor é:

- Sensor 1: cintura
- Sensor 2: coxa esquerda
- Sensor 3: canela direita
- Sensor 4: braço direito.

Por fim, temos a *feature* "class". Trata-se da variável que queremos prever e que pode assumir os seguintes valores: 'sitting', 'sittingdown', 'standing', 'standingup', 'walking'

Durante a coleta de dados, que consistiu em 8 horas de atividades de quatro indivíduos distintos, foram geradas 165362 amostras dos sensores. Trata-se de um bom volume de dados para aplicação dos algoritmos.

Após verificação foi constatada a ausência de *missing values*, o que é um aspecto muito positivo já que não será necessário inferir valores fictícios como a média ou a mediana para os atiributos que porventura não existissem.

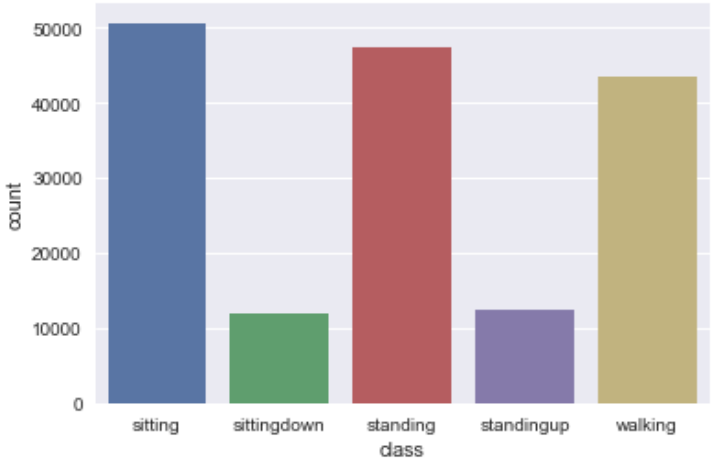
O atributo *gender* foi convertido para valores binários sendo 0 para *Woman* e 1 para *Man*.

O atributo *class* foi convertido da seguinte forma:

0 – sitting
1 – sittingdown
2 – standing
3 – standingup
4 – walking

Visualização Exploratória

A distribuição das classes de classificação do dataset estão distribuídas da seguinte forma:



- sittingdown = 7,15%
- standingup = 7,50%
- walking = 26,23%
- standing = 28,64%
- sitting = 30,45%

As classes "sittingdown" e "standingup" estão sub-representadas e teremos que balancear o dataset para termos uma distribuição melhor entre as classes.

Para lidarmos com esta situação temos algumas opções. Podemos deletar registros das classes mais representativas para termos o mesmo nível de dados das classes menos representativas. Ou então podemos fazer o inverso: criar dados sintéticos das classes menos representativas. Neste projeto será utilizada a técnica SMOTE – Synthetic Minority Over-sampling Technique – ou seja, criaremos dados para as classes menos representativas para nivelarmos a quantidade de todas as classes. Note-se que esta técnica deve ser aplicada apenas no dataset de treinamento e não no dataset de testes.

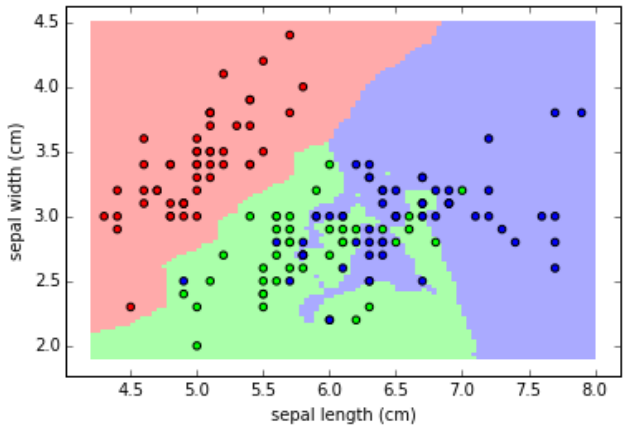
Algoritmos e Técnicas

Neste projeto iremos testar 3 algoritmos:

1) K-Nearest Neighbors
2) Random Forest
3) Gaussian Naive Bayes

K-Nearest Neighbors

O KNN tenta capturar o relacionamento entre as instâncias medindo a distância entre eles. Esta distância pode ser medida pela distância Euclidiana ou Manhattan.



Ao lado temos a classificação do dataset "Iris" após a execução do algoritmo, formando regiões a partir da distância entre os elementos do dataset.

Vantagens:

- Treinamento rápido em datasets grandes.
- Algoritmo simples.

Desvantagens:

- Por tratar-se de um algoritmo *lazy learning*, isto é, não gera uma função no treinamento para ser aplicado nos testes. Ao invés disso ele "memoriza" o resultado do dataset de aprendizado. Dessa forma a predição de novas entradas são computacionalmente custosas para serem realizadas.
- A escolha do valor "K", ou seja, o número de vizinhos próximos, não é trivial.

Random Forest

O algoritmo Random Forest é um algoritmo que utiliza um conjunto combinado de árvores de decisão para criar um modelo de classificação, por isso é chamado de "Ensemble Learning".

Vantagens:

- É capaz de lidar com uma grande quantidade de *features*.
- Trabalha bem com *missing values* e *outliers*.

- Custo computacional menor para os dados de testes ou produção.

Desvantagens:

- Custo computacional de treino maior.
- Difícil interpretação

Gaussian Naive Bayes

O algoritmo Gaussian Naive Bayes é baseado no Naive Bayes clássico, porém suporta apenas valores contínuos. O algoritmo pressupõe independência entre as *features*, isto é, as *features* não tem relação umas com as outras.

Vantagens:

- Rápido de ser executado.
- Lida bem com dimensionalidade alta.
- Aceita *missing values*.

Desvantagens:

- A premissa de independência entre as *features* é muito forte. A ausência desta premissa irá gerar um mau resultado do classificador.

Benchmark

Neste projeto teremos dois modelos de referência:

1. A execução sem refinamento dos outros três algoritmos.
2. O resultado do artigo original

Benchmark execução sem refinamento

	Precision	*Recall*	*F1 Score*
KNN	0,9901	0,9905	0,9903
Random Forest	0,9915	0,9903	0,9909
Gaussian NB	0,6738	0,6467	0,6452

Benchmark artigo original

No artigo original as métricas são dadas por classe, conforme tabela abaixo:

	Precision	*Recall*	*F1 Score*	*ROC AUC*
Sitting	1	0,999	0,999	1
Sitting down	0,969	0,971	0,970	0,999
Standing	0,998	0,999	0,999	1
Standing up	0,969	0,962	0,965	0,999
Walking	0,994	0,994	0,994	1

III. Metodologia

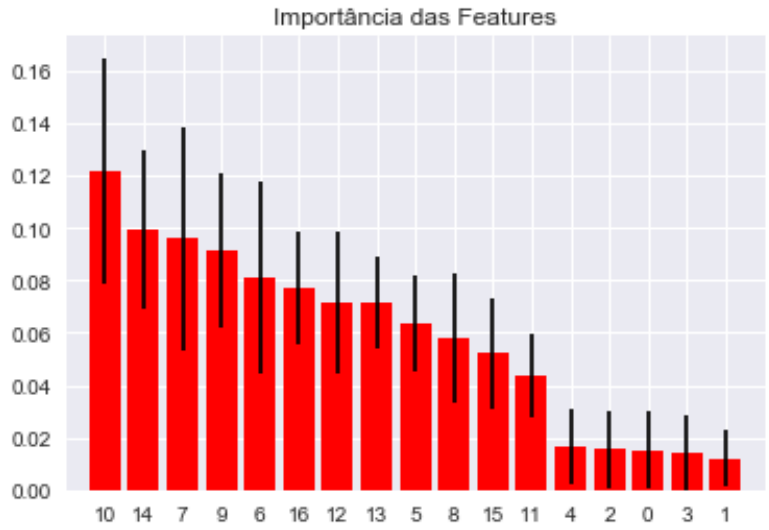
Data Preprocessing

Nas sessões anteriores já foram explicados dois pré-processamentos que fizemos no nosso dataset:

1. Convertemos as variáveis categóricas que estavam como *string* para valores numéricos.
2. Aplicamos a técnica de criação de dados sintéticos (SMOTE) para balancear as classes do dataset.

Após estes pré-processamentos, colocamos todos os valores em uma escala de 0 a 1. Esta prática é importante pois o resultado pode ser distorcido por variáveis numéricas muito díspares. Por exemplo: imagine um algoritmo que tenha as *features* peso e salário. A ordem de grandeza destas *features* são diferentes, tendo o peso valores de dezenas e no máximo de centenas e os salários valores de milhares. Isso pode ter um péssimo efeito para determinados algoritmos. Dessa forma utilizamos a classes `MinMaxScaler` do pacote `sklearn.preprocessing` para atribuir valores entre 0 e 1 nas *features*.

Buscando reduzir a dimensionalidade do dataset para evitar a utilização de *features* pouco significativas para os modelos e que impactem no tempo de execução, foi utilizado o classificador `ExtraTreeClassifier` para determinar a importância das variáveis no dataset. A ordem de importância de cada variável está descrita abaixo:



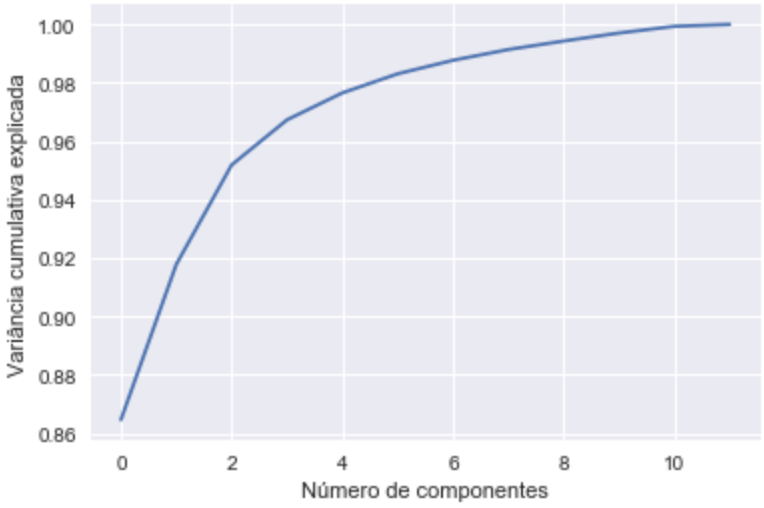
1. feature Z2
2. feature X4
3. feature Z1
4. feature Y2
5. feature Y1
6. feature Z4
7. feature Y3
8. feature Z3
9. feature X1
10. feature X2
11. feature Y4
12. feature X3
13. feature body_mass_index
14. feature how_tall_in_meters
15. feature gender
16. feature weight
17. feature age

Pelo gráfico podemos ver que as variávies cujo índice estão no *range* de 0 a 4 pouco influenciam no modelo. Tratam-se de variáveis de características pessoais dos dados coletados: altura em metros (how_tall_in_meters), índice de massa corpórea (body_mass_index), peso (weight), gênero (gender) e idade (age). Portanto estas variáveis foram removidas do dataset.

Em seguida foi aplicada o PCA (*Principal Component Analysis*). Esta técnica utiliza princípios de álgebra linear para transformar variáveis, possivelmente correlacionadas, em um número menor de variáveis chamadas de Componentes Principais.

Usemos com exemplo um dataset onde existam as *features* idade e anos escolaridade. Suponha que estas duas *features* estão correlacionadas entre si, ou seja, quanto maior a idade, maior os anos de escolaridade do indivíduo. Aplicando-se a técnica do PCA, estas duas *features* poderiam ser transformadas em uma única, ou um componente principal.

Para o nosso exemplo, podemos verificar que entre 8 e 9 componentes explicam 99% da variância do da dataset (ver gráfico abaixo). Isso significa que podemos transformar nosso dataset para que fique com 9 *features*, ou neste caso, 9 componentes principais, reduzindo a dimensionalidade inicial de 17 *features* para 9.



Apesar da redução significativa da dimensionalidade, a cada fase do processo após o escalonamento dos dados houve uma perda nas métricas comparadas com o benchmark. Portanto, na fase de refinamento dos algoritmos utilizaremos o dataset com os dados escalonados, ou seja, com o número de *features* original, porém com os dados numa escala de 0 a 1.

Implementação

Para cada ciclo de pré-processamento era feita a execução dos três algoritmos: Knn, Randon Forest e GaussianNB, e era registrado seus resultados.

Todas as execuções foram executadas com o dataset dividido da seguinte forma:

- 65% dos registros para treinamento dos modelos.
- 35% dos resgistros para teste dos modelos.

Como mencionado na sessão "Visualização Exploratória", a técnica SMOTE deve ser aplicada apenas nos 65% dos dados para treinamento.

Contudo, o escalonamento dos dados, a remoção das variáveis e a transformação do dataset através do PCA foram aplicadas tanto nos dados de treinamento como nos dados de testes.

Refinamento

Para o refinamento do modelo foi utilizada a classe `GridSearchCV` do pacote `sklearn.model_selection`.

Com esta classe é possível fazer uma busca exaustiva pelo melhor modelo, passando uma lista de hiperparâmetros para o classificador. Isto é, a classe irá combinar a lista de hiperparâmetros executando o classificador até encontrar o melhor dado um escore específico. Abaixo seguem os hiperparâmetros para cada classificador.

Conforme dito anteriormente, o dataset utilizado será o escalonado, pois com ele tivemos a melhor performance até aqui.

K-Nearest Neighbors

- *n_neighbors*: número de vizinhos para classificar uma nova entrada. Valores: 5, 6, 7, 8 e 9
- *metric*: distância utilizada para a medição. Valores: Manhattan, Euclidean e minkowski.
- *weights*: pesos usados nas predições. Valores: *uniform* (todos os pontos em cada vizinhança tem o mesmo peso), *distance* (pesos inversamente proporcionais à distância, ou seja, vizinhos mais próximos tem um peso maior).

Random Forest

- *n_estimators* : número de árvores que serão combinadas. Valores: 10 e 250 *estimators*.
- *min_samples_split*: número mínimo de amostras para a quebra do nó. Valores: 2, 4 e 6.
- *min_samples_leaf*: número mínimo de amostras para que um nó seja folha (o último nó da árvore). Valores: 3 e 5.
- *criterion:* métrica da qualidade da quebra. Valores: *gini* e *entropy*.

Gaussian NB

O classificado GaussianNB não possui parâmetros para serem refinados.

IV. Resultados

Avaliação e validação dos modelos

Nesta sessão iremos comparar os resultados obtidos com o benchmark.

Resultados gerais

Benchmark

	Precision	*Recall*	*F1 Score*
KNN	0,9901	0,9905	0,9903
Random Forest	0,9915	0,9903	0,9909
Gaussian NB	0,6738	0,6467	0,6452

Após refinamento

	Precision	*Recall*	*F1 Score*
KNN	0,9904	0,9932	0,9918
Random Forest	0,9939	0,9941	0,9940
Gaussian NB	0,6485	0,6545	0,6340

Podemos observar que tivemos um ganho de performance nos algoritmos K-Nearest Neighbors e no Random Forest em todos os indicadores. O benchmark já partia de um patamar alto, com bom desempenho logo no início.

O procedimento que auferiu maiores ganhos durante o pré processamento de dados foi o balanceamento do dataset. Dessa forma conseguimos criar mais instâncias para as classes que eram subrepresentadas.

Após esta última fase de refinamento, a melhor performance foi o *Random Forest*, que obteve as métricas *Precision*, *Recall* e *F1* mais altas.

Resultados por classe

Benchmark da PUC Rio

	Sitting	*Sitting down*	*Standing*	*Standing Up*	*Walking*
Precision	1,000	0,969	0,998	0,969	0,994
Recall	0,999	0,971	0,999	0,962	0,994
F1 Score	0,999	0,970	0,999	0,965	0,994
ROC AC	1,000	0,999	1,000	0,999	1,000

K-Nearest Neighbors

	Sitting	*Sitting down*	*Standing*	*Standing Up*	*Walking*
Precision	1,000	0,979	0,993	0,982	0,999
Recall	0,999	0,994	0,998	0,982	0,988
F1 Score	1,000	0,986	0995	0,984	0,993
ROC AC	1,000	0,996	0,998	0,993	0,994

Random Forest

	Sitting	*Sitting down*	*Standing*	*Standing Up*	*Walking*
Precision	1,000	0,990	0,998	0,986	0,996
Recall	0,999	0,991	0,997	0,986	0,998
F1 Score	1,000	0,990	0,997	0,986	0,997
ROC AC	1,000	0,995	0,998	0,992	0,998

Gaussian Naive Bayes

	Sitting	*Sitting down*	*Standing*	*Standing Up*	*Walking*
Precision	0,958	0,435	0,699	0,279	0,872
Recall	0,865	0,702	0,921	0,279	0,639
F1 Score	0,909	0,537	0,795	0,192	0,738
ROC AC	0,924	0,816	0,881	0,558	0,803

O benchmark e os algoritmos Knn e Random Forest tem desempenho muito próximos, mas o benchmark tem um desempenho superior quando comparamos as métricas F1 e ROC AUC.

Dado todos os testes realizados, acredito que temos resultados robustos que validam os algoritmos utilizados tanto no benchmark original, tanto utilizados neste projeto.

V. Conclusão

Considero o estudo realizado de grande utilidade para os projetos de Reconhecimento de Atividades Humanas (HAR).

Tanto algoritmo utilizado no artigo original, como o Random Forest e o K-Nearest Neighbors poderiam ser utilizados para detecção de posições do corpo humano.

Como exemplos de uso, podemos citar o monitoramento de indivíduos da terceira idade com dificuldade de locomoção, a melhoria de performance de atletas, o monitoramento de pacientes em recuperação de tratamentos ortopédico, entre outras.

Embora tenhamos um ótimo resultado, acredito que a adição de sensores em pontos específicos poderia melhorar a assertividade dos modelos.

Referências

Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.

Read more: http://groupware.les.inf.puc-rio.br/har#ixzz577kG5mmP

Receiver Operating Characteristic (ROC)

Confusion matrix

Feature importances with forests of trees