Terceiro trabalho prático

Herberth Amaral e Marcelino Macedo

Departamento de Ciência da Computação
Universidade Estadual de Montes Claros
Professor Dr. Renato Dourado Maia

10 de novembro de 2015

1 Introdução

A tarefa de avaliar o desempenho de diferentes algoritmos de classificação na área de Mineração de Dados e Aprendizagem de Máquina (Machine Learning) vêm sendo discutida nos últimos anos, devido a principalmente à falta de validade estatística dos resultados. Em [4][5][8] são discutidos as principais falhas no ponto de vista de análise estatística dos resultados como o uso incorreto de testes paramétricos *t-pareado (comparação pareada de resultados)* e sugere o uso de testes não paramétricos como *Wilcoxon Signed Rank Test (comparação pareada)*

No presente trabalho iremos utilizar 15 algoritmos de classificação do [9] em três data sets e iremos obter o desempenho médio da acurácia (taxa de acerto) de cada algoritmo em um data set, depois será gerado um rank do desempenho dos mesmos em cada dataset. Tal rank não possui grande confiabilidade estatística, pois apenas a média não permite inferir nada sobre as distribuições dos dados. Sendo assim, iremos utilizar testes não-paramétricos para avaliar o desempenho dos 2 algoritmos campeões em um data set, no caso , o teste *Wilcoxon Signed Rank*, um teste não paramétrico, onde dois classificadores são comparados entre si de forma pareada, com o objetivo de verificar a hipótese nula de que o desempenho dos dois são equivalentes, a hipótese alternativa de que o desempenho de um é melhor do que do outro.

1.1 Objetivos

Os objetivos deste trabalho são:

1. Explorar opções de algoritmos de classificação;
2. Analisar o desempenho desses algoritmos segundo critérios de precisão, tempo de treinamento e de execução.
3. Utilizar testes estatísticos não paramétricos para avaliação de desempenho de classificadores em situações de um problema e em casos de multi-problemas.

2 Metodologia

2.1 Data sets

Para o desenvolvimento desse trabalho foram escolhidos 3 data sets do repositório da UCI para Machine Learning [12], sendo elas : Abalone[1] Bank Note Authentication [2] Car Evaluation [3], respectivamente com múltiplas classes (28 classes diferentes, sendo bastante desbalanceada),duas classes balanceadas, e um com 4 classes balanceadas).

A escolha das bases foi feita com o objetivo testar o desempenho dos algoritmos de classificação com relação a variação do número de classes disponíveis nas bases.

2.2 Algoritmos

A lista de algoritmos utilizados neste trabalho estão disponíveis no scikit-learn[9],foram estão selecionados 15 de diversos tipos (probabilísticos,árvores de decisão, modelos lineares,e outros).

1. *sklearn.linear_model.SGDClassifier*;
2. *sklearn.linear_model.Perceptron*;
3. *sklearn.linear_model.PassiveAggressiveClassifier*;
4. *sklearn.lda.LDA*;
5. *sklearn.kernel_ridge.KernelRidge*;
6. *sklearn.svm.SVC*;
7. *sklearn.svm.NuSVC*;
8. *sklearn.svm.LinearSVC*;
9. *sklearn.neighbors.RadiusNeighborsClassifier*;
10. *sklearn.neighbors.KNeighborsClassifier*;
11. *sklearn.naive_bayes.GaussianNB*;
12. *sklearn.naive_bayes.MultinomialNB*;
13. *sklearn.naive_bayes.BernoulliNB*;
14. *sklearn.tree.DecisionTreeClassifier*;
15. *sklearn.ensemble.GradientBoostingClassifier*;

2.3 Descrição do Ambiente de Experimento

A implementação dos algoritmos será a padrão disponível no módulo sklearn [9]. Os data sets passaram por uma fase de pré-processamento e normalização para se ajustar a alguns algoritmos.

Para as fases de treinamento, teste e validação , foi utilizada a estratégia de validação cruzada ou *(*cross validation) utilizando a técnica *k-fold* no caso especificamente o 10-fold , onde foram gerados 10 folds, que corresponde a utilizar 1 fold para a teste e outros 9 para treinamento , equivalente a treinar com 90% dos dados e testar em 10%. Para melhor variância dos dados no processo de treinamento e teste cada algoritmo será executado 100 vezes em cada amostra de um data set.

Um arquivo irá armazenar a métrica de desempenho do classificador (acurácia) e outras métricas (tempos de treinamento e teste,etc).Cada linha desse arquivo representa os resultados do algoritmo em uma interação.

2.4 Análise de Desempenho

Os algoritmos serão rankeados pelo seu desempenho médio da acurácia em cada data set, com média e desvio padrão. Para justificar o uso de testes não paramétricos , iremos verificar se de fato os resultados obtidos são de uma distribuição normal ou não, para isso serão utilizados os testes de normalidades citados no artigo [5] , como o *Kolmogorov-Smirnov* implementado em Python como *kstest* [10] e outros testes que garantem ou não o uso seguro de testes paramétricos.

Como os dados obtidos não seguem todos os critérios de normalidade, serão utilizados testes não paramétricos, que não pressupõem nenhuma informação a respeito de qual distribuição os dados são, assim podemos utilizá-los para avaliar o desempenho dos dados de forma estatisticamente válida.

Utilizaremos na comparação os testes de Wilcoxon Signed Rank, para teste pareado para identificar o melhor algoritmo dentre os dois melhores colocados no rank realizado inicialmente, como o objetivo de verificar e validar se de fato o desempenho entre um deles é estatisticamente superior ao do outro. Os dois melhores em um data set serão comparados.

3 Desenvolvimento

O presente trabalho foi desenvolvido na linguagem Python 2 e seu respectivo código (com as devidas instruções de execução) pode ser encontrado no endereço `https://github.com/herberthamaral/mestrado/tree/master/MD/pratica3`.

Ferramentas auxiliares foram utilizadas no desenvolvimento deste trabalho:

1. Numpy [6];
2. Scikit-learn [9];
3. Scipy [10]

Todos os testes foram executados em um Intel Core i5 de segunda geração (2 processadores, 4 *threads*) com 6GB de RAM.

Os seguintes algoritmos foram avaliados:

1. *sklearn.linear_model.SGDClassifier*;
2. *sklearn.linear_model.Perceptron*;
3. *sklearn.linear_model.PassiveAggressiveClassifier*;
4. *sklearn.lda.LDA*;
5. *sklearn.kernel_ridge.KernelRidge*;
6. *sklearn.svm.SVC*;

7. *sklearn.svm.NuSVC*;
8. *sklearn.svm.LinearSVC*;
9. *sklearn.neighbors.RadiusNeighborsClassifier*;
10. *sklearn.neighbors.KNeighborsClassifier*;
11. *sklearn.naive_bayes.GaussianNB*;
12. *sklearn.naive_bayes.MultinomialNB*;
13. *sklearn.naive_bayes.BernoulliNB*;
14. *sklearn.tree.DecisionTreeClassifier*;
15. *sklearn.ensemble.GradientBoostingClassifier*;

3.1 Técnicas de implementação

Aproveitando do fato que as classes que implementam os algoritmos de classificação seguem a mesma interface, implementamos o algoritmo de testes dos classificadores utilizando técnicas de reflexão. Essas técnicas permitiram que o algoritmo de teste ficasse mais generalista e resumido, uma vez que não é necessário implementar um teste específico para cada algoritmo.

Devido ao alto tempo de execução e a alta possibilidade de paralelismo, utilizamos o módulo de multiprocessamento do Python para diminuir o tempo de execução dos testes e fazer melhor uso dos recursos computacionais. A quantidade de subprocessos é determinada pela quantidade de processadores disponíveis no ambiente que o algoritmo é executado (4 subprocessos utilizando a máquina de testes descrita anteriormente).

Além da execução paralela, duas pequenas otimizações foram feitas com o intuito de dimiuir o tempo de processamento. Duas técnicas foram utilizadas: *lazy load* dos datasets e uma otimização do algoritmo *minmax* em que reduzimos a complexidade de $O(n^2)$ para $O(n)$.

Pelo mesmo motivo de tempo de execução apontado anteriormente, implementamos um mecanismo de retomada da execução do algoritmo de testes: a cada uma das cem iterações salvamos o estado da execução. O algoritmo continua a execução de onde parou caso uma parada aconteça.

3.2 Pré-processamento de dados

Com exceção da base de dados *banknot*, as bases de dados contêm atributos não-numéricos que precisam ser tratados antes. Esses atributos foram substituídos por valores inteiros com o intuito de permitir o uso nos classificadores. Esse pré-processamento pode ser analisado no arquivo *main.py* na função *trata_datasets()*.

Além disso, todos os dados numéricos (exceto as classes) foram normalizados utilizando a técnica **minmax** ($minmax(X) = \frac{x - Min(X)}{Max(X) - Min(X)}, \forall x \in X$). O minmax normaliza

os dados no intervalo $[0, 1]$ e isso é especialmente interessante para os classificadores Bayesianos, os quais não aceitam entradas negativas.

3.3 Execução e pós-processamento

Com 16 algoritmos de classificação, três datasets e validação cruzada 10-fold, cada iteração demora cerca de seis minutos com a configuração detalhada anteriormente. Como executamos cem iterações, a execução total do nosso algoritmo ficou em cerca de dez horas.

A implementação de recursos de paralelismo ajudou a diminuir o tempo de execução, porém, notamos que boa parte do tempo é gasta sintetizando os resultados do teste, o que usa apenas um processador. Fazendo uma analogia com o modelo de programação MapReduce (colocar referência), uma quantidade significativa de tempo foi gasta na fase de redução, porém uma boa fatia de tempo foi economizada na fase de mapeamento.

Salvamos um arquivo com os dados produzidos pelo nosso algoritmo no formato JSON a cada iteração. Esse arquivo contém as seguintes informações:

1. Tamanho do dataset;
2. Tempo (em segundos) usado para treinamento;
3. Tempo (em segundos) usado para validação;
4. Precisão (no intervalo $[0..1]$) do modelo;
5. Quantidade de erros de validação;
6. *Datset* utilizado;
7. Algoritmo de classificação utilizado;
8. Matriz de confusão;

Para facilitar a análise dos dados, desenvolvemos um utlitário para unificar os arquivos JSON e converte-los para CSV (com exceção da matriz de confusão).

4 Resultados Preliminares

Nas tabelas 4,4 e 4 estão os resultados dos algoritmos nos data sets *Abalone, Bank Note Authentication e Car Evaluation* respectivamente. Como pode ser notado na Tabela 1, o desempenho de todos os algoritmos foram relativamente baixos na base *Abalone*. Esse resultado foi de certa forma esperado uma vez que a base não foi discretizada durante a fase de pré-processamento, então foram mantida as características de 28 classes e as mesmas não são balanceadas e a maioria delas possuem apenas 1 elemento por classe o que dificulta a tarefa de classificação dos algoritmos.

Alguns algoritmos apresentaram desempenho praticamente determinístico com pouco ou até nenhuma diferença de desempenho nas 100 execuções realizadas sobre essa base (Tabela 1), com exceções do *LinearSVC*,*Gradient Boosting*,*SGD*,*Decision Tree*, *Passive Aggressive* que apresentaram uma variação da taxa de acúracia durante o experimento.

Como o objetivo deste trabalho é verificar e comparar o desempenho dos algoritmos no ambiente de teste, não será discutido os detalhes de porquê um determinado algoritmo obteve acúrácia maior que outro. Detalhes como: se o problema (data set) é linearmente separável ou não, ou como internamente um algorítmo trata conjuntos densos ou esparços , ou ainda, qual a técnica utilizada para classificação múltipla interna do algoritmo está fora do escopo desse trabalho.

Antes de realizar os testes estatísticos de comparação de desempenho, foi feito testes de normalidade e de heteroscidacidade (equivalência de variância) nos resultados dos algorítmos para que se verificasse a possibilidade de uso de testes paramétricos de forma segura , como sugerido por [5]. Os testes a serem utilizados foram os de Dagostino-Pearson para normalidade e o de Levene para heteroscidacidade citados [5] e disponíveis no pacote scipy.stats [10] como *normaltest* e outros e *levene* respectivamente .

Os testes avaliarão a normalidade ou não dos dados da variável precisão (acurácia) obtida nas 100 execuções de cada algorítmo em cada data set. Durante a análise foi verificado que a maioria dos algoritmos apresentou falha nos testes de normalidade em todos os três data sets, cerca de 26 dos 45 testes deram não normal .(3 bases x 15 algoritmos em cada)

Tabela 1: *Rank do Desempenho dos Algoritmos no Data Set Abalone*

Posição	Algoritmo	Acurácia Média e Desvio Padrão
1	LDA	$0.2523 +/- 0.0000$
2	LinearSVC	$0.2360 +/- 0.0001$
3	Gradient Boosting	$0.2299 +/- 0.0001$
4	KNN	$0.2298 +/- 0.0000$
5	SVC	$0.2212 +/- 0.0000$
6	Decision Tree	$0.1975 +/- 0.0001$
7	MultinomialNB	$0.1877 +/- 0.0000$
8	Kernel Ridge	$0.1853 +/- 0.0000$
9	Radius Neighbors	$0.1841 +/- 0.0000$
10	BernoulliNB	$0.1779 +/- 0.0000$
11	Perceptron	$0.1592 +/- 0.0000$
12	SGD	$0.1866 +/- 0.0159$
13	Passive Aggressive	$0.1523 +/- 0.0146$
14	GaussianNB	$0.1877 +/- 0.0000$
15	NuSVC	$0.0000 +/- 0.0000$

Tabela 2: *Rank do Desempenho dos Algoritmos no Data Set Bank Note Authentication*

Posição	Algoritmo	Acurácia Média e Desvio Padrão
1	KNN	$0.9315 +/- 0.0000$
2	Decision Tree	$0.9006 +/- 0.0021$
3	Gradient Boosting	$0.8976 +/- 0.0004$
4	SVC	$0.8535 +/- 0.0000$
5	Perceptron	$0.8513 +/- 0.0000$
6	NuSVC	$0.8506 +/- 0.0000$
7	LDA	$0.8484 +/- 0.0000$
8	LinearSVC	$0.8448 +/- 0.0000$
9	Kernel Ridge	$0.8163 +/- 0.0000$
9.5	Passive Aggressive	$0.7908 +/- 0.0501$
9.5	SGD	$0.7908 +/- 0.0465$
12	GaussianNB	$0.7179 +/- 0.0159$
13	BernoulliNB	$0.6778 +/- 0.0000$
14	MultinomialNB	$0.6596 +/- 0.0000$
15	Radius Neighbors	$0.05620 +/- 0.0000$

Tabela 3: *Rank do Desempenho dos Algoritmos no Data Set Car Evaluation*

Posição	Algoritmo	Acurácia Média e Desvio Padrão
1	Decision Tree	$0.0814 +/- 0.0003$
2	Gradient Boosting	$0.0810 +/- 0.0000$
3	KNN	$0.0804 +/- 0.0000$
4	GaussianNB	$0.0793 +/- 0.0000$
5.5	Kernel Ridge	$0.0787 +/- 0.0000$
5.5	LinearSVC	$0.0787 +/- 0.0000$
7	LDA	$0.8484 +/- 0.0000$
8	SGD	$0.0754 +/- 0.0059$
9	Passive Aggressive	$0.0742 +/- 0.0069$
12	MultinomialNB	$0.0729 +/- 0.0000$
12	BernoulliNB	$0.0729 +/- 0.0000$
12	Perceptron	$0.0729 +/- 0.0000$
12	SVC	$0.0729 +/- 0.0000$
12	Radius Neighbors	$0.0729 +/- 0.0000$
15	NuSVC	$0.0000 +/- 0.0000$

4.1 Avaliações de Desempenho : Análise Single Problem

Nessa etapa utilizaremos os testes não paramétricos para validar ou não rank obtido nas bases de dados . No caso como foi verificado que a maioria dos algoritmos não passaram nos testes de normalidade, o teste estatísticos não paramétrico de Wilcoxon - *Wilcoxon Signed Rank* entre os dois algoritmos que tiveram a melhor acurácia.

4.2 Data Set Abalone

Como visto na tabela 1, os algoritmos de *LDA* e *LinearSVC* foram os que melhor obtiveram sucesso na tarefa de classificação na base Abalone. Para verificar se de fato essa diferença entre eles é estatisticamente significativa ou fruto do mero acaso, foi utilizado o método de Wilcoxon Signed Rank [5][10] o pacote onde a hipótese nula é de as duas amostras relacionados são da mesma população, caso a hipótese nula seja rejeitada , podemos concluir que não são da mesma população e a diferença é de fato significativa. Utilizando a implementação do teste na linguagem Python pelo módulo *stats.wilcoxon* entre o algoritmo *LDA* e o *LinearSVC*, no base *abalone*, considerando $\alpha = 0.05$, o *p-value* obtido foi de $1.27050005636171e-21$. Como o p-value é menor do que o nosso α , a hipótese nula de que as amostras são da mesma população foi rejeitada, e podemos concluir que a diferença apresentada pelo *LDA* foi de fato significativa em relação ao *LinearSVC* na base *abalone*

4.3 Data Set Bank

Na tabela 2, os algoritmos *KNN* e *Decision Tree* foram os que obtiveram maior acurácia média na base *Bank*, acima de 90%. Para validar o *KNN* como campeão em relação ao *Decision Tree*, foi utilizado o teste de Wilcoxon Signed Rank, que é uma alternativa não paramétrica ao método paramétrico *T Pareado*.Considerando nosso $\alpha = 0.05$, que é o nosso nível de significância , ou em outras palavras, a chance de rejeitarmos a hipótese nula sendo ela verdadeira.O *p-value* calculado entre o algoritmo *KNN* e o *Decision Tree* foi de $(3.4816449004796852e-18)$, o que é menor do que nosso α, portanto podemos concluir que o desempenho foi significativo.

4.4 Data Set Car

Na tabela 3, os algoritmos *Decision Tree* e *Gradient Boosting* foram os melhores na base *Car Evaluation*, com uma pequena vantagem ao *Decision Tree* com relação ao *Gradient Boosting*. Com o teste de Wilcoxon queremos verificar se essa pequena diferença é significativa estatisticamente. Utilizando como $\alpha = 0.05$, o *p-value* obtido entre o *Decision Tree* e o *Gradient Boosting* foi de $3.8779181150546794e-18$.Comprovando que o desempenho foi estatisticamente superior na base *Car*

5 Conclusões

O uso de médias para representar o desempenho de algoritmos é uma prática muito utilizada em artigos científicos , porém , apesar de fácil entendimento dos resultados, o seu uso não traz nenhuma validade estatistica uma vez que ela pressume que há alguma medida comum entre as amostras, o que nem sempre é verdade, por isso, a validação dos resultados com testes não paramétricos como o caso do Wilcoxon para testes pareados se faz oportuno e confiável.[8]. Como os testes não paramétricos não são feitas suposições sobre os dados , de qual distribuição são ou qualquer outra característica que deve ser verificada como nos casos dos métodos paramétricos. A proposta do trabalho foi validar os desempenhos médios de alguns algoritmos de classificação em alguns data sets de forma válida estatisticamente. O trabalho conseguiu validar os resultadas das médias dos algoritmos nas bases de dados através da confirmação por meio de testes não-paramétricos que as diferenças encontradas entre os algoritmos foram de fato significativas e não aleatórias.

6 Considerações finais

6.1 Dificuldades

Esta subseção trata das dificuldades encontradas na execução deste trabalho.

6.1.1 Planejamento

Em alguns programas de pós graduação (como por exemplo o PPGEE, da UFMG) há oferta de disciplina de Análise de Experimentos. Encontramos referências de livros abertos na Web que tratam de análise de experimentos (utilizando testes paramétricos, entretanto) em que o volume chega a quase 700 páginas [11]. Tanto a disciplina quanto o livro referido condensam informações sobre técnicas para realizar e tirar conclusões de experimentos como os deste trabalho. Obviamente, o tempo necessário para absorver os conhecimentos necessários para fazer um trabalho minimamente bem fundamentado é, no melhor dos casos, escasso.

Portanto, tentou-se aplicar técnicas básicas de análise de experimentos para avaliar a *performance* dos algoritmos e mesmo assim sem grande critério. Um exemplo é o número de *datasets* necessários para o teste de Friedman: segundo [10] seria necessário testes com pelo menos seis datasets e dez algoritmos de classificação.

6.1.2 Classificadores

Pelo fato da base de conhecimentos sobre estatística dos autores deste trabalho ser insuficiente, procurou-se focar na remediação desta deficiência em detrimento das demais atividades do trabalho.

Sendo assim, com o tempo despendido não foi possível entender com profundidade cada um dos classificadores utilizados para este trabalho. São 15 classificadores em que apenas uma análise superficial foi feita de cada um. Desta forma, o primeiro objetivo deste trabalho ficou comprometido.

Referências

[1] Abalone Data Set http://mlr.cs.umass.edu/ml/datasets/Abalone

[2] Banknote Authentication http://mlr.cs.umass.edu/ml/datasets/banknote+authentication

[3] Car Evaluation http://mlr.cs.umass.edu/ml/datasets/Car+Evaluation

[4] DEMSAR,JANEZ.Statistical Comparisons of Classifiers over Multiple Data Sets.Journal of Machine Learning Research.Ljubljana.Eslovênia.Págs 1-30.2006

[5] GARCIA,Salvador.MOLINA,Daniel.LOZANO,Manuel.HERRERA,Franciso.A study on the use of non-parametric for analyzing the evolutionary algorithm's behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization.Springer.2008.

[6] Numpy.org http://www.numpy.org

[7] Portal Action http://www.portalaction.com.br

[8] SALZBERG,Steven L.On Comparing Classifiers: Pitfalls To Avoid and a Recommended Approach.Data Mining and Knowledge Discovery.Kluwer Academic Publishres,Boston,USA.Págs.317-328.1997.

[9] Scikit Learnig scikit-learn.org

[10] Scipy.org. Statistical Functions http://docs.scipy.org/doc/scipy-0.14.0/reference/stats.html

[11] OEHLERT, Gary W. A First Course in Design and Analysis of Experiments http://users.stat.umn.edu/~gary/book/fcdae.pdf

[12] UCI Machine Learning Repository http://mlr.cs.umass.edu/ml/index.html