Projeto de AAED 2018

Rodney Rick

UNIFESP-SJC, ICT
rodneyrick@gmail.com

Abstract

Poker atualmente é um dos jogos mais envolventes e com crescente número de novos jogadores a cada dia. Um cenário ideal para exploração de técnicas estatísticas e reconhecimento de padrões. Neste trabalho está descrito como explorar e melhorar este conjunto de dados a fim de predizer qual o é a mão do jogador.

1 Introdução

Para o trabalho da Disciplina de Análise de Algoritmos de Estrutura de Dados de 2018, é apresentado neste artigo maneira de tratar dados brutos e explorá-los a fim de enriquecê-los e obter melhores resultados para uma classificação mais assertiva.

Esta análise é composta pela abordagem de:

- Interpretação dos dados;
- Técnicas de Aprendizado de Máquina para classificação;
- Uso de bibliotecas *Open-Source* para desenvolvimento;
- E métricas para mensurar resultados.

Dado o conjunto de dados apresentados, este trabalho visa a trabalhar com técnicas da área de Aprendizado Supervisionado, onde é conhecido o identificador (*label*) de cada um dos registros da base de treinamento, e o qual visa o reconhecimento dos novos dados com suas predições para cada um dos identificadores conhecidos.

2 O conjunto de dados

2.1 Descrição de Dados

Da área de Jogos de Azar e um dos conjuntos de dados (*dataset*) mais populares, desde 2007, para trabalhar com classificação de dados, o *dataset* conhecido como *Poker Hand*[1] [8].

[1] *Poker Hand Data Set* - `https://archive.ics.uci.edu/ml/datasets/Poker+Hand`

Sobre um conjunto de dados de 1.025.010 registros (ou mão de poker), são descritos de forma categórica (ordinais) e inteiros. Cada dupla de coluna do *dataset* é representada por S_x e C_x, respectivamente, Tipo da Carta (ou *Naipe*) e seu Valor Numérico, onde x descreve qual a carta entre $[1-5]$. Conforme exemplo abaixo:

S1 "Tipo da Carta 1"
Ordinal (1-4) representando os Naipes (♡♢♣♠);

C1 "Rank da Carta 1"
Numérico (1-13) representando: As, 2, 3, ... , Rainha, Rei.

Como se trata de um *dataset* para classificação existe uma coluna de classificação de cada registro, e é representada como Ordinal entre o intervalo $[0-9]$, conforme abaixo.

0. : Nada na mão; não é uma mão de poker reconhecida;
1. : Um par; um par de fileiras iguais dentro de cinco cartas;
2. : Dois pares; dois pares de fileiras iguais dentro de cinco cartas;
3. : Três de um tipo; três fileiras iguais dentro de cinco cartas;
4. : Em linha reta; cinco cartas, sequencialmente classificadas sem lacunas;
5. : *Flush*; cinco cartas com o mesmo naipe;
6. : Casa cheia; par + classificação diferente três de um tipo;
7. : Quatro de um tipo; quatro fileiras iguais dentro de cinco cartas;
8. : *Straight flush*; *straight* + *flush*;
9. : *Royal flush*; Ás, Rei, Rainha, Valete, Dez + *flush*

2.2 Estatísticas da Distribuição das cartas

A distribuição das mãos de poker apresenta alta desigualdade entre as classes $[0-9]$. Na tabela 2 é possível visu-

alizar a distribuição das classes disponíveis nos *datasets* de treino e teste.

Tabela 1: Porcentagem de Distribuição das Classes nos *Datasets*

Poker Hand	Classe	Treino	Teste
Nothing in hand	0	49.9	50.12
One pair	1	42.3	42.24
Two pairs	2	4.8	4.76
Three of a kind	3	2.0	2.11
Straight	4	0.3	0.38
Flush	5	0.2	0.19
Full house	6	0.1	0.14
Four of a kind	7	0.0	0.02
Straight flush	8	0.0	0.00
Royal flush	9	0.0	0.00

3 Técnicas de *Machine Learning*

Técnicas de *Machine Learning* visam aprender como atuar e interpretar os dados a ponto de identificar qual a possível classe/*label* que um registro representa. Nesta seção são demonstrados algumas técnicas utilizadas para este trabalho.

3.1 *Support Vector Machines* - SVM

Abrevida como SVM, conhecida como *Support Vector Machines* e desenvolvida por [3], esta técnica visa trabalhar com dados complexos. Para este trabalho está sendo explorado para um caso mais simples, o qual não apresenta uma complexidade de dados alta - grande dimensionalidade de atributos - e sim grande diferença de quantidade de dados presente em cada um das classes.

SVM é composta por três mais comuns funcionalidades - *Kernel Functions* - para a identificação de registros. Cada *Kernel Function* apresenta suas particularidades na classificação dos registros. Ainda, são agrupados em duas frentes.

1. **Linear**
2. **Non-Linear**
   (a) **Polinomial de Potência p**
   (b) **Gaussiana**

Antes de começar a explicação das funcionalidades, deve-se esclarecer um ponto importante de atuação com SVM. Uma abordagem importante em como irá ser feita a atuação d desenvolvimento da codificação está no formato da análise da mesma. Existem duas formas de separação dos dados, *One-Vs-One* (OVO) e *One-Vs-All* (OVA). Essa abordagem está bem esclarecida pelos autores em [14], com algumas melhorias em suas devidas abordagens e ilustrada pela figura 1[2].

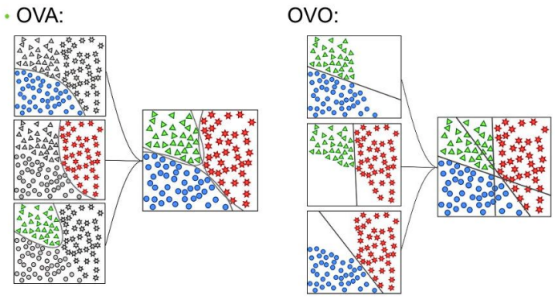

Figura 1: OVO vs OVA

Em OVO, consiste na atuação de análise dos classes dos registros de forma a considerar cada uma das classes de forma independentes. No caso de OVA, considera-se a análise de forma que uma classe será separada e classificada de forma binária contra todas as outras restantes, como se existisse duas classes no momento de execução.

Esclarecido a diferença entre OVO e OVA, para este trabalho utilizou-se a abordagem de SVM com *Kernel* Linear, tem-se que classificar os registros de forma binária no espaço dos atributos. Ou seja, através da análise de cada um dos atributos, aplica-se a verificar da melhor separação dos dados a cada novo atributo analisado. No entanto, qual seria a linear entre as classes dos registros (figura 2).

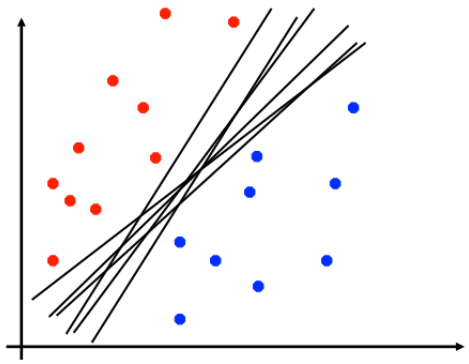

Figura 2: Possibilidades de separação linear.

Para atuação deste projeto, será utilizado somente a abordagem de SVM Linear. A qual aplica a otimização da separação entre as classes ao utilizar os registros que se encontram nas margens entre as classes. Tais registros são designados como *Support Vectors* para o cálculo da distância de cada registro das classes. A equação 1 demonstra como é feito o

[2]Figura retirada do site `http://slideplayer.com/slide/7872916`

cálculo. E a figura 3, representa a análise entre dois atributos[3].

$$r = \frac{w^T x_i + b}{\| w \|} \quad (1)$$

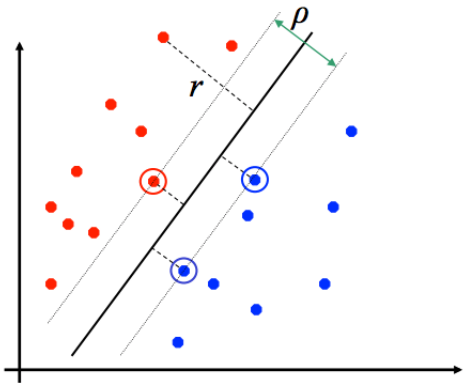

Figura 3: Cálculo da equação de distância

A complexidade da análise para SVM Linear é representada por $O(max(n,d)min(n,d)^2)$, onde n representa o número de registros e d o número de atributos analisados. Os autores [1] representam os custos da complexidade de codificação.

Existe a possibilidades de diminuir o valor d, utilizando outras técnicas como Regularização dos dados ou *Principal component analysis* (PCA) - esta trabalha na transformação dos atributos para a dimensão ortogonal, criado por [6].

3.2 *Random Forest* - RF

Outro classificador da área clássica de *Machine Learning* é a *Random Forest*, a qual apresenta um conjunto de *Decision Trees* (DT). Técnica esta desenvolvida em 1995 pelo pesquisador [7], que inicialmente utilizou para a análise de *Data Mining* com diversas *Decision Trees*.

Antes de entender o RF, é importante descrever a complexidade e forma de atuação de uma DT.Uma *Decision Tree* é bem próxima sua atuação com uma árvore binária, com a diferença da função que realiza quebra da árvore para o próximo nível. Sua complexidade é da ordem de $O(v * n * log(n))$, onde v e n, respectivamente, representam o número de atributos (ou variáveis) e números de registros.

Como *Random Forest* é uma floresta de *Decision Trees*, a complexidade é representa pela combinação de:

- $nTree$ - número de DTs;
- v - número de atributos;
- n - números de registros

Sendo sua complexidade representada como $O(nTree * v * n * log(n))$.

3.3 *Neural Networks* - NN

Quando se estuda qualquer técnica em *Machine Learning* é possível chegar em um consenso sobre qual a ordem de complexidade de cada uma. No entanto, quando se trata de *Neural Networks*, a pergunta mais adequada a se fazer seria: "Qual o tempo de treinamento?".

Como realizar a medida de complexidade depende muito de qual a arquitetura está sendo utilizada, qual a função de custo, função de ativação, se será desconsiderado um certa porcentagem de neurônios em cada camada interna, se existe *back-propagation*. Este universo de diferentes arquiteturas está demonstrada de maneira simples no site [15].

Os autores do artigo [2] descrevem da complexidade para algumas arquiteturas, mas esclarecem da dependência e aderência de cada problema abordado, em outras palavras, existe a necessidade de customizações em cada utilização.

É possível considerar a seguinte anotação básica para uma medida de referência, que seria: $O(n^2 * H)$, onde n representa o número de neurônios e H a quantidade de camadas de neurônios utilizadas para o desenvolvimento. Esta seria uma arquitetura padrão.

Neste trabalho, utilizou-se uma abordagem utilizando duas bibliotecas:

Keras [4]
Biblioteca desenvolvida para utilização sob a linguagem de programação Python e muito difundida na área de *Machine Learning*;

TensorFlow [11]
Biblioteca desenvolvida pela Google, a qual traz como foco construções desde simples *Neural Networks*, até redes mais complexas conhecidas como *Deep Learning*, como descreve o artigo [9].

3.4 *Cross-validation*

A técnica de *Cross-Validation*, ou Validação Cruzada, visa trabalhar o *dataset* de treinamento de forma a dividir em k blocos proporcionais e utilizar parte dos blocos no treinamento do modelo preditivo e parte para validação do mesmo. Essa abordagem visa um melhor aproveitamento dos blocos para o momento de treinamento, os quais podem apresentar

[3]Figura retirada da apresentação - `http://www.ic.unicamp.br/~rocha/teaching/2018s1/mo444/classes/mo444-class-materials-08.pdf`

maiores características aderentes a cada uma das classes do *dataset*.

Existem algumas modalidades de *Cross-Validation* apresentadas na biblioteca *Scikit-Learn* [12], entre elas:

KFold
Essa abordagem descreve a quebra do *dataset* em k blocos de tamanhos iguais, e selecionados aleatoriamente para uso de treinamento e validação;

StratifiedKFold
Para esta validação, e sendo uma variação do *KFold*, trabalha com a divisão de blocos com o seguinte critério: manter a proporcionalidade de dados entre as classes, assim como existe no *dataset* original;

ShuffleSplit
Nesta separação de dados, em blocos de tamanhos diferentes, não garante no momento de treinamento ou validação as classes presentes em suas devidas proporções originais, o que pode causar um desbalanceamento no treinamento e não aprender a interpretar uma devida classe, normalmente com quando apresentam poucos registros;

StratifiedShuffleSplit
Nesta validação é uma mescla de *StratifiedKFold* e *ShuffleSplit*, a qual visa retorna de forma aleatória os blocos, no entanto mantendo a porporção de dados para cada uma das classes. Neste caso, foi utilizada para este trabalho.

3.5 Parâmetros das técnicas

Uma área ainda muito explorada representa na escolha dos parâmetros para cada técnica. A demonstração desta importância de abordagem está descrita pelo artigo [10], o qual os autores esclarecem possíveis técnicas de como explorar cada universo de possíveis combinações entre os parâmetros. Na figura 4[4] é possível verificar como é feita a exploração.

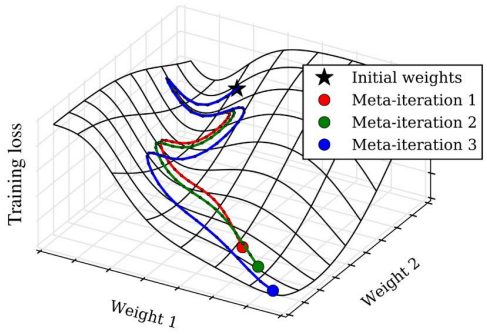

Figura 4: Otimização da combinação de parâmetros

[4]Figura retirada do artigo [10].

A literatura cita duas técnicas mais utilizadas, logo após de *Gradient Descent*[16]. *Grid Search* que considera dado um conjunto de valores para cada um dos parâmetros existentes nas técnicas de *Machine Learning*. E *Randomized Search Cross Validation* que considera dentro conjunto de valores, possíveis combinações através da escolha dos valores em um determinado espaço dos valores. Ambas as técnicas estão disponíveis na biblioteca *Scikit-Learn*.

Para esta exploração utilizou-se a abordagem da opção *Randomized Search Cross Validation*.

4 Resultados

Nesta seção estão apresentados os resultados, entretanto, todo resultado necessita ser comparado com alguma métrica.

As métricas utilizadas para este artigo foram:

Acurácia
Esta métrica verifica o resultado total do modelo preditivo e as classes a qual cada registro realmente pertence. No entanto, está aderente na quantidade total de registros. Um conjunto de dados desbalanceado não necessária é uma boa métrica.

Precision
Em português, Precisão, mais também conhecido como *Positive predictive value* (PPV), essa medida verifica todos os registros preditos como Positivos (TP) dividido por ele mesmo mais o valores de Falso Positivos (FP). A equação que representa esta métrica é: $PPV = \frac{TP}{TP+FP}$. Valores mais próximos a 100% são melhores resultados do modelo preditivo.

Recall
Também conhecido como *Sensitivity* ou *True positive Rate* (TPR), essa medida verifica todos os registros preditos como Positivos (TP) dividido por ele mesmo mais o valores de Falso Negativos (FN). A equação que representa esta métrica é: $TPR = \frac{TP}{TP+FN}$. Assim como *Precision*, valores mais próximos a 100% apresentam melhores do modelo preditivo.

ROC Curve
Abreviada da nomenclatura *Receiver Operating Characteristic* (ROC), realiza a métrica de forma binária do desempenho do modelo preditivo para cada classe do *dataset*. O foco está em considerar em cada um dos eixos a seguinte abordagem: **(a)** Positivos Verdadeiros Preditos dos total de Positivos Totais Reais e **(b)** Positivos Falsos Preditos dos total de Negativos Totais Reais. Para o melhor aderência do modelo preditivo, este valor deve estar próximo ao topo do gráfico. Para mais detalhes desta medida, segue artigo [13].

Esclarecidos algumas métricas de uso e funcionamento, nas figuras 5, 6 e 7 deste capítulo estão apresentados os resultados das técnicas enunciadas.

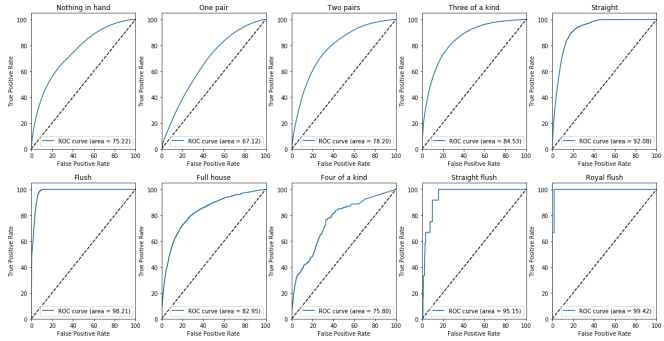

Figura 5: *ROC Curve* para *Random Forest*.

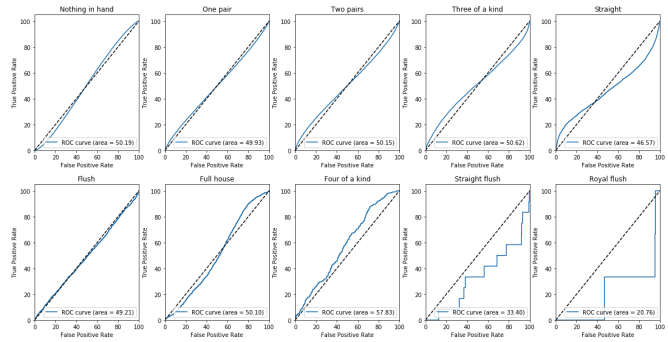

Figura 6: *ROC Curve* para *SVM*.

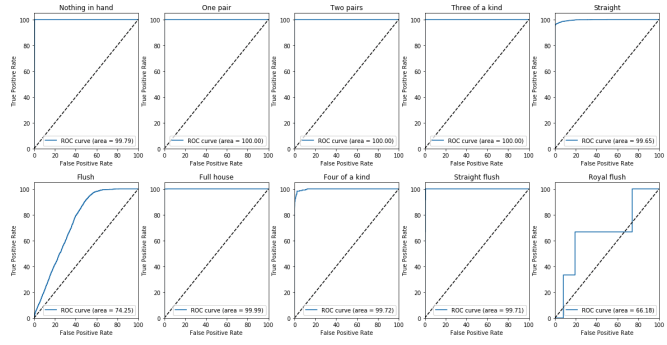

Figura 7: *ROC Curve* para *Neural Networks*.

4.1 Tabela comparativa

Na tabela 2, segue o resultados das métricas globais das técnicas utilizadas e dos tempos de processamento.

5 Conclusão

Cada técnica e seus resultados apresentados existem suas particularidades. Em *datasets* onde o problema apresenta diversidade dos registros classificados, muitos *outliers*[5] e classificação binária, SVM normalmente apresenta melhores resultados, pelo fato de conseguir maximizar as margens de separação entre as classes.

Tabela 2: Médias métricas de execuções.

Abordagem	Tempo	Acurácia	Precisão	Recall
SVM	14.4min	49.9%	43%	37.2%
Random Forest	28.6s	59.8%	58%	60%
Neural Network	6.47min	99.7%	99.8%	99.7%

Para problemas multi-classes, como este do *Poker Hand*, a *Random Forest*, com um tempo de execução surpreendente com treinamento e validações cruzadas, apresentou mais aderência na identificação de cada um dos registros, porém apenas 10% acima considerando a medida de Acurácia da SVM.

Atualmente os estudos de muitos pesquisadores estão voltados para *Neural Networks*, mais especificamente *Deep Learning*, devidos a algumas desvantagens que as técnicas em *Classic Machine Learning* apresentam, como por exemplo, crescimento dos *datasets* tanto em quantidade de registros quanto em tamanho de dimensionalidade. Porém, neste caso, deve-se ter alguns cuidados, por exemplo, um treinamento com uma quantidade de épocas muito alto do modelo, por tornar facilmente um modelo com *Overfitting*, como explica este autor [5].

Apêndice

Este projeto, artigo e codificação, dos resultados apresentados estão disponíveis no repositório do Github atráves do link[6].

Referências

[1] A. Abdiansah e R. Wardoyo. "Time Complexity Analysis of Support Vector Machines (SVM) in LibSVM". Em: *International Journal of Computer Applications* 128.3 (out. de 2015), pp. 28–34. DOI: 10.5120/ijca2015906480.

[2] M. Bianchini e F. Scarselli. "On the Complexity of Neural Network Classifiers: A Comparison Between Shallow and Deep Architectures". Em: *IEEE Transactions on Neural Networks and Learning Systems* 25.8 (ago. de 2014), pp. 1553–1565. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2013.2293637.

[5] Registros com dados de entradas que distorcem e enganam o processo de treinamento dos algoritmos de aprendizado de máquina, resultando em tempos de treinamento mais longos, modelos menos precisos e, em última análise, resultados mais pobres. Em outras palavras, valores discrepantes.

[6] https://github.com/rodneyrick/unifesp_aaed_2018_01/tree/master/projeto

[3] Bernhard E. Boser, Isabelle M. Guyon e Vladimir N. Vapnik. “A Training Algorithm for Optimal Margin Classifiers”. Em: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT ’92. Pittsburgh, Pennsylvania, USA: ACM, 1992, pp. 144–152. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130401. URL: http://doi.acm.org/10.1145/130385.130401.

[4] François Chollet et al. *Keras*. https://keras.io. 2015.

[5] Pedro Domingos. “A Few Useful Things to Know About Machine Learning”. Em: *Commun. ACM* 55.10 (out. de 2012), pp. 78–87. ISSN: 0001-0782. DOI: 10.1145/2347736.2347755. URL: http://doi.acm.org/10.1145/2347736.2347755.

[6] Karl Pearson F.R.S. “LIII. On lines and planes of closest fit to systems of points in space”. Em: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720.

[7] Tin Kam Ho. “Random decision forests”. Em: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. Ago. de 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994.

[8] S. Jabin. “Poker hand classification”. Em: *2016 International Conference on Computing, Communication and Automation (ICCCA)*. Abr. de 2016, pp. 269–273. DOI: 10.1109/CCAA.2016.7813761.

[9] Hugo Larochelle et al. “An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation”. Em: Corvallis, OR, 2007, pp. 473–480. DOI: http://doi.acm.org/10.1145/1273496.1273556. URL: http://oregonstate.edu/conferences/icml2007/paperlist.html.

[10] Dougal Maclaurin, David Duvenaud e Ryan Adams. “Gradient-based hyperparameter optimization through reversible learning”. Em: *International Conference on Machine Learning*. 2015, pp. 2113–2122.

[11] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[12] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. Em: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[13] David Powers. “Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness Correlation”. Em: 2 (jan. de 2008).

[14] A. Rocha e S. K. Goldenstein. “Multiclass From Binary: Expanding One-Versus-All, One-Versus-One and ECOC-Based Approaches”. Em: *IEEE Transactions on Neural Networks and Learning Systems* 25.2 (fev. de 2014), pp. 289–302. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2013.2274735.

[15] Fjodor Van Veen. *The Neural Network Zoo*. 2016. URL: http://www.asimovinstitute.org/neural-network-zoo/ (acedido em 14/09/2016).

[16] Tong Zhang. “Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms”. Em: *Proceedings of the Twenty-first International Conference on Machine Learning*. ICML ’04. Banff, Alberta, Canada: ACM, 2004, pp. 116–. ISBN: 1-58113-838-5. DOI: 10.1145/1015330.1015332. URL: http://doi.acm.org/10.1145/1015330.1015332.