Análise comparativa de algoritmos de classificação para predição de jogadas no poker

Daniel Dias Barbosa[1], **Flávio Vezono Filho**[1]

[1]Faculdade de Computação – Universidade Federal de Uberlândia (UFU)
Uberlândia – MG – Brazil

daniel.dias@ufu.br, flavio.filho@ufu.br

Abstract. *The work consists of a comparative analysis of different classification methods used to predict the next move of a poker player. In the end, they were compared using the parameters of accuracy, training time, and prediction time. The evaluated methods are: k-nearest neighbor, support vector classifier, Decision Tree, Multilayer Perceptrons, and Naive Bayes.*

Resumo. *O trabalho consiste em uma análise comparativa de diferentes métodos de classificação usados para predizer a próxima jogada de um jogador de poker. Ao final foram comparados utilizando os parâmetros de acurácia, tempo de treino e tempo de predição. Os métodos avaliados são: k nearest neighbor, support vector classifier, Árvore de Decisão, Perceptrons Multinível e Naive Bayes.*

1. Introdução

O objetivo desse trabalho é verificar se é possível predizer a próxima jogada de um jogador de poker considerando o estado atual de sua mão e do jogo. Para o trabalho tomou-se como base a modalidade de Poker Texas Hold'em, que é uma das variantes mais populares do poker e é jogado com um baralho padrão de 52 cartas. Segundo [888Poker 2023], o objetivo do jogo é formar a melhor mão de cinco cartas possível, utilizando suas duas cartas pessoais e as cinco cartas comunitárias compartilhadas na mesa. O jogo consiste em quatro rodadas de apostas, que são:

1. Pré-flop: Depois que as blinds são colocadas, cada jogador recebe duas cartas pessoais. O jogador à esquerda do big blind começa a agir e tem três opções: desistir (fold), pagar a aposta do big blind (call) ou aumentar a aposta (raise).
2. Flop: Após a primeira rodada de apostas, o dealer revela as três primeiras cartas comunitárias na mesa. Essas cartas são chamadas de "flop". Os jogadores podem usar essas cartas em combinação com suas cartas pessoais para formar uma mão de poker. Começando pelo jogador à esquerda do dealer, uma nova rodada de apostas começa.
3. Turn: Depois da segunda rodada de apostas, o dealer revela uma quarta carta comunitária, conhecida como "turn". Os jogadores têm a opção de apostar, verificar (passar a vez) ou desistir. A rodada de apostas começa com o primeiro jogador à esquerda do dealer.
4. River: Após a terceira rodada de apostas, o dealer revela a quinta e última carta comunitária, chamada de "river". Os jogadores têm uma última oportunidade de apostar. A rodada de apostas começa novamente com o primeiro jogador à esquerda do dealer.

Se mais de um jogador permanecer após a rodada final de apostas, ocorre o showdown. Os jogadores revelam suas cartas pessoais e a melhor mão de poker possível é formada, combinando suas duas cartas pessoais com as cinco cartas comunitárias na mesa. O jogador com a melhor mão de poker vence o pote. Em caso de empate, o pote é dividido igualmente entre os jogadores empatados.

2. Coleta dos dados e pré-processamento

Os dados usados no trabalho foram gentilmente cedidos pelo professor Murilo e estão disponíveis no artigo [Carneiro and de Lisboa 2018]. Os dados foram extraídos de partidas reais de poker e possuem informações sobre qual era o estado da partida e qual foi a ação tomada pelo jogador. As informações foram todas normalizadas utilizando-se do algoritmo min-max, no qual o menor valor é 0 e o maior é 1. Os dados foram disponibilizados conforme formatação apresentada na Figura 1.

```
1.  Title: ZP-River data set
        Updated July 11 2018 by M. G. Carneiro

2.  Feature Information:
            NAME                VALUES                                          NORMALIZATION
        2.1 POSITION            {SB, BB, UTG, MP, CO, BU}                       min-max
        2.2 EHS                 REAL                                            min-max
        2.3 TOTAL_POT           REAL                                            min-max
        2.4 POT_ODDS            REAL                                            min-max
        2.5 BOARD_SUIT          {Rainbow, TwoSuited, Monotone}                  min-max
        2.6 BOARD_CARDS         {NoPared, Pared, Triplet}                       min-max
        2.7 BOARD_CONNECT       {Connect, SemiConnect, Disconnect}              min-max
        2.8 PREV_ROUND_ACTION   {Check, Call, Bet, Raise}                       min-max
        2.9 PREVIOUS_ACTION     {NoAction, Check, Bet, BetAndCall,              min-max
                                BetAndRaise}
        2.10BET_VILLAIN         REAL                                            min-max
        2.11AGG                 {IPvsAgg, HeroAgg, OOPvsAgg}                    min-max
        2.12IP_VS               {0, 1, 2, 3, 4, 5}                              min-max
        2.13OOP_VS              {0, 1, 2, 3, 4, 5}                              min-max
        2.14ACTION_HERO         {Fold, Check, Call, Bet, Raise}

3.  Metadata:
        #Objects                #Attributes                 #Classes [Class Distribution]
        10342                   14                          5 [11.7%, 52.9%, 8.9%, 24.8%, 1.7%]

4.  Target Class: ACTION_HERO

5.  For a complete description about this data set, please read the following paper:
        Murillo G. Carneiro and Gabriel A. Lisboa, What's the next move? Learning Player Strategies in Zoom Poker Games.
        In IEEE Congress on Evolutionary Computation, 2018, pp. 1951-1958.

6.  Contact information:
        mgcarneiro@ufu.br
        gabriel_alves@ufu.br

7.  Please cite the following paper when using this data set:
        @inproceedings{carneiro_alves2018,
        title={What's the next move? Learning Player Strategies in Zoom Poker Games},
        author={Murillo G. Carneiro and Gabriel A. Lisboa},
        booktitle={IEEE Congress on Evolutionary Computation},
        pages={1951--1958},
        year={2018}
        }
```

Figure 1. Formatação dos dados

Para computar esses dados, foi utilizado da biblioteca Pandas [pandas 2023] em python, que facilitou muito a manipulação desses dados para a futura análise. Os dados foram importados conforme figura 2.

```python
path = "../data/ZP-River.dat"
quantidade_teste = 10


def load_data(path):
    diretorio = os.path.join(os.path.dirname(__file__), path)

    nomes = [
        "POSITION",
        "EHS",
        "TOTAL_POT",
        "POT_ODDS",
        "BOARD_SUIT",
        "BOARD_CARDS",
        "BOARD_CONNECT",
        "PREV_ROUND_ACTION",
        "PREVIOUS_ACTION",
        "BET_VILLAIN",
        "AGG",
        "IP_VS",
        "OOP_VS",
        "ACTION_HERO",
    ]

    return pd.read_table(
        filepath_or_buffer=diretorio, header=None, names=nomes, sep=" "
    )
```

	POSITION	EHS	TOTAL_POT	POT_ODDS	BOARD_SUIT	BOARD_CARDS	BOARD_CONNECT	PREV_ROUND_ACTION	PREVIOUS_ACTION	BET_VILLAIN	AGG	IP_VS	OOP_VS	ACTION_HERO
0	0.6	0.513636	0.079095	0.00	0.000000	0.0	0.5	0.0	0.000000	0.00	1.0	0.00	0.25	2.0
1	0.2	0.787879	0.000533	0.00	0.000000	1.0	0.0	0.5	0.000000	0.00	0.5	0.00	0.25	2.0
2	0.8	0.912626	0.085486	0.00	0.000000	0.0	1.0	0.0	0.333333	0.25	0.5	0.25	0.00	4.0
3	1.0	0.263822	0.005593	0.00	0.000000	0.0	0.5	0.0	0.000000	0.25	0.5	0.25	0.25	2.0
4	0.0	0.376263	0.019973	0.00	0.000000	0.0	0.0	0.0	0.000000	0.00	1.0	0.00	0.25	2.0
5	0.6	0.902020	0.125699	0.00	0.000000	0.0	0.0	0.0	1.000000	0.25	0.0	0.25	0.00	2.0
6	0.2	0.813131	0.016511	0.00	0.000000	1.0	0.0	0.5	0.000000	0.00	1.0	0.00	0.25	4.0
7	0.6	0.501515	0.045273	0.36	0.011923	0.0	0.5	0.0	0.333333	0.50	0.0	0.25	0.00	1.0
8	0.6	0.730303	0.028495	0.56	0.013413	0.0	0.5	0.0	0.000000	0.50	0.5	0.25	0.00	1.0
9	0.0	0.600505	0.032490	0.00	0.000000	1.0	0.0	1.0	0.333333	0.00	0.5	0.00	0.25	2.0

Figure 2. Código utilizado para ler os dados e cabeçalho dos dados lidos

3. Metodos de classificação

Todos os métodos descritos abaixo foram implementados através da biblioteca do [scikit learn 2023] em que seu site sugere múltiplos algoritmos de classificação que podem ser utilizados. Escolhemos, dentre esses, os que foram abordados em aula e que tinham mais relevância em fóruns.

3.1. K Nearest Neighbor (KNN)

O primeiro método de classificação escolhido foi o KNN por ser o de mais simples implementação e ser praticamente sempre o primeiro a ser estudado em tópicos de machine learning. O princípio utilizado para classificar objetos é com base na similaridade com seus vizinhos mais próximos. Ele é considerado um algoritmo de aprendizado preguiçoso (lazy learning), pois não requer um estágio de treinamento explícito. Em vez disso, ele armazena o conjunto de dados de treinamento e, quando um novo objeto precisa ser classificado, ele busca pelos k vizinhos mais próximos desse objeto no espaço de características.

Para classificar o novo objeto, o algoritmo KNN utiliza a maioria das classes dos vizinhos encontrados. Isso significa que, se a maioria dos vizinhos pertence a uma determinada classe, o novo objeto será classificado nessa classe. A escolha do valor de k é um parâmetro importante no algoritmo KNN, pois determina a influência dos vizinhos

no processo de classificação. Valores maiores de k consideram um número maior de vizinhos, levando a uma decisão mais robusta, mas também potencialmente mais sujeita a ruído ou padrões locais.

O algoritmo KNN é relativamente simples de entender e implementar, e geralmente é eficaz em conjuntos de dados menores ou de baixa dimensionalidade. No entanto, seu desempenho pode ser afetado por conjuntos de dados grandes e de alta dimensionalidade, pois a busca pelos vizinhos mais próximos pode se tornar computacionalmente custosa. Além disso, o KNN não captura informações sobre a estrutura dos dados ou a importância relativa dos atributos, o que pode afetar sua precisão em certos cenários. No entanto, com os ajustes adequados de parâmetros e o pré-processamento adequado dos dados, o algoritmo KNN pode ser uma ferramenta útil em problemas de classificação.

3.2. Support Vector Classifier (SVC)

O algoritmo Support Vector Classifier (SVC) é um método de classificação baseado em máquinas de vetores de suporte. É um algoritmo de aprendizado supervisionado que busca encontrar um hiperplano ótimo para separar as classes no espaço de características.

O SVC busca encontrar o hiperplano que maximiza a margem entre as classes, ou seja, a distância entre o hiperplano e os pontos de dados mais próximos de cada classe. Esses pontos de dados são chamados de vetores de suporte, pois são fundamentais para definir o hiperplano de decisão. Ao maximizar a margem, o SVC busca uma separação mais robusta e com menor probabilidade de erro de classificação.

Em casos onde as classes não são linearmente separáveis, o SVC pode utilizar truques matemáticos chamados de ”kernel tricks”. Esses truques mapeiam os dados para um espaço de alta dimensionalidade onde as classes possam se tornar linearmente separáveis. Os kernels mais comumente usados são o kernel linear, o kernel polinomial e o kernel gaussiano (RBF).

Uma das vantagens do SVC é sua capacidade de lidar com conjuntos de dados de alta dimensionalidade. Além disso, o SVC é resistente a outliers, pois os vetores de suporte são selecionados para estar mais próximos dos pontos de dados de cada classe. No entanto, o SVC pode ser computacionalmente exigente, especialmente quando o conjunto de dados é muito grande. Além disso, o ajuste de parâmetros, como o parâmetro de regularização C e a escolha do kernel, é importante para obter um bom desempenho do SVC.

3.3. Árvore de decisão

A árvore de decisão é um algoritmo de aprendizado de máquina amplamente utilizado para tarefas de classificação e regressão. Ela é construída a partir de um conjunto de regras de decisão hierárquicas, representadas por uma estrutura de árvore. Cada nó interno na árvore representa uma decisão baseada em um atributo específico, e cada ramo representa uma possível saída ou consequência dessa decisão. As folhas da árvore são os nós terminais que representam as classes ou os valores de regressão finais.

A construção da árvore de decisão é baseada em algoritmos que procuram encontrar o melhor atributo para dividir os dados em cada nó interno. A medida de impureza, como a entropia ou o índice de Gini, é usada para avaliar a qualidade da divisão em termos de pureza das classes resultantes. O algoritmo continua recursivamente dividindo

os dados com base nos atributos até que uma condição de parada seja alcançada, como atingir uma profundidade máxima ou quando todos os exemplos em um nó pertencem à mesma classe.

Uma das vantagens das árvores de decisão é a sua interpretabilidade, pois as regras de decisão podem ser facilmente visualizadas e compreendidas. Além disso, elas são capazes de lidar com conjuntos de dados grandes e de alta dimensionalidade, e também podem lidar com atributos categóricos e numéricos. As árvores de decisão também são menos sensíveis a outliers em comparação com alguns outros algoritmos.

No entanto, as árvores de decisão podem ser propensas ao overfitting, especialmente quando a árvore se torna muito profunda ou quando o conjunto de dados é ruidoso. Para mitigar esse problema, estratégias como a poda da árvore, o ajuste de parâmetros e o uso de algoritmos ensemble, como o Random Forest, podem ser empregados.

3.4. Multilayer Perceptrons

O MLP (Multilayer Perceptron) é um algoritmo de aprendizado de máquina amplamente utilizado para tarefas de classificação. É um tipo de rede neural artificial que consiste em várias camadas de neurônios interconectados. Cada neurônio em uma camada está conectado a todos os neurônios da camada seguinte, formando uma arquitetura em forma de camadas.

No MLP, a camada de entrada recebe os atributos do conjunto de dados, que são propagados para a camada oculta através de pesos atribuídos às conexões entre os neurônios. Cada neurônio na camada oculta recebe as entradas ponderadas e aplica uma função de ativação não-linear para produzir uma saída. Essa saída é então propagada para a próxima camada e o processo se repete até chegar à camada de saída, onde a classificação final é feita.

Durante o treinamento do MLP, os pesos das conexões são ajustados iterativamente para minimizar uma função de perda, como o erro quadrático médio ou a entropia cruzada, utilizando técnicas de otimização, como o gradiente descendente. O objetivo é encontrar os pesos que permitem que o MLP faça previsões precisas em um conjunto de dados de treinamento.

Uma das vantagens do MLP é sua capacidade de aprender representações complexas e não lineares dos dados. Ele pode lidar com conjuntos de dados de alta dimensionalidade e é robusto em relação a ruídos. Além disso, o MLP é flexível e pode ser adaptado para tarefas de classificação com várias classes.

No entanto, o treinamento do MLP pode ser computacionalmente intensivo, especialmente para conjuntos de dados grandes e complexos. Além disso, o MLP é sensível à inicialização dos pesos e ao ajuste de hiperparâmetros, como a taxa de aprendizado e o número de neurônios nas camadas ocultas. O overfitting também pode ser um desafio, e técnicas como regularização, dropout e validação cruzada podem ser aplicadas para mitigar esse problema.

3.5. Naive-Bayes

O Naive Bayes Gaussiano é um algoritmo de classificação que se baseia no Teorema de Bayes e assume que as features (atributos) dos dados de entrada seguem uma distribuição

gaussiana (normal). É chamado de ”naive” (ingênuo) porque assume independência condicional entre as features, o que significa que a presença de uma determinada feature não afeta a presença de outras features.

A classificação com o Naive Bayes Gaussiano envolve calcular a probabilidade posterior de cada classe para um determinado conjunto de features e, em seguida, atribuir o objeto à classe com a maior probabilidade posterior. A probabilidade posterior é calculada utilizando a fórmula do Teorema de Bayes, combinando a probabilidade a priori da classe com a verossimilhança das features dadas a classe.

Para aplicar o Naive Bayes Gaussiano, é necessário estimar os parâmetros das distribuições gaussianas para cada classe e cada feature. Esses parâmetros incluem a média e o desvio padrão para cada feature em cada classe. Durante a fase de treinamento, esses parâmetros são estimados a partir dos dados de treinamento. Durante a fase de teste, os parâmetros estimados são utilizados para calcular as probabilidades posteriores e realizar a classificação.

Uma das vantagens do Naive Bayes Gaussiano é sua simplicidade e eficiência computacional. Ele é especialmente adequado para conjuntos de dados grandes e com alta dimensionalidade. Além disso, o Naive Bayes pode lidar com dados categóricos e numéricos e é relativamente robusto em relação a dados faltantes.

No entanto, uma das limitações do Naive Bayes Gaussiano é sua suposição de independência condicional entre as features, o que pode não ser verdadeiro em alguns conjuntos de dados. Isso significa que, se as features forem altamente correlacionadas, o desempenho do Naive Bayes pode ser prejudicado. Além disso, ele pode ter dificuldades em lidar com classes desbalanceadas ou com classes que possuem sobreposição significativa nos dados.

4. Benchmark

Escolhido os algoritmos, foi desenvolvido um código que irá fazer o benchmark das possibilidades. Para validar os testes, primeiramente se decide quantas vezes irá executar a rotina. A cada iteração um novo subconjunto será escolhido para ser o treinamento e outro para validação. A proporção escolhida foi de 2/3 para treino e 1/3 para teste.

O fluxo de execução do código segue a seguinte lógica:

1. Inicializa-se uma lista de tuplas com o nome e um objeto da classe que implementa o classificador
2. O código carrega os dados a partir de um arquivo usando a função load_data(). Os dados sã divididos em features de entrada (input) e a classe alvo (output).
3. O conjunto de dados é dividido em conjuntos de treinamento e teste usando a função train_test_split().
4. O código itera sobre a lista de métodos de classificação definidoss. Para cada método, a função classificar() é chamada. Ela recebe os conjuntos de treinamento e teste, ajusta o modelo aos dados de treinamento e realiza a predição nos dados de teste. A acurácia da predição, o tempo de treinamento e o tempo de predição são registrados. Os resultados de cada método são armazenados em um dicionário chamado resultado.

5. Após iterar por todos os métodos, o dicionário resultado é retornado pela função benchmark(). No trecho final do código, é realizado o benchmark múltiplas vezes, determinado pelo valor da variável quantidade_teste. Os resultados de cada iteração são acumulados e a média é salva no dicionário final.
6. Os resultados acumulados são formatados em uma tabela utilizando a biblioteca tabulate e, em seguida, impressos na saída.

5. Resultados

Após discutir sobre os métodos, as métricas de desempenho escolhidas foram acurácia, tempo de treinamento e tempo de predição. A acurácia é uma medida de precisão do modelo, indicando a proporção de instâncias corretamente classificadas em relação ao total de instâncias. Quanto maior o valor de acurácia, melhor é o desempenho do algoritmo em classificar corretamente os dados.

O tempo de treinamento e o tempo de predição representam o tempo necessário para ajustar o modelo aos dados de treinamento e para realizar a predição nos dados de teste, respectivamente. Essas medidas são importantes para avaliar a eficiência dos algoritmos, especialmente em conjuntos de dados grandes ou em cenários em tempo real, onde a velocidade de resposta é crucial.

Os resultados estão mostrados na figura 3

Nome método	Acurácia	Tempo Treino	Tempo Predição
KNN	0.746462	0.026701	0.197117
SVC	0.795215	0.943046	1.73253
Arvore	0.753857	0.0270036	0.00180075
MLP	0.801247	4.08594	0.00240219
Naive Bayes	0.72036	0.00449929	0.00239954

Figure 3. Tabela de resultados

6. Conclusão

Com base nos resultados obtidos no benchmark de desempenho dos algoritmos de classificação, podemos tirar algumas conclusões relevantes:

Acurácia: Observamos que o algoritmo MLP obteve a maior acurácia, com 80.12%, seguido pelo SVC (Support Vector Classifier) com 79.52%. Ambos os algoritmos apresentaram um desempenho superior em relação aos demais. O KNN (K-Nearest Neighbors) e a Árvore de Decisão também mostraram um desempenho razoável, com acurácias em torno de 74% e 75%, respectivamente. O Naive Bayes teve a menor acurácia, com 72.04%, indicando que talvez não seja tão eficaz para este conjunto de dados específico.

Tempo de Treinamento: O algoritmo MLP foi o mais lento para treinar o modelo, levando em média 4.09 segundos. O SVC também exigiu um tempo considerável, com média de 0.94 segundos. KNN e a Árvore de Decisão foram muito mais rápidos, com tempos de treinamento em torno de 0.03 segundos. O Naive Bayes foi o mais rápido, com um tempo médio de treinamento de apenas 0.0045 segundos.

Tempo de Predição: O SVC foi o algoritmo mais lento para fazer predições nos dados de teste, levando em média 1.73 segundos. O KNN e a Árvore de Decisão tiveram tempos de predição moderados, em torno de 0.20 segundos e 0.002 segundos, respectivamente. O MLP e o Naive Bayes foram os mais rápidos, com tempos médios de predição de 0.0024 segundos.

Com base nessas conclusões, a escolha do algoritmo de classificação ideal dependerá do equilíbrio entre a acurácia desejada, o tempo de treinamento e o tempo de predição. O MLP apresentou a melhor acurácia, mas é mais lento em termos de treinamento. O KNN é rápido para se treinar, mas com uma acurácia um pouco menor. O Naive Bayes e a Árvore de decisão são os algoritmo mais rápidos de se treinar e prever, mas com a menor acurácia. O SVC se destaca pelo seu desempenho de treinamento, mas lentidão ao predizer.

É observável e relevante notar que a base de dados não é homogênea e é tendenciosa a objetos da classe Fold, Check e Bet, mas mesmo assim obteve-se um desempenho considerável com todos os algoritmos, mostrando que é possível sim, com um certo nível de certeza (70% a 80%) predizer a próxima jogada de de um jogador poker dado seu estado atual.

7. References

References

888Poker (2023). 8 passos simples para dominar o texas hold'em. In *https://br.888poker.com/poker-games/texas-holdem/*.

Carneiro, M. G. and de Lisboa, G. A. (2018). What's the next move? learning player strategies in zoom poker games. In *Advances in Computer Science*.

pandas (2023). pandas - python data analysis library. In *https://pandas.pydata.org*.

scikit learn (2023). scikit-learn machine learning in python. In *https://scikit-learn.org/stable/index.html*.