Métodos Computacionais em Física de Altas Energias

Exercício #2

Decision tree – Instruções gerais
Dados: simulação de eventos no contexto do experimento LHCb com uma coluna de rótulos binários (*label*=1 para *sinal*, 0 para *contaminação/ruído*) e múltiplas *features* (*e.g.* cinemática).

Dataset:/shared_dataset/Met_Stat_HEP_AI_School/Kstarmumu_Combined_10K.csv
Imports: *numpy, pandas, scikit-learn, matplotlib*

Exercise 1: Construindo uma Árvore de Decisão de Duas Camadas

Implemente uma árvore de decisão simples de duas camadas com dois cortes sucessivos:

a) Defina inicialmente os limiares (t_x, t_y) como as **medianas** das duas variáveis escolhidas;

b) Aplique as regras:

- se $x < t_x$: classifique como sinal se $y < t_y$, caso contrário background;
- se $x \geq t_x$: classifique como sinal se $y \geq t_y$, caso contrário background.

c) Plote as regiões de decisão e compare com os rótulos verdadeiros;

d) Em seguida, ajuste manualmente os valores de corte e compare o desempenho com o caso das medianas.

e) Calcule a eficiência de sinal (*true positive rate*); e a rejeição de fundo (1− *false positive rate*).

f) (Opcional) Use o algoritmo `DecisionTreeClassifier` do `scikit-learn` com apenas duas variáveis. Treine o modelo para encontrar automaticamente os cortes que melhor separam sinal e fundo. Em seguida, calcule novamente a eficiência e a rejeição de fundo obtidas.

Exercise 2: Modelos de Classificação Supervisionada com `scikit-learn`

Neste exercício, você irá treinar e comparar diferentes modelos de Machine learning para separar sinal e background (contaminação) em um conjunto de dados de física de partículas.

a) **Decision tree:** Treine um modelo `DecisionTreeClassifier` do scikit-learn com todas as variáveis disponíveis no dataset. Separe os dados em treino e teste, e avalie a performance inicial do modelo.

b) **Análise de Performance:** Crie gráficos para explorar e avaliar o modelo:

- Matriz de correlação entre as variáveis, por exemplo usando `np.corrcoef(df_num.T)`;
- Curva de aprendizado (*learning curve*) do *sklearn.model_selection*, sendo importado *learning_curve* e sugiro usar também, `StratifiedKFold` para cross-validation;
- Curva ROC e valor de AUC, sendo que agora pode usar diretamente *roc_curve*, *roc_auc_score* do *sklearn.metrics*. Note que para obter os "resultados" do classificador voce precisa de *model.predict_proba(X_test)*;
- Visualizar o poder de separação do sinal e background analisando os *scores* obtidos do treino/test.

c) **Ajuste de Hiperparâmetros:** Implemente uma rotina simples que varie parâmetros principais da árvore (`max_depth`, `min_samples_split`, `min_samples_leaf`, etc.) em alguns intervalos, e compare novamente as métricas obtidas. Neste caso, você pode simplificar usando o *GridSearchCV* do *sklearn.model_selection*.

d) **Seleção de Atributos:** Explore a importância das variáveis. Teste a performance do modelo ao remover ou adicionar *features* (por exemplo, usando `SelectFromModel` ou `feature_importances_`). Avalie como o desempenho se altera.

e) **Validação Cruzada:** Aplique *k-fold cross validation* (por exemplo, com $k = 5$) para estimar a estabilidade do modelo. Compare os resultados médios de ROC AUC e acurácia entre os folds.

f) **Floresta Aleatória:** Treine um `RandomForestClassifier` e compare as métricas (ROC AUC, acurácia, eficiência e rejeição) com a Decision tree. Discuta como a combinação de múltiplas árvores afeta o resultado.

g) **Gradiente Boosting:** Treine um modelo `GradientBoostingClassifier` e compare suas métricas com os modelos anteriores. Em seguida, varie `n_estimators` e `max_depth` e analise a evolução do AUC.

h) **Adaboost e XGBoost:** Treine modelos `AdaBoostClassifier` e `XGBClassifier`. Compare com os modelos anteriores e discuta as diferenças de desempenho e tempo de treino.

i) **Aplicação a Dados Reais:** Adicione os resultados dos diferentes treinos a amostra de dados reais e aplique um corte de tipicamente de 90% de rejeição de background e compare a performance dos diferentes algoritmos.