Aprendizado de Máquina

Fluxo de Trabalho em Machine Learning e Técnicas de Otimização

Prof. Klayton R. Castro

IDP
Instituto Brasileiro de Ensino, Desenvolvimento e Pesquisa

11 de setembro de 2024

Práticas em Laboratório

- No decorrer do curso, nossas práticas em laboratório serão desenvolvidas no ambiente Jupyter, uma ferramenta amplamente utilizada por pesquisadores, educadores, engenheiros, analistas e cientistas de dados para criar documentos que integram texto, código, equações e visualizações.
- No Jupyter, o *notebook* é um documento interativo que pode ser salvo e compartilhado no formato `.ipynb`, facilitando a execução individual de células de código, documentação de equações, inserção de imagens, depuração e prototipagem de soluções que envolvem exploração de dados, aprendizado de máquina, dentre outros processos no fluxo de trabalho.

Datasets e Notebooks

- Os Datasets e Jupyter Notebooks utilizados durante o curso estão disponíveis no GitHub para referência, revisão ou prática adicional.
- Visite o repositório `idp-machinelearning` na seguinte URL:

`https://github.com/klaytoncastro`

Fluxo de Trabalho em Machine Learning

O fluxo de trabalho padrão em machine learning inclui várias etapas, que podem ser adaptadas conforme a necessidade específica do projeto:

- Definição do Problema: Esclarecer qual problema você deseja resolver e definir claramente o objetivo.
- Coleta de Dados: Obter dados relevantes que possam ser usados para treinar o modelo. Isso pode incluir a coleta de dados novos, a utilização de datasets existentes ou a combinação de ambos. Aqui aferimos a qualidade dos dados pois, às vezes, é preciso realizar correções ou melhorias.
- Análise Exploratória de Dados (EDA): Analisar os dados para entender padrões, tendências e anomalias, e para formular hipóteses sobre os dados. Isso inclui visualização de dados, estatísticas descritivas, identificação de possíveis outliers e correlação entre variáveis.

Fluxo de Trabalho em Machine Learning (cont.)

- Pré-processamento de Dados: Limpar e formatar os dados adequadamente. Isso pode incluir tratamento de valores ausentes, codificação de variáveis categóricas, normalização ou padronização de variáveis numéricas e remoção de colunas irrelevantes. Importante considerar técnicas de balanceamento de classes, como SMOTE, para Oversampling ou Undersampling, no caso de problemas de classificação desequilibrada.
- Engenharia de Recursos: Criar novos recursos a partir dos dados existentes (feature engineering) para melhorar a capacidade do modelo de aprender padrões significativos.

Fluxo de Trabalho em Machine Learning (cont.)

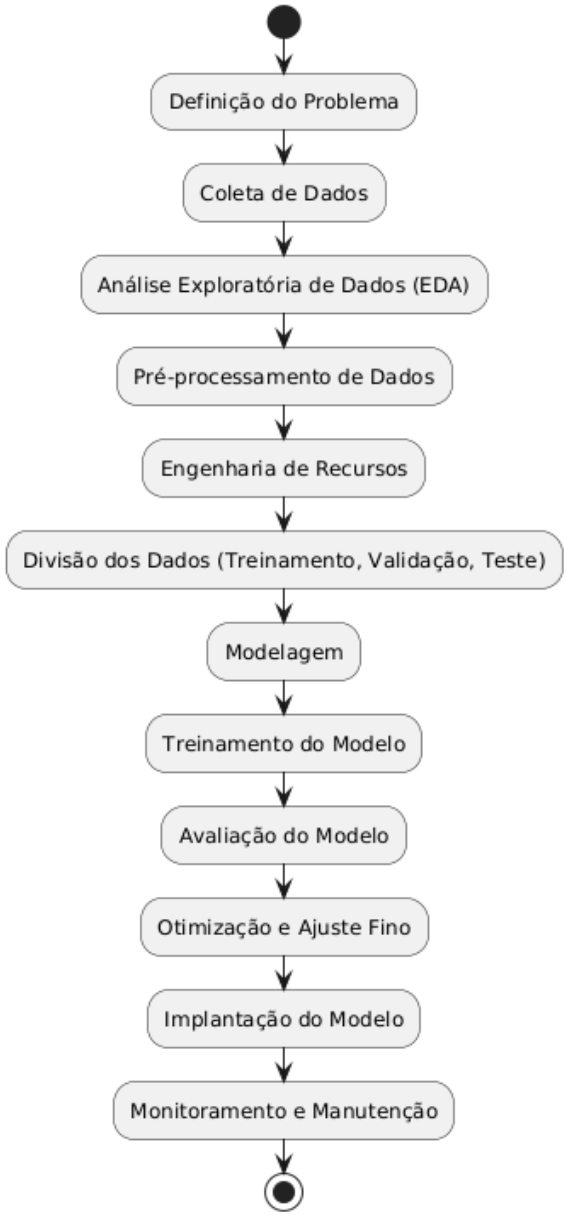
- Divisão dos Dados: Dividir o dataset em conjuntos de treinamento, validação (cross-validation) e teste.
- Modelagem: Escolher um ou mais algoritmos de machine learning adequados para o problema. Isso pode incluir modelos lineares, árvores de decisão, stacking / ensemble methods, etc.
- Treinamento e Avaliação: Treinar o modelo, usando dados de treinamento. Avaliar o desempenho do modelo usando métricas apropriadas (como precisão, recall, AUC-ROC, R2, MAE, MSE, RMSE, etc.) no conjunto de teste.

Fluxo de Trabalho em Machine Learning (cont.)

- Otimização e Ajuste Fino: Refinar o modelo e ajustar seus hiperparâmetros, usando o conjunto de validação, para maximizar o desempenho.
- Implantação: Aplicar o modelo em um ambiente de produção, onde ele possa receber dados novos e fazer previsões (MLOps).
- Monitoramento e Manutenção: Monitorar o desempenho do modelo ao longo do tempo para garantir que ele continue performando bem conforme novos dados surgem.

Visão Geral do Fluxo de Trabalho

- Aqui está uma visão geral do processo:



Definição de Otimização

- Processo que visa encontrar a melhor solução possível para um determinado problema, sujeito a restrições, por meio da maximização ou minimização de uma função objetivo.
- Envolve a busca sistemática por soluções viáveis em um espaço de busca, ajustando variáveis ou parâmetros de forma iterativa para melhorar progressivamente a função objetivo.

Definição de Otimização (cont.)

- Pode ser aplicado em uma ampla variedade de domínios, desde problemas de engenharia e logística até problemas de design de sistemas ou do cotidiano, como encontrar a rota mais eficiente para um deslocamento.
- Por exemplo, seu objetivo pode ser o atendimento a determinados critérios: alcançar uma meta de desempenho, maximizar o lucro, minimizar os custos, maximizar a precisão, recall ou ambos de maneira balanceada (f1-score) em um modelo de aprendizado de máquina.

Definição de Hiperparâmetros

- Hiperparâmetros são variáveis que controlam o comportamento do algoritmo de machine learning durante o treinamento e afetam sua capacidade de aprender padrões nos dados.
- A escolha adequada de hiperparâmetros pode levar a modelos mais precisos e eficientes, enquanto hiperparâmetros mal ajustados podem resultar em desempenho insatisfatório.

Otimização de Hiperparâmetros

- A otimização de hiperparâmetros, portanto, visa encontrar a configuração ideal dos parâmetros externos ao algoritmo, tal que maximizem o desempenho do modelo, influenciando seu poder de generalização e eficácia.
- Durante o curso, trabalharemos com alguns dos métodos mais populares para otimização de modelos de Machine Learning:
  - Busca em Grade (Grid Search)
  - Busca Aleatória (Random Search)
  - Algoritmos baseados no Teorema de Bayes (Otimização Bayesiana).

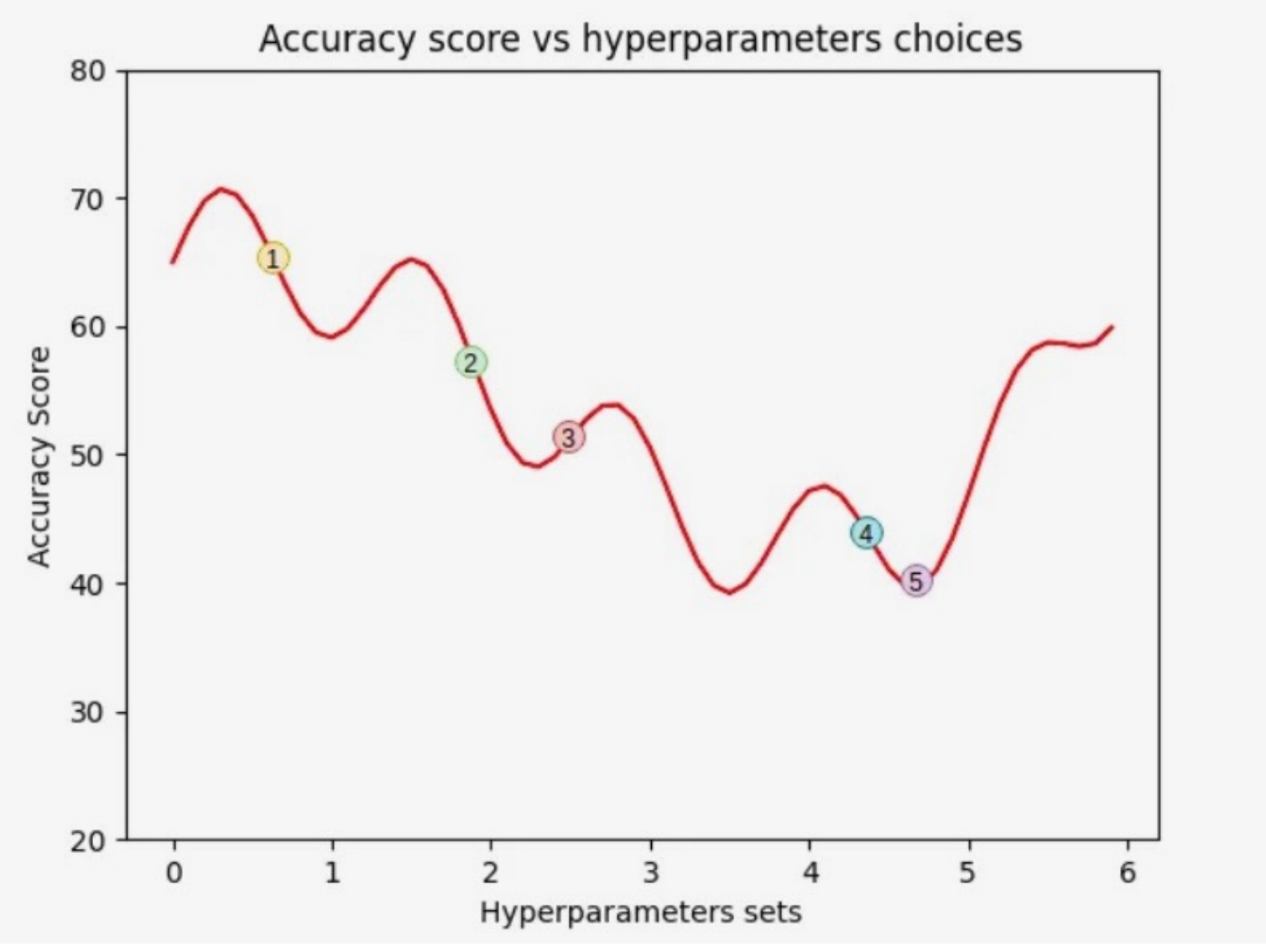
Exemplos de Hiperparâmetros

- Taxa de aprendizado em algoritmos de otimização (Gradient Boosting);
- Número de árvores em algoritmos baseados em Árvore de Decisão (Decision Tree, Random Forest, Extra Trees);
- Número de clusters em algoritmos de clusterização (K-means, DBSCAN).

Exemplos de Hiperparâmetros (Cont.)

- **Regressão Linear:**
  - Coeficiente de regularização (α)
  - Tipo de regularização (L1, L2)
- **Regressão Logística:**
  - Coeficiente de regularização (C)
  - Tipo de regularização (L1, L2)
  - Método de otimização (solver)
- **SVM (Support Vector Machine):**
  - Tipo de kernel (rbf, linear, poly)
  - Parâmetro de suavização (C)
  - Parâmetro do kernel (gamma)

Critério de Escolha



Principais Desafios

- Alta dimensionalidade do espaço de busca;
- Limitações computacionais;
- Complexidade nas combinações entre os hiperparâmetros;
- Necessidade de evitar overfitting ao otimizar os hiperparâmetros.

Grid Search

- Abordagem sistemática que testa todas as combinações possíveis de hiperparâmetros especificados em uma grade.
- Fácil implementação e compreensão, sendo uma boa opção para espaços de busca pequenos.
- Porém, pode ser computacionalmente caro para espaços de busca grandes.
- Não leva em conta resultados anteriores para informar as próximas escolhas de hiperparâmetros.

GridSearchCV

```
1  from sklearn.model_selection import GridSearchCV
2  from sklearn.svm import SVC
3  from sklearn.datasets import load_iris
4
5  # Carregar conjunto de dados
6  iris = load_iris()
7  X = iris.data
8  y = iris.target
9
10 # Definir o modelo
11 model = SVC()
12
13 # Definir os par metros para Grid Search
14 param_grid = {'C': [0.1, 1, 10, 100], 'gamma':
       [0.1, 0.01, 0.001], 'kernel': ['rbf', 'linear
       ']}
```

GridSearchCV cont.

```
# Criar o objeto GridSearchCV
grid_search = GridSearchCV(estimator=model,
    param_grid=param_grid, cv=3)

# Executar a busca em grade
grid_search.fit(X, y)

# Exibir os melhores par metros e o melhor
    score
print("Melhores par metros:", grid_search.
    best_params_)
print("Melhor score:", grid_search.best_score_)
```

Limitações do Grid Search

- Impraticável para hiperparâmetros contínuos ou espaços de busca grandes.
- Não aproveita informações sobre o desempenho do modelo em iterações anteriores.
- Pode resultar em uma busca ineficiente e demorada.

Randomized Search

- Seleciona aleatoriamente combinações de hiperparâmetros para avaliação.
- Mais eficiente que o Grid Search, especialmente para hiperparâmetros contínuos.
- Explora melhor o espaço de busca, mesmo com menos iterações.
- No entanto, pode ser menos eficiente em espaços de busca menores.

RandomizedSearchCV

```
1 from sklearn.model_selection import
      RandomizedSearchCV
2 from scipy.stats import uniform
3 from sklearn.ensemble import
      RandomForestClassifier
4 from sklearn.datasets import load_iris
5
6 # Carregar conjunto de dados
7 iris = load_iris()
8 X = iris.data
9 y = iris.target
```

RandomizedSearchCV cont.

```
1
2 # Definir o modelo
3 model = RandomForestClassifier()
4
5 # Definir a distribui  o dos par metros para
    Random Search
6 param_dist = {'n_estimators': [10, 100, 200,
    300],
7                'max_depth': [3, 5, 10, 20],
8                'min_samples_split': uniform(0.1,
                    0.9)}
```

RandomizedSearchCV cont. 2

```
# Criar o objeto RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=
    model, param_distributions=param_dist, n_iter
    =10, cv=3)

# Executar a busca aleat ria
random_search.fit(X, y)

# Exibir os melhores par metros e o melhor
    score
print("Melhores par metros:", random_search.
    best_params_)
print("Melhor score:", random_search.best_score_
    )
```

Otimização Bayesiana

- Utiliza modelos probabilísticos para guiar a busca pelos melhores hiperparâmetros.
- Modela a função objetivo como uma distribuição probabilística.
- Atualiza continuamente suas crenças com base nas observações anteriores.
- Mais eficiente e converge mais rapidamente para boas configurações que o Grid Search e o Random Search.

Algoritmos de Otimização Bayesiana

- Gaussian Process Regression (GPR) e Tree Parzen Estimator (TPE) são algoritmos comuns.
- GPR modela a função objetivo como um processo gaussiano.
- TPE utiliza uma árvore de decisão para dividir o espaço de busca.
- Ambos são eficazes em espaços de hiperparâmetros contínuos e discretos.

Bayesian Optimization

```
from skopt import BayesSearchCV
from sklearn.svm import SVC
from sklearn.datasets import load_iris

# Carregar conjunto de dados
iris = load_iris()
X = iris.data
y = iris.target
```

Bayesian Optimization cont.

```
# Definir o modelo
model = SVC()

# Definir os limites dos par metros para a
    otimiza  o bayesiana
param_space = {'C': (0.1, 100.0, 'log-uniform'),
               'gamma': (0.01, 1.0, 'log-uniform
                  '),
               'kernel': ['rbf', 'linear']}
```

Bayesian Optimization cont. 2

```
# Criar o objeto BayesSearchCV
bayes_search = BayesSearchCV(estimator=model,
    search_spaces=param_space, n_iter=10, cv=3)

# Executar a otimiza  o bayesiana
bayes_search.fit(X, y)

# Exibir os melhores par metros e o melhor
    score
print("Melhores par metros:", bayes_search.
    best_params_)
print("Melhor score:", bayes_search.best_score_)
```

Vantagens da Otimização Bayesiana

- Eficiente mesmo para espaços de hiperparâmetros grandes e complexos.
- Converge mais rapidamente para boas configurações que outras abordagens.
- Usa informações anteriores para direcionar a busca, resultando em uma exploração mais inteligente do espaço de busca.
- Minimiza o número de iterações necessárias para encontrar uma boa solução.

Processo de Otimização

- O dilema *Exploration-Exploitation*: Devo ir para regiões menos exploradas caso esteja faltando alguma coisa? Ou continuar tentando pontos próximos às regiões que já conheço e apresentam resultados promissores?
- E se minha função de otimização for não convexa, não linear e ruidosa (*noisy*)? E se eu ficar preso no **máximo local** pensando que é a melhor solução (**máximo global**)?
- Uma abordagem mais simples seria testar o acionamento o algoritmo com múltiplos valores de inicialização (*random state*), para explorar diferentes partes do espaço de busca e evitar ficar preso em máximos locais.
- Uma abordagem mais complexa seria a utilização de técnicas de otimização mais avançadas, como algoritmos genéticos, algoritmos de enxame, ou métodos de otimização estocástica, que são mais robustos ao lidar com ambientes não convencionais.

Conclusão

- A escolha da técnica de otimização de hiperparâmetros depende do problema específico e das restrições computacionais.
- Além disso, a escolha adequada de hiperparâmetros é crucial para o desempenho e a generalização dos modelos de machine learning, especialmente quando adotada em conjunto com cross-validation.
- Abordagens avançadas de otimização de hiperparâmetros, como a Otimização Bayesiana, oferecem uma alternativa eficiente e eficaz ao Grid Search e ao Randomized Search.

Descrição da Atividade

Objetivo

Cada estudante ou dupla otimizará um algoritmo de machine learning, utilizando as três técnicas de otimização de hiperparâmetros essenciais e consultando a documentação específica de cada algoritimo.

Algoritmos Atribuídos

- Decision Tree
- Random Forest
- Naive Bayes
- Logistic Regression
- KNN
- SVM (SVC)
- Extra Trees
- XGBoost

Recursos e Diretrizes

Documentação

Consulte a documentação oficial do **scikit-learn** e **XGBoost** para entender e aplicar as técnicas de otimização.

Técnicas de Otimização

- Grid Search
- Random Search
- Otimização Bayesiana

Métricas de Avaliação

Compare o desempenho utilizando métricas como precisão, AUC-ROC, ou F1-score.

Formato do Relatório

Conteúdo do Relatório

Documente as etapas realizadas, hiperparâmetros testados, e resultados obtidos.

Apresentações

Prepare uma apresentação dos resultados para compartilhar com a turma.

Links Úteis

- Scikit-learn User Guide
- XGBoost Documentation