Relatório de Análise de Machine Learning

Dataset Yeast - Predição de Localização de Proteínas

---

1. Descrição dos Dados

1.1 Apresentação do Dataset

O dataset **Yeast** (UCI Machine Learning Repository) contém informações sobre proteínas de levedura (*Saccharomyces cerevisiae*) e tem como objetivo predizer a localização celular das proteínas com base em características bioquímicas.

1.2 Características do Dataset

- **Total de instâncias**: 1.484 exemplos
- **Número de features**: 8 atributos numéricos contínuos
- **Número de classes**: 10 localizações celulares diferentes
- **Tipo de problema**: Classificação multiclasse

1.3 Atributos (Features)

As 8 features representam scores de análise de sequência:

1. **mcg**: Presença de sítios de reconhecimento de McGeoch
2. **gvh**: Presença de sítios de reconhecimento de von Heijne
3. **alm**: Score de Alom et al.
4. **mit**: Score de presença na mitocôndria
5. **erl**: Score de presença no retículo endoplasmático
6. **pox**: Score de presença no peroxissoma
7. **vac**: Score de presença no vacúolo
8. **nuc**: Score de presença no núcleo

1.4 Classes (Localizações Celulares)

- CYT (citoplasma)
- NUC (núcleo)
- MIT (mitocôndria)
- ME3 (membrana 3)
- ME2 (membrana 2)
- ME1 (membrana 1)

- EXC (excretado)
- VAC (vacúolo)
- POX (peroxissoma)
- ERL (retículo endoplasmático)

1.5 Desbalanceamento das Classes

O dataset apresenta desbalanceamento significativo entre as classes, com algumas classes tendo poucos exemplos. Isso justifica o uso de amostragem **estratificada** em todas as divisões de dados.

---

2. Organização dos Experimentos

2.1 Pré-processamento dos Dados

2.1.1 Codificação de Classes

- As classes categóricas foram convertidas em valores numéricos usando `LabelEncoder` do scikit-learn
- Esta conversão é necessária para os algoritmos de aprendizado de máquina

2.1.2 Normalização

- Todas as features foram normalizadas usando `StandardScaler`
- **Método**: Padronização Z-score (média = 0, desvio padrão = 1)
- **Justificativa**:
    - Coloca todas as features na mesma escala
    - Essencial para algoritmos baseados em distância (K-Neighbors, Regressão Logística)
    - Melhora a convergência de algoritmos iterativos
- **Importante**: O scaler foi **ajustado apenas no conjunto de treino** e depois aplicado no conjunto de teste para evitar vazamento de informação (data leakage)

2.2 Divisão dos Dados

2.2.1 Divisão Inicial (70-30)

- **Conjunto de Treino**: 70% dos dados (1.039 amostras)
- **Conjunto de Teste**: 30% dos dados (445 amostras)
- **Método**: Amostragem estratificada (`stratify=y`)
- **Semente aleatória**: 42 (para reprodutibilidade)
- **Justificativa da estratificação**: Mantém a proporção de cada classe nos conjuntos de treino e teste, crítico devido ao desbalanceamento das classes

2.2.2 Divisão para Curvas de Aprendizagem

- **Tamanhos de treino**: 5%, 10%, 15%, ..., 90%, 95%
- **Passo**: 5%
- **Total de divisões**: 19 diferentes proporções
- **Método**: Amostragem estratificada em cada divisão
- **Objetivo**: Avaliar como o desempenho varia com a quantidade de dados de treinamento

2.3 Validação e Otimização de Hiperparâmetros

2.3.1 Estratégia de Validação

- **Método**: Grid Search com Validação Cruzada Estratificada
- **Número de folds**: 5 (5-fold cross-validation)
- **Estratificação**: Aplicada em cada fold para manter proporção das classes
- **Métrica de otimização**: F1-Score ponderado (`f1_weighted`)
    - Escolhido por balancear precisão e recall
    - Versão ponderada considera o desbalanceamento das classes

2.3.2 Garantia de Avaliação Correta

CRÍTICO: O conjunto de teste **NUNCA** foi usado durante:

- Treinamento dos modelos
- Seleção de hiperparâmetros (Grid Search)
- Ajuste do normalizador (StandardScaler)

O conjunto de teste foi usado **APENAS** para avaliação final dos modelos com os melhores hiperparâmetros encontrados.

---

3. Classificadores e Hiperparâmetros

3.1 Árvore de Decisão (`DecisionTreeClassifier`)

3.1.1 Hiperparâmetros Testados

- **max_depth**: [5, 10, 15, 20, None]
    - Profundidade máxima da árvore
    - None = sem limite de profundidade
- **min_samples_split**: [2, 5, 10]
    - Número mínimo de amostras para dividir um nó interno
- **min_samples_leaf**: [1, 2, 4]
    - Número mínimo de amostras em uma folha
- **criterion**: ['gini', 'entropy']
    - Função para medir a qualidade da divisão

3.1.2 Espaço de Busca

- **Total de combinações testadas**: 5 × 3 × 3 × 2 = 90 combinações
- **Avaliações totais**: 90 × 5 folds = 450 treinamentos

3.1.3 Melhores Hiperparâmetros Encontrados

[Serão preenchidos após execução do código]

3.2 Naive Bayes Gaussiano (`GaussianNB`)

3.2.1 Hiperparâmetros Testados

- **var_smoothing**: [1e-9, 1e-8, 1e-7, 1e-6]
    - Porção da variância da maior feature adicionada às variâncias para estabilidade
    - Previne divisão por zero

3.2.2 Espaço de Busca

- **Total de combinações testadas**: 4 combinações
- **Avaliações totais**: 4 × 5 folds = 20 treinamentos

3.2.3 Características do Algoritmo

- Assume que as features seguem distribuição Gaussiana
- Assume independência condicional entre features
- Muito eficiente computacionalmente
- Baseline simples para comparação

3.2.4 Melhores Hiperparâmetros Encontrados

[Serão preenchidos após execução do código]

3.3 Regressão Logística (`LogisticRegression`)

3.3.1 Hiperparâmetros Testados

- **C**: [0.01, 0.1, 1, 10, 100]
    - Inverso da força de regularização
    - Valores menores = regularização mais forte
- **penalty**: ['l2']
    - Tipo de regularização (Ridge)
- **solver**: ['lbfgs', 'saga']
    - Algoritmo de otimização
    - lbfgs: eficiente para datasets pequenos/médios
    - saga: suporta penalidades l1 e l2

3.3.2 Configurações Fixas

- **max_iter**: 1000 (para garantir convergência)
- **random_state**: 42

3.3.3 Espaço de Busca

- **Total de combinações testadas**: 5 × 1 × 2 = 10 combinações
- **Avaliações totais**: 10 × 5 folds = 50 treinamentos

3.3.4 Melhores Hiperparâmetros Encontrados

[Serão preenchidos após execução do código]

3.4 K-Vizinhos Mais Próximos (`KNeighborsClassifier`)

3.4.1 Hiperparâmetros Testados

- **n_neighbors**: [3, 5, 7, 9, 11, 15, 20]
  - Número de vizinhos a considerar
  - Valores ímpares preferíveis para evitar empates
- **weights**: ['uniform', 'distance']
  - uniform: todos os vizinhos têm peso igual
  - distance: peso inversamente proporcional à distância
- **metric**: ['euclidean']
  - Distância Euclidiana (conforme especificação do projeto)

3.4.2 Espaço de Busca

- **Total de combinações testadas**: 7 × 2 × 1 = 14 combinações
- **Avaliações totais**: 14 × 5 folds = 70 treinamentos

3.4.3 Justificativa da Normalização

- **CRUCIAL** para K-NN: features com escalas diferentes dominam o cálculo de distância
- Normalização garante que todas as features contribuam igualmente

3.4.4 Melhores Hiperparâmetros Encontrados

[Serão preenchidos após execução do código]

3.5 Random Forest (`RandomForestClassifier`)

3.5.1 Hiperparâmetros Testados

- **n_estimators**: [50, 100, 200]
    - Número de árvores na floresta
- **max_depth**: [10, 20, 30, None]
    - Profundidade máxima de cada árvore
- **min_samples_split**: [2, 5, 10]
    - Número mínimo de amostras para dividir nó
- **min_samples_leaf**: [1, 2, 4]
    - Número mínimo de amostras em folha

3.5.2 Configurações Fixas

- **random_state**: 42
- Outras configurações padrão do scikit-learn

3.5.3 Espaço de Busca

- **Total de combinações testadas**: 3 × 4 × 3 × 3 = 108 combinações
- **Avaliações totais**: 108 × 5 folds = 540 treinamentos

3.5.4 Características do Algoritmo

- Ensemble de árvores de decisão
- Reduz overfitting através de agregação
- Robusto a outliers e ruído

3.5.5 Melhores Hiperparâmetros Encontrados

[Serão preenchidos após execução do código]

---

4. Métricas de Avaliação

4.1 Métricas Utilizadas

4.1.1 Precisão (Precision)

- **Definição**: Proporção de predições positivas corretas
- **Fórmula**: TP / (TP + FP)
- **Interpretação**: Quando o modelo prediz uma classe, qual a probabilidade de estar correto?

4.1.2 Cobertura (Recall)

- **Definição**: Proporção de positivos reais que foram identificados
- **Fórmula**: TP / (TP + FN)
- **Interpretação**: De todos os exemplos de uma classe, quantos foram identificados?

4.1.3 F1-Score

- **Definição**: Média harmônica entre precisão e recall
- **Fórmula**: 2 × (Precision × Recall) / (Precision + Recall)
- **Interpretação**: Balanceia precisão e cobertura em uma única métrica
- **Uso principal**: Métrica principal para comparação de modelos

4.1.4 Acurácia (Accuracy)

- **Definição**: Proporção de predições corretas
- **Fórmula**: (TP + TN) / Total
- **Limitação**: Pode ser enganosa em datasets desbalanceados

4.2 Versões Ponderadas

- Todas as métricas usam a versão **weighted** (ponderada)
- Calcula a métrica para cada classe e faz média ponderada pelo número de exemplos
- **Justificativa**: Apropriado para datasets desbalanceados como o Yeast

---

5. Resultados

5.1 Comparação dos Classificadores

[Tabela será preenchida após execução]

Classificador	Precision	Recall	F1-Score	Accuracy
Decision Tree	-	-	-	-
Naive Bayes	-	-	-	-
Logistic Regression	-	-	-	-
K-Neighbors	-	-	-	-
Random Forest	-	-	-	-

5.2 Melhores Hiperparâmetros por Classificador

[Será preenchido após execução]

---

6. Análise das Curvas de Aprendizagem

6.1 Objetivo

As curvas de aprendizagem mostram como o desempenho dos modelos varia com a quantidade de dados de treinamento disponíveis.

6.2 Metodologia

- Para cada proporção (5% a 95% com passo de 5%):
    1. Dividir dados estratificadamente
    2. Treinar modelo com melhores hiperparâmetros
    3. Avaliar em treino e teste
    4. Registrar métricas

6.3 Interpretação das Curvas

6.3.1 Overfitting (Sobreajuste)

- **Indicador**: Grande diferença entre curvas de treino e teste
- **Curva de treino**: Alta e estável
- **Curva de teste**: Baixa ou instável
- **Causa**: Modelo memorizou dados de treino
- **Solução**: Mais dados, regularização, modelo mais simples

6.3.2 Underfitting (Subajuste)

- **Indicador**: Ambas as curvas em valores baixos
- **Curvas próximas mas ambas baixas**
- **Causa**: Modelo muito simples
- **Solução**: Modelo mais complexo, mais features

6.3.3 Bom Ajuste

- **Indicador**: Curvas próximas e em valores altos
- **Gap pequeno entre treino e teste**
- **Convergência das curvas**

6.3.4 Tendências Esperadas

- Com mais dados de treino:
    - Desempenho no treino tende a diminuir (menos overfitting)
    - Desempenho no teste tende a aumentar (melhor generalização)
    - As curvas convergem

---

7. Comentários e Observações

7.1 Pontos Fortes da Metodologia

1. **Estratificação consistente**: Mantém proporção das classes em todas as divisões

2. **Validação cruzada**: Reduz viés na seleção de hiperparâmetros
3. **Separação rigorosa**: Teste nunca usado em treinamento/validação
4. **Normalização adequada**: Ajustada apenas em treino
5. **Múltiplas métricas**: Avaliação abrangente do desempenho

7.2 Desafios do Dataset

1. **Desbalanceamento de classes**: Algumas classes têm poucos exemplos
2. **Complexidade do problema**: 10 classes diferentes
3. **Features limitadas**: Apenas 8 atributos

7.3 Limitações

1. **Grid Search**: Explora espaço discreto (pode não encontrar ótimo global)
2. **Computational**: Random Forest com muitas combinações é custoso
3. **Tamanho do dataset**: ~1.500 exemplos é moderado

7.4 Recomendações Futuras

1. **Random Search**: Alternativa mais eficiente ao Grid Search
2. **Feature Engineering**: Criar novas features ou transformações
3. **Ensemble Methods**: Combinar múltiplos modelos
4. **Técnicas de balanceamento**: SMOTE, undersampling, etc.
5. **Otimização Bayesiana**: Para busca mais inteligente de hiperparâmetros

---

8. Conclusão

Este relatório apresentou uma metodologia rigorosa para avaliação de 5 classificadores diferentes no dataset Yeast. A organização dos experimentos garantiu:

✅ Avaliação justa e sem viés
✅ Seleção apropriada de hiperparâmetros
✅ Reprodutibilidade dos resultados
✅ Comparação válida entre modelos

As curvas de aprendizagem forneceram insights sobre:

- Capacidade de generalização de cada modelo
- Quantidade de dados necessária para bom desempenho
- Presença de overfitting ou underfitting

Os resultados obtidos permitem escolher o modelo mais adequado para o problema de predição de localização celular de proteínas, considerando o trade-off entre precisão, recall e complexidade computacional.

Autor: [Seu Nome]
Data: [Data da Execução]
Ferramentas: Python 3.x, scikit-learn, pandas, numpy, matplotlib, seaborn