Guia Completo para Treinamento de Redes Neurais com Dados Tabulares

Seu Nome
Orientador: Nome do Orientador

28 de agosto de 2025

Resumo

Este documento apresenta um guia passo a passo para o desenvolvimento de projetos de Inteligência Artificial utilizando dados tabulares. O material foi desenvolvido como parte de um projeto de Iniciação Científica em Engenharia de Software, documentando todo o processo desde o pré-processamento de dados até o treinamento e avaliação de modelos de rede neural. O enfoque didático inclui erros comuns, boas práticas e observações críticas para cada etapa do processo.

Sumário

1. Introdução

Este material foi desenvolvido como parte de um projeto de Iniciação Científica com o objetivo de criar um recurso educacional completo para estudantes de Engenharia de Software que desejam aprender Inteligência Artificial na prática.

Objetivos do projeto:

- Fornecer um roteiro claro para treinamento de redes neurais com dados tabulares
- Documentar erros e acertos ao longo do processo
- Criar um material de referência para futuros projetos
- Desenvolver habilidades práticas em pré-processamento, modelagem e avaliação

2. Pré-Processamento de Dados

2.1. Análise Exploratória de Dados (EDA)

O que fazer:

- Carregar o dataset e examinar suas dimensões
- Verificar tipos de dados e valores missing
- Analisar distribuições estatísticas e correlações
- Visualizar dados com gráficos apropriados

Exemplo de código:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Carregar dados
df = pd.read_csv('dataset.csv')

# Informa  es b sicas
print(df.info())
print(df.describe())

# Verificar valores missing
print(df.isnull().sum())

# Visualizar correla  es
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

Listing 1: Análise Exploratória Básica

O que o aluno deve perceber:

✓ A qualidade dos dados impacta diretamente no desempenho do modelo

✓ Valores missing podem distorcer completamente os resultados

✓ Correlações fortes podem indicar multicolinearidade

✓ Distribuições desbalanceadas podem exigir tratamento especial

2.2. Limpeza e Transformação de Dados

O que fazer:

- Tratamento de valores missing (imputação, remoção)
- Codificação de variáveis categóricas (One-Hot, Label Encoding)
- Normalização/Padronização de características numéricas
- Tratamento de outliers

Exemplo de código:

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Definir transforma  es para colunas num ricas e categ ricas
numeric_features = ['age', 'income']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_features = ['gender', 'category']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='
       missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Combinar transforma  es
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])
```

Listing 2: Pré-processamento de Dados

O que o aluno deve perceber:

✓ Diferentes estratégias de imputação produzem resultados diferentes

✓ One-Hot Encoding pode aumentar significativamente a dimensionalidade

✓ A normalização é crucial para algoritmos baseados em gradiente

✓ Decisões de pré-processamento devem ser documentadas cuidadosamente

2.3. Divisão dos Dados

O que fazer:

- Dividir dados em conjuntos de treino, validação e teste
- Considerar estratificação para manter distribuições similares

- Garantir que não haja vazamento de informação entre os conjuntos

Exemplo de código:

```
from sklearn.model_selection import train_test_split

# Separar caracter sticas e r tulo
X = df.drop('target', axis=1)
y = df['target']

# Divis o inicial: treino + temporario
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42)

# Divis o do temporario: validacao e teste
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state
       =42)
```

Listing 3: Divisão Estratificada dos Dados

O que o aluno deve perceber:

✓ A estratificação é especialmente importante em conjuntos desbalanceados

✓ O vazamento de dados é um erro comum e catastrófico

✓ O conjunto de teste deve ser tocado apenas na avaliação final

✓ O random$_{s}tategarantereproducibilidade$

3. Configuração da Rede Neural

3.1. Seleção da Arquitetura

O que fazer:

- Definir número de camadas e neurônios por camada
- Escolher funções de ativação apropriadas
- Considerar a complexidade do problema para definir a arquitetura

Exemplo de código:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(input_dim,)))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))  # Para classificacao
   binaria
```

Listing 4: Definição da Arquitetura da Rede

O que o aluno deve perceber:

✓ Arquiteturas muito complexas podem levar a overfitting

✓ Arquiteturas muito simples podem não capturar padrões complexos

✓ A função de ativação da camada final depende do problema

✓ É necessário experimentar diferentes arquiteturas

3.2. Inicialização e Regularização

O que fazer:

- Escolher estratégias de inicialização de pesos
- Aplicar técnicas de regularização (L1/L2, Dropout)
- Considerar Batch Normalization para estabilizar o treinamento

Exemplo de código:

```
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import BatchNormalization

model = Sequential()
model.add(Dense(64, activation='relu',
                kernel_regularizer=l2(0.01),
                input_shape=(input_dim,)))
model.add(BatchNormalization())
model.add(Dropout(0.3))
# ... mais camadas
```

Listing 5: Adicionando Regularização

O que o aluno deve perceber:

✓ A regularização ajuda a prevenir overfitting

✓ Dropout "desliga"neurônios aleatoriamente durante o treinamento

✓ Batch Normalization acelera a convergência e estabiliza o treinamento

✓ É preciso balancear a força da regularização

3.3. Compilação do Modelo

O que fazer:

- Escolher uma função de perda apropriada ao problema
- Selecionar um otimizador e ajustar sua taxa de aprendizado
- Definir métricas de avaliação relevantes

Exemplo de código:

```python
from tensorflow.keras.optimizers import Adam

model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',  # Para classificacao
                 binaria
              metrics=['accuracy',
                       'precision',
                       'recall'])
```

Listing 6: Compilação do Modelo

O que o aluno deve perceber:

✓ A função de perda deve refletir o objetivo do problema

✓ Diferentes otimizadores têm características diferentes

✓ A taxa de aprendizado impacta diretamente na convergência

✓ Métricas adicionais fornecem visões complementares do desempenho

4. Treinamento e Avaliação

4.1. Treinamento e Monitoramento

O que fazer:

- Definir número de épocas e tamanho do batch
- Implementar callbacks para early stopping e ajuste de learning rate
- Monitorar loss e métricas em treino e validação

Exemplo de código:

```python
from tensorflow.keras.callbacks import EarlyStopping,
   ReduceLROnPlateau

callbacks = [
    EarlyStopping(patience=10, restore_best_weights=True),
    ReduceLROnPlateau(factor=0.1, patience=5)
]

history = model.fit(X_train, y_train,
                    epochs=100,
                    batch_size=32,
                    validation_data=(X_val, y_val),
                    callbacks=callbacks,
                    verbose=1)
```

Listing 7: Treinamento com Callbacks

O que o aluno deve perceber:

✓ Early stopping previne overfitting interrompendo o treinamento

✓ A redução de LR ajuda a escapar de mínimos locais

✓ Batch size muito pequeno pode tornar o treinamento instável

✓ É crucial visualizar a curva de aprendizado

4.2. Validação e Ajustes

O que fazer:

- Avaliar o modelo no conjunto de validação
- Analisar matriz de confusão e métricas detalhadas
- Realizar ajustes de hiperparâmetros com base nos resultados

Exemplo de código:

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Previsoes no conjunto de validacao
y_pred = (model.predict(X_val) > 0.5).astype("int32")

# Matriz de confusao
cm = confusion_matrix(y_val, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.show()

# Relatorio de classificacao
print(classification_report(y_val, y_pred))
```

Listing 8: Avaliação do Modelo

O que o aluno deve perceber:

✓ A accuracy sozinha pode ser enganosa

✓ Precision e recall são especialmente importantes em dados desbalanceados

✓ A matriz de confusão revela padrões de erro específicos

✓ O ajuste de hiperparâmetros é um processo iterativo

4.3. Documentação de Erros e Lições Aprendidas

O que fazer:

- Manter um registro detalhado de todas as experiências
- Documentar hiperparâmetros testados e seus resultados
- Registrar insights e descobertas ao longo do processo

Exemplo de estrutura de documentação:

```
# Experimento 12 - 2023-11-15
# Objetivo: Testar regularizacao L2 com diferentes valores

# Hiperpar metros:
# - Arquitetura: 64-32-1
# - Learning rate: 0.001
# - L2: 0.01 (camadas 1 e 2)
# - Batch size: 32
# - Epocas: 50

# Resultados:
# - Train accuracy: 0.89
# - Val accuracy: 0.85
# - Overfitting reduzido em comparacao com exp. 11
# - LR pode estar muito alto para esta configuracao

# Pr ximos passos:
# - Testar LR=0.0005 com mesma configuracao
# - Experimentar dropout adicional na ultima camada
```

Listing 9: Registro de Experimentos

O que o aluno deve perceber:

✓ A documentação detalhada economiza tempo no longo prazo

✓ Erros são oportunidades de aprendizado valiosas

✓ A reprodução de resultados exige documentação precisa

✓ Padrões emergem após múltiplas iterações

5. Conclusão

Este guia apresentou um fluxo completo para o desenvolvimento de projetos de IA com dados tabulares, desde o pré-processamento até a avaliação final. O processo iterativo de experimentação e documentação é fundamental para o aprendizado e para o sucesso em projetos de machine learning.

Próximos passos sugeridos:

- Experimentar com diferentes arquiteturas de rede
- Implementar técnicas avançadas de pré-processamento
- Explorar métodos de ensemble e transfer learning
- Considerar implantação do modelo em produção