Projeto 6: Análise e identificação de SPAM em mensagens

Contextualização: Dataset SMS Spam Collection

O **SMS Spam Collection** é um conjunto de dados público amplamente utilizado para estudos de detecção de spam em mensagens de texto (SMS). Ele contém 5.574 mensagens, sendo 4.827 legítimas (*ham*) e 747 spam, coletadas de fontes como fóruns públicos, bases acadêmicas e contribuições voluntárias. Esse dataset é relevante porque:

- **Reflete desafios reais**: Mensagens SMS são curtas (até 160 caracteres), contêm abreviações, erros de digitação e linguagem informal, tornando a classificação complexa.
- **É desbalanceado**: A maioria das mensagens é legítima, exigindo cuidado na escolha de métricas de avaliação.
- **Serve como benchmark**: Foi utilizado em pesquisas acadêmicas para comparar algoritmos de machine learning, como SVM e Naive Bayes.

Você pode acessar o dataset diretamente no (UCI Archive):
https://archive.ics.uci.edu/dataset/228/sms+spam+collection

Situação Problema
Uma operadora de telecomunicações está recebendo reclamações de clientes sobre mensagens de spam em seus celulares. Essas mensagens não só irritam os usuários, mas também podem levar a golpes financeiros. A empresa contratou sua equipe para desenvolver um **modelo de classificação automática** que identifique mensagens spam com alta precisão, evitando bloquear mensagens legítimas (especialmente alertas importantes, como notificações bancárias ou emergenciais).

Objetivo: Criar um modelo de machine learning que classifique SMS como *ham* ou *spam* com base no conteúdo textual.

Passos para a Solução
Sigam as seguintes etapas em um notebook Jupyter (.ipynb):

1. **Carregamento e Exploração Inicial**
    - Importar o dataset (disponível no link do UCI).
    - Analisar a distribuição das classes (*ham* vs. *spam*).
    - Verificar exemplos de mensagens de cada classe.
2. **Pré-processamento de Texto**
    - Limpar o texto: remover caracteres especiais, números e converter para minúsculas.
    - Tokenizar as mensagens (dividir em palavras ou termos).

- *Opcional*: Testar técnicas como remoção de stopwords ou stemming, mas observar se melhoram a precisão (referência: no artigo original, essas técnicas não foram usadas).

3. **Vetorização dos Dados**
   - Converter texto em features numéricas usando **TF-IDF** ou **Bag of Words**.
4. **Divisão dos Dados**
   - Separar o dataset em conjuntos de treino (70%) e teste (30%).
   - Considerar técnicas para balanceamento de classes (ex: SMOTE para oversampling).
5. **Treinamento de Modelos**
   - Testar algoritmos como:
     - **SVM Linear** (destaque no artigo original).
     - **Naive Bayes Multinomial** (comum em NLP).
     - **Random Forest** ou **Regressão Logística**.
6. **Avaliação dos Modelos**
   - Calcular métricas: **Acurácia**, **Precisão**, **Recall**, **F1-Score** e **Matriz de Confusão**.
   - Justificar a escolha da métrica principal (ex: F1-Score é adequado para dados desbalanceados).
7. **Otimização**
   - Ajustar hiperparâmetros com **GridSearchCV** ou **RandomizedSearchCV**.
   - Comparar o desempenho antes e após a otimização.
8. **Conclusão e Recomendações**
   - Indicar o melhor modelo com base nas métricas.
   - Sugerir melhorias futuras (ex: usar n-grams, incorporar embeddings de palavras).

Entrega Esperada
Um notebook contendo:

- Código comentado e organizado.
- Visualizações claras (gráficos de distribuição, matriz de confusão).
- Análise crítica dos resultados.
- Referência ao artigo original e ao dataset do UCI.

Dica: Replique parcialmente o experimento do artigo (ex: testar SVM com tokenização específica) e comparar com suas próprias implementações!