Relatório Aprendizado de Máquina

Definição:

Nosso trabalho tem como objetivo desenvolver modelos baseados em AM para diagnosticar tumores a partir de um conjunto de categorias físicas de células extraídas de pacientes. Para isso usamos o conjunto de dados de câncer de mama de Wisconsin disponível em: https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic.

Dados médicos podem conter informações sensíveis e privadas que podem levar a identificação de pacientes. Este problema pode ser amenizado através da geração de dados sintéticos a serem utilizados em treinamento de modelos.

Comparamos os desempenhos dos modelos quando utilizados no dataset original e quando utilizado em dados sintéticos gerados a partir de diferentes arquiteturas de Generative Adversarial Networks.

Descrição dos dados:

O conjunto de dados utilizado possui 569 instâncias de células com 30 atributos cada, as instâncias podem ser de duas classes: Benigno (357 instâncias) ou Maligno (212 instâncias).

Todos os atributos são valores numéricos reais computados a partir de imagens das células, os valores são de raio, textura, perímetro, área, suavidade, compacidade, severidade das porções côncavas do contorno, número de porções côncavas no contorno, simetria e dimensão fractal. Para cada um desses existem 3 valores que foram calculados: o valor médio, o erro padrão e o pior ou maior do atributo.

Metodologia:

Para a implementação foi utilizada a linguagem Python com as bibliotecas sklearn, torch e numpy.

Os algoritmos utilizados para classificação foram: *K-nearest Neighbors (KNN)*, *Logistic Regression*, *Support Vector Machine (SVM)* e *Random Forest,* importados da biblioteca sklearn. Foram utilizadas Generative Adversarial Networks (GANs) para criação de dados sintéticos.

Utilizando K-Cross Fold Validation, os resultados dos algoritmos foram comparados quando treinados utilizando os dados de treino e quando treinados em dados sintéticos gerados a partir dos dados de treino.

Pré-Processamento

O dataset escolhido não possui dados duplicados ou faltantes, possui 569 instâncias, 30 atributos e possui proporção de 1.6 entre Benignos/Malignos. Portanto, não foram utilizadas técnicas de pré-processamento além de normalização dos dados.

Código:

```python
def kcv(data,target,k,print_info=False,randomize=False):
    random_state = randint(0,100) if randomize else 42
    data, target = shuffle(data,target, random_state=random_state)
    b_rows = []
    m_rows = []
    for i, row in enumerate(data):
        if target[i] == 1:
            # Add row as an element of b_rows
            b_rows += [row]
        else:
            m_rows += [row]

    proportion = len(b_rows)/len(m_rows)
    if (print_info):
      print(f'Benign to Malignant proportion: {proportion} ({len(b_rows)}/{len(m_rows)})')

    # Use a list comprehension to create new arrays with the added element
    b_rows = [np.append(arr, 0) for arr in b_rows]
    m_rows = [np.append(arr, 1) for arr in m_rows]

    b_splits = divide_list_into_k_parts(b_rows, k)
    m_splits = divide_list_into_k_parts(m_rows, k)

    folds = np.array(join_lists_of_lists(b_splits, m_splits), dtype=object)

    partitions = []

    for i in range(k):
        X_test = np.array([arr[:-1] for arr in folds[i]])
        y_test = np.array([arr[-1] for arr in folds[i]])

        X_train = np.concatenate([([arr[:-1] for arr in folds[j]]) for j in range(k) if j != i])
        y_train = np.concatenate([([arr[-1] for arr in folds[j]]) for j in range(k) if j != i])

        # Calculate the mean and standard deviation of each column
        mean = np.mean(X_train, axis=0)
        std_dev = np.std(X_train, axis=0)

        # Subtract the mean from each element and divide by the standard deviation
        training_set = (X_train - mean) / std_dev
        test_set = (X_test - mean) / std_dev

        if (print_info):
          print(f'Test fold {i + 1}: Instances for training: {len(training_set)}, Instances for testing: {len(test_set)})')
        partitions.append((training_set, y_train, X_test, y_test))
    return partitions
```

Recebe os atributos e a classificação e retorna o dataset dividido em uma lista de k elementos. A proporção original do dataset é mantida em cada elemento.
Cada k elemento possui 4 listas: X_train, y_train, X_test, y_test, que são os dados de treino/testes a serem utilizados.

```python
def join_lists_of_lists(list1, list2):
    if len(list1) != len(list2):
        raise ValueError("Input lists must have the same number of sublists")

    result = []
    for sublist1, sublist2 in zip(list1, list2):
        result.append(sublist1 + sublist2)

    return result


def divide_list_into_k_parts(input_list, k):
    n = len(input_list)
    avg = n // k
    remainder = n % k

    result = []
    start = 0

    for i in range(k):
        end = start + avg + (1 if i < remainder else 0)
        result.append(input_list[start:end])
        start = end
    return result
```

```python
class confusionMatrix:
    def __init__(self,test,pred,beta=1):
        self.VP,self.FP,self.VN, self.FN = 0,0,0,0
        self.size = len(pred)
        self.analysis(pred,test)
        self.accuracy = self.accuracy()
        self.precision = self.precision()
        self.recall = self.recall()
        self.npv = self.NPV()
        self.f_score = self.f_score(beta)

    def analysis(self,pred,test):
        for i in range(0,len(pred)):
          if(pred[i]==0 and test[i]==0):
            self.VN=self.VN+1
          if(pred[i]==1 and test[i]==0):
            self.FP=self.FP+1
          if(pred[i]==0 and test[i]==1):
            self.FN=self.FN+1
          if(pred[i]==1 and test[i]==1):
            self.VP=self.VP+1
    def accuracy(self):
        return (self.VP+self.VN)/self.size
    def precision(self):
        return (self.VP)/(self.VP + self.FP)
    def recall(self):
        return (self.VP)/(self.VP + self.FN)
    def NPV(self):
        return (self.VN)/(self.VN + self.FN)
    def f_score(self,beta):
        return (1+pow(beta,2))*(self.precision*self.recall)/(pow(beta,2)*self.precision + self.recall)
```

```python
from math import sqrt
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from confusionmatrix import confusionMatrix

def train(X_train, y_train, X_test, y_test):
  scaler = StandardScaler()
  X_test_scaled = scaler.fit_transform(X_test)
  X_train_scaled = scaler.fit_transform(X_train)

  # KNN
  knn_classifier = KNeighborsClassifier(n_neighbors=int(sqrt(len(X_train))))
  knn_classifier.fit(X_train_scaled, y_train)
  knn_predictions = knn_classifier.predict(X_test_scaled)
  knn = confusionMatrix(y_test, knn_predictions)

  # Logistic Regression
  logreg_classifier = LogisticRegression()
  logreg_classifier.fit(X_train_scaled, y_train)
  logreg_predictions = logreg_classifier.predict(X_test_scaled)
  logreg = confusionMatrix(y_test, logreg_predictions)

  # SVM
  svm_classifier = SVC(kernel='linear')
  svm_classifier.fit(X_train_scaled, y_train)
  svm_predictions = svm_classifier.predict(X_test_scaled)
  svm = confusionMatrix(y_test, svm_predictions)

  # Random Forest
  rf_classifier = RandomForestClassifier(n_estimators=100)
  rf_classifier.fit(X_train_scaled, y_train)
  rf_predictions = rf_classifier.predict(X_test_scaled)
  rf = confusionMatrix(y_test, rf_predictions)

  return [knn, logreg, svm, rf]
```

```python
folds = kcv(data.data, data.target,k_number)
for i,fold in enumerate(folds):
  X_train, y_train, X_test, y_test = fold
  results = train(X_train, y_train, X_test, y_test)
```

A função train() é uma função auxiliar que recebe dados de treino/teste e retorna uma confusionMatrix (classe definida anteriormente) para cada um dos modelos utilizados (knn, svm, logistic regression e random forest). Os resultados são guardados em uma lista e analisados posteriormente.

GAN:

```
for i,fold in enumerate(folds):
  …
  X_train, y_train, X_test, y_test = fold
  final_table =
create_table(X_train,y_true_indexes,y_false_indexes,hidden_size)
     train_results = train(final_table[:,:30], final_table[:,30],
X_test, y_test)
```

A função create_table gera um novo dataset da seguinte maneira:
Os dados são separados em B/M, e utilizados para treinar uma GAN em cada classe. Após o treinamento, cada GAN gera dados que são combinados respeitando a proporção original entre classes do dataset recebido.
Hiperparâmetros utilizados:

```
# GAN parameters
GANS = [
   [256,512],
   [256],
   [128,256],
   [128],
]
# Hyperparameters
input_size = 30 # n atributos
batch_size = 16
learning_rate = 0.0012
num_epochs = 5000
num_samples = 569 # gera mesma qtt de dados no dataset
```

Análise de Resultados:

Todos algoritmos tiveram um bom desempenho em todas as métricas, com resultados similares. Por ser uma tarefa de classificação médica, atribuímos maior importância às métricas *Negative Predictive Value* e *Recall*, onde os modelos Logistic Regression e SVM apresentaram melhor desempenho.

Resultados obtidos em uma execução com k = 5
Resultados e desvio padrão do treino no dataset original:

Attribute	knn	logreg	svm	randomforest
accuracy	0.961(0.012)	0.981(0.012)	0.979(0.013)	0.960(0.008)
precision	0.995(0.011)	0.991(0.013)	0.986(0.013)	0.957(0.030)
recall	0.901(0.032)	0.957(0.035)	0.957(0.031)	0.934(0.025)
f_score	0.945(0.018)	0.973(0.016)	0.971(0.019)	0.945(0.011)
npv	0.944(0.016)	0.976(0.020)	0.975(0.018)	0.962(0.014)

Resultados obtidos nos dados sintéticos gerados pelas seguintes arquiteturas:
[256,512], [256], [128, 256], [128]

Attribute	knn	logreg	svm	randomforest
accuracy	0.907(0.029)	0.917(0.026)	0.924(0.037)	0.870(0.033)
precision	0.967(0.023)	0.973(0.018)	0.965(0.041)	0.985(0.021)
recall	0.778(0.092)	0.802(0.087)	0.830(0.111)	0.660(0.078)
f_score	0.859(0.055)	0.876(0.048)	0.888(0.064)	0.789(0.061)
npv	0.884(0.040)	0.895(0.040)	0.909(0.051)	0.832(0.033)

Attribute	knn	logreg	svm	randomforest
accuracy	0.932(0.022)	0.958(0.015)	0.956(0.011)	0.895(0.032)
precision	0.988(0.016)	1.000(0.000)	0.990(0.014)	0.993(0.015)
recall	0.826(0.050)	0.887(0.039)	0.891(0.037)	0.722(0.083)
f_score	0.899(0.035)	0.940(0.022)	0.938(0.017)	0.834(0.056)
npv	0.906(0.025)	0.937(0.020)	0.939(0.019)	0.859(0.038)

Attribute	knn	logreg	svm	randomforest
accuracy	0.907(0.031)	0.907(0.029)	0.919(0.012)	0.899(0.055)
precision	0.977(0.037)	0.977(0.023)	0.984(0.024)	0.983(0.025)
recall	0.769(0.079)	0.768(0.081)	0.797(0.045)	0.744(0.155)
f_score	0.858(0.054)	0.858(0.051)	0.880(0.022)	0.839(0.102)
npv	0.879(0.035)	0.879(0.036)	0.892(0.020)	0.872(0.068)

Attribute	knn	logreg	svm	randomforest
accuracy	0.944(0.024)	0.954(0.013)	0.960(0.016)	0.926(0.020)
precision	0.994(0.012)	0.990(0.021)	0.971(0.009)	0.977(0.039)
recall	0.854(0.062)	0.887(0.042)	0.920(0.049)	0.826(0.094)
f_score	0.918(0.038)	0.935(0.020)	0.944(0.023)	0.891(0.035)
npv	0.921(0.031)	0.937(0.022)	0.954(0.027)	0.908(0.048)

Conclusão:
Para os dados originais foram obtidos desempenhos satisfatórios, mesmo sem a otimização de hiperparâmetros.
Não conseguimos atingir um desempenho similar para dados originais e sintéticos, havendo piora significativa nas métricas de interesse (Recall e NPV). A expectativa era que obtivéssemos resultados similares aos obtidos no artigo *Data Augmentation Using GANs,* onde houve um desempenho similar ou até superior nos dados sintéticos. Não tentamos reproduzir os resultados do artigo, apenas decidimos por aplicar o tema no trabalho.
Diferentemente do artigo, utilizamos kcv e o modelos mais fortes (no artigo foi utilizado uma árvore de decisão com acurácia reportada de 0.888). Ao invés de treinar uma GAN em todos os dados de treino e depois transformar a classe em 0 ou 1, optamos por treinar uma GAN para cada classe e adicionar a classe dos dados gerados baseado na GAN utilizada. Também optamos por gerar os dados respeitando a proporção original do conjunto recebido para treino.
Fizemos algumas modificações nos hiperparâmetros utilizados no artigo através de testes com holdout, pois o treino de redes neurais é computacionalmente exigente para ser repetido diversas vezes dentro do kcv. Optamos por aumentar o learning rate, o batch size e o número de epochs.

Referências:
Fabio Henrique Kiyoiti dos Santos Tanaka, & Claus Aranha. (2019). Data Augmentation Using GANs.
Varoquaux, G., Colliot, O. (2023). Evaluating Machine Learning Models and Their Diagnostic Value. In: Colliot, O. (eds) Machine Learning for Brain Disorders. Neuromethods, vol 197. Humana, New York, NY.
Wolberg,William, Mangasarian,Olvi, Street,Nick, and Street,W.. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository.
https://doi.org/10.24432/C5DW2B.