Renan Moreira - 1611061

Yan Gurevitz - 1610886

**T3 INF1771 Relatório:**

A base de dados utilizada foi a do Tic-Tac-Toe (“Jogo da velha”) que possuí nove atributos relativos às posições que podem ser ocupadas numa partida. Os atributos são nominais e são podem ter os valores de ‘x’, ‘o’ e ‘b’ que são as possíveis formas de ocupar as posições no jogo, sendo que o ‘b’ representa deixar em branco. A base de dados possui 958 instancias que representam todas as formas possíveis de se terminar uma partida e é um problema de classificação para definir se o jogador usando ‘x’, que começa, ganhou (“positive”) ou não (“negative”) a partida.

Decidimos fazer comparar os algoritmos de Árvores de Decisão e K-Nearest Neighbor (KNN)

**Árvores de Decisão:**

Decidimos implementar esse algoritmo em C++.

Para isso começamos realizando testes no Weka na árvore J48 nos quais chegamos à conclusão que o melhor era manter todos os atributos para uma classificação mais precisa, pois, qualquer atributo ou conjunto de atributos que fossem removidos causou uma diminuição no percentual de acerto.

Após isso realizamos diversos testes usando o modelo de dados. Os melhores foram:

**Uso de 66% dos dados como treinamento**

Time taken to build model: 0 seconds

Time taken to test model on test split: 0 seconds

Correctly Classified Instances 274 84.0491 %

**Cross-validation Folds: 12**

Time taken to build model: 0 seconds

Correctly Classified Instances 833 86.952 %

Com isso começamos a implementar a arvore de decisão em C++.

A arvore é criada a partir de uma função recursiva que analisa todos os dados fornecidos para treinamento e conta, para cada opção possível em cada atributo, a quantidade de resultados positivos e negativos. Com isso calculamos, através da entropia, qual é o melhor atributo para ficar no topo da arvore e em seguida chamamos novamente a função recursiva para cada opção de escolha desse atributo removendo do conjunto de opções as que não representam essa escolha.

Para separar o conjunto de testes e treinamento dos dados da arvore foi feita uma função que a partir de um percentual passado como parâmetro gera os dois arquivos separados com o conjunto de treinamento possuindo esse percentual do total de dados. Essa função usa rand() para escolher as instancias que irão para o conjunto de treinamento.

Após variar esse percentual realizando diversos testes o melhor resultado obtido foi usando 73% do total para o conjunto de treinamento, mas obtivemos um resultado não muito diferente com 34%. Os melhores resultados obtidos na arvore implementada também foram testados no Weka.

**Uso de 34% dos dados como treinamento - implementado**

Tempo gasto com treinamento: 0.03 s.

Tempo gasto com testes: 0.04 s.

Resultados Corretos: 542, de: 642

Percentual de acerto: 84.423676

**Uso de 34% dos dados como treinamento - Weka**

Time taken to build model: 0.02 seconds

Time taken to test model on supplied test set: 0.02 seconds

Correctly Classified Instances 518 80.6854 %

**Uso de 73% dos dados como treinamento - implementado**

Tempo gasto com treinamento: 0.08 s.

Tempo gasto com testes: 0.01 s.

Resultados Corretos: 229, de: 259

Percentual de acerto: 88.416992

**Uso de 73% dos dados como treinamento – Weka**

Time taken to build model: 0.01 seconds

Time taken to test model on supplied test set: 0 seconds

Correctly Classified Instances 225 86.8726 %

Além desse modo de separar implementamos um que pegaria metade de resultados positivos e metade de negativos, isso foi feito escolhendo de forma randômica e testando se ele seria positivo ou negativo. Caso ocorresse desse teste cair no errado mais de dez vezes consecutivas este seria aceito. Usando esse algoritmo nosso melhor resultado para os testes foi:

**Uso de 68% dos dados como treinamento - implementado**

Tempo gasto com treinamento: 0.07 s.

Tempo gasto com testes: 0.01 s.

Resultados Corretos: 271, de: 307

Percentual de acerto: 88.273613

**Uso de 68% dos dados como treinamento - Weka**

Time taken to build model: 0.01 seconds

Time taken to test model on supplied test set: 0 seconds

Correctly Classified Instances 268 87.2964 %

Vale ressaltar que para conjuntos de treinamento abaixo de 40% do total esse segundo método foi pior do que o primeiro. Acreditamos que isso se deu, pois os dados possuem uma quantidade significativamente maior de casos positivos do que de negativos e esse tipo de separação fez com que os exemplos não demonstrassem a realidade.

Também testamos usando todos os dados para o treinamento e para os testes:

**Usando todos os dados para o treinamento e para os testes - implementado**

Tempo gasto com treinamento: 0.07 s.

Tempo gasto com testes: 0.01 s.

Resultados Corretos: 271, de: 307

Percentual de acerto: 88.273613

**Usando todos os dados para o treinamento e para os testes - Weka**

Time taken to build model: 0 seconds

Time taken to test model on training data: 0 seconds

Correctly Classified Instances 898 93.737 %

**K-Nearest Neighbor (KNN):**