EP3 - Aprendizagem Supervisionada

MAC0425 - Inteligência Artificial

Renato Lui Geh
NUSP:8536030

1 Weka

Foi usada a ferramenta Weka para pré-processamento e classificação.

2 Pré-processamento

No EP dividimos as mensagens de SMS em várias categorias e três classes:

1. Type
   - `spam`
   - `ham`
2. Tokens
   - (“`buy`”, f_0)
   - (“`NEW`”, f_1)
   - (“`NOKIA`”, f_2)
   - (”`NOW`”, f_3)
   - ...
3. Rules
   - `length` - tamanho da mensagem.
   - `spec_char` - se é um caractere especial.
   - `is_cap` - se é uma letra maiúscula.
   - `digits` - se é um dígito (0, 9).

A classe Type classifica a mensagem como uma mensagem de spam ou de ham. Tokens é toda sequência de caracteres delimitada por “`.,;:'"()?!`”. Rules são características de cada caractere de cada mensagem.

O feature vector resultante fica então da seguinte forma:

$$V_i = [\text{type}, \text{length}, \text{spec\_char}, \text{is\_cap}, \text{digits}, t_0, t_1, ..., t_n] \tag{1}$$

Onde `type` $\in$ {`ham`,`spam`}, t_i é um Token e `length, spec_char, is_cap, digits` são números inteiros. Cada Token t_i possue um valor $t_i[0]$ que é uma `string` e representa uma sequência de caractere na mensagem e $t_i[1]$ que é o número de vezes que $t_i[0]$ ocorre em todas as mensagens. Cada V_i representa uma mensagem.

2.1 Weka Preprocess

No Weka foi usado o filtro `StringToWordVector` para gerar os Tokens. Foram escolhidos os argumentos `WordTokenizer` com delimitadores "`.,;:'"()?!`", `outputWordCounts=true` para que conte a frequência dos tokens ao invés de só presença e `useStoplist=true` para que use as stopwords.

3 Classificação

Foram usados os seguintes métodos de classificação:

1. NearestNeighbor (IBk)
   - Euclidean
   - Manhattan
   - Levenshtein
   - Chebyshev
2. DecisionTree
   - j48
     - LaPlace
     - $\neg$ LaPlace
   - BFTree (Best-first Decision Tree)
     - Gini
     - $\neg$ Gini
3. NaiveBayes
   - $\neg$ KernelEstimator $\wedge$ $\neg$ SupervisedDiscretization
   - KernelEstimator
   - SupervisedDiscretization

Para todos os testes foi utilizado 66% do conjunto de mensagens como treino e o resto como conjunto de teste.

3.1 NearestNeighbor

Os resultados da classificação com NearestNeighbor:

```
Nearest Neighbour
  @Euclidean
    -KNN 1
      correct = 95.1323 %
      mean abs error = 0.0489
      relative abs error = 21.167 %
    -KNN 5
      correct = 94.9735 %
      mean abs error = 0.0554
      relative abs error = 23.9808
  @Manhattan
    -KNN 1
      correct = 95.0794 %
      mean abs error =  0.0493
      relative abs error = 21.3168 %
    -KNN 5
      correct = 95.0794 %
      mean abs error = 0.0531
      relative abs error = 22.9949 %
  @Levenshtein
    -KNN 1
      correct = 95.0794 %
      mean abs error = 0.0493
      relative abs error = 21.3168 %
    -KNN 5
      #error
  @Chebyshev
    -KNN 1
      correct = 91.0053 %
      mean abs error = 0.185
      relative abs error = 80.0721 %
    -KNN 5
      correct = 87.3545 %
      mean abs error = 0.2259
      relative abs error = 97.7585%
```

Onde `KNN` é o número de vizinhos usados na média, `correct` é a porcentagem de acertos na classificação, `mean abs error` é a média de erro absoluto e `relative abs error` é o erro relativo.

Portanto, a melhor configuração de parâmetros é `KNN=1` e com algoritmo de distância Euclidiana.

3.2 DecisionTree

Os resultados da classificação com DecisionTree:

```
DecisionTree
  @j48 algorithm (w/ confidenceFactor = 0.25)
    -LaPlace=false
       correct = 98.2011 %
       mean abs error = 0.0287
       relative abs error = 12.3106 %
    -LaPlace=true
       correct = 98.2011 %
       mean abs error = 0.032
       relative abs error = 13.8544 %
```

```
@BFTree (Best-first Decision Tree)
  -Gini=true
     correct = 97.672 %
     mean abs error = 0.0425
     relative abs error = 18.4121 %
  -Gini=false
     correct = 97.5132 %
     mean abs error = 0.0427
     relative abs error = 18.4906 %
```

Portanto, a melhor configuração de parâmetros é uma `j48 tree` com LaPlace.

3.3 NaiveBayes

Os resultados da classificação com NaiveBayes:

```
NaiveBayes
  @Simple
     correct = 95.0265 %
     mean abs error = 0.0497
     relative abs error = 21.4892 %
  @KernelEstimator
     correct = 98.4127 %
     mean abs error = 0.0161
     relative abs error = 6.9828 %
  @SupervisedDiscretization
     correct = 98.4656 %
     mean abs error = 0.0174
     relative abs error = 7.5302 %
```

Portanto, a melhor configuração de parâmetros é uma `NaiveBayes` com `SupervisedDiscretization`. Para discretizar os valores, ao invés de usar a frequência de cada palavra, usou-se se a palavra ocorre ou não na mensagem (1 se ocorre e 0 se não ocorre). De forma similar discretizou-se também todas as regras para caracteres.

3.4 Cross-validation

A validação cruzada com todos os possíveis parâmetros discutidos acima demorou 2h11 (:O) para ser completada, com os parâmetros de Cross-Validation 10-folds no Weka em um Linux Mint 17.1 Rebecca com CPU Intel(R) Core i7-4500U @ 1.80 GHz e 16 GB de RAM.

Dataset	correct % ± StdDev
lazy.IBk '-K 5 -W 0 -A ËinearNNSearch -A ChebyshevDistance -R first-last	86.8777 %± 0.2356
lazy.IBk '-K 1 -W 0 -A ËinearNNSearch -A ChebyshevDistance -R first-last	91.8813 %± 0.7479
lazy.IBk '-K 5 -W 0 -A ËinearNNSearch -A ManhattanDistance -R first-last	93.2716 %± 0.7195
lazy.IBk '-K 5 -W 0 -A ËinearNNSearch -A EuclideanDistance -R first-last	93.5522 %± 0.7072
lazy.IBk '-K 1 -W 0 -X -A ËinearNNSearch -A EuclideanDistance -R first-last	95.9730 %± 0.6780
lazy.IBk '-K 1 -W 0 -A ËinearNNSearch -A ManhattanDistance -R first-last	95.9802 %± 0.6614
bayes.NaiveBayes	96.9982 %± 0.7774
trees.BFTree '-S 1 -M 2 -N 5 -C 1.0 -P PREPRUNED'	97.4838 %± 0.5662
trees.BFTree '-S 1 -M 2 -N 5 -G -C 1.0 -P PREPRUNED'	97.5647 %± 0.6164
trees.J48 '-C 0.25 -M 2'	98.1079 %± 0.5173
trees.J48 '-C 0.25 -M 2 -A'	98.1079 %± 0.5173
bayes.NaiveBayes -D	98.1313 %± 0.5213
bayes.NaiveBayes -K	98.5612 %± 0.4494
Average	95.5762 %

3.5 Respostas para validação

1. Pode-se ver que o melhor classificador no conjunto de validação é a Naive Bayes com Kernel Estimator, já que tem maior porcentagem de acerto com um baixo desvio padrão.

2. Quando feito apenas com 66% do conjunto como treino e o restante como teste, é possível que tenhamos tido sorte com os testes e portanto o resultado fica viesado. Com a validação cruzada testamos vários testes com diferentes treinos, o que torna o resultado menos viesado (ainda que com um tanto de viés).

3. A Naive Bayes com Kernel Estimator teve melhores resultados pois usa as propriedades de dependência de uma Rede Bayesiana. Por isso, podemos achar probabilidades que não acharíamos por exemplo no Nearest Neighbour, já que este apenas vê seus vizinhos mais próximos e não acharia relações que poderiam tornar a classificação melhor.

4. Comparações nunca vão ser completamente confiáveis, já que nunca poderemos usar um conjunto infinito de treino para termos 100% de confiança que os resultados são corretos. No entanto, para melhorarmos a confiança das comparações podemos sempre usar os mesmos conjuntos de treinos e testes em diferentes classificadores, tendo então comparações menos viesadas.

4 Algumas árvores :)

O Weka, em modo debug, imprime as árvores de decisão usadas. Visualizando as árvores usadas no método `j48tree`, `BFTree` com corte e `BFTree` sem corte, pode-se ver que apesar da árvore `j48` ter se saído melhor na classificação vista na seção anterior, a árvore gerada é maior que as geradas pela `BFTree` (tanto com corte e sem corte).

```
digits <= 4
|   mobile <= 0
|   |   reply <= 0
|   |   |   Your <= 0
|   |   |   |   digits <= 1
|   |   |   |   |   Dear <= 0: ham (4324.0/27.0)
|   |   |   |   |   Dear > 0
|   |   |   |   |   |   co <= 0: ham (30.0)
|   |   |   |   |   |   co > 0: spam (3.0)
|   |   |   |   digits > 1
|   |   |   |   |   your <= 0
|   |   |   |   |   |   with <= 0
|   |   |   |   |   |   |   is_cap <= 10: ham (290.0/5.0)
|   |   |   |   |   |   |   is_cap > 10
|   |   |   |   |   |   |   |   Reply <= 0
|   |   |   |   |   |   |   |   |   To <= 0: ham (54.0/6.0)
|   |   |   |   |   |   |   |   |   To > 0: spam (3.0)
|   |   |   |   |   |   |   |   Reply > 0: spam (3.0)
|   |   |   |   |   |   with > 0
|   |   |   |   |   |   |   about <= 0
|   |   |   |   |   |   |   |   is_cap <= 9: ham (13.0)
|   |   |   |   |   |   |   |   is_cap > 9: spam (3.0)
|   |   |   |   |   |   |   about > 0: spam (3.0)
|   |   |   |   |   your > 0
|   |   |   |   |   |   spec_char <= 6: ham (20.0/1.0)
|   |   |   |   |   |   spec_char > 6: spam (12.0/1.0)
|   |   |   Your > 0
|   |   |   |   digits <= 1: ham (26.0/2.0)
|   |   |   |   digits > 1: spam (4.0)
|   |   reply > 0
|   |   |   is_cap <= 9: ham (24.0)
|   |   |   is_cap > 9: spam (11.0/1.0)
|   mobile > 0
|   |   your <= 0
|   |   |   digits <= 2: ham (13.0)
|   |   |   digits > 2: spam (2.0)
|   |   your > 0: spam (13.0)
digits > 4
|   3 <= 0
|   |   digits <= 10
|   |   |   good <= 0
|   |   |   |   me <= 0
|   |   |   |   |   I <= 0
|   |   |   |   |   |   is_cap <= 4
|   |   |   |   |   |   |   your <= 0
|   |   |   |   |   |   |   |   service <= 0
|   |   |   |   |   |   |   |   |   is_cap <= 1
```

```
|   |   |   |   |   |   |   |   |   |   length <= 34: spam (2.0)
|   |   |   |   |   |   |   |   |   |   length > 34: ham (3.0/1.0)
|   |   |   |   |   |   |   |   |   is_cap > 1: ham (9.0)
|   |   |   |   |   |   |   |   service > 0: spam (2.0)
|   |   |   |   |   |   |   your > 0: spam (5.0)
|   |   |   |   |   |   is_cap > 4: spam (78.0/4.0)
|   |   |   |   |   I > 0: ham (6.0/1.0)
|   |   |   |   me > 0: ham (6.0)
|   |   |   good > 0: ham (7.0)
|   |   digits > 10: spam (555.0/5.0)
|   3 > 0
|   |   or <= 0
|   |   |   2 <= 0
|   |   |   |   so <= 0: spam (8.0)
|   |   |   |   so > 0: ham (3.0)
|   |   |   2 > 0: ham (17.0)
|   |   or > 0: spam (8.0)
```

Listing 1: Uma `j48tree` onde os nós decisões são as regras ou tokens mencionados na seção de pré-processamento e as folhas são as classificações.

```
digits < 4.5
|  FREE < 0.5
|  |  co < 0.5
|  |  |  STOP < 0.5
|  |  |  |  http < 0.5
|  |  |  |  |  mobile < 0.5
|  |  |  |  |  |  Reply < 0.5
|  |  |  |  |  |  |  5 < 1.5
|  |  |  |  |  |  |  |  Box < 0.5
|  |  |  |  |  |  |  |  |  ringtone < 0.5
|  |  |  |  |  |  |  |  |  |  £1 < 0.5
|  |  |  |  |  |  |  |  |  |  |  is_cap < 7.5: ham(4386.0/18.0)
|  |  |  |  |  |  |  |  |  |  |  is_cap >= 7.5
|  |  |  |  |  |  |  |  |  |  |  |  e < 0.5
|  |  |  |  |  |  |  |  |  |  |  |  |  order < 0.5
|  |  |  |  |  |  |  |  |  |  |  |  |  |  new < 0.5
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  since < 0.5: ham(346.0/14.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  since >= 0.5: spam(2.0/1.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  |  new >= 0.5: spam(2.0/1.0)
|  |  |  |  |  |  |  |  |  |  |  |  |  order >= 0.5: spam(2.0/0.0)
|  |  |  |  |  |  |  |  |  |  |  |  e >= 0.5: spam(3.0/0.0)
|  |  |  |  |  |  |  |  |  |  |  £1 >= 0.5: spam(2.0/0.0)
|  |  |  |  |  |  |  |  |  |  ringtone >= 0.5: spam(2.0/0.0)
|  |  |  |  |  |  |  |  |  Box >= 0.5: spam(2.0/0.0)
|  |  |  |  |  |  |  |  5 >= 1.5: spam(2.0/0.0)
|  |  |  |  |  |  |  Reply >= 0.5
|  |  |  |  |  |  |  |  with < 0.5: spam(6.0/1.0)
|  |  |  |  |  |  |  |  with >= 0.5: ham(5.0/0.0)
|  |  |  |  |  |  mobile >= 0.5
|  |  |  |  |  |  |  your < 0.5
|  |  |  |  |  |  |  |  is_cap < 11.5: ham(13.0/0.0)
|  |  |  |  |  |  |  |  is_cap >= 11.5: spam(2.0/0.0)
|  |  |  |  |  |  |  your >= 0.5: spam(8.0/0.0)
|  |  |  |  |  http >= 0.5: spam(5.0/0.0)
|  |  |  |  STOP >= 0.5: spam(6.0/1.0)
|  |  |  co >= 0.5: spam(9.0/0.0)
|  FREE >= 0.5: spam(11.0/1.0)
digits >= 4.5
|  7 < 0.5
|  |  digits < 9.5
|  |  |  is_cap < 8.5
|  |  |  |  spec_char < 11.5
|  |  |  |  |  is_cap < 5.5
|  |  |  |  |  |  your < 0.5
|  |  |  |  |  |  |  length < 31.5: spam(2.0/0.0)
|  |  |  |  |  |  |  length >= 31.5
|  |  |  |  |  |  |  |  service < 0.5: ham(16.0/2.0)
|  |  |  |  |  |  |  |  service >= 0.5: spam(2.0/0.0)
|  |  |  |  |  |  your >= 0.5: spam(4.0/0.0)
|  |  |  |  |  is_cap >= 5.5: spam(10.0/0.0)
|  |  |  |  spec_char >= 11.5: ham(12.0/0.0)
|  |  |  is_cap >= 8.5
|  |  |  |  is_cap < 49.0: spam(55.0/2.0)
|  |  |  |  is_cap >= 49.0: ham(3.0/1.0)
|  |  digits >= 9.5
|  |  |  one < 0.5
|  |  |  |  spec_char < 26.5: spam(571.0/2.0)
|  |  |  |  spec_char >= 26.5: ham(2.0/0.0)
|  |  |  one >= 0.5: ham(3.0/0.0)
|  7 >= 0.5
|  |  digits < 15.0: ham(18.0/1.0)
|  |  digits >= 15.0: spam(3.0/0.0)
```

Listing 2: Uma `BFTree` sem corte. Esta árvore é um pouco menor que a `j48`, mas sua classificação gerou piores resultados.

```
digits < 4.5
|  FREE < 0.5
|  |  co < 0.5
|  |  |  STOP < 0.5
|  |  |  |  http < 0.5: ham(4753.0/65.0)
|  |  |  |  http >= 0.5: spam(5.0/0.0)
|  |  |  STOP >= 0.5: spam(6.0/1.0)
|  |  co >= 0.5: spam(9.0/0.0)
|  FREE >= 0.5: spam(11.0/1.0)
digits >= 4.5
|  7 < 0.5
|  |  digits < 9.5
|  |  |  is_cap < 8.5
|  |  |  |  spec_char < 11.5
|  |  |  |  |  is_cap < 5.5
|  |  |  |  |  |  your < 0.5
|  |  |  |  |  |  |  length < 31.5: spam(2.0/0.0)
|  |  |  |  |  |  |  length >= 31.5
|  |  |  |  |  |  |  |  service < 0.5: ham(16.0/2.0)
|  |  |  |  |  |  |  |  service >= 0.5: spam(2.0/0.0)
|  |  |  |  |  |  your >= 0.5: spam(4.0/0.0)
|  |  |  |  |  is_cap >= 5.5: spam(10.0/0.0)
|  |  |  |  spec_char >= 11.5: ham(12.0/0.0)
|  |  |  is_cap >= 8.5
|  |  |  |  is_cap < 49.0: spam(55.0/2.0)
|  |  |  |  is_cap >= 49.0: ham(3.0/1.0)
|  |  digits >= 9.5
|  |  |  one < 0.5
|  |  |  |  spec_char < 26.5: spam(571.0/2.0)
|  |  |  |  spec_char >= 26.5: ham(2.0/0.0)
|  |  |  one >= 0.5: ham(3.0/0.0)
|  7 >= 0.5
|  |  digits < 15.0: ham(18.0/1.0)
|  |  digits >= 15.0: spam(3.0/0.0)
```

Listing 3: Uma `BFTree` com pre-corte. Esta árvore é muito menor que as outras e ainda teve melhores resultados que a mesma árvore sem corte.

5 Conclusões