

Montage d'un classificateur sur les performances de différents algorithmes de compression de données

Session 1 : semaine du 13 au 17 août 2007.

Le problème

On dispose d'un tableau mesurant différentes performances pour trois algorithmes à comparer. En outre, chaque algorithme est caractérisé par différents paramètres, dont les valeurs sont comprises entre 0 et 1.

Id	Nom_methode	performance	att1	att2	att3	att4	att5	att6
1	Meth1	0,1						
2	Meth2	0,3						
3	Meth3	0,45						

Nous appellerons « Méthode 1 », « Méthode 2 » et « Méthode 3 » les noms des différents algorithmes.

Ne disposant pas des valeurs exactes, un tableau sera généré aléatoirement en tenant compte :

- D'une variable polynomiale « Nom_methode » à trois instances
- D'une variable numérique « Taux_de_Compression » qui sera la performance. Cette variable pourra jouer plusieurs rôles
- De six attributs numériques jouant le rôle de paramètres pour chaque méthode.

L'analyse du problème

Le besoin

Mon interlocuteur souhaitait pouvoir disposer d'une méthode formelle permettant de choisir entre ces trois algorithmes. La méthode se présenterait sous forme « si paramètre 1 prend valeur X=> choix algo Y » et prendrait en compte la performance pour orienter la décision.

Mon interlocuteur souhaitait a priori un arbre de classification ou un jeu de règles de décision. Il souhaitait pouvoir mettre en concurrence différents algorithmes de classification pour comparer les résultats, et voulait une analyse discriminante à but descriptif pour comprendre l'influence de chacun des paramètres sur la performance de classification.

La faiblesse du nombre de données

C'est une problématique que l'on retrouve en génétique, en text mining, ainsi qu'en diagnostic médical : le ratio entre le nombre d'attributs et le nombre d'individus est « grand », joint à un faible nombre d'individus, ce qui compromet un apprentissage efficace.

« grand » signifie certaines valeurs qui dépendent du domaine considéré : Le cas du diagnostic médical revient à prendre une décision « malade : oui/non » sur une dizaine de patients atteints de maladie rare, avec une quarantaine d'attributs, voire plus.

Ici, ne disposant pas de plus de trois individus (algorithmes), deux risques se présentent :

- classificateur mal entraîné : le classificateur ne prédit aucune valeur juste pour aucun cas de figure
- classificateur sur-entraîné : le classificateur est « boosté » par des méta-schémas, le menant à des performances maximales sur les trois exemples. Lorsqu'un nouvel algorithme, ou une nouvelle palette de mesure sur un des algorithmes, est présenté au classificateur, la décision est catastrophique.

L'analyse des attributs

Je n'envisage donc qu'une seule solution :

- Bien évidemment privilégier l'analyse des attributs, donc effectuer l'analyse discriminante AVANT la classification. L'idée étant que nous avons peu d'attributs, ils contiennent une information utile et du bruit, qu'il faut absolument extraire et séparer au mieux.
- Aggréger de nouveaux attributs comme fonctions non-linéaires des attributs existants. Vu le domaine des télécommunications, je privilégierai les fonctions « + », « - », « sin », « exp » et « log »; la dernière fonction revient à effectuer une analyse log-linéaire.
- Donc après avoir agrégé des attributs aux six existants (famille F0), j'obtiendrai plusieurs dizaines de nouveaux attributs qui, aggrégés à F0 donneront la famille F1. De cette famille, je vais extraire soit les composantes principales soit toute autre composante maximisant l'inter-entropie. Ces composantes formeront la famille F2.
- Puis j'aggrèrerai F1 et F2 pour avoir F3
- Il me reste à sélectionner les attributs les plus significatifs au vu de la classification.

L'analyse discriminante

Deux problèmes se posent pour la classification :

- Quelle est la variable expliquée : Le nom de la méthode, ou la valeur de la performance ? Si c'est le Nom_methode, c'est une classification « multi-label » sachant que 60% des classificateurs sont « bi-label » (utilisation d'un méta-schéma de multi-labellisation). Si c'est la performance (attribut numérique), c'est une classification que l'on appelle « régression » (pas forcément linéaire...)
- Par ailleurs, si je veux utiliser un classificateur pour discriminer les attributs, c'est le serpent qui se mord la queue : Pour choisir les bons attributs, il me faut un classificateur, et pour avoir le bon classificateur, il me faut les attributs

L'idée que j'ai eue est la suivante :

La variable expliquée ne sera pas la même pour l'analyse discriminante des attributs et pour le montage du classificateur.

Pour l'analyse discriminante, j'utilise un *empilement concurrentiel* de classificateurs à régression, et la variable expliquée sera « Taux_de_Compression ». Je mettrai donc en lumière les variables ayant le plus d'influence sur la performance, et je m'en servirai pour classer les labels de « Nom_Methode ».

Après divers tâtonnements, je me suis aperçu que RapidMiner dispose d'une série de classificateurs

polynominaux qui acceptent une variable « individus pondérés », soit « weighted examples ». Ce sont soit des arbres, soit des jeux de règles : Cela tombe très bien !! Dans le flux de calcul, j'ai pris l'algo « DecisionStump », mais cela aurait pu être un des autres algorithmes.

Le flux de calcul

```
<operator name="Root" class="Process">
  <operator name="CreationExemple" class="OperatorChain">
    <operator name="ExampleSetGenerator" class="ExampleSetGenerator"
breakpoints="after">
      <parameter key="attributes_lower_bound" value="0.0"/>
      <parameter key="attributes_upper_bound" value="1.0"/>
      <parameter key="number_examples" value="3"/>
      <parameter key="number_of_attributes" value="6"/>
      <parameter key="target_function" value="random"/>
    </operator>
    <operator name="ChangeAttributeName" class="ChangeAttributeName">
      <parameter key="new_name" value="Taux_de_Compression"/>
      <parameter key="old_name" value="label"/>
    </operator>
    <operator name="IdTagging" class="IdTagging">
      <parameter key="create_nominal_ids" value="true"/>
    </operator>
    <operator name="ChangeAttributeName (2)" class="ChangeAttributeName"
breakpoints="after">
      <parameter key="new_name" value="Nom_methode"/>
      <parameter key="old_name" value="id"/>
    </operator>
  </operator>
  <operator name="ApprocheFilter" class="OperatorChain">
    <operator name="CompleteFeatureGeneration"
class="CompleteFeatureGeneration" breakpoints="after">
      <parameter key="use_diff" value="true"/>
      <parameter key="use_exp" value="true"/>
      <parameter key="use_log" value="true"/>
      <parameter key="use_plus" value="true"/>
      <parameter key="use_sin" value="true"/>
    </operator>
    <operator name="FeatureNameFilter" class="FeatureNameFilter"
activated="no">
      <parameter key="skip_features_with_name" value="att1"/>
    </operator>
    <operator name="PCA" class="PCA">
      <parameter key="dimensionality_reduction" value="fixed number"/>
      <parameter key="number_of_components" value="5"/>
    </operator>
    <operator name="IOMultiplier" class="IOMultiplier">
      <parameter key="io_object" value="ExampleSet"/>
    </operator>
    <operator name="ModelApplier" class="ModelApplier" breakpoints="after">
      <list key="application_parameters">
      </list>
      <parameter key="keep_model" value="true"/>
    </operator>
    <operator name="ExampleSetJoin" class="ExampleSetJoin"
breakpoints="after">
      <parameter key="remove_double_attributes" value="false"/>
    </operator>
    <operator name="FastICA" class="FastICA">
      <parameter key="number_of_components" value="40"/>
    </operator>
```

```

        <operator name="ModelApplier (4)" class="ModelApplier"
breakpoints="after">
        <list key="application_parameters">
        </list>
        <parameter key="keep_model" value="true"/>
        </operator>
    </operator>
    <operator name="AnaDiscriDescriptive" class="OperatorChain">
        <operator name="FeatureSelection" class="FeatureSelection"
breakpoints="after">
        <operator name="XValidation (2)" class="XValidation">
            <parameter key="create_complete_model" value="true"/>
            <parameter key="keep_example_set" value="true"/>
            <parameter key="leave_one_out" value="true"/>
            <parameter key="sampling_type" value="linear sampling"/>
            <operator name="train (2)" class="OperatorChain">
                <operator name="Stacking (2)" class="Stacking">
                    <parameter key="keep_example_set" value="true"/>
                    <operator name="JMySVMLEARNER (2)"
class="JMySVMLEARNER">
                        <parameter key="keep_example_set" value="true"/>
                        <parameter key="kernel_type" value="radial"/>
                    </operator>
                    <operator name="NearestNeighbors (2)"
class="NearestNeighbors">
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                    <operator name="NeuralNet (2)" class="NeuralNet">
                        <parameter key="default_number_of_hidden_layers"
value="4"/>
                        <list key="hidden_layer_types">
                        </list>
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                    <operator name="W-DecisionTable (2)" class="W-
DecisionTable">
                        <parameter key="R" value="true"/>
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                    <operator name="W-M5P (2)" class="W-M5P">
                        <parameter key="L" value="true"/>
                        <parameter key="R" value="true"/>
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                    <operator name="W-DecisionStump (2)" class="W-
DecisionStump">
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                    <operator name="W-GaussianProcesses (2)" class="W-
GaussianProcesses">
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                    <operator name="W-IsotonicRegression (2)" class="W-
IsotonicRegression">
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                    <operator name="W-KStar (2)" class="W-KStar">
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                    <operator name="W-LWL (2)" class="W-LWL">
                        <parameter key="keep_example_set" value="true"/>
                    </operator>
                </operator>
            </operator>
        </operator>
    </operator>

```

```

        <operator name="test (2)" class="OperatorChain">
            <operator name="ModelApplier (2)" class="ModelApplier">
                <list key="application_parameters">
                </list>
                <parameter key="keep_model" value="true"/>
            </operator>
            <operator name="Performance (2)" class="Performance">
                <parameter key="keep_example_set" value="true"/>
            </operator>
        </operator>
    </operator>
    <operator name="AttributeWeightsApplier" class="AttributeWeightsApplier"
activated="no">
    </operator>
</operator>
    <operator name="ClassifCout" class="OperatorChain">
        <operator name="ChangementLabel" class="OperatorChain"
breakpoints="after">
            <operator name="Label2Cost" class="ChangeAttributeType">
                <parameter key="name" value="Taux_de_Compression"/>
                <parameter key="target_type" value="weight"/>
            </operator>
            <operator name="Id2Label" class="ChangeAttributeType">
                <parameter key="name" value="Nom_methode"/>
                <parameter key="target_type" value="label"/>
            </operator>
        </operator>
    </operator>
    <operator name="OperatorChain" class="OperatorChain">
        <operator name="XValidation (3)" class="XValidation">
            <parameter key="create_complete_model" value="true"/>
            <parameter key="keep_example_set" value="true"/>
            <operator name="DecisionTree" class="DecisionTree">
                <parameter key="keep_example_set" value="true"/>
            </operator>
            <operator name="OperatorChain (2)" class="OperatorChain">
                <operator name="ModelApplier (3)" class="ModelApplier">
                    <list key="application_parameters">
                    </list>
                    <parameter key="keep_model" value="true"/>
                </operator>
                <operator name="Performance (3)" class="Performance">
                    <parameter key="keep_example_set" value="true"/>
                </operator>
            </operator>
        </operator>
    </operator>
</operator>

```

En faisant « copier-coller » de ce fichier dans l'onglet « XML » de RapidMiner, on peut alors visualiser le flux tel que je l'ai généré chez moi.

Les résultats

J'ai transmis le flux à mon interlocuteur, qui me répondra incessamment sous peu. En revanche, j'ai pu faire quelques observations à mon niveau. Je les liste ci-dessous :

Pas de données réelles

La détermination des attributs et des méthodes de sélection/réduction ne peut se faire qu'au vu de la physionomie des données réelles. C'est l'une des raisons pour lesquelles un projet de KDD ne peut être qu'empirique, comme je le mentionne sur le site.

En l'occurrence, la première « boîte » du flux génère un exemple aux valeurs aléatoires. Or l'histogramme des valeurs réalisées et des valeurs possibles pour chaque attribut en sera affecté : Une approche « filtering » doit toujours être accompagnée d'une visualisation des données.

C'est pourquoi je ne suis pas catégorique du tout quant à mon choix d'aggrégation/réduction...

Résultats limités en classification dans le wrapper

Après avoir empilé une petite dizaine de classificateurs dans la boîte d'analyse discriminante (empilement « stacking » et wrapper « FS » méthode « forward selection »), j'ai bien entendu obtenu une sélection d'une dizaine d'attributs sur la famille F3.

J'ai également vérifié la performance du classificateur hybride résultant de l'empilement : l'erreur RMS est de 30% environ. Un résultat prédit sur trois est faux concernant la régression, et nous avons trois exemples...

Si l'on considère que les valeurs initiales de F0 ont été générées au hasard, c'est plus que raisonnable !

Forcer le nombre de composantes principales

Avec mon tableau généré aléatoirement, je disposais de 2 composantes principales lorsque je fixais le seuil de variance résiduelle à 90%. J'ai forcé le nombre de composantes à 5, sachant qu'il faut éviter de dépasser le nombre initial d'attributs, sous peine d'introduire de l'information parasite liée à l'échantillonnage des données.

(F2) : 5 composantes

Après avoir aggloméré F2 et F1, je réduisis F3 par « ICA » (maximisation de l'inter-entropie comme pour le problème de la « cocktail party » où des micros dispersés écoutent l'ambiance sonore sans connaître le nombre d'interlocuteurs ni aucun a priori sur la manière dont ils parlent), qui permet de séparer les sources en aveugle. J'avais spécifié que ICA me sorte 40 composantes, par rapport aux centaines de F3.

Ces 40 composantes seraient sélectionnées par le wrapper au vu de la régression sur la performance.

Le comportement du classificateur pondéré

J'ai rapidement pu constater que le classificateur avait tout simplement tendance à ignorer l'algorithme ayant la plus mauvaise performance : Logique, si les valeurs des paramètres sont

aléatoires, autant ignorer la performance la plus faible...

Remarques a posteriori

Concernant l'aggrégation, j'aurais pu effectuer une aggrégation sur les attributs tels quels d'une part, mais aussi sur un clone de la table de valeurs que j'aurais normalisé auparavant. En effet, l'exemple généré est uniforme quant à la plage de valeurs des attributs, toutes comprises entre 0 et 1, mais rien n'indique que les données réelles aient les mêmes plages d'un attribut à l'autre...

J'aurais donc obtenu deux tables, une sur les valeurs originales munie de ses fonctions d'aggrégation, l'autre sur les colonnes normalisées munie des mêmes fonctions mais à valeurs différentes et j'aurais aggrégé les deux tables avant de passer à la réduction par ACP...