

TP 2

Clustering avec WEKA

Dans la suite des TP, nous allons utiliser le logiciel WEKA pour effectuer différentes tâches de datamining : clustering, classification et règles d'association. Les bases de données nécessaires au TP sont sur la page suivante : <https://github.com/renatopp/arff-datasets/tree/master/>

Prise en main de WEKA

Lancer WEKA avec la commande.

Une fenêtre s'ouvre, cliquer sur « Explorer ». Une nouvelle fenêtre s'affiche avec plusieurs onglets. L'onglet « Preprocess » permet de gérer les données utilisées. On peut alors charger une base de données et y appliquer un filtre si nécessaire.

Ouvrir la base de donnée iris.arff utilisée au TP précédent. Vous pouvez alors visualiser la distribution des différents attributs. Ces attributs permettent-ils de prédire la classe ?

Cliquer sur l'onglet « Visualize ».

- Quels premiers résultats pouvez-vous tirer des différents graphiques ?
- Que remarquez-vous sur les graphiques dessinés en fonction de la classe ? Certains attributs sont-ils discriminants ?

Cliquer sur l'onglet « Cluster ». Vous pouvez alors choisir la méthode de clustering. Choisissez pour commencer la méthode K-means. En cliquant sur la barre « K-means » vous accéder alors aux paramètres de la méthode.

- Quels sont-ils ? Expliquer leur rôle.

Iris ayant 3 classes, nous allons commencer par un K-means avec K=3. Exécuter l'algorithme en cliquant sur « Start » :

- Quel résultat obtenez-vous ? Semble-t-il correct ?
- Quels attributs sont utilisés par l'algorithme pour effectuer le clustering ? Que remarquez-vous ?

En effet, dans cette base de données, la classe apparaît, il faut donc ignorer cet attribut pour effectuer un clustering cohérent. Relancer l'algorithme plusieurs fois :

- Quels sont les clustering obtenus pour chacune des exécutions ? Qu'observez-vous ?

Ici, vous devez voir l'effet du paramètre *seed* sur l'exécution de l'algorithme. En effet, celui-ci peut modifier le résultat de l'algorithme quand une part d'aléatoire est présente.

- Où trouve-t-on un choix aléatoire dans cet algorithme ?

Il est possible de visualiser le résultat du clustering. Faire un clic droit sur la ligne résultat et choisir « Vizualize cluster assignment ». Une fenêtre s'ouvre en faisant apparaître les différents clusters.

Maintenant, nous allons utiliser l'information de la classe pour connaître les instances mal classées. Pour cela, cliquer sur « Classes to clusters evaluation » et choisir l'attribut classe. Relancer l'algorithme et visualiser le résultat.

L'utilisation d'un algorithme présentant autant de paramètres doit être précédée d'une phase d'analyse sur les performances de l'algorithme (ici le taux de bonne classification) en fonction des paramètres définis pour une instance donnée.

Le protocole expérimental est le suivant : étudier la performance moyenne sur 5 exécutions (donc 5 *seed* différentes) de l'algorithme K-means par rapport à la distance pour identifier les clusters (distance euclidienne ou distance de Manhattan) pour K=3 (pour cette instance, le nombre de classes identifiées est de 3).

	Seed=1	Seed=2	Seed=3	Seed=4	Seed=5	Performance moyenne
Distance euclidienne						
Distance de Manhattan						

Faites varier les différents paramètres de l'algorithme comme précisé dans le tableau cidessus. Calculer la performance moyenne.

- Quelle distance conseillez-vous d'utiliser pour cette instance ?
- Que proposez-vous pour réduire le taux d'erreur de classement ?

D'autres algorithmes de clustering sont disponibles :

- Quel est le principe Hierarchical clusterer ? Quels sont ses paramètres dans WEKA ?
- Comparer leurs performances en faisant varier le nombre de clusters $N = \{2, 3, 4\}$. N'oubliez pas qu'il est important d'exécuter plusieurs fois l'algorithme.
- Quel algorithme conseillez-vous d'utiliser ?

Dans la suite du TP, vous allez travailler sur différentes bases de données.

Travail demandé

Etudier les bases de données :

vote.arff, weather.numeric.arff, weather.nominal.arff

Description de chaque base de données :

- Nombres d'instances ? d'attributs ? de classes ?
- Types de données ?

Paramètres

1) Que remarque-t-on en faisant varier la seed ? le nombre de clusters ? la méthode de calcul de la distance entre les instances ?

Pour chaque base, quel est le meilleur paramétrage selon vous ?

Interprétation des résultats

2) Comment interprétez-vous le centre des clusters ? Quelles informations donne-t-il ?

3) Comment est calculé « Incorrectly clustered instances » ?

4) Comment classer une nouvelle instance ?

TP 3

Classification supervisée -- Arbres de décisions

Les bases de données nécessaires au TP sont sur la page suivante :

<https://github.com/renatopp/arff-datasets/tree/master/classification>

Algorithme des k-voisins "à la main"

Supposons que l'on a un problème de classification qui consiste à déterminer la classe d'appartenance de nouvelles instances X_i . Le domaine de valeurs des classes est : {1,2,3}.

Selon la base de connaissance suivante, déterminez la classe de l'instance X6, dont les valeurs pour les attributs A1 à A5 (numériques) sont <3, 12, 4, 7, 8>, à l'aide de l'algorithme des k-voisins les plus proches (K-NN).

Instances	A1	A2	A3	A4	A5	Classe
X1	3	5	4	7	1	1
X2	4	6	10	3	2	1
X3	8	3	4	2	6	2
X4	2	1	4	3	6	2
X5	2	5	1	4	8	3

- Quelle distance choisissez-vous ?
- Faites varier K dans {1,2,3}. Quelle est l'influence du paramètre K sur la classification ?

Classification avec WEKA

Bases de données utilisées : « iris » -- « vote »

Algorithmes de classification : onglet « Classify » de l'« Explorer ».

- K-NN : Choose > lazy > IBk
- [arbre de décision] C4.5 (amélioration de ID3) : Choose > tree > J48

Ce TP a pour but de tester ces algorithmes sur deux bases de données et d'appréhender la difficulté liée au choix d'une méthode de classification et de son paramétrage.

Chargez la base de donnée « iris » et choisissez la méthode K-NN de Weka.

- Quelle est la valeur prédéfinie pour K ?
- Quel est l'impact sur la méthode de classification ?
- A quoi correspond le paramètre *crossValidate* ?

Apprentissage et validation des données : 4 méthodes disponibles

« Use training set » -- « Supplied test set » -- « Cross-validation » -- « Percentage split »

- Expliquez leur principe.

Lancez l'algorithme avec chacune de ces méthodes.

- Quelles informations sont données en résultats ?
- Donnez la signification de la valeur « Total number of instances »
- A quoi correspond la matrice de confusion ?

Bases de données et prétraitement :

Ouvrez la base de donnée « iris » dans un éditeur de texte, que remarquez-vous ? Choisissez dans Weka, comme « Test options », l'option « Percentage split » avec 66% de taux d'entraînement. Lancez l'algorithme.

- Quel taux d'erreur de classement obtenez-vous ?

Maintenant, cliquez sur « More options » et cochez « Preserve order for % Split ». Puis relancez l'algorithme.

- Que remarquez-vous ? Donnez une explication sur ce phénomène.

Il est possible dans Weka de rendre la base de donnée utilisée aléatoire par un prétraitement de ses données grâce à l'onglet « Preprocess ». Choisissez le filtre : filters > unsupervised > instance > Randomize. Cliquez sur « Apply ». Relancez l'algorithme.

- Que remarquez-vous ?
- Quel pourrait-être selon vous le désavantage d'utiliser un tel filtre plutôt que de laisser la méthode d'apprentissage se charger des données ?

Pour la suite du TP, redécochez l'option « Preserve order for % Split ».

Rechargez la base « iris ».

Paramètre K et méthode d'apprentissage et de validation

Lancez l'algorithme K-NN avec $K = \{1, 15, 20\}$ et un « Percentage split » = {50,66} % ou un

« Cross-validation »

- Comparez pour ces 9 combinaisons de paramètres, le taux d'instances bien classées.
- Quelle est la sensibilité au paramètre K, à la méthode d'apprentissage/validation ?
- Quel pourrait être le paramétrage choisi par un décideur ? La taille de la base de données joue-t-elle un rôle dans cette décision ?

	K=1	K=15	K=20
Split 50%			
Split 66%			
Cross-validation			

Réitérer cette étude en passant le paramètre *crossValidate* à 'True'.

	K=1	K=15	K=20
Split 50%			
Split 66%			
Cross-validation			

- Obtenez-vous les mêmes conclusions ?

Dans la suite du TP, on utilisera l'option « Cross-validation » pour l'apprentissage et la validation de la classification.

Choisissez la méthode d'arbre de décision (C4.5) de Weka. L'arbre est visualisable en faisant un clic droit sur la ligne de résultat.

- A quoi correspond le paramètre minNumObj (noté M) ?

Paramètre M

Lancez l'algorithme C4.5 avec $M = \{2, 5, 7\}$.

- Quel nombre de feuilles possède l'arbre trouvé pour chaque valeur de M ?
- Quels attributs sont utilisés ? Aurait-on pu le prédire ?
- Existe-t-il un compromis entre le nombre de feuilles et le taux d'erreur ? Quel pourrait être le paramétrage choisi par un décideur ?

Etude des paramètres des algorithmes de classification supervisée Chargez la base de données « vote ».

- Quels sont les attributs ?
- Combien y'a-t-il de classes différentes ?

Lancez K-NN avec $K = \{1, 9, 10, 11, 20\}$.

- Analysez la sensibilité à ce paramètre. Que pouvez-vous dire ?

Lancer C4.5 avec $M = \{2, 6, 10\}$.

- Analyser la sensibilité à ce paramètre. Que pouvez-vous dire ?
- Que dire du compromis entre le nombre de feuilles et le taux d'erreur ?

De manière générale, que pouvez-vous conclure sur le choix des méthodes de classification supervisée et sur leurs paramètres ?

Lancez les réseaux de neurones sous Weka pour ce problème de classification

TP4

Règles d'association

Les bases de données nécessaires au TP sont sur la page suivante :

<https://github.com/renatopp/arff-datasets/tree/master/classification>

Algorithme Apriori

Exercice 1

Répondez aux questions suivantes :

- Expliquez la notion « d'itemsets fréquents »
- Expliquez le fonctionnement d'Apriori
- Sur quel type de données travaille Apriori
- Donnez un exemple classique d'utilisation

Exercice 2

Trouvez les ensembles fréquents du problème ci-dessous représenté de façon verticale (il conviendra peut être de modifier la présentation des données) ...

1	2	3	4	5	6	7	8	9	10
A	B	A	A	A	A	D	A	B	A
B	D	B	B	D	B	E	B	C	B
C	E	C	C	E	C		D	D	D
D			E					E	

- ... pour un support de 40%
- ... pour un support de 60%

Exercice 3

Appliquez Apriori sur l'exemple comportant les transactions suivantes. On prendra un seuil de support égal à 60% et un seuil de confiance de 90%.

{A, C, G, H} ; {B, D, F, G} ; {A, C, D, G, H} ; {B, C, H} ; {A, D, G, H}

Apriori avec WEKA

Bases de données utilisées : « soybean » , « weather.nominal »

<http://storm.cis.fordham.edu/~gweiss/data-mining/datasets.html>

Algorithmes de classification : onglet « Associate » de l' « Explorer ».

- Apriori

Ce TP a pour but de tester l'algorithme Apriori et d'appréhender la difficulté liée au choix des règles d'association.

Description de chaque base de données :

- Nombres d'instances ? d'attributs ? de classes ?
- Types de données ? Paramètres

de l'algorithme A priori :

- Quels sont les paramètres proposés par WEKA ?
- Quel est leur rôle ?

Expérimentations :

Lancez l'algorithme et analysez le résultat affiché.

- Combien de règles ont été trouvées ?
- Les règles sont-elles toutes intéressantes ? Faites varier ce nombre
- Quelles règles trouvez-vous intéressantes ?
- Comment fixer l'attribut conséquence (attribut à droite de la règle) à l'attribut classe. Voir le paramètre *car* d'Apriori

Le choix des règles est associé à une métrique.

- Quelles sont les 4 métriques proposées dans WEKA ? A quoi correspondent-elles ?
- A quel moment interviennent-elles ?
- Comparez les règles obtenues avec différentes métriques.

En cliquant du bouton droit dans la fenêtre en face du bouton Choose, on a accès aux paramètres de l'algorithme apriori. Le bouton More détaille chacune de ces options.

delta : fait décroître le support minimal de ce facteur, jusqu'à ce que soit le nombre de règles demandées a été trouvé, soit on a atteint la valeur minimale du support
lowerBoundMinSupport

lowerBoundMinSupport : valeur minimale du support (minsup en cours). Le support part d'une valeur initiale, et décroît conformément à delta.

metricType : la mesure qui permet de classer les règles. Supposons que L désigne la partie gauche de la règle et R la partie droite. Il y en a quatre (L désigne la partie gauche de la règle et R la partie droite) :

- Confidence : la confiance.

- Lift : l'amélioration.
- Leverage : proportion d'exemples concernés par les parties gauche et droite de la règle, en plus de ce qui seraient couverts, si les deux parties de la règle étaient indépendantes :
- Conviction : similaire à l'amélioration, mais on s'intéresse aux exemples où la partie droite de la règle n'est pas respectée. Le rapport est inversé.

minMetric : la valeur minimale de la mesure en dessous de laquelle on ne recherchera plus de règle.

numRules : Le nombre de règles que l'algorithme doit produire.

removeAllMissingCols : enlève les colonnes dont toutes les valeurs sont manquantes.

significanceLevel : test statistique

upperBoundMinSupport : valeur initiale du support.

Exercice optionnel

L'épicerie de nuit de la rue "remplacez par votre rue favorite" a décidé à l'approche des fêtes de fin d'année de lancer une vaste opération de promotion. Son patron, fervent adepte des nouvelles technologies et de la fouille de données (ça arrive), vous demande d'utiliser les règles d'associations pour trouver des règles intéressantes pour ses futures promotions. Il va donc réutiliser le bilan d'achats de l'année dernière à la même date :

Achats	Produit 1	Produit 2	Produit 3	Produit 4	Produit 5
Mme Michou	X			X	X
Tonton Gérard	X	X			X
Mme Guénolet					X
Mr Robert			X	X	X
Mr Sar	X	X	X	X	X
Mr causy	X				X
Mme mimi	X			X	X
Mme Fillon		X	X		

1. Générer un fichier ARFF contenant les données du bilan d'achat
2. Extraire les règles d'associations avec un support de 0.5, 0.3 puis de 0.1
3. Que pouvez-vous conseiller comme promotion au patron ?

TP5 Réseaux de neurones (Optionnel)

On souhaite appliquer les réseaux de neurones à la base de données iris.arff. Allez dans le répertoire Classifiers : Functions : MultiLayerPerceptron. Les **principales options disponibles dans l'Explorer** (pour les perceptrons multicouche) sont les suivantes :

- *GUI* : permet l'utilisation d'une interface graphique
- *autobuild* : Connecte les couches cachées : le laisser à True.
- *decay* : si vrai, faire décroître le taux d'apprentissage : les poids sont moins modifiés au fur et à mesure de l'apprentissage.
- *hiddenLayers* : permet de décrire le nombre et la taille des couches cachées. La description est :
 - Soit une suite d'entiers (le nombre de neurones par couche) séparés par des virgules.
 - Soit les valeurs spéciales déterminant une seule couche cachée :
 - a : (nombre d'attributs+nombre de classes)/2
 - i : nombre d'attributs
 - o : nombre de classes
 - t : nombre d'attributs+nombre de classes
- *learning rate* : η (algorithme du gradient)
- *momentum* : α (algorithme du gradient)

$$\Delta w(n) = \eta wx + \alpha \Delta(n-1)$$

- *nominalToBinaryFilter* : transforme les attributs nominaux en attributs binaires : un attribut pouvant prendre k valeurs différentes sera transformé en k attributs binaires, un seul de ces attributs valant 1.
- *normalizeAttributes* : les valeurs des attributs (y compris les attributs nominaux qui seront passés dans le filtre nominalToBinaryFilter) seront toutes ramenées entre -1 et 1.
- *normalizeNumericClass* : si la classe est numérique, on la normalise (de façon interne) : permet d'améliorer les résultats .
- *training time* : le nombre de fois (epochs en anglais) où l'on fera passer l'ensemble d'apprentissage à travers le réseau.

Réalisez un apprentissage à l'aide d'un perceptron multicouche sur la base iris.arff. Après apprentissage, qui peut prendre un peu de temps, vous devriez obtenir le résultat dans la fenêtre de droite « Classifier output ». Analyse du résultat. Si vous avez utilisé la méthode de validation croisée, vous devriez obtenir ensuite un tableau de résultats du type :

```

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      117           78      %
Incorrectly Classified Instances    33           22      %
Kappa statistic                    0.67
Mean absolute error                 0.2936
Root mean squared error             0.3444
Relative absolute error             66.0635 %
Root relative squared error         73.0663 %
Total Number of Instances          150

=== Detailed Accuracy By Class ===

TP Rate    FP Rate    Precision    Recall    F-Measure    Class
1          0.04        0.926       1          0.962       Iris-setosa
0.34       0           1           0.34      0.507       Iris-versicolor
1          0.29        0.633       1          0.775       Iris-virginica

=== Confusion Matrix ===

  a  b  c   <-- classified as
50  0  0 |  a = Iris-setosa
 4 17 29 |  b = Iris-versicolor
 0  0 50 |  c = Iris-virginica

```

D'après ce tableau, on voit que 78% des exemples ont été classés correctement. La matrice de confusion en bas, indique que ces erreurs ont toutes concerné la classe « Iris-versicolor » pour laquelle seule 17 exemples sur 50 sont correctement classés, alors que 4 sont classés comme des « Iris-setosa » et 29 comme des « Iris-Verginica ».

Effets du choix des valeurs de paramètres sur les résultats

Question 1 : En demandant de visualiser les erreurs (click droit dans Result List, puis Visualize classifiers errors), retrouvez les instances sur lesquelles le classifieur se trompe. Peut-on trouver les raisons des mauvaises classifications ?

Question 2 : Relancer l'apprentissage, en fixant maintenant un temps de calcul de 25 epochs, au lieu de 500. Regardez de nouveau où se situent les instances mal classées.

Question 3 : Combien faut-il d'epochs pour que tous les exemples soient bien classés?

Analyse des réseaux

Reprenez un apprentissage de 500 epochs, mais demandez maintenant la visualisation du réseau (GUI dans la fenêtre des options du **MultilayerPerceptron**)

Maintenant, lorsqu'on appuie sur Start une fenêtre s'ouvre, qui nous montre le réseau construit par **Weka**.

La couche cachée contient 3 neurones (choix par défaut de Weka $((4 + 3)/2 = 3)$)

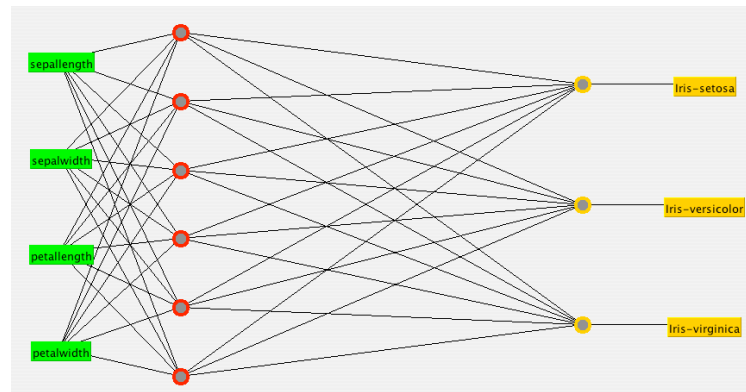
Pour lancer l'apprentissage, il faut maintenant, dans la fenêtre représentant le réseau, appuyer sur « Start », puis sur « Accept » pour fermer la fenêtre et obtenir les statistiques dans la fenêtre Classifier output.

Les nœuds sont numérotés de 0 à n , en commençant par la couche la plus à droite, et par le neurone du haut. Après apprentissage, la fenêtre Classifier « output » nous donne la valeur des poids dans tout le réseau. Par exemple, les poids en entrée du nœud « node 0 », celui qui correspond à la sortie « iris setosa », ont les valeurs indiquées dans la figure 2

```
Sigmoid Node 0
  Inputs  Weights
Threshold -2.428722952755603
Node 3    -0.4668883326080196
Node 4     6.159683060398803
Node 5    -3.9410809046292963
```

Fig. 2 – Les poids en entrée de « Node 0 »

Question 4 : Nous allons maintenant faire varier le nombre de neurones de la couche cachée. Pour cela, cliquez sur le bouton « MultilayerPerceptron ... ». Puis mettez « autobuild » à true. Puis choisissez le nombre de neurones de la couche cachée dans hidden layer. Par exemple, si vous mettez 6, et que vous mettez « GUI » à True, vous obtiendrez le réseau suivant :



Vous devriez observer que le nombre de neurones en couche cachée n'affecte pas beaucoup les performances en généralisation. Cela devrait être différent avec d'autres jeux de données.

Mêmes expériences avec d'autres jeux de données

On refait les mêmes expériences avec d'autres jeux de données dont : tic-tac-toe.arff et credit.arff

Comparaison avec un autre algorithme d'apprentissage : les SVM et Random Forest

Utilisez maintenant l'algorithme SVM et Random Forest pour les mêmes expériences. Pour cela, il faut choisir : classifieurs : SMO et RandomForest

Question 5 : Est-ce que les erreurs de classification commises sont les mêmes que pour le perceptron multicouche



TP6 - Ensemble Algorithms (Optionnel)

We are going to take a tour of 5 top ensemble machine learning algorithms in Weka. Each algorithm that we cover will be briefly described in terms of how it works, key algorithm parameters will be highlighted and the algorithm will be demonstrated in the Weka Explorer interface. The 5 algorithms that we will review are:

1. Bagging
2. Random Forest
3. AdaBoost
4. Voting
5. Stacking

These are 5 algorithms that you can try on your problem in order to lift performance. A standard machine learning classification problem will be used to demonstrate each algorithm. Specifically, the *Ionosphere binary classification problem*. This is a good dataset to demonstrate classification algorithms because the input variables are numeric and all have the same scale the problem only has two classes to discriminate.

Each instance describes the properties of radar returns from the atmosphere and the task is to predict whether or not there is structure in the ionosphere or not. There are 34 numerical input variables of generally the same scale. You can learn more about this dataset on the [UCI Machine Learning Repository](#). Top results are in the order of 98% accuracy.

Start the Weka Explorer:

1. Open the Weka GUI Chooser.
2. Click the “Explorer” button to open the Weka Explorer.
3. Load the Ionosphere dataset from the *data/ionosphere.arff* file
4. Click “Classify” to open the Classify tab.

Bootstrap Aggregation (Bagging)

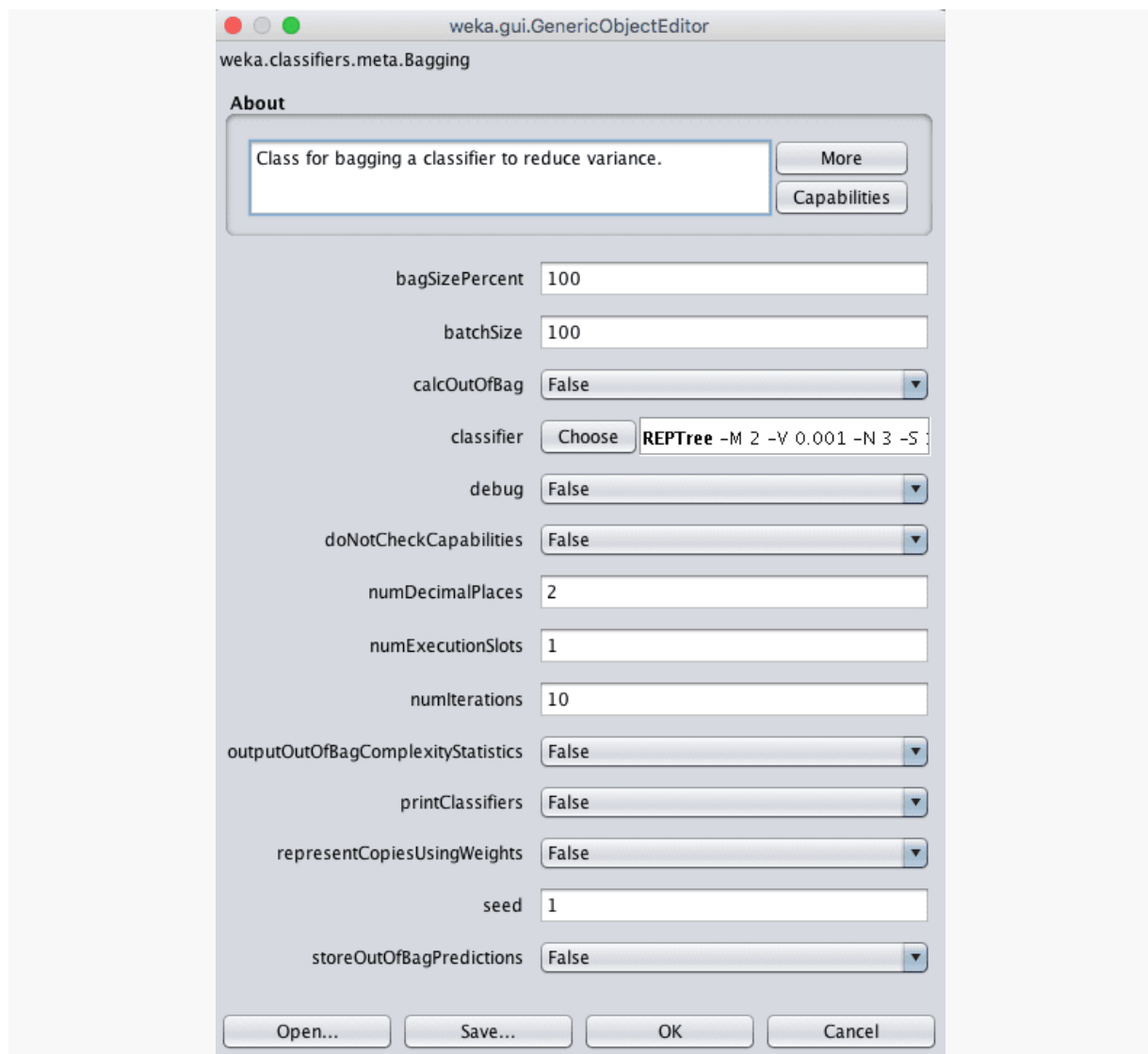
Bootstrap Aggregation or Bagging for short is an ensemble algorithm that can be used for classification or regression. Bootstrap is a statistical estimation technique where a statistical quantity like a mean is estimated from multiple random samples of your data (with replacement). It is a useful technique when you have a limited amount of data and you are interested in a more robust estimate of a statistical quantity.

This sample principle can be used with machine learning models. Multiple random samples of your training data are drawn with replacement and used to train multiple different machine learning models. Each model is then used to make a prediction and the results are averaged to give a more robust prediction.

It is a technique that is best used with models that have a low bias and a high variance, meaning that the predictions they make are highly dependent on the specific data from which they were trained. The most used algorithm for bagging that fits this requirement of high variance are **decision trees**.

Choose the bagging algorithm:

1. Click the “Choose” button and select “Bagging” under the “meta” group.
2. Click on the name of the algorithm to review the algorithm configuration.



Weka Configuration for the Bagging Algorithm

A key configuration parameter in bagging is the **type of model being bagged**. The default is the REPTree which is the Weka implementation of a standard decision tree, also called a Classification and Regression Tree or CART for short. This is specified in the classifier parameter.

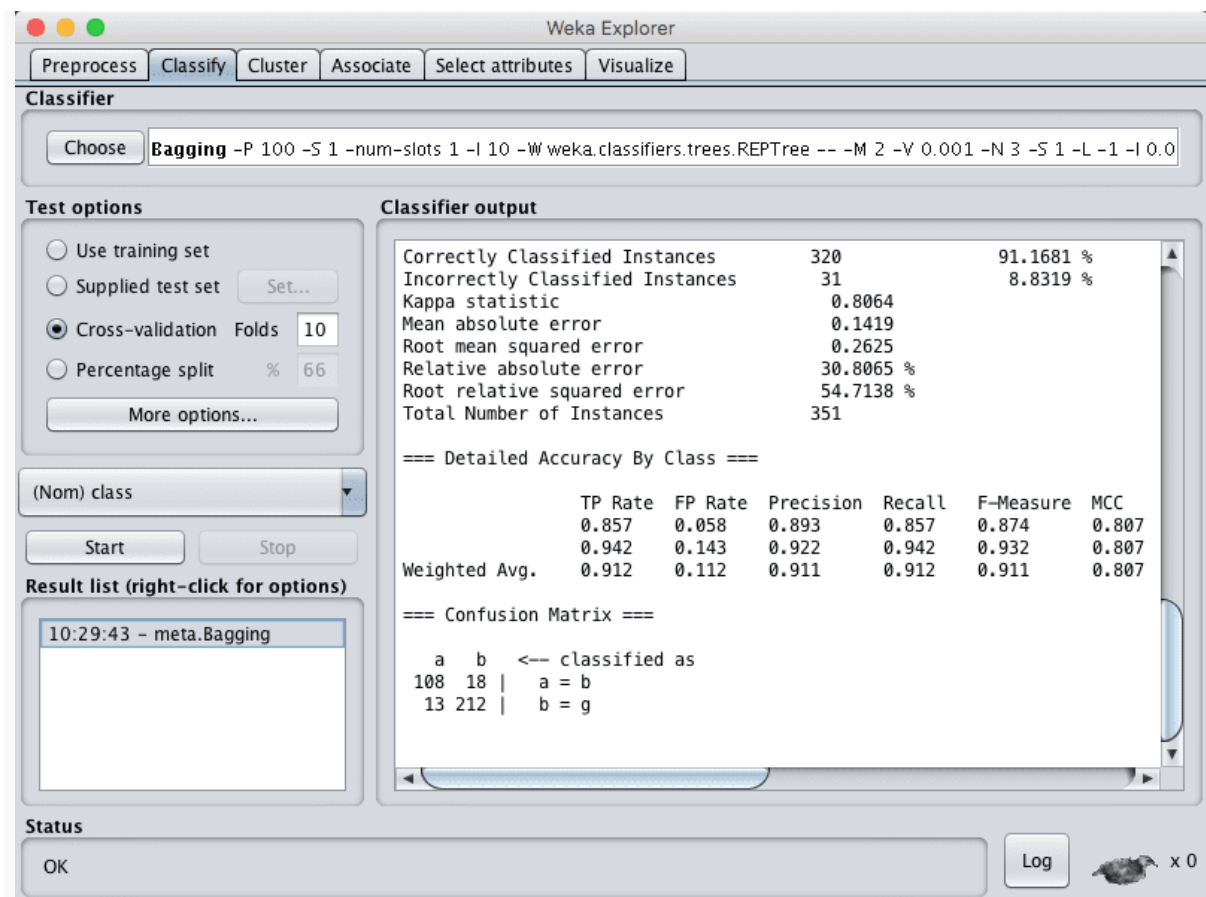
The size of each random sample is specified in the **bagSizePercent**, which is a size as a percentage of the raw training dataset. The default is 100% which will create a new random sample the same size as the training dataset, but will have a different composition.

This is because the random sample is drawn with replacement, which means that each time an instance is randomly drawn from the training dataset and added to the sample, it is also added back into the training dataset (replaced) meaning that it can be chosen again and added twice or more times to the sample.

Finally, the number of bags (and number of classifiers) can be specified in the **numIterations** parameter. The default is 10, although it is common to use values in the hundreds or thousands. Continue to increase the value of numIterations until you no longer see an improvement in the model, or you run out of memory.

1. Click “OK” to close the algorithm configuration.
2. Click the “Start” button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that bagging achieves an accuracy of 91%.



The screenshot shows the Weka Explorer window with the 'Classify' tab selected. The 'Classifier' dropdown is set to 'Bagging'. The command line shows: `Bagging -P 100 -S 1 -num-slots 1 -I 10 -W weka.classifiers.trees.REPTree -- -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0`.

Test options:

- ☐ Use training set
- ☐ Supplied test set (Set...)
- ☒ Cross-validation (Folds: 10)
- ☐ Percentage split (%: 66)
- More options...

Classifier output:

```

Correctly Classified Instances      320           91.1681 %
Incorrectly Classified Instances    31            8.8319 %
Kappa statistic                    0.8064
Mean absolute error                 0.1419
Root mean squared error             0.2625
Relative absolute error             30.8065 %
Root relative squared error         54.7138 %
Total Number of Instances          351

=== Detailed Accuracy By Class ===
               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
Weighted Avg.   0.857   0.058   0.893     0.857   0.874     0.807
               0.942   0.143   0.922     0.942   0.932     0.807
               0.912   0.112   0.911     0.912   0.911     0.807

=== Confusion Matrix ===
  a  b  <-- classified as
108 18 | a = b
 13 212 | b = g

```

Result list (right-click for options):

- 10:29:43 - meta.Bagging

Status: OK

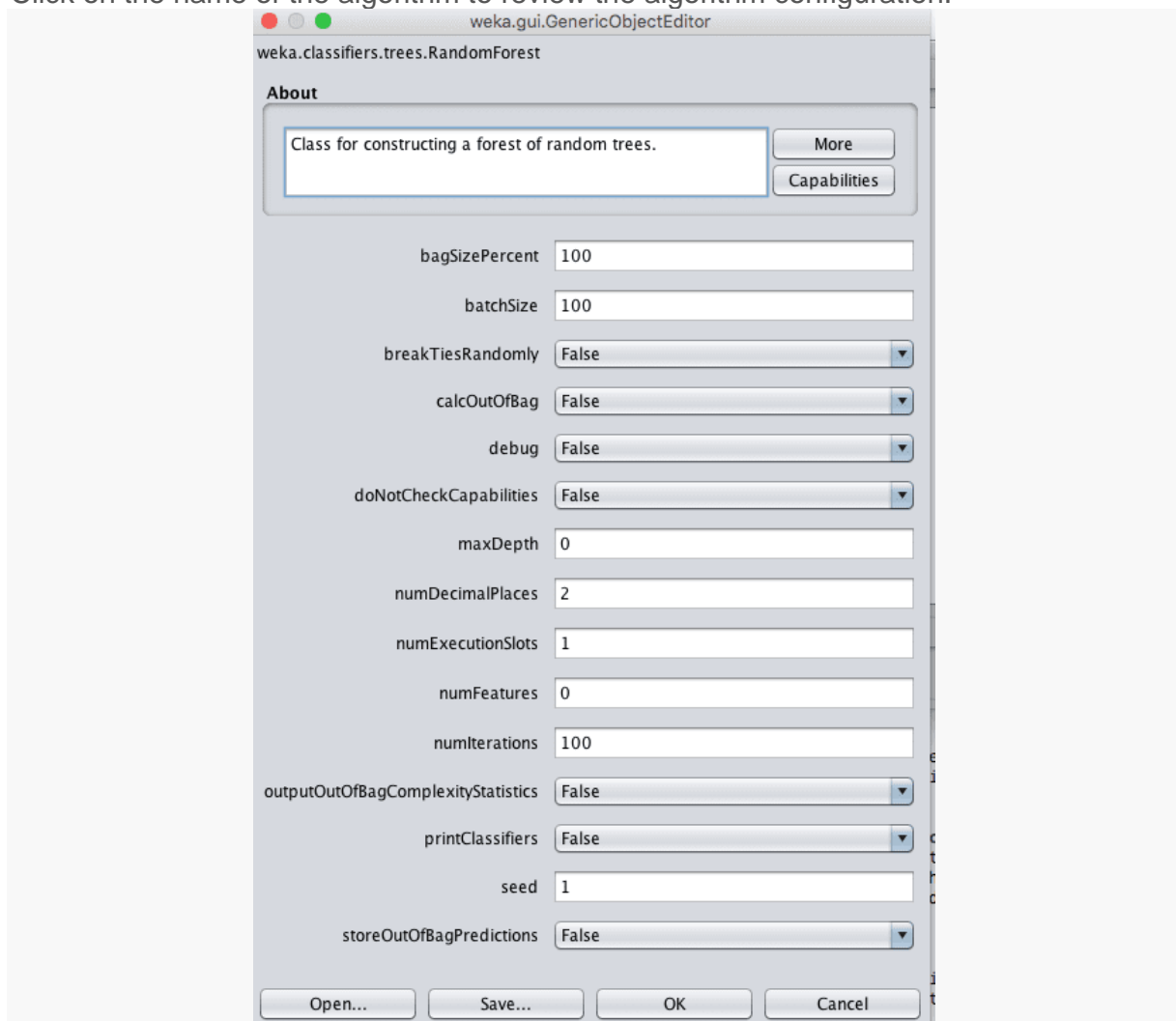
Weka Classification Results for the Bagging Algorithm

Random Forest

Random Forest is an extension of bagging for decision trees that can be used for classification or regression. A down side of bagged decision trees is that decision trees are constructed using a greedy algorithm that selects the best split point at each step in the tree building process. As such, the resulting trees end up looking very similar which reduces the variance of the predictions from all the bags which in turn hurts the robustness of the predictions made.

Random Forest is an improvement upon bagged decision trees that disrupts the greedy splitting algorithm during tree creation so that split points can only be selected from a random subset of the input attributes. This simple change can have a big effect decreasing the similarity between the bagged trees and in turn the resulting predictions. Choose the random forest algorithm:

1. Click the “Choose” button and select “RandomForest” under the “trees” group.
2. Click on the name of the algorithm to review the algorithm configuration.



Weka Configuration for the Random Forest Algorithm

In addition to the parameters listed above for bagging, a key parameter for random forest is the number of attributes to consider in each split point. In Weka this can be controlled by the **numFeatures attribute**, which by default is set to 0, which selects the value automatically based on a rule of thumb.

1. Click “OK” to close the algorithm configuration.
2. Click the “Start” button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that random forests achieves an accuracy of 92%.

Classifier

Choose **RandomForest** -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1

Test options

☐ Use training set
☐ Supplied test set Set...
☒ Cross-validation Folds
☐ Percentage split %
 More options...

(Nom) class

Start Stop

Result list (right-click for options)

10:40:49 - trees.RandomForest

Classifier output

```

Correctly Classified Instances      326      92.8775 %
Incorrectly Classified Instances    25       7.1225 %
Kappa statistic                    0.8428
Mean absolute error                 0.1287
Root mean squared error             0.2255
Relative absolute error             27.951 %
Root relative squared error         47.0057 %
Total Number of Instances          351

=== Detailed Accuracy By Class ===
               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
               0.865   0.036   0.932     0.865   0.897     0.844
               0.964   0.135   0.927     0.964   0.946     0.844
Weighted Avg.   0.929   0.099   0.929     0.929   0.928     0.844

=== Confusion Matrix ===
  a  b  <-- classified as
109 17 |  a = b
 8 217 |  b = g
  
```

Status

OK Log x 0

Weka Classification Results for the Random Forest Algorithm

AdaBoost

AdaBoost is an ensemble machine learning algorithm for classification problems.

It is part of a group of ensemble methods called **boosting**, that add new machine learning models in a series where subsequent models attempt to fix the prediction errors made by prior models. AdaBoost was the first successful implementation of this type of model.

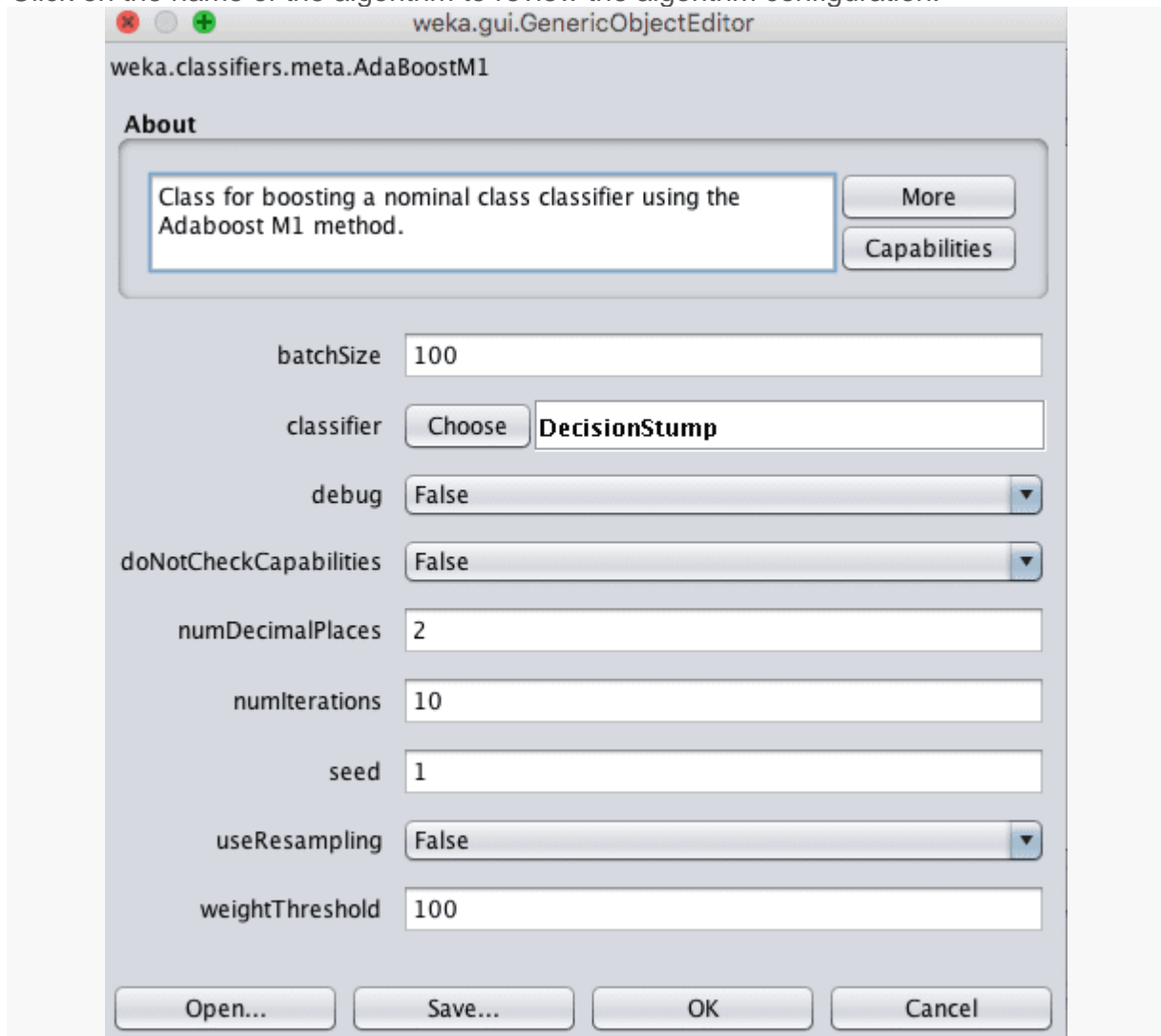
Adaboost was designed to use **short decision tree models**, each with a single decision point. Such short trees are often referred to as decision stumps.

The first model is constructed as per normal. Each instance in the training dataset is weighted and the weights are updated based on the overall accuracy of the model and

whether an instance was classified correctly or not. Subsequent models are trained and added until a minimum accuracy is achieved or no further improvements are possible. Each model is weighted based on its skill and these weights are used when combining the predictions from all of the models on new data.

Choose the AdaBoost algorithm:

1. Click the “Choose” button and select “AdaBoostM1” under the “meta” group.
2. Click on the name of the algorithm to review the algorithm configuration.



Weka Configuration for the AdaBoost Algorithm

The weak learner within the AdaBoost model can be specified by the classifier parameter. The default is the decision stump algorithm, but other algorithms can be used. a key parameter in addition to the **weak learner** is the **number of models to create** and add in series. This can be specified in the *numIterations* parameter and defaults to 10.

1. Click “OK” to close the algorithm configuration.

- Click the “Start” button to run the algorithm on the Ionosphere dataset.
You can see that with the default configuration that AdaBoost achieves an accuracy of 90%.

Classifier

Choose **AdaBoostM1** -P 100 -S 1 -I 10 -W weka.classifiers.trees.DecisionStump

Test options

☐ Use training set
☐ Supplied test set Set...
☒ Cross-validation Folds **10**
☐ Percentage split % **66**
 More options...

(Nom) class

Start Stop

Result list (right-click for options)

10:48:08 - meta.AdaBoostM1

Classifier output

```

Correctly Classified Instances      319      90.8832 %
Incorrectly Classified Instances    32       9.1168 %
Kappa statistic                    0.7925
Mean absolute error                 0.1605
Root mean squared error             0.2733
Relative absolute error             34.8447 %
Root relative squared error         56.9762 %
Total Number of Instances          351

=== Detailed Accuracy By Class ===
               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
Weighted Avg.   0.770    0.013    0.970     0.770    0.858     0.804
               0.987    0.230    0.884     0.987    0.933     0.804
               0.909    0.152    0.915     0.909    0.906     0.804

=== Confusion Matrix ===
  a  b  <-- classified as
 97 29 |  a = b
  3 22 |  b = g
  
```

Status

OK Log x 0

Weka Classification Results for the AdaBoost Algorithm

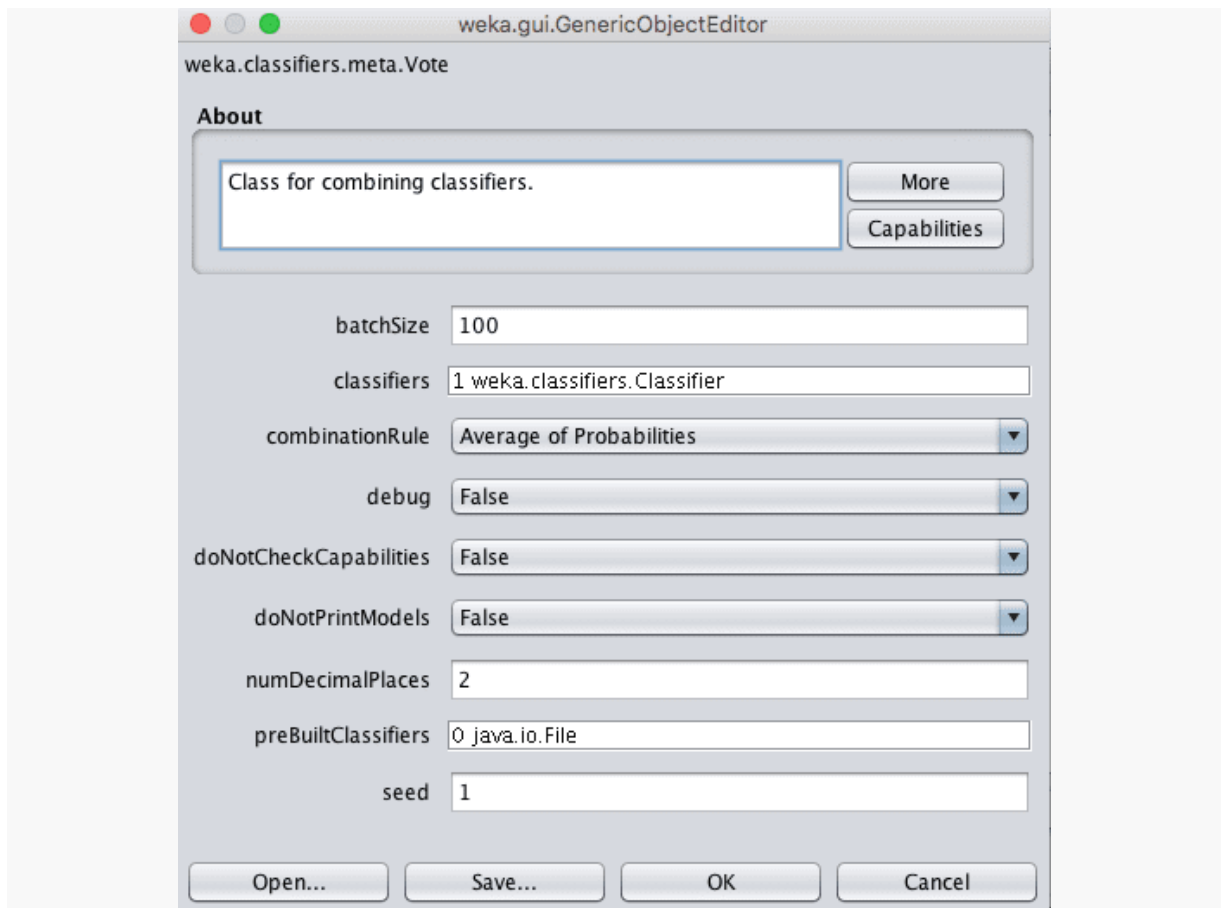
Voting

Voting is perhaps the simplest ensemble algorithm, and is often very effective. It can be used for classification or regression problems.

Voting works by creating two or more sub-models. Each sub-model makes predictions which are combined in some way, such as by taking the mean or the mode of the predictions, allowing each sub-model to vote on what the outcome should be.

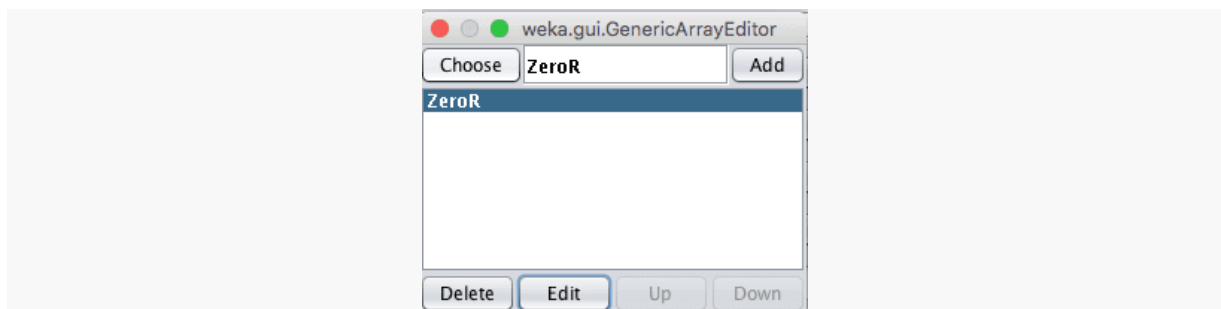
Choose the Vote algorithm:

- Click the “Choose” button and select “Vote” under the “meta” group.
- Click on the name of the algorithm to review the algorithm configuration.



Weka Configuration for the Voting Ensemble Algorithm

The key parameter of a Vote ensemble is the selection of sub-models. Models can be specified in Weka in the classifier parameter. Clicking this parameter lets you add a number of classifiers.



Weka Algorithm Selection for the Voting Ensemble Algorithm

Clicking the “Edit” button with a classifier selected lets you configure the details of that classifier. An objective in selecting sub-models is to select models that make quite different predictions (uncorrelated predictions). As such, it is a good rule of thumb to select very different model types, such as trees, instance based methods, functions and so on. Another key parameter to configure for voting is how the predictions of the sub models are

combined. This is controlled by the **combinationRule parameter** which is set to take the average of the probabilities by default.

1. Click “OK” to close the algorithm configuration.
2. Click the “Start” button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that Vote achieves an accuracy of 64%.

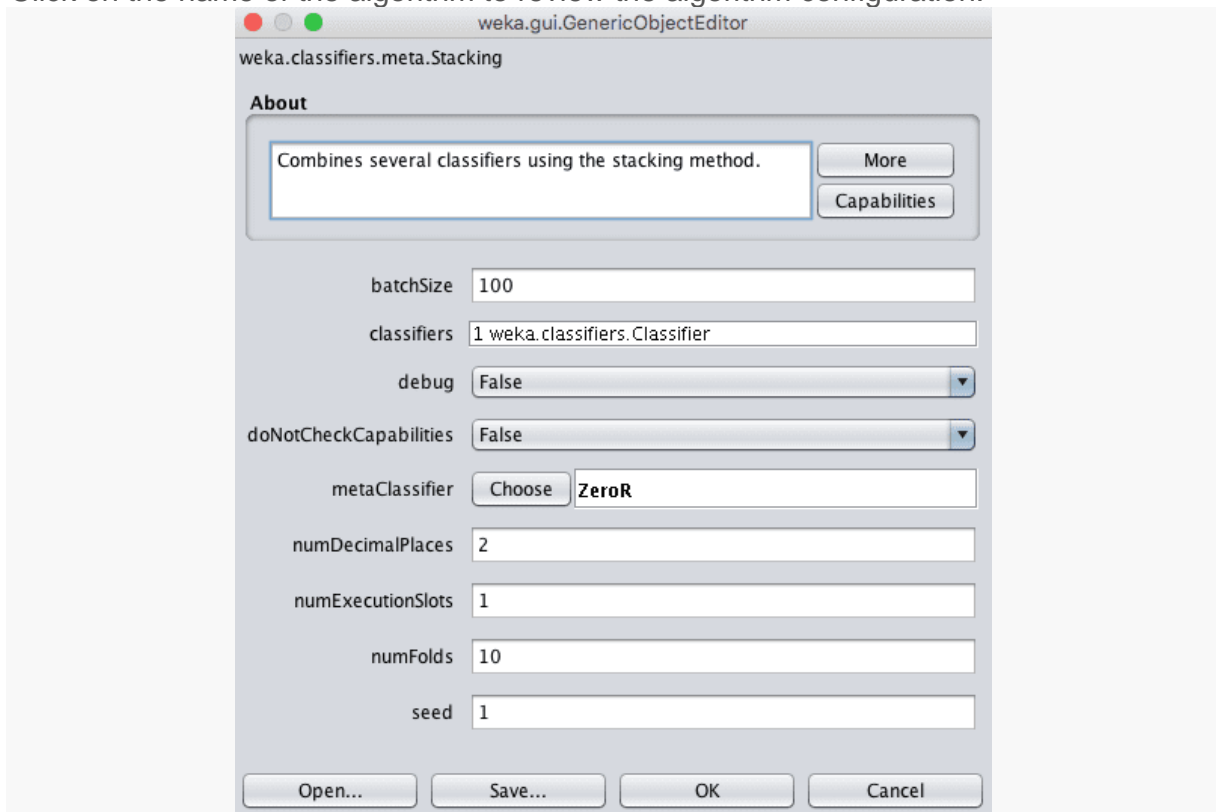
Obviously, this technique achieved poor results because only the ZeroR sub-model was selected. Try selecting a collection of 5-to-10 different sub models.

Stacked Generalization (Stacking)

[Stacked Generalization or Stacking](#) for short is a simple extension to Voting ensembles that can be used for classification and regression problems. In addition to selecting multiple sub-models, stacking allows you **to specify another model to learn how to best combine the predictions from the sub-models**. Because a meta model is used to best combine the predictions of sub-models, this technique is sometimes called blending, as in blending predictions together.

Choose the Stacking algorithm:

1. Click the “Choose” button and select “Stacking” under the “meta” group.
2. Click on the name of the algorithm to review the algorithm configuration.



Weka Configuration for the Stacking Ensemble Algorithm

As with the Vote classifier, you can specify the sub-models in the classifiers parameter.

The model that will be trained to learn how to best combine the predictions from the sub model can be specified in the metaClassifier parameter, which is set to ZeroR (majority vote or mean) by default. It is common to use a linear algorithm like linear regression or logistic regression for regression and classification type problems respectively. This is to achieve an output that is a simple linear combination of the predictions of the sub models.

1. Click "OK" to close the algorithm configuration.
2. Click the "Start" button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that Stacking achieves an accuracy of 64%.

Again, the same as voting, Stacking achieved poor results because only the ZeroR sub-model was selected. Try selecting a collection of 5-to-10 different sub models and a good model to combine the predictions.

The screenshot shows the Weka Explorer window with the 'Classify' tab selected. The 'Classifier' section shows 'Stacking' with parameters: `-X 10 -M "weka.classifiers.rules.ZeroR" -S 1 -num-slots 1 -B "weka.classifiers.rules.ZeroR"`. The 'Test options' section has 'Cross-validation' selected with 'Folds' set to 10. The 'Classifier output' section displays the following results:

Metric	Value	Percentage
Correctly Classified Instances	225	64.1026 %
Incorrectly Classified Instances	126	35.8974 %
Kappa statistic	0	
Mean absolute error	0.4605	
Root mean squared error	0.4797	
Relative absolute error	100	%
Root relative squared error	100	%
Total Number of Instances	351	

Below this, the 'Detailed Accuracy By Class' table is shown:

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
Weighted Avg.	0.641	0.641	0.411	0.641	0.501	0.000

The 'Confusion Matrix' section shows:

```
a  b  <-- classified as
0 126 | a = b
0 225 | b = g
```

The 'Result list' shows a single entry: '11:09:07 - meta.Stacking'. The 'Status' bar at the bottom shows 'OK'.

Weka Classification Results for the Stacking Ensemble Algorithm

