Machine Learning: EXPLORING WEKA

What’s Weka

- Collection of Machine Learning algorithm for data mining tasks
- 100+ algorithms for classification
- Data pre-processing
- Feature selection assistance
- Clustering, finding association rules etc.

HOW do we use Weka

- Load data into Weka and explore it
- Use filters and preprocess it
- Apply classification algorithms
- Interpret the output
- Understand evaluation methods and their implication
- Understand various representations for models
- Explain how popular machine learning algorithms work
- Be aware of common pitfalls with data mining

To do

- Install Weka
- Explore the “Explorer” interface
- Explore some datasets
- Build a classifier
- Interpret the output
- Use filters
- Visualize your data set
Comparing classifiers

Algorithms choice: basic and simple (first)

- **Baseline first**
  - Rule > ZeroR
  - Simply chooses the most likely class
  - Test on the training set
  - If attribute non significant > lower performance than the baseline

- **Simple first**
  - Rule > OneR
  - One rule does the whole job

OVERFITTING

- Using all features for classifying is not good...
  - Select OneR, for classifying a dataset with branching numeric attributes
  - Change the option MinBucketSize from 6 to 1, run the test using the Training Set:
    - Branching from «temperature» in the Weather dataset: perfect prediction, very accurate, but no generalization
More Algorithms: Naïve Bayes

- Naïve Bayes methods
  - OneR (one attribute does all the work) vs. Naïve Bayes (use all the attributes)
- Two assumptions
  - Attributes are equally important a priori
  - Statistically independent (given the class value)
    - i.e., knowing the value of one attribute says nothing about the value of another (if the class is known, especially in cases where there is not a large amount of training data, Ng & Jordan 2002)

More Algorithms: Naïve Bayes

- Probability of event $H$ (class) given evidence $E$ (instance)
  $$P[H|E] = \frac{P[E|H]P[H]}{P[E]}$$
- $P[H]$ is a priori probability of $H$ (the probability of event before evidence is seen)
- $P[H|E]$ is a posteriori probability of $H$ (the probability of event after evidence is seen)
- "Naïve" assumption: evidence splits into parts that are independent
  $$P[H|E] = \frac{P[E_1|H]P[E_2|H]...P[E_n|H]P[H]}{P[E]}$$

More Algorithms: Decision trees

- E.g. J48 (statistical classifier) C4.5 (Quinlian) = J48
- Top-Down recursive divide and conquer
  - Select attribute for root node (create branch for each possible attribute value)
  - Split instances into subsets (one for each branch extending from the root node)
  - Repeat recursively for each branch (using only instances that reach the branch)
  - Stop if all instances have the same class
More Algorithms: Decision trees

- **Pruning**: perform worse on the test data, but generalize better
  - Don’t continue splitting if the nodes get very small (J48 minNumObj parameter, default value 2)
  - Build full tree and then work back from the leaves, applying a statistical test at each stage (confidenceFactor parameter, default value 0.25)
  - Sometimes it’s good to prune an interior node, raising the subtree beneath it up one level (subtreeRaising, default true)
  - Messy ... complicated ... not particularly illuminating

More Algorithms: Nearest neighbor

- "Rote learning": simplest form of learning
  - to classify a new instance, search training set for one that’s “most like” it
  - the instances themselves represent the “knowledge”
  - lazy learning: do nothing until you have to make predictions
- "Instance-based" learning = “nearest-neighbor” learning

More Algorithms: Nearest neighbor

- \( k \)-nearest-neighbors choose majority class among several neighbors \((k \text{ of them})\)
- lazy\text{-IBk} (instance-based learning)
  - Try \( k = 1, 5, 20 \) with 10-fold cross-validation on a rather small vs large dataset

Boundary Visualizer

- Internal representation of the dataset
More Algorithms: LINEAR REGRESSION

- Data sets so far: nominal and numeric attributes, but only nominal classes
- Now: numeric classes (classical statistical method from 1805!)
- Calculate weights from training data
- Ok if there are more instances than attributes
  - Nominal values must be converted in 1 and 0
- MSP (linear regression patches at each node of a Decision Tree)

More Algorithms: classification by REGRESSION

- Two-class problem
  - Training: call the classes 0 and 1
  - Prediction: set a threshold for predicting class 0 or 1
- Multi-class problem: “multi-response linear regression”
  - Training: perform a regression for each class
    - Set output to 1 for training instances that belong to the class,
    - 0 for instances that don’t
  - Prediction: choose the class with the largest output ... or use “pairwise linear regression”, which performs a regression for every pair of classes

Investigate two-class classification by regression

- Open file diabetes.arff
- Use the NominalToBinary attribute filter to convert to numeric
  - but first set Class: class (Nom) to No class, because attribute filters do not operate on the class value
- Choose functions>LinearRegression
- Run
- Set Output predictions option

More Algorithms: logistic REGRESSION

- Two-class problem
  - Training: call the classes 0 and 1
  - Prediction: set a threshold for predicting class 0 or 1
- Multi-class problem: “multi-response linear regression”
  - Training: perform a regression for each class
    - Set output to 1 for training instances that belong to the class,
    - 0 for instances that don’t
  - Prediction: choose the class with the largest output ... or use “pairwise linear regression”, which performs a regression for every pair of classes
More Algorithms:
Support Vector MAchine

- Maximum margin hyperplane among support vectors
- Linearly separable: too easy... real life is harder!
- "Kernel trick" > different shape
- S(quential)M(inimal)O(ptimization)
- libSVM (external download)

More Algorithms:
ensamble learning

- Bagging: resampling the training set (meta>Bagging)
- Random forests: alternative branches in decision trees – (trees>RandomForests)
- Boosting: focus on where the existing model makes errors (meta>AdaBoostM1)
- Stacking: combine results using another learner (instead of voting) (meta>Stacking)

Comparing classifiers

* = significantly worse
v = significantly better