Data Mining

Practical Machine Learning Tools and Techniques

Slides for Chapter 11, Beyond supervised and unsupervised learning

of *Data Mining* by I. H. Witten, E. Frank,
M. A. Hall and C. J. Pal

Semi-supervised and multi-instance learning

- Semisupervised learning
  - Clustering for classification
  - Cotraining
  - EM and cotraining
  - Neural network approaches
- Multi-instance learning
  - Converting to single-instance learning
  - Upgrading learning algorithms
  - Dedicated multi-instance methods

Semisupervised learning

- *Semisupervised learning*: attempts to use unlabeled data as well as labeled data
  - The aim is to improve classification performance
- Why try to do this? Because unlabeled data is often plentiful and labeling data can be expensive
  - Web mining: classifying web pages
  - Text mining: identifying names in text
  - Video mining: classifying people in the news
- Leveraging the large pool of unlabeled examples would be very attractive

Clustering for classification

- We have seen how to use EM for learning a mixture model for clustering, with one mixture component per cluster
- Naïve Bayes can be viewed as applying a mixture model with one component distribution per class
- Can we combine the two?
- Idea: use naïve Bayes on labeled examples and then apply EM
  1. Build naïve Bayes model on labeled data
  2. Label unlabeled data based on class probabilities ("expectation" step)
  3. Train new naïve Bayes model based on all the data ("maximization" step)
  4. Repeat 2 and 3 step until convergence
- Essentially the same as EM for clustering with fixed cluster membership probabilities for the labeled data

Comments on this approach

- Assumes conditional independence
- Has been applied successfully to document classification:
  - Certain phrases are indicative of classes
  - Some of these phrases occur only in the unlabeled data, some in both the labeled and the unlabeled data
  - EM can generalize the model beyond the labeled data by taking advantage of co-occurrences of these phrases
- Refinement 1: reduce weight of unlabeled data
  - Introduce parameter that enables the user to give less weight to the unlabeled data during the learning process
- Refinement 2: allow multiple clusters per class
  - We can extend the mixture model to have multiple components per class, not just one component
  - Modify maximization step to not only probabilistically label each example with classes but to assign it probabilistically to components within a class

Co-training

- *Cotraining* is another well-known method for semi-supervised learning
- Exploits *multiple views* (multiple sets of attributes) for learning from labeled and unlabeled data
- Web pages: classic example of data with multiple views
  - First set of attributes describes content of web page
  - Second set of attributes describes links that link to the web page
- Cotraining algorithm:
  1. Build classification model from each view
  2. Use models to assign labels to unlabeled data
  3. Select those unlabeled examples that were most confidently predicted (ideally, preserving ratio of classes)
  4. Add those examples to the training set
  5. Go to Step 1 until unlabeled data has been exhausted
- Assumption: views are independent (but cotraining appears to work also when views are dependent)

Combining EM and cotraining

- We can combine EM and cotraining for semi-supervised learning to yield the *co-EM* method
- Works like the basic EM approach, but view/classifier is switched in each iteration of EM
  - Uses all the unlabeled instances, weighted using the classifiers' class probability estimates, for training in each iteration
  - Basic cotraining method assigns hard labels instead
- The *co-EM* method has also been used successfully with support vector machines, adapted to deal with weights
  - Logistic models are fit to the output of the SVMs to obtain class probability estimates
- Cotraining and co-EM even seem to work even when views are chosen randomly
  - Why? Possibly because cotrained classifier is more robust

Neural network approaches

- Semi-supervised learning can also be applied in deep learning of neural network classifiers
- Unsupervised pre-training is a form of semi-supervised learning in deep learning
  - Purely supervised deep learning is very effective when large amounts of labeled data are available
  - Unsupervised pre-training based on unlabeled data can be useful when labeled data is scarce
- It is also possible to extend autoencoders, which are unsupervised, to include supervision when available
  - Add branch to output layer of autoencoder that predicts class label
  - Apply composite loss function that measures both reconstruction performance and classification performance

Multi-instance learning

- Multi-instance learning can be interpreted as a form of weakly supervised learning
  - We do not get labels for the individual instances when learning, only labels for entire bags of instances
- An appealing approach to multi-instance learning is to transform the problem into a single-instance learning one
- We have already seen aggregation of *input* or *output* as very simple approaches to do this
  - These approaches often work surprisingly well in practice
- Will fail in some situations, at least in theory
  - Aggregating the input loses a lot of information because attributes are condensed to summary statistics individually and independently
  - Aggregating the output requires labeling all training instances with their bag's label, which may not be ideal
- Can we do better?

Converting to single-instance learning

- Can we condense each bag into a single instance without loosing as much as information as with basic aggregation?
- Yes, but more attributes are needed in the "condensed" representation
- Basic idea: partition the instance space into *regions*
  - One attribute per region in the single-instance representation
- Simplest case → Boolean attributes
  - Attribute corresponding to a region is set to true for a bag if it has at least one instance in that region
- Can use numeric counts instead of Boolean attributes to preserve more information
- Resulting instance summarizes the distribution of the original bag of instances in instance space

How to find suitable partitions?

- Main problem: how to partition the instance space?
- Simple approach → partition space into equal sized *hypercubes*
  - Only works for few attributes/dimensions
- More practical → use unsupervised learning
  - Take all instances from all bags (minus class labels) and cluster them
  - Create one attribute per cluster (region)
- But: clustering ignores the class membership
- Instead, use a decision tree to partition the space
  - Each leaf corresponds to one region of the instance space
- How to learn the tree when class labels apply to entire bags?
  - Method applied in *aggregating the output* can be used: take the bag's class label and attach it to each of its instances
  - Many labels will be incorrect, however, they are only used to obtain a partitioning of the space, not the final classification

Soft partitions

- Using *k*-means clustering or decision trees yields "hard" partition boundaries
- Can make region membership "soft" by using distance – transformed into similarity – to compute attribute values
  - Each attribute corresponds to the similarity of the bag of instances to a particular reference point in instance space
  - Requires a way of aggregating similarity scores between a bag and a reference point into a single value
  - For example, by taking the maximum similarity between each instance in a bag and the reference point
- Remaining question: which points should be the reference points?
- The *MILES* method uses all training instances from all bags as reference points
  - Generates a high-dimensional single-instance representation for a bag of instances

Multi-instance Learning

- Each individual example comprises a *set* of instances
  - All instances are described by the same attributes
  - One or more instances within an example may be responsible for its classification
- Goal of learning is still to produce a concept description
- Important real world applications
  - e.g. drug activity prediction

Example: Learning to Rank

- Query consists of multiple words (terms)
- Example: document containing one or more of the query terms
(= instances)

R	N	R	N
data	data		data
mining	mining	mining	
software		software	software

- Attributes for each query term in the document:
  - TI? - does word occur in doc. Title?
  - #occ - #occurences of word in document
  - #anchor? - does word occur in an anchor text?
  - IDF – inverse doc. frequency of word in collection
- Overall #attributes depends on #query_terms
-> single-instance method not applicable

Multi-instance learning

- Simplicity-first methodology can be applied to multi-instance learning with surprisingly good results
- Two simple approaches, both using standard single-instance learners:
  - Manipulate the input to learning
  - Manipulate the output of learning

Aggregating the input

- Convert multi-instance problem into single-instance one
  - Summarize the instances in a bag by computing mean, mode, minimum and maximum as new attributes
  - "Summary" instance retains the class label of its bag
  - To classify a new bag the same process is used
- Results using summary instances with minimum and maximum + support vector machine classifier are comparable to special purpose multi-instance learners on original drug discovery problem

Aggregating the output

- Learn a single-instance classifier directly from the original instances in each bag
  - Each instance is given the class of the bag it originates from
- To classify a new bag:
  - Produce a prediction for each instance in the bag
  - Aggregate the predictions to produce a prediction for the bag as a whole
  - One approach: treat predictions as votes for the various class labels
  - A problem: bags can contain differing numbers of instances → give each instance a weight inversely proportional to the bag's size

Output aggregation: LTR example

- Training query: "data mining software"
(Each instance is given the class of the bag it originates from)

R		N		R		N	
data	R	data	N			data	N
mining	R	mining	N	mining	R		
software	R			software	R	software	N

- Testing query: "decision tree"
(aggregating predictions via averaging of weights)

0.5		0.3		0.7		0.5	
decision	0.8	decision	0.4	decision	0.8	decision	0.6
tree	0.2	tree	0.2	tree	0.6	tree	0.4

Multi-instance learning

- Converting to single-instance learning
- Already seen aggregation of *input* or *output*
  - Simple and often work well in practice
- Will fail in some situations
  - Aggregating the input loses a lot of information because attributes are condensed to summary statistics individually and independently
- Can a bag be converted to a single instance without discarding so much info?

Labeling instances for multi-instance learning

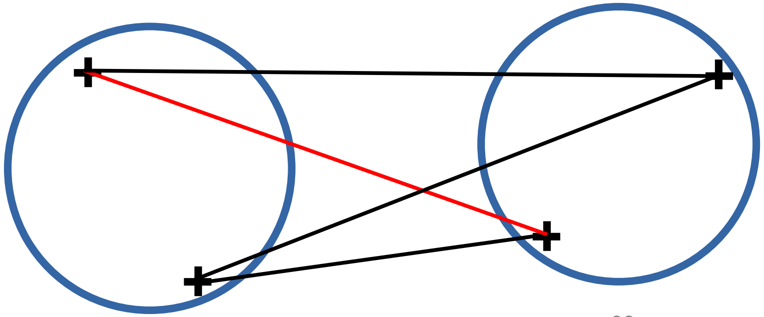
- An alternative approach to applying single-instance learning is to try to label all the instances in each bag
- Just assigning the bags' labels, as done in *aggregating the output*, often works well but is clearly naïve
- Instead an iterative approach has been proposed:
  1) Label each instance with its bag's label
  2) Learn a (single-instance) classifier
  3) Relabel the instances based on the classifier's class assignments
  4) Go back to 2) until convergence
- When the so-called *standard multi-instance assumption holds*, we know the following about our data:
  - All instances in a bag with a negative class label are truly negative
  - At least one instance in a positive bag is positive
- We can enforce these two constraints in Step 3 of the above iterative algorithm

Upgrading learning algorithms

- Converting to single-instance learning is appealing because many existing algorithms can be applied after conversion
  - But: may not be the most effective approach
- Alternative: upgrade single-instance algorithm to the multi-instance setting
  - Can be achieved elegantly for distance/similarity-based methods (e.g., nearest neighbor classifiers or SVMs)
  - All we need is a measure to compute distance/similarity between two bags of instances
  - Many other algorithms, e.g., tree and rule learners, require more substantial internal changes
  - However, models that are mathematical functions learned by optimizing a loss function can often be adapted quite easily
- We only consider distance/similarity-based methods here

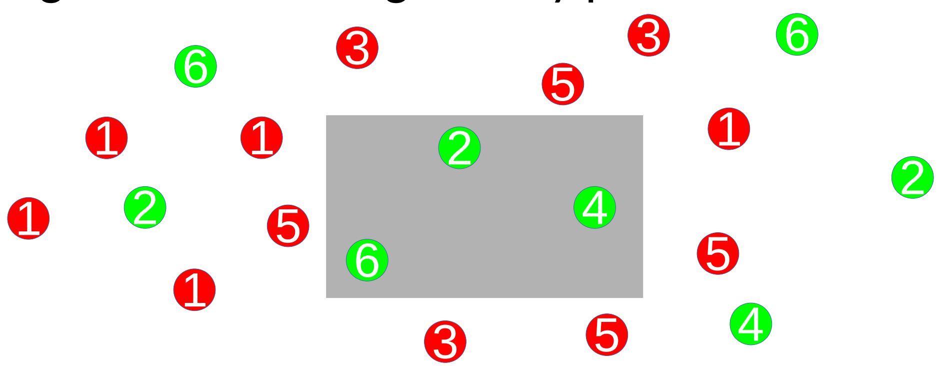
Upgrading similarity-based learning

- Kernel-based methods: similarity measure must be a proper kernel function to apply methods such as SVMs
- One example – so called *set kernel*
  - Given a kernel function for pairs of instances, the set kernel sums this kernel function over all pairs of instances from the two bags being compared
  - Can be applied with **any** single-instance kernel function
- Nearest neighbor learning: we can apply variants of the *Hausdorff distance* to find the nearest bag(s) for a particular test bag
  - Hausdorff distance is a distance measure define for sets of points
  - Given: two bags and a distance function between pairs of instances
  - Hausdorff distance: largest distance from any instance in one bag to its closest instance in the other bag
  - Can be made more robust to outliers by using the *n*th-largest distance



Dedicated multi-instance methods

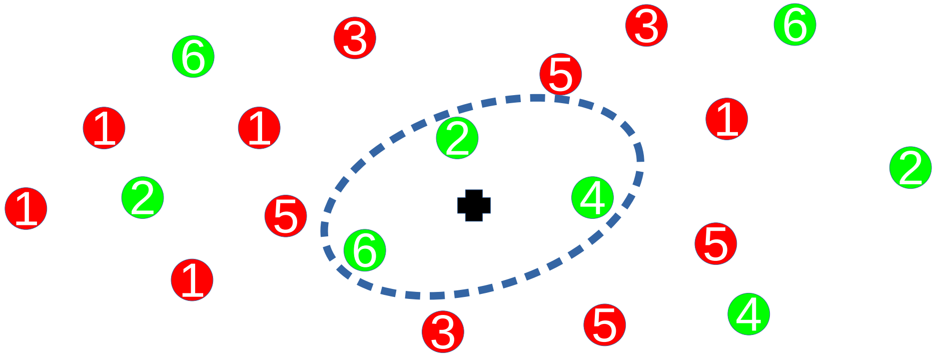
- Some well-known multi-instance learning methods are not based directly on single-instance algorithms
- One famous approach for learning with the standard multi-instance assumption
  → find a single hyperrectangle that contains
  - at least one instance from each positive bag and
  - no instances from any negative bags
- This rectangle will enclose an area of the instance space where all positive bags overlap
  - Originally designed for the drug activity problem mentioned in Chapter 2



- Can use other shapes – e.g., hyperspheres (balls)
  - Can also use boosting to build an ensemble of balls

Diverse density learning

- Previously described methods have hard decision boundaries – an instance either falls inside or outside a hyperrectangle/ball
- *Diverse-density* learning uses a probabilistic approach to learn a model for the standard multi-instance assumption:
  - Learns a single reference point in instance space
  - The probability that an *instance* is positive decreases with increasing distance from the reference point (by applying a bell-shaped distribution)
  - Instances' probabilities are combined using the "noisy-OR" (probabilistic version of logical OR) to obtain the probability that a *bag* is positive
  - All instance-level probabilities 0 → bag-level probability is 0
  - At least one instance-level probability is 1 → bag-level probability is 1



Diverse density learning

- Gradient descent can be used to maximize the likelihood of the model
- Diverse density is maximized when the reference point is located in an area where positive bags overlap and no negative bags are present