Sherry Zhao's Individual Report

December 10, 2021

I implemented miVLAD and EM-DD algorithms for multiple instance learning. The algorithms are implemented using python. Numpy is imported for dealing with arrays, sklearn is imported for split training and testing data, calculating the evaluation metrics, and classification with svm. The datasets are in .csv files.

1 miVLAD

1.1 Implementation

miVLAD is a embedded-based, vocabulary-based multiple instance learning method. According to [2], in miVLAD, the algorithm use K-means clustering method to find K centroids, all the instance b_{ij} are assigned to the nearest centroids in vocabulary, and the centroids are stored in vocabulary $C = \{c_1, ..., c_k, ..., c_K\}$, where i indicates the i^{th} bag in the dataset, j indicates the j^{th} instance in a bag, and k indicates the k^{th} attribute of the centroid.
Then a mapping function M(B, C) = $\vec{\mathbf{v}}$ is used to map the instances to a K-dimensional feature vector, where B is a bag and V is vocabulary. $\vec{\mathbf{v}}_i = \{v_{i1}, ..., v_{ik}, ...v_{in}\}$ is the new feature vector obtained from the mapping function, and each bag is represented in a $\vec{\mathbf{v}}_i$. To map each bag to a feature vector, the function is:

$$v_{ikl} = \sum_{b_{ij} \in \{b_{ij} | NN(b_{ij}) = c_k\}} b_{ijl} - c_{kl}$$

$b_j - c_k$ indicates the likelihood of an instance belonging to the c_k class. v_{ikl} represents the l^{th} attribute mapped to the k^{th} centroid in v for bag i. b_{ijl} represents the l^{th} attribute in j^{th} instance of bag i, and c_{kl} represents the l^{th} attribute in the kth centroid.
Because some of the attributes can be so large that the significance of other attributes is reduced, which is called burstiness. In order to get an accurate prediction, miVLAD needs to reduce burstiness in data. Intra normalization is used to reduce the burstiness, in the miVLAD algorithm in this project, the feature vector is normalized by sign square root (SSR) followed by l_2-nomalization:

$$SSR: \ \mathbf{v}_i \leftarrow sign(\mathbf{v}_i)\sqrt{\mathbf{v}_i}$$

$$l_2 - nomalization: \mathbf{v}_i \leftarrow \frac{\mathbf{v}_i}{||\mathbf{v}_i||_2}$$

After obtaining the normalized feature vector, we use a supervised classifier SVM to train a model. The training set is T = where v is the feature vector we get from the mapping function, y_i is the label of training bag B_i. The training function can be represented as:

$$f(X) = g(M(X,V))$$

where f is miVLAD algorithm and g is svm classification.
When predicting a bag, we map the bags to the vocabulary we learned in the training process to find feature vectors, normalize the feature vectors with SSR and l_2-nomalization, then put them into the SVM to predict the label of each bag.

1.2 Result

	musk1	musk2	tiger	elephant	fox
Accuracy	0.821±0.078	0.776±0.085	0.822±0.044	0.828±0.065	0.613±0.069
Precision	0.795±0.085	0.735±0.149	0.800±0.062	0.853±0.049	0.598±0.071
Recall	0.897±0.100	0.820±0.133	0.860±0.061	0.832±0.110	0.721±0.091
AUC	0.825±0.075	0.797±0.085	0.824±0.046	0.836±0.061	0.617±0.068

From the result table, musk1, musk2, tiger, and elephant have similar accuracy, precision, recall, and auc than other dataset. The performance of miVLAD on fox dataset is lower than others. Two extensions are implemented for miVLAD.

1.3 Extension — Soft Assignment

In miVLAD, the vocabulary is essential for classification. The performance of clustering method influnce the performance of the entire algorithm because the classification is based on the vocabulary learned from clustering. This extension is trying to improve the performance of clustering. K-means is a hard assignment method. One instance is assigned to one centroid, and it's sensitive to noise in data. For extension of miVLAD, a soft assignment method can be used to cluster the instances, which allows each data to belong to more than one class. For some data that are overlapped between classes, fuzzy c-means clustering should perform bettern than k-means clustering. Fuzzy c-means clustering is implemented as the soft assignment method. The goal of fuzzy c-means clustering is to minimize:

$$J(\Gamma, V) = \sum_i \sum_j (\gamma_{ij}^m ||x_i - v_j||^2)$$

where v is the centroids, γ is the membership values, m is the fuzziness index, and $||x_i - v_j||^2$ calculates the Euclidean distance. In fuzzy c-means clustering, k centoids are initiated and the

following functions are repeated until the functions converge:

$$v_j = \frac{\sum_i (\gamma_{ik}^m \cdot x_k)}{\sum_i \gamma_{ik}^m}$$

$$\gamma_{ij} = \left[\sum_k \left(\frac{d_{ki}^2}{d_{kj}^2} \right)^{\frac{1}{m-1}} \right]^{-1}$$

where d is the Euclidean distance. Fuzzy c-means algorithm replaced k-means clustering in this extention to build the vocabulary, and here are the results:

	musk1	musk2	tiger	elephant	fox
Accuracy	0.817±0.061	0.787±0.068	0.843±0.057	0.850±0.048	0.604±0.071
Precision	0.765±0.069	0.820±0.181	0.833±0.076	0.863±0.075	0.598±0.087
Recall	0.922±0.078	0.623±0.182	0.888±0.054	0.841±0.081	0.698±0.114
AUC	0.824±0.069	0.764±0.071	0.843±0.056	0.851±0.047	0.609±0.063

The performance of miVLAD with soft assignemnt is similar to miVLAD with hard assignment. The change of accuracy of musk1 and fox are less then 1%, and though the increase of accuracy of musk2, tiger, and elephant is small. Because the change is too small, it is inconclusive that if the soft assignemnt can improve the performance a little or it cause by other reasons like getting better training set during cv split.
From the two performance table for miVLAD, it shows that in most of the experiment, the recall is higher than precision, which meeans that it is easier for miVLAD to give false positive than false negative.

2 EM-DD

2.1 Implementation

Expectation-Maximization Diverse Density (EM-DD) is an instance-based multiple instance learning method. EM-DD is based on the SMI assumption, which states that there is at least one positive instance in a positive bag, and there is no positive instance in a negative bag. A point is chosen to be the target concept, and the target concept should be a positive point. According to [1], DD is the measure of the distance of positive bags and negative bags to the target concept point. DD makes the assumption that the bags are conditionally independent. DD is defined by:

$$DD(t) = \frac{P(\mathbf{B}|y=t)P(y=t)}{P(\mathbf{B})}$$

where **B** is the bags, and y is the labels. To find the true concept, DD needs to be maximized. Because the prior probability $P(y=t)$ is uniform and constant and the probability of evidence $P(\mathbf{B})$ is also constant, maximizing likelihood can also maximize DD, applying the Baysian rule:

$$arg\max_x \prod_i P(x=t|B_i^+) \prod_i P(x=t|B_i^-)$$

where B_i^+ indicates a positive bag, and B_i^- indicates a negative bag. The instances in positive bags should be close to the target concept and the instances in negative bags should be far away from the target concept. The most-likely-model is used to calculate the $\prod_i P(x=t|B_i^+) \prod_i P(x=t|B_i^-)$ term in above equation.The model is calculated by:

$$P(x=t|B_i^+) = arg\max_j P(x=t|B_{ij}^+)$$

$$P(x=t|B_i^-) = 1 - arg\max_j P(x=t|B_{ij}^-)$$

$P(x=t|B_i^+)$ is the probability of the instance being positive, and $P(x=t|B_i^-)$ is the probability of an instance being negative. Because there are two classes among the bags and assuming the probabilities of positive and negative bags are the same before we look at the attributes, the threshold is set to 0.5. If the highest probability among the instances in the bag is greater than or equal to 0.5, the bag is classified as positive, and if none of the instances in the bag has a probability greater than or equal to 0.5, the bag is classified as negative. Euclidean distance is used to calculate the distance between an instance and hypothesis, and the probability of the bag belongs to positive bag or negative bag can be represented by a Gussian-like distribution:

$$P(x=t|B_{ij}) = \exp(-\sum_k s_k^2 (B_{ijk} - x_k)^2)$$

where x is the target point vector, and s is the scaling vector for the target point vector. B_{ijk} represents the k^{th} attribute of the j^{th} instance in the ith bag. Combining x and s, the target concept = $[[x_1,...,x_k],[s_1,...,s_k]]$, which is the vector that EM-DD algorithm looks for.

In the expectation step, the distances between each instance and the hypothesis are calculated, and the instance representing the bag the most is selected as a presentative instance, which means the instance closest to the hypothesis is selected in positive bags, and the instance most far away from the hypothesis is selected in negative bags.

In the maximization step, an optimal algorithm is needed to find the maximum DD. According to [3], finding the maximum DD equals finding the minimum of negative log DD (NLDD). Gradient descent is used to minimize the NLDD. EM-DD is very time consuming because gradient descent has to take one step at a time, so not all the instances in the training bags are used as a start point to find global minimum. Because gradient descent is likely to be stuck in local minimums, multiple start points are selected from the bags in the training data to help find the global minimum.

Here is the results of EM-DD algorithm using instances in difference number of largest positive bags as start points to classify Musk1:

	all	1	2	3	5
Accuracy	0.721±0.125	0.532±0.136	0.674±0.137	0.686±0.102	0.674±0.100
Precision	0.704±0.138	0.254±0.314	0.610±0.238	0.657±0.212	0.680±0.105
Recall	0.834±0.165	0.400±0.490	0.783±0.310	0.712±0.336	0.830±0.196
AUC	0.719±0.117	0.548±0.081	0.669±0.116	0.671±0.082	0.650±0.085
Runtime	763.13±0.281	114.448±107.816	170.878±103.739	259.90±133.93	438.890±240.396

From the experiment, the accuracy of EM-DD is very depend on the starting point of hypothesis. If a starting point near optimal is chosen, we can get high accuracy in prediction. If a starting point far away from optimal is chosen, it is less likely for the gradient cannot bring the hypothesis to the optimal. However, using all the positive points as starting points is too time comsuming for the algorithm. To balance the accuracy and time comsuming problem, instances in 3 largest bags in the training set is used as starting point of hypothesis.

2.2 Result

	musk1	tiger	elephant	fox
Accuracy	0.686±0.102	0.672±0.121	0.675±0.056	0.545±0.078
Precision	0.657±0.212	0.493±0.327	0.600±0.218	0.507±0.206
Recall	0.712±0.336	0.559±0.371	0.672±0.240	0.585±0.219
AUC	0.671±0.082	0.669±0.119	0.664±0.072	0.547± 0.078

The overall performance of EM-DD is not as good as miVLAD. It is a weak classifier and its accuracy of most dataset is between 60% - 70%. According to experiments in [1], the accuracy of EM-DD should be higher. The reason caused the poor performance might be difference in parameters. If good parameters are used, like a more suitble learning rate or maximum iteration in gradient descent, the performance will probably be better. Because EM-DD is very time comsuming, trying for parameters will takes a large mount of time, I have only tried modifying the number of staring point for hypothesis as above.

2.3 Extension — Feature Selection

The feature selection method compute the ANOVA F-value by:

$$F = \frac{variation\,between\,sample\,means}{variation\,within\,the\,samples}$$

and choose the features with high f-values. Sklearn is used to calculate f-value and feature selection.

	musk1	tiger	elephant	fox
Accuracy	0.679±0.126	0.660±0.118	0.653±0.068	0.543±0.070
Precision	0.616±0.117	0.615±0.235	0.607±0.218	0.532±0.075
Recall	0.820±0.242	0.637±0.251	0.517±0.284	0.582±0.106
AUC	0.669± 0.122	0.658±0.098	0.640±0.094	0.545± 0.069

Runtime of Musk1 is 219.64±98.873 seconds, which is slightly shorter than EM-DD algorithm without feature selection. The change of accuracy is not significant. For tiger, elephant, and fox, the difference between precision and recall is reduced, which means the probability of false positive and false negtive are reduced.

I haven't implemented other extension for EM-DD due to the limitation of time, but I have some ideas to improve the EM-DD algorithm. The biggest problem of EM-DD from my view is its runtime, it needs hours to learn a model, which is not efficient. I can think of two solutions for it. The first is to reduce the size of data, for example using PCA to reduce the dimension or use correlation to reduce the number of attributes to make the algorithm run faster. For the second method, because gradient descent uses first derivative test, the path to optimum can be zigzag and long, if a second derivative optimization algorithm is used, maybe the function can converge to the optimum faster, resulting in reducing the runtime.

References

[1] L. Dong. A comparison of multi-instance learning algorithms. 2006.

[2] X. S. Wei, J. Wu, Zhou, and Z. H. Scalable algorithms for multi-instance learning. *IEEE transactions on neural networks and learning systems*, 24(4):975–987, 2016.

[3] Q. Zhang and S. A. Goldman. An improved multiple-instance learning technique. *Advances in neural information processing systems (pp. 1073-1080)*, 2002.