CSE250B Programming Project 1: Prototype selection for nearest neighbor

Kevin Tan

February 4, 2019

1. **High-level description for prototype selection**

   Prototype selection (PS) algorithms select a subset S of the original training set TR. S is defined such that $|S| < |TR|$ and $S \subseteq TR$. Then we can use S instead of TR for classification, such that 1-NN classification with S can classify the examples almost as accurately as 1-NN does with TR.

   In this assignment, I will implement a PS algorithm inspired by condensed nearest neighbor (CNN) [1] which is designed to reduce the dataset size for k-NN classification. Condensation refers to a technique for the type of search carried out by a PS algorithm, which aims to retain the points which are closer to the decision boundaries, also called the border points. Intuitively, the internal points do not affect the decision boundaries as much as the border points, and thus can be removed (*absorbed*) with relatively little effect on classification accuracy.

2. **Pseudocode**

Algorithm 1 Prototype selection with CNN

Inputs:

TR: the labeled training set with $(\boldsymbol{x_i}, y_i)$ pairs, $i = 1...n$

M: the size of the selected prototype subset, $M < |TR|$

Outputs:

S: the selected prototype subset of TR, $|S| = M$

shuffle TR ▷ random traversal ordering of the dataset for each experiment
initialize the set S
insert first sample from TR into S
while $|S| < M$ **do**
    fit 1-NN classifier to the current contents of S
    for each $(\boldsymbol{x_i}, y_i)$ in TR **do**
        if 1-NN$(\boldsymbol{x_i}; S) \neq y_i$ **then** ▷ 1-NN$(\boldsymbol{x_i}; S)$ is the nearest prototype given the contents of S
            remove x from TR and add it to S
            break
return S

3. **Experimental results**

To evaluate the quality of our prototype selection algorithm, we compare its performance to that of a uniform-random selection. For $M = 10000, 5000, 1000$, we construct a prototype subset and random selection, both of size M, and then compare their performance using a 1-NN classifier. We implement data processing, Algorithm 1 (CNN), and the code for running multiple experiments in Python and NumPy. In our implementation, we use the KNeighborsClassifier [2] with Minkowski distance ($p = 2$) from sklearn for the 1-NN classifier (lines 5,7 in Algorithm 1) as the prototype selection rule. Note that our implementation induces randomness due to the initial shuffling of TR (line 1).

However, in practice, we found that it takes a very long time for large values of M, e.g $M = 10000$, so instead we help alleviate this by only fitting the 1-NN based on the current contents of S when $|S|$ increases by 50 (as opposed to 1). Furthermore, we also saved computational costs by limiting the number of chosen prototypes, by running Algorithm 1 until $|S|$ was a small fraction of M (say, 1/10), and then filling the rest of $M - |S|$ with randomly selected points, until $M = |S|$.

We run 8 experiments for all values of M, and report the results with the mean, followed by std dev, in the table below.

	M=1000		M=5000		M=10000	
	mean	std	mean	std	mean	std
random selection	0.8851	0.0027	0.9354	0.0017	0.9490	0.0011
ours (CNN)	0.8869	0.0043	0.9354	0.0025	0.9496	0.0012

4. **Critical Evaluation**

Unfortunately, it turns out that our method is not a clear improvement against random selection. The results suggest that our method is, for the most part, as good as a random one.

The biggest limitation to our experiments was that we had to restrict ourselves to a small fraction of carefully chosen border prototypes (1/10 of M) due to computational costs. From prior experiments on smaller datasets (e.g. $M = 50, 100$), we extrapolate and hypothesize that if we were to carry out our CNN experiments in full, we would observe improvements in classification accuracy of around 2% to 5%.

Future work could focus on implementing Fast CNN [3] or Generalized CNN [4], which are slightly more advanced algorithms that claim to reduce the learning time complexity to being quadratic time in the worst case.

References

[1] Hart, P. E. (1968). The condensed nearest neighbor rule. IEEE Transactions on Information Theory 14, 515-516.

[2] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E., *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, vol. 12, 2011.

[3] F Angiulli, Fast Condensed Nearest Neighbor, ACM International Conference Proceedings, Vol 119, pp 25-32.

[4] Chien-Hsing C, Bo-Han K, Fu C (2006) The generalized condensed nearest neighbor rule as a data reduction method. In: Proceedings of the 18th international conference on pattern recognition. IEEE Computer Society, Hong-Kong, pp 556559