Programming Assignment 1

K-NN AND POCKET ALGORITHM

Course: CIS4526 - Foundations of Machine Learning

Professor: Kai Zhang

Name: Hieu Khuong

TUID: 915399644

Email: hieukhuong@temple.edu

1. Abstract

Using the Letter Recognition Data Set from UCI Machine Learning Repository, which has 20,000 vectors consisting of a letter at the head of vector and numbers as attributes for the rest of the vector, to train 2 algorithm: k-Nearest Neighbors and Pocket Algorithm.

2. Algorithm

The dataset of 20,000 vectors was divided into 2 parts: the training set consisting of 15,000 vectors and the test set consisting of 5,000 vectors. The code that the professor provided already map the letter to numbers and divided the dataset into data vectors and label vectors.

a) K-NN Algorithm:

K-NN algorithm will proceed in the following order:

- Load the training and testing dataset.
- Calculate all the distances for every data point in testX to all points in trainX.
- Select the majority number of class to decide the result, put the result into pred_y.
- Compare the testY and pred_y vectors, calculate the accuracy percentage.

b) Pocket Algorithm:

Pocket Algorithm and OVA Algorithm will proceed in the following order:

- Initialize a weight vector for Perceptron Learning Algorithm (PLA).
- Choose a number of iterations for PLA.
- For each letter number (for example: A - 0, B - 1, etc), convert the label of trainY to 1 and -1. 1 is for the letter number we chose, and -1 for the others.
- Train the PLA algorithm with the initialized weight, update the weight every time hitting a misclassified point.
- Run the pocket algorithm: after a certain number of iterations, keep the weight which provide best accuracy for that letter number. Add this weight to an array.
- After doing pocket algorithm for all letter numbers, use the weight array to predict the label for testX. The label will be decided by the point with maximum value when multiplying it with all the weights. For example: if point testX[i] * weights[2] is the maximum value, then testX[i] will be predicted as letter number 2.
- Compare and compute accuracy of pred_y to testY.

3. Performance

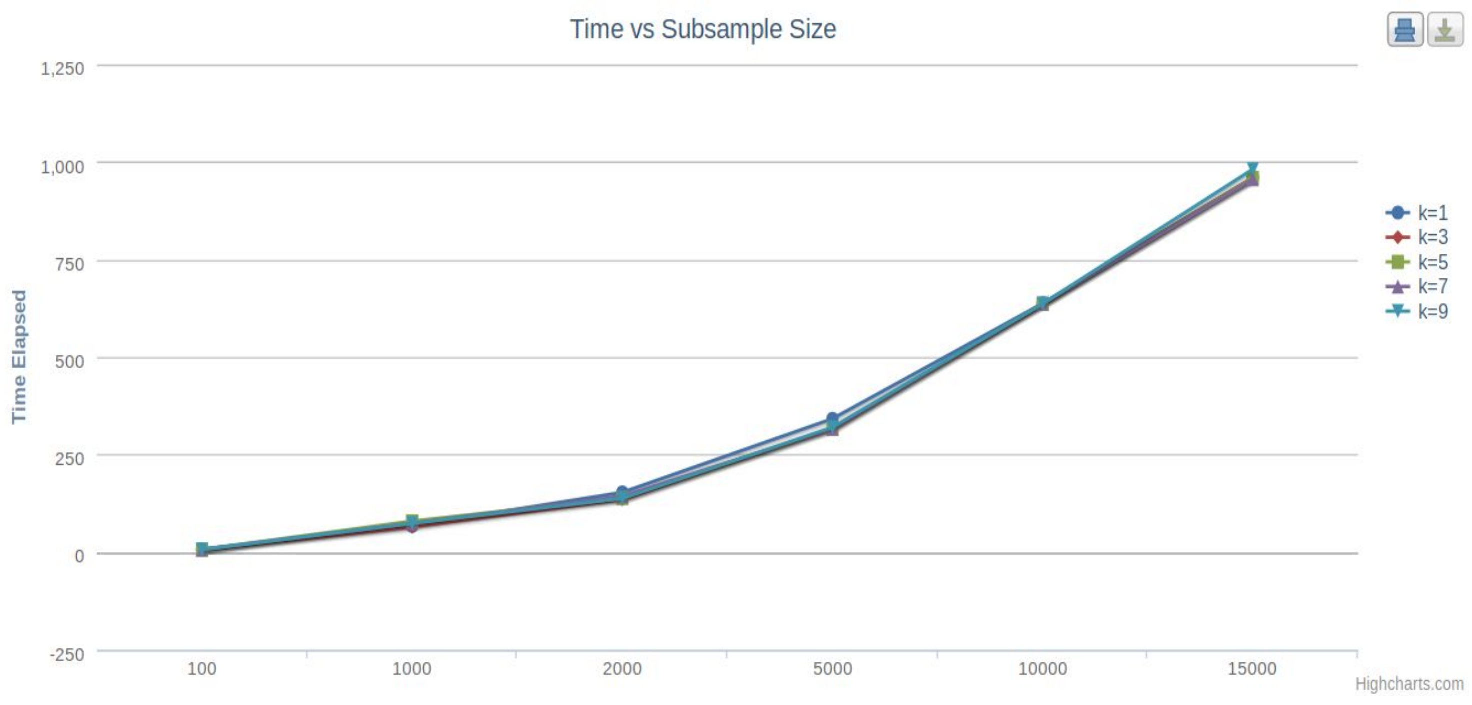
There are a total of 36 experiments, executed with different subsample training size and algorithm:

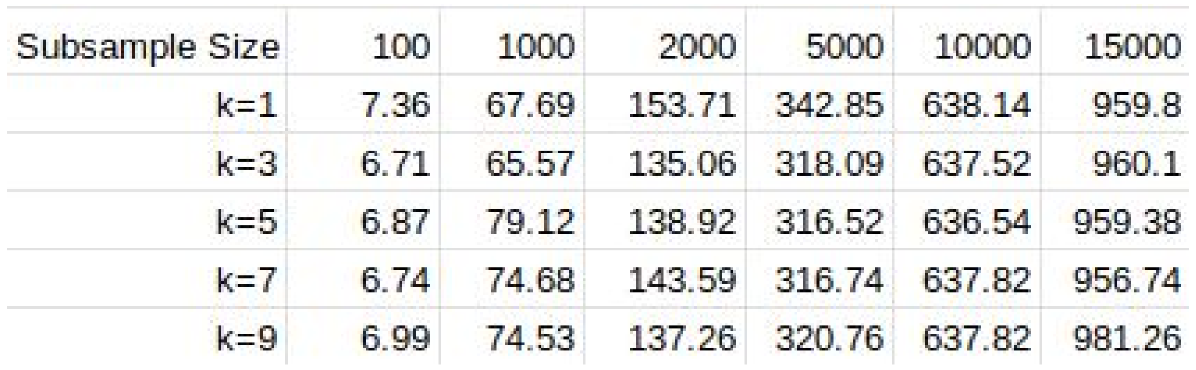
- The 6 subsample size values for training set are: [100, 1000, 2000, 5000, 10000, 15000]
- We run 6 algorithms: [1-NN, 3-NN, 5-NN, 7-NN, 9-NN, Pocket]

(k-NN denotes k Nearest Neighbors)

a) This is the performance of k-NN algorithm:

This is the graph for the k-NN time performance:

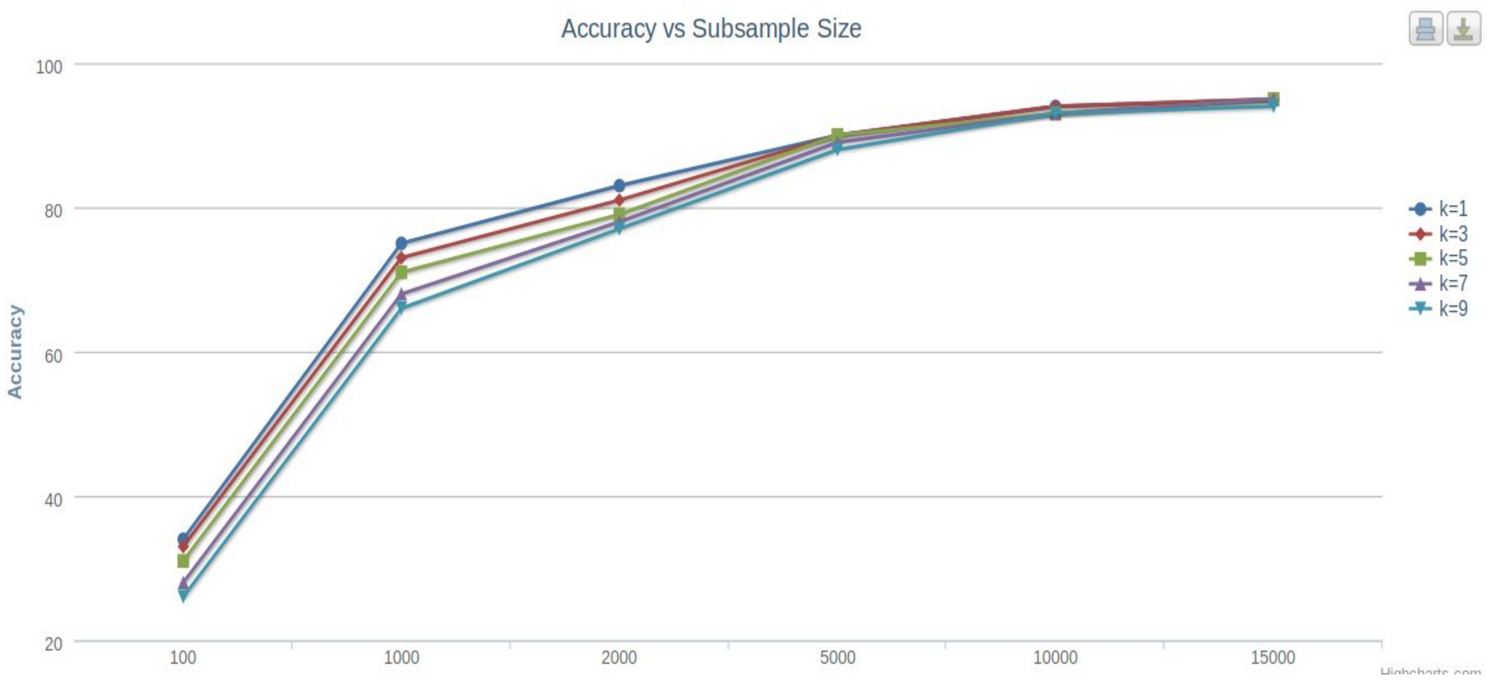


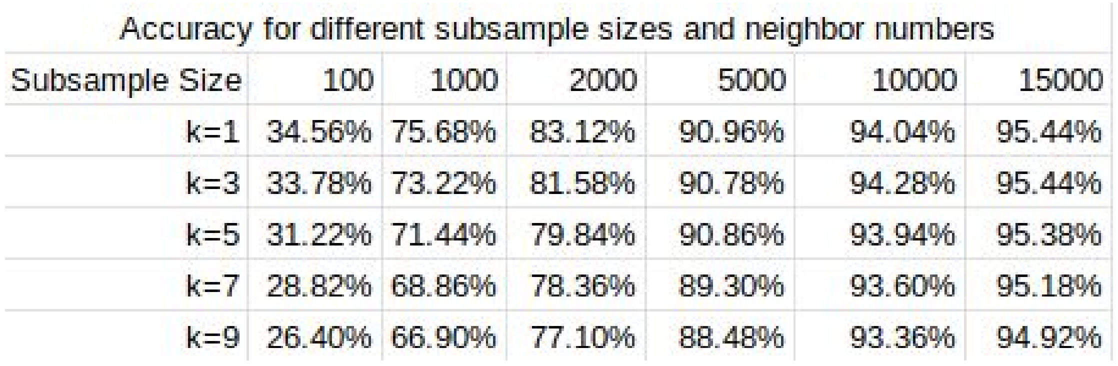


Subsample Size	100	1000	2000	5000	10000	15000
k=1	7.36	67.69	153.71	342.85	638.14	959.8
k=3	6.71	65.57	135.06	318.09	637.52	960.1
k=5	6.87	79.12	138.92	316.52	636.54	959.38
k=7	6.74	74.68	143.59	316.74	637.82	956.74
k=9	6.99	74.53	137.26	320.76	637.82	981.26

As we can see, the number of neighbor does not have too much influence on the performance of the k-NN. The execution time mostly depends on the subsample training size: the more data points in subsample, the more points will be calculated to find the majority label.

This is the graph for the k-NN time performance:

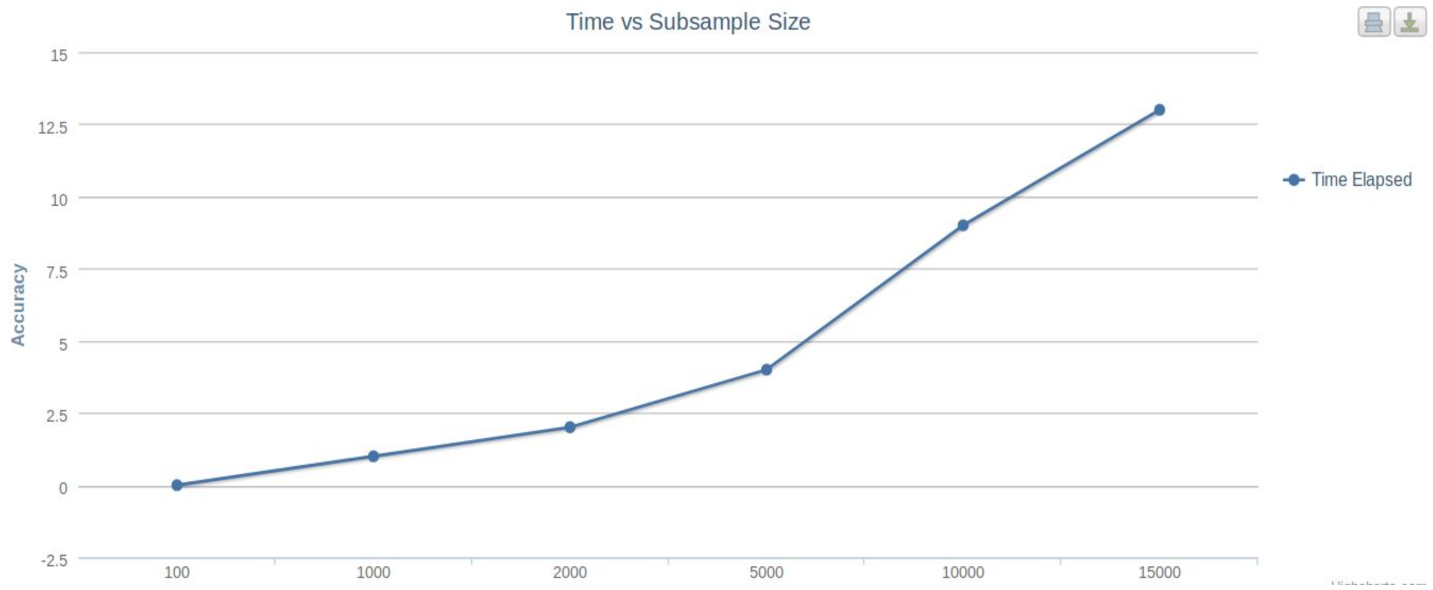


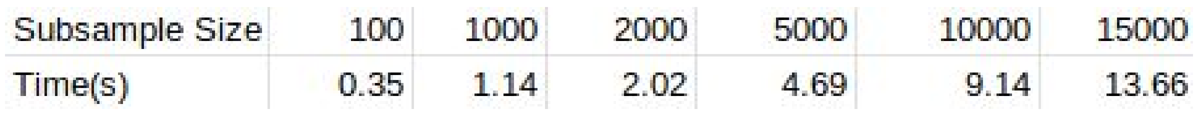

Accuracy for different subsample sizes and neighbor numbers

Subsample Size	100	1000	2000	5000	10000	15000
k=1	34.56%	75.68%	83.12%	90.96%	94.04%	95.44%
k=3	33.78%	73.22%	81.58%	90.78%	94.28%	95.44%
k=5	31.22%	71.44%	79.84%	90.86%	93.94%	95.38%
k=7	28.82%	68.86%	78.36%	89.30%	93.60%	95.18%
k=9	26.40%	66.90%	77.10%	88.48%	93.36%	94.92%

From the graph and the table, we can see that as subsample size increases, accuracy increases. This makes perfect sense because as the bigger training data set get, the better our algorithm will decide the label for the test points. Also, by looking at the neighbor number, we can see that k=3 perform the best among other k when the subsample size is big (10000, 15000). When the subsample size is 100, 1000, 2000, 5000, k=1 seems to perform best. So we can conclude that in order to keep the accuracy high, we should not choose k to be big, or subsample training size to be small.Big value of k can cause distraction, as many points in the k nearest neighbor may be the noise to the decision making process.

b) This is the performance of Pocket Algorithm and OVA classifier:
I choose the number of iterations for Pocket Algorithm to be 10.
This is the time taken to run Pocket and OVA with different subsample size

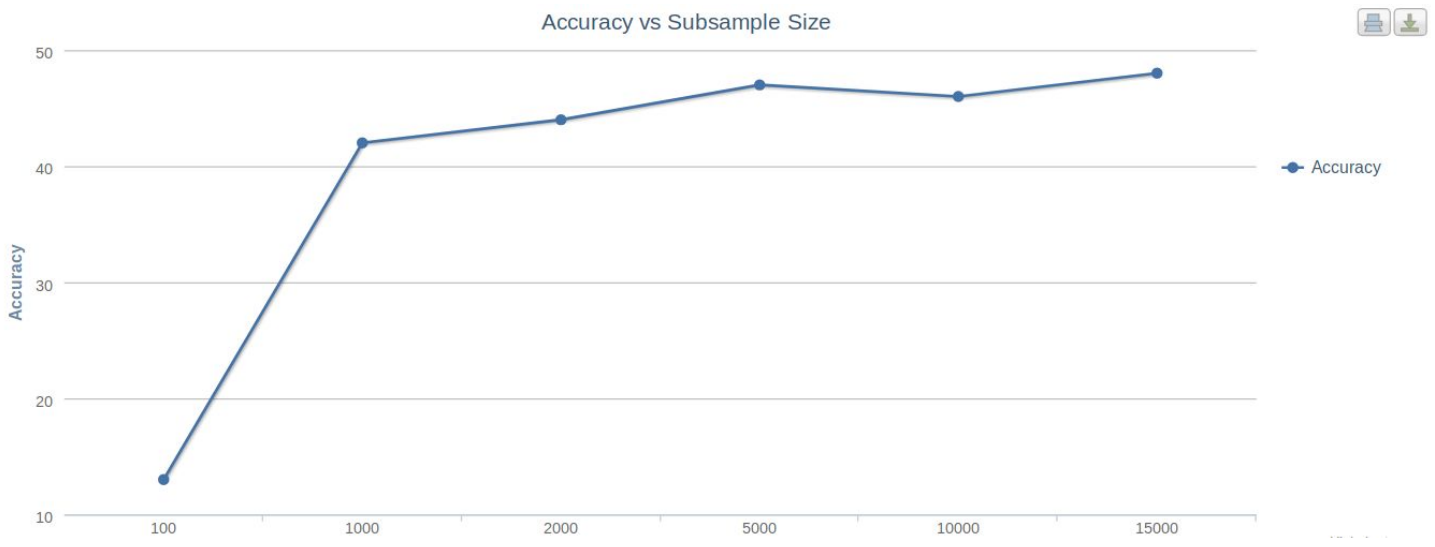




Subsample Size	100	1000	2000	5000	10000	15000
Time(s)	0.35	1.14	2.02	4.69	9.14	13.66

The time taken to run Pocket Algorithm was much less than it took to run k-NN. This is easy to understand because we do not need to calculate all the distances like the kNN algorithm. Instead we only need to run a number of iterations, which is much faster. Also the label are converted into 1 and -1, which makes it very fast to calculate and compare.

This is the accuracy of Pocket and OVA:



Subsample Size	100	1000	2000	5000	10000	15000
Accuracy	13.54%	42.84%	44.92%	47.26%	46.42%	48.64%

Pocket and OVA algorithm may take much less time to finish, but they produce lower accuracy. As subsample size get bigger, the accuracy does not increase proportionally. We can see in the graph that the accuracy somehow gets stuck at 46-48%, so I can predict that if the subsample size increase to 20000, or 30000, the accuracy may not increase much: it may only reach about 50%. I think the reason that Pocket is not good at classifying multiple class is that Pocket Algorithm use Perceptron, which is mostly used to classify binary class. OVA may divide the sample space by many 'lines', or hyperplane, so it would not be accurate.

4. Extra:

I created a confusion matrix of kNN algorithm, running with subsample size of 2000 and 3 neighbor.

		Actual																								
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Predicted	A	182	1	0	3	0	0	0	1	1	0	1	0	0	1	0	0	0	1	1	0	8	0	0	2	4
	B	0	146	0	2	1	2	1	6	0	0	1	0	1	0	0	0	0	9	1	0	1	1	0	1	0
	C	0	0	153	0	4	0	5	2	0	0	2	0	0	0	2	1	0	0	0	0	2	0	0	0	0
	D	0	7	0	185	0	0	0	13	1	0	0	0	0	0	1	0	0	7	1	0	0	0	0	0	0
	E	0	3	6	0	134	1	13	3	0	0	7	0	0	0	0	1	0	1	6	0	0	0	1	9	0
	F	1	2	0	1	1	152	0	0	1	2	0	0	0	3	0	26	0	0	0	3	0	0	0	2	0
	G	0	6	3	2	0	1	162	4	0	1	1	0	2	0	9	1	10	3	0	0	0	0	2	1	0
	H	1	5	1	12	4	0	2	117	0	0	9	1	3	0	11	0	0	5	1	0	1	0	0	4	0
	I	0	1	0	0	0	2	0	1	172	15	3	0	0	0	0	3	0	0	3	0	0	0	0	5	0
	J	0	0	0	0	2	1	0	0	5	172	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0
	K	0	1	1	0	0	0	1	15	0	0	138	0	2	0	0	0	0	8	1	0	1	0	0	9	0
	L	2	1	2	0	1	0	1	0	0	2	1	180	0	0	0	3	1	1	1	0	2	0	0	7	0
	M	4	1	0	0	0	0	2	0	0	0	0	0	166	4	0	0	0	1	0	0	0	4	5	0	0
	N	0	0	0	8	0	0	0	2	0	0	3	0	6	153	5	0	1	9	0	0	0	10	1	0	0
	O	0	0	0	7	0	0	1	2	0	0	0	0	0	1	159	2	5	1	0	0	0	1	2	0	0
	P	0	5	0	6	0	19	0	4	0	0	0	0	1	1	1	168	0	0	0	0	0	1	0	0	1
	Q	2	1	0	5	0	0	5	1	0	0	0	0	0	0	30	5	165	1	0	0	0	2	0	0	0
	R	0	21	0	10	0	0	3	9	0	0	9	2	2	1	0	2	0	144	1	0	0	1	3	0	0
	S	4	3	1	5	7	4	0	2	0	0	2	1	0	0	0	0	1	2	155	0	1	0	0	1	1
	T	1	1	1	3	0	4	0	2	0	1	0	0	0	0	0	1	1	1	4	157	0	2	0	2	2
	U	0	8	0	2	0	0	0	6	0	1	0	0	2	3	8	0	0	1	1	1	181	0	1	0	0
	V	0	2	0	0	0	1	0	0	0	0	0	0	1	0	2	2	0	0	1	0	0	144	7	0	8
	W	0	0	1	0	0	0	0	0	0	0	1	0	4	6	1	1	0	4	0	0	0	3	145	0	1
	X	1	1	0	6	5	0	0	1	0	2	11	0	0	0	0	0	0	1	1	0	1	0	0	156	0
	Y	0	1	0	2	0	2	1	4	0	1	0	0	0	1	1	4	3	0	4	21	0	6	0	1	131
	Z	0	0	0	2	2	0	0	1	1	3	0	1	0	0	0	0	4	0	6	3	0	0	0	8	1