Enhancing Accuracy in KNN by using weights depending on Information gain and Gini impurity of attributes

Debadrita Roy, Jadavpur University

ABSTRACT

The standard K-Nearest Neighbours algorithm assumes that all the attributes of a dataset are equally important when classifying a test instance, which is not the case in reality. It cannot learn that one feature is more discriminative than another.
This paper proposes some different ways of weighting each attribute with respect to their information gain and gini impurity. We compare the accuracy of each weightage with the others and with the KNN algorithm available in the scikit-learn package of Python. We have used the Diabetes dataset and the Iris dataset for our analysis. After analysis, we have found that some ways of weightage work better than the others and mostly have more accuracy than the scikit-learn KNN algorithm.

OVERVIEW / PROBLEM STATEMENT

K-Nearest Neighbours algorithm is one of the simplest and most popular algorithms used in classification. It assumes that all data instances correspond to points in n-dimensional space, n being the number of the attributes of the data. The test data instances are classified according to the class to which most of its nearest neighbours belong, the number of the nearest neighbours considered for this being k, which must be specified at the beginning. The usual distance measures used for defining the nearest neighbours of an instance are as follows:

- Euclidean distance
- Manhattan distance
- Minkowski distance

All of the above distance measures give equal weightage to each attribute, assuming that each attribute is equally important for classifying a test instance. However, in practice, this is not so. Usually, some attributes are more important in making a decision than others. So, we propose some methods of weighing each attribute based on their information gain values and/or gini impurity values and using weighted Euclidean distance to define nearest neighbours.

RELATED WORK

Many algorithms have been proposed over the years to improve the standard KNN algorithm. They mainly belong to the following variants:

- Choosing a proper (optimised) value of k to reduce bias towards majority class

- Giving different weights to the attributes to assign different degrees of importance to them by either coding by hand or by cross-validation
- Weighting the selected k nearest neighbours by their distance from the test instance to find the class of the test instance

Wang Baobao, Mao Jinsheng, Shao Minru proposed an enhancement of the K-Nearest Neighbour Algorithm using Information Gain and Extension Relativity [3]. There they calculated the weightage as $\lambda_i = Gain(i) \times e^{Gain(i)} / \sum_{i=1}^{n} Gain(i)$ and used it to calculate the degree of importance of the attributes.
Another enhancement was proposed by Xinjiang Xiao and Huafeng Ding based on Weighted Entropy of Attribute Value [2]. There, they used the information gain value as it is to weight each attribute and showed that the accuracy of classification is enhanced.
Gini index or gini impurity has also been used as a feature selection and a weighting method, especially in text classification [5].

PROPOSED METHOD

KNN is an instance-based, lazy learning algorithm to classify data.
KNN Training algorithm:
For each training example (x , f (x)) , add the example to the list training examples
KNN Classification algorithm:
Given a query instance x_q to be classified,
Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q based on distance measure used.
Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

[1]
where δ(a, b) = 1 if a = b and where δ(a, b) = 0 otherwise.

One practical issue in applying the above algorithm is that the distance between instances is calculated based on all attributes. So, if we apply it to a problem in which each instance is described by 20 attributes, but only 2 of these attributes are relevant to determining the classification for the particular target function, then instances that have identical values for the 2 relevant attributes may nevertheless be distant from one another in the 20-dimensional instance space. As a result, the similarity metric will be misleading. The distance between neighbours will be dominated by a large number of irrelevant attributes. This difficulty, which arises when many irrelevant attributes are present, is sometimes referred to as the curse of dimensionality.

An approach to overcoming this problem is to weight each attribute differently when calculating the distance between two instances. The amount by which each axis should be stretched can be determined automatically using a cross-validation approach. However, the problem with this is that it takes time and necessitates having sufficient data for cross-validation.

In this paper, we have worked on finding the k nearest neighbours of the instance to be classified based on weighted euclidean distance, the weights of the attributes calculated using information gain and/or gini impurity of the corresponding attributes. In this way, the weights of the attributes can be calculated at the beginning and reused for each classification.

In this section, we will define information gain and gini impurity. Then in the next section, we propose our different attribute weighting methods.

INFORMATION GAIN:
In order to define information gain precisely, we begin by defining a measure commonly used in information theory, called entropy, that characterises the (im)purity or randomness of an arbitrary collection of examples. If a collection S has *n* classes of examples, then

$$Entropy = -\sum_{i=1}^{n} p_i log_2(p_i)$$

where, $p_i = probability\ of\ an\ example\ belonging\ to\ class\ i$

Information gain is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. More precisely, the information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S,\ A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where, Values(A) = set of all possible values for attribute A,
and S_v = subset of S for which attribute A has value v.

GINI IMPURITY:
Gini impurity or gini index was proposed by Breiman in 1984 and is widely used in CART algorithm for decision tree learning. It can be defined as follows:
If a collection S has *n* classes of examples, then

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2$$

where $p_i = probability\ that\ an\ example\ belongs\ to\ class\ i$

We now propose and compare different attribute weighting methods based on information gain and/or gini impurity values.
Case #1: The information gain values of the attributes are used as weightages as they are.

$$w_1(A) = info.gain(A)$$

Case #2: The weight of attribute A is calculated as

$$w_2(A) = \frac{info.gain(A) - info.gain_{min}}{info.gain_{max} - info.gain_{min}}$$

Case #3: The weight of attribute A is calculated as

$$w_3(A) = \frac{info.gain(A)}{info.gain_{min}}$$

Case #4: The weight of attribute A is calculated as

$$w_4(A) = 1 - gini(A)$$

Case #5: The weight of attribute A is calculated as

$$w_5(A) = \frac{w_4(A) - w_{4_{min}}}{w_{4_{max}} - w_{4_{min}}}$$

Case #6: The weight of attribute A is calculated as

$$w_6(A) = \frac{w_4(A)}{w_{4_{min}}}$$

Case #7: The weight of attribute A is calculated as

$$w_7(A) = \frac{1}{gini(A)}$$

Case #8: The weight of attribute A is calculated as

$$w_8(A) = \frac{w_7(A) - w_{7_{min}}}{w_{7_{max}} - w_{7_{min}}}$$

Case #9: The weight of attribute A is calculated as

$$w_9(A) = \frac{w_7(A)}{w_{7_{min}}}$$

Case #10: The weight of attribute A is calculated as

$$w_{10}(A) = w_1(A) \times w_4(A)$$

Case #11: The weight of attribute A is calculated as

$$w_{11}(A) = w_2(A) \times w_5(A)$$

Case #12: The weight of attribute A is calculated as

$$w_{12}(A) = w_3(A) \times w_6(A)$$

Case #13: The weight of attribute A is calculated as

$$w_{13}(A) = w_1(A) \times w_7(A)$$

Case #14: The weight of attribute A is calculated as

$$w_{14}(A) = w_2(A) \times w_8(A)$$

Case #15: The weight of attribute A is calculated as

$$w_{15}(A) = w_3(A) \times w_9(A)$$

Case #16: The weight of attribute A is calculated as

$$w_{16}(A) = \frac{w_{13}(A) - w_{13_{min}}}{w_{13_{max}} - w_{13_{min}}}$$

Case #17: The weight of attribute A is calculated as

$$w_{17}(A) = \frac{w_{13}(A)}{w_{13_{min}}}$$

The above weights are used in the weighted Euclidean distance function to define nearest neighbours. A data point is of the form (x_{A_1}, x_{A_2}, …, x_{A_n}) for n attributes / features.

$$d = \sqrt{\sum_{i=1}^{n}[w(A_i) \times (x_{A_i} - x^{test}_{A_i})^2]}$$

where,

$n \quad = number\ of\ attributes/features\ for\ the\ data\ points$

$w(A_i) = weight\ of\ the\ corresponding\ attribute\ according\ to\ the\ scheme\ chosen$

$x_{A_i} \quad = value\ of\ the\ training\ data\ point\ w.r.t\ A_i$

$x^{test}_{A_i} = value\ of\ the\ test\ data\ point\ w.r.t\ A_i$

Once we get the distances for all training data points, we can determine which are the k nearest neighbours to the test data point. The class of training data points which dominates them is chosen as the class for the test data point.

Case #18 in the following observation tables gives the accuracy of the scikit-learn KNN algorithm for the same k and train-test split.

EVALUATION METRIC

Accuracy has been used to compare the different methods. It is given by

$$Accuracy = \frac{TP + TN}{TP+TN+FP+FN}$$

where, $TP = no.\ of\ actually\ positive\ instances\ which\ are\ classified\ as\ positive$

$TN = no.\ of\ actually\ negative\ instances\ which\ are\ classified\ as\ negative$

$FP = no.\ of\ actually\ negative\ instances\ which\ are\ classified\ as\ positive$

$FN = no.\ of\ actually\ positive\ instances\ which\ are\ classified\ as\ negative$

EXPERIMENTAL DATA ANALYSIS AND RESULTS

We have tested the enhanced algorithm with the different weight measures on the diabetes and the iris datasets to determine their feasibility and performance. For the following tables, each observation was taken 5 times and then averaged to get the accuracy. The algorithms were tested with different values of k and with different train-test splits.

COMPARISON OF ACCURACY OF THE DIFFERENT WEIGHTS

- TRAIN-TEST SPLIT = 50:50

  Dataset: Diabetes

	K=1	K=3	K=5	K=7	K=9	K=11	K=13	K=15
#1	67.34	72.14	73.7	73.7	74.69	72.92	74.01	72.97
#2	66.82	71.88	73.44	73.49	73.33	74.06	73.59	72.45
#3	67.45	72.03	73.65	73.7	74.64	72.86	74.22	72.71

#4	66.46	69.69	73.33	72.5	74.38	72.97	73.12	72.6
#5	67.19	72.08	73.54	73.54	73.23	74.06	73.18	72.14
#6	66.15	70.05	73.23	72.55	73.7	72.97	72.97	72.5
#7	66.41	69.74	73.28	72.4	74.27	72.76	73.07	72.71
#8	67.03	71.41	73.65	73.75	73.33	73.8	73.23	72.55
#9	66.15	70.05	73.23	72.55	73.7	72.97	72.97	72.5
#10	67.19	72.19	73.44	73.75	74.38	72.71	73.54	72.34
#11	66.41	71.82	73.18	74.06	73.39	73.54	73.85	71.72
#12	67.45	72.03	73.65	73.7	74.64	72.86	74.22	72.71
#13	67.24	71.88	73.33	73.8	74.11	72.86	73.65	72.45
#14	65.73	71.77	72.81	74.06	73.59	73.33	74.06	72.03
#15	67.45	72.03	73.65	73.7	74.64	72.86	74.22	72.71
#16	66.93	71.46	73.28	73.8	73.49	73.91	73.7	72.19
#17	67.24	72.03	73.65	73.85	73.96	72.71	73.59	72.4
#18	66.15	70.05	73.23	72.55	73.7	72.97	72.97	72.5

Dataset: Iris

	K=1	K=3	K=5	K=7	K=9	K=11	K=13	K=15
#1	96.53	96.27	96	96.53	96.27	95.47	96.27	96.27
#2	96.53	96.8	96.53	97.33	96.27	96.27	96.27	96
#3	96.27	96.53	96.27	96.8	96	95.73	96.53	96.27
#4	96	96.8	96	96.53	96.53	94.93	95.73	96.53
#5	96.53	96.53	96.27	97.33	96.27	96	96.27	96
#6	96	96.53	95.47	96.53	96.53	94.13	95.2	96
#7	96.27	96.8	96	96.53	96.8	95.73	96	96.27
#8	96.53	96.27	96.27	97.33	96.27	96.27	96	95.73
#9	96	95.73	96.8	96.53	96.53	95.73	95.47	96.27

#10	96	96	96.53	96.53	96.27	95.2	96.27	96.27
#11	97.33	96	96.53	97.33	95.73	96.27	96	95.73
#12	96.53	96.53	96	96.8	96	95.73	96.53	96.27
#13	96.27	95.73	96.53	96.53	96.27	95.73	96.27	96
#14	97.07	96	96.53	97.33	95.47	96.27	96.8	95.73
#15	96.27	95.73	96.27	96.53	96	96	95.73	96.27
#16	96.53	96	96.27	97.33	96	96.27	96	95.47
#17	96.27	96	96.53	96.53	96.27	95.73	96	96
#18	96	96.53	94.93	96.53	96.27	94.13	95.2	96

From the above tables, we see that if we use measures 1, 6, 9, 12, 13 and 15, we get higher or equal accuracy than the standard KNN algorithm in most situations.

- TRAIN-TEST SPLIT = 60:40
  Dataset: Diabetes

	K=1	K=3	K=5	K=7	K=9	K=11	K=13	K=15
#1	68.51	74.35	73.70	76.62	76.95	76.95	77.92	77.27
#2	70.45	74.35	74.68	75.65	75.65	73.38	75.00	75.97
#3	67.53	73.38	74.03	75.65	75.32	76.62	77.60	77.60
#4	67.21	68.83	72.40	72.73	73.38	75.65	75.97	75.65
#5	70.45	74.35	74.68	75.32	75.00	73.70	74.35	75.97
#6	67.21	69.48	72.40	72.40	73.05	74.03	75.97	75.97
#7	66.88	69.16	71.75	72.08	74.35	75.97	76.30	75.97
#8	70.78	75.65	74.68	75.00	74.35	74.68	75.00	75.00
#9	67.21	69.48	72.40	72.40	73.05	74.03	75.97	75.97
#10	68.83	75.65	74.35	77.27	75.65	76.95	78.25	77.60
#11	71.10	75.32	75.32	75.97	74.35	75.32	75.32	74.35
#12	67.53	73.38	74.03	75.65	75.32	76.62	77.60	77.60
#13	68.83	75.32	74.68	77.60	75.32	76.95	77.27	77.92

#14	70.45	75.65	76.30	75.97	74.03	75.32	74.67	74.67
#15	67.53	73.38	74.03	75.65	75.32	76.62	77.60	77.60
#16	71.10	75.00	74.68	75.32	75.00	75.0	74.02	76.62
#17	68.51	75.65	76.30	77.27	76.95	76.95	77.27	75.97
#18	67.21	69.48	72.40	72.40	73.05	74.03	75.97	75.97

Dataset: Iris

	K=1	K=3	K=5	K=7	K=9	K=11	K=13	K=15
#1	95	95.33	95.67	96	95.33	96.33	96	95.33
#2	94.67	96	96	95.67	95.33	95.67	95.67	95
#3	95	95.67	96	96	95	96.33	95.67	95.33
#4	94	95.33	95.33	95.67	95	96	95.33	94
#5	94.67	96	96	95.67	95.67	95.67	96	95
#6	94.33	95.33	95.33	94.67	95.67	96	95	94
#7	94	95.67	95.33	95.67	95.33	97	95	94.33
#8	95	96	96	96	95.67	95.67	95.67	95
#9	94.33	95.67	95.67	96	95.33	97	95	95.33
#10	95	95.67	96.33	96.67	95.33	96	96	95.33
#11	94.67	96	96.67	96	95.67	95.67	95.67	95.33
#12	95	95.67	96	96	95	96	96	95.33
#13	95	95.67	96	96.33	95.33	96	96	95.33
#14	94.67	96	96	95.33	95.67	95.33	95.67	95
#15	94.67	95.67	96	96.67	96	95.67	96	95.33
#16	95	96	96	96	95.67	96	95.67	95
#17	94.67	95.67	96	96.33	95.33	96	96	95.33
#18	94	95.33	95.33	94.67	95.33	96	95.33	94

From the above tables, we see that if we use measures 1, 3, 6, 9, 10, 12, 13, 15 and 17, we get higher or equal accuracy than the standard KNN algorithm in most situations.

- TRAIN-TEST SPLIT = 75:25

  Dataset: Diabetes

	K=1	K=3	K=5	K=7	K=9	K=11	K=13	K=15
#1	68.44	70.83	74.9	73.44	74.79	75.94	76.25	75.62
#2	68.54	71.35	72.6	72.08	75.73	74.48	75.52	75
#3	69.37	70.21	74.38	73.33	74.27	75.94	75.31	76.25
#4	68.85	67.92	70.83	73.23	74.27	72.4	76.15	74.9
#5	68.13	71.67	72.4	71.98	75	74.27	75.31	75.31
#6	69.06	67.92	71.15	73.54	74.58	72.71	75.42	75.21
#7	68.85	67.92	70.73	72.92	74.48	72.5	76.15	75.42
#8	67.81	71.77	72.08	71.88	75	74.27	74.79	75.52
#9	69.06	67.92	71.15	73.54	74.58	72.71	75.42	75.21
#10	68.75	71.15	75.1	73.33	74.48	75.42	76.04	76.25
#11	68.02	69.79	71.67	72.6	73.44	74.69	74.06	75.52
#12	69.37	70.21	74.38	73.33	74.27	75.94	75.31	76.25
#13	69.06	71.15	74.9	73.12	74.48	75.83	76.25	76.46
#14	67.92	69.9	71.35	73.23	73.02	74.69	73.75	75.62
#15	69.37	70.21	74.38	73.33	74.27	75.94	75.31	76.25
#16	68.44	71.35	72.29	72.6	75.21	74.48	75.1	75.21
#17	68.54	71.35	74.48	73.33	73.65	75.83	75.94	76.15
#18	69.06	67.92	71.15	73.54	74.58	72.71	75.42	75.21

  Dataset: Iris

	K=1	K=3	K=5	K=7	K=9	K=11	K=13	K=15
#1	93.68	97.89	96.84	93.16	95.79	95.79	95.79	96.32
#2	94.21	97.37	96.84	93.68	95.79	96.32	95.26	96.32
#3	93.68	97.89	96.84	93.16	95.79	95.79	95.79	96.32

#4	93.68	97.37	96.84	94.74	95.79	96.32	96.32	96.84
#5	94.21	97.37	96.84	93.68	95.79	96.32	95.26	96.32
#6	93.68	97.37	96.84	93.68	95.79	95.79	95.79	96.84
#7	93.68	97.37	96.84	94.21	95.79	96.32	96.32	96.84
#8	94.21	97.37	97.37	93.68	95.79	96.32	95.26	95.79
#9	93.68	97.89	96.84	93.68	95.79	96.32	95.26	97.37
#10	93.68	97.89	96.84	94.21	95.79	95.79	95.26	96.84
#11	94.74	97.89	97.37	93.68	95.79	96.84	95.26	95.79
#12	93.68	97.89	96.84	93.16	95.79	95.79	95.79	96.32
#13	93.68	97.37	96.84	94.21	95.79	95.79	95.26	96.84
#14	95.79	97.89	97.89	94.21	95.26	96.84	94.74	95.79
#15	93.68	97.37	96.84	94.21	95.79	96.32	95.26	97.37
#16	94.21	97.37	97.37	93.68	95.79	96.32	95.26	95.79
#17	93.68	97.37	96.84	94.21	95.79	96.32	95.26	96.84
#18	93.68	97.37	96.84	94.21	95.26	95.79	95.79	96.84

From the above tables, we see that if we use measures 3, 6, 9, 12, 13 and 15, we get higher or equal accuracy than the standard KNN algorithm in most situations.

Thus, considering the different train-test splits, we find that methods 6, 9, 12, 13 and 15 give the best results overall.

CONCLUSION

We tried to tackle the deficiency of the KNN algorithm regarding considering all attributes to be of equal importance and not considering that one attribute may be more discriminative than other. We proposed different weighting methods based on information gain and/or gini impurity of the attributes and compared their performance accuracy with each other as well as with the standard KNN algorithm. We found that almost all the measures perform better than the standard in most situations, with a few standing out as better than the others overall.

FUTURE SCOPE

We can weight the k nearest neighbours found using our proposed distance measures with the corresponding distances from the test data point when determining the majority class.

We can also use the proposed weighting methods with other distance measures like Manhattan distance, Minkowski distance, etc. Also, we can use the proposed distance measures in clustering algorithms like K- Means, K-Medoid, etc.

REFERENCES

1. *Mitchell, Tom M., and Tom M. Mitchell. Machine learning. Vol. 1. No. 9. New York: McGraw-hill, 1997.*
2. *Xiao, Xingjiang, and Huafeng Ding. "Enhancement of K-nearest neighbor algorithm based on weighted entropy of attribute value." 2012 5th International Conference on BioMedical Engineering and Informatics. IEEE, 2012.*
3. *Baobao, Wang, Mao Jinsheng, and Shao Minru. "An enhancement of K-Nearest Neighbor algorithm using information gain and extension relativity." 2008 International Conference on Condition Monitoring and Diagnosis. IEEE, 2008.*
4. *Taneja, Shweta, et al. "An enhanced k-nearest neighbor algorithm using information gain and clustering." 2014 Fourth International Conference on Advanced Computing & Communication Technologies. IEEE, 2014.*
5. *Shang, Wenqian, et al. "An adaptive fuzzy knn text classifier based on gini index weight." 11th IEEE Symposium on Computers and Communications (ISCC'06). IEEE, 2006.*