WEIGHTED K NEAREST NEIGHBOR

Siddharth Deokar
CS 8751
04/20/2009
deoka001@d.umn.edu

Outline

- Background
- Simple KNN
- KNN by Backward Elimination
- Gradient Descent & Cross Validation
  - Instance Weighted KNN
  - Attribute Weighted KNN
- Results
- Implementation
- DIET

Background

- K Nearest Neighbor
  - Lazy Learning Algorithm

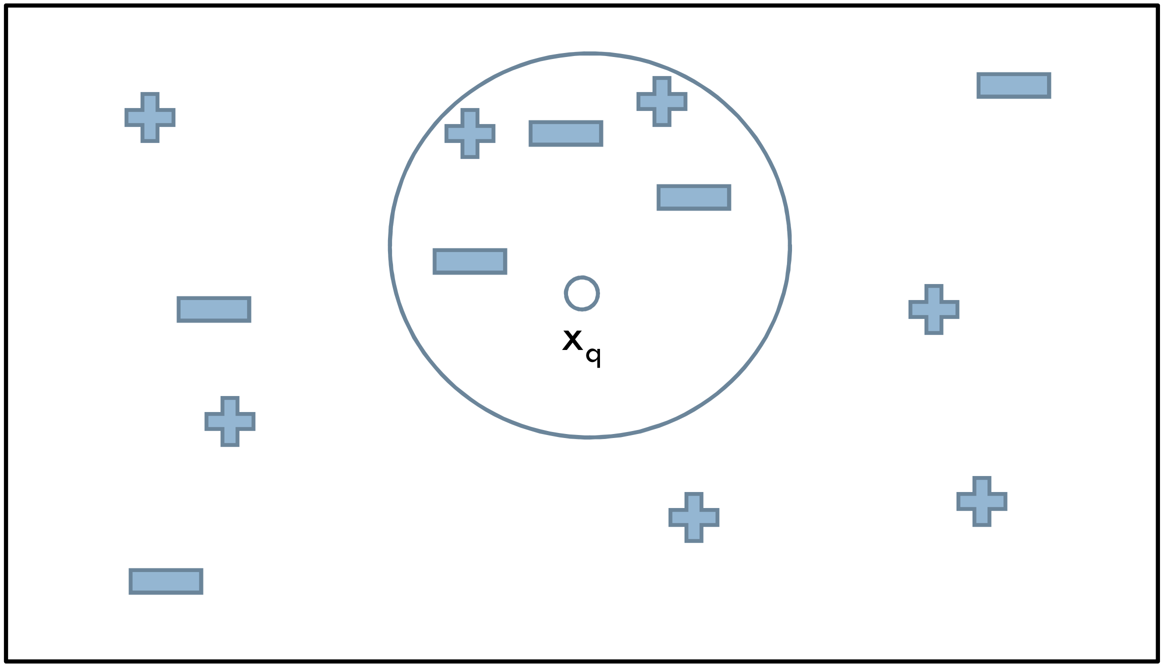
    Defer the decision to generalize beyond the training examples till a new query is encountered
  - Whenever we have a new point to classify, we find its K nearest neighbors from the training data.
  - The distance is calculated using one of the following measures
    - Euclidean Distance
    - Minkowski Distance
    - Mahalanobis Distance

Simple KNN Algorithm

- For each training example <x,f(x)>, add the example to the list of training_examples.

- Given a query instance x_q to be classified,
  - Let $x_1, x_2 \ldots x_k$ denote the k instances from training_examples that are nearest to x_q .
  - Return the class that represents the maximum of the k instances.

KNN Example



If K = 5, then in this case query instance x_q will be classified as negative since three of its nearest neighbors are classified as negative.

Curse of Dimensionality

- Distance usually relates to all the attributes and assumes all of them have the same effects on distance
- The similarity metrics do not consider the relation of attributes which result in inaccurate distance and then impact on classification precision. Wrong classification due to presence of many irrelevant attributes is often termed as the curse of dimensionality
- For example: Each instance is described by 20 attributes out of which only 2 are relevant in determining the classification of the target function. In this case, instances that have identical values for the 2 relevant attributes may nevertheless be distant from one another in the 20 dimensional instance space.

Weighted K Nearest Neighbor

- Approach 1
  - Associate weights with the attributes
  - Assign weights according to the relevance of attributes
    - Assign random weights
    - Calculate the classification error
    - Adjust the weights according to the error
    - Repeat till acceptable level of accuracy is reached
- Approach 2
  - Backward Elimination
  - Starts with the full set of features and greedily removes the one that most improves performance, or degrades performance slightly

Weighted K Nearest Neighbor

- Approach 3 (Instance Weighted)
  - Gradient Descent
  - Assign random weights to all the training instances
  - Train the weights using Cross Validation
- Approach 4 (Attribute Weighted)
  - Gradient Descent
  - Assign random weights to all the attributes
  - Train the weights using Cross Validation

Definitions

- Accuracy
  - Accuracy = (# of correctly classified examples / # of examples) X 100

- Standard Euclidean Distance
  - $d(x_i, x_J) = \sqrt{(\text{For all attributes } a \sum (x_{i,a} - x_{J,a})^2)}$

Backward Elimination

- For all attributes do
  - Delete the attribute
  - For each training example x_i in the training data set
    - Find the K nearest neighbors in the training data set based on the Euclidean distance
    - Predict the class value by finding the maximum class represented in the K nearest neighbors
    - Calculate the accuracy as

    Accuracy = (# of correctly classified examples / # of training examples) X 100
  - If the accuracy has decreased, restore the deleted attribute

Weighted K-NN using Backward Elimination

- Read the training data from a file <x, f(x)>
- Read the testing data from a file <x, f(x)>
- Set K to some value
- Normalize the attribute values in the range 0 to 1.
  - Value = Value / (1+Value);
- Apply Backward Elimination
- For each testing example in the testing data set
  - Find the K nearest neighbors in the training data set based on the Euclidean distance
  - Predict the class value by finding the maximum class represented in the K nearest neighbors
  - Calculate the accuracy as
    - Accuracy = (# of correctly classified examples / # of testing examples) X 100

Example of Backward Elimination

- # training examples 100
- # testing examples 100
- # attributes 50
- K 3
- Simple KNN
  - Accuracy/Correctly Classified Examples (training set) = 56 with all the 50 attributes
  - Accuracy/Correctly Classified Examples (test set) = 51 with all the 50 attributes
- Applying the backward elimination, we eliminate 16 irrelevant attributes
  - Accuracy/Correctly Classified Examples (training set) = 70 with 34 attributes
  - Accuracy/Correctly Classified Examples (test set) =64 with 34 attributes

Instance Weighted K-NN using Gradient Descent

- Assumptions
  - All the attribute values are numerical or real
  - Class attribute values are discrete integer values
    - For example: 0,1,2…..
- Algorithm
  - Read the training data from a file <x, f(x)>
  - Read the testing data from a file <x, f(x)>
  - Set K to some value
  - Set the learning rate α
  - Set the value of N for number of folds in the cross validation
  - Normalize the attribute values in the range 0 to 1
    - Value = Value / (1+Value)

Instance Weighted K-NN using Gradient Descent Continued…

- Assign random weight w_i to each instance x_i in the training set
- Divide the number of training examples into N sets
- Train the weights by cross validation
  - For every set N_k in N, do
    - Set N_k = Validation Set
    - For every example x_i in N such that x_i does not belong to N_k do
      - Find the K nearest neighbors based on the Euclidean distance
      - Calculate the class value as
        - $\sum w_k \times x_{j,k}$ where j is the class attribute
      - If actual class != predicted class then apply gradient descent
        - Error = Actual Class – Predicted Class
        - For every W_k
          - $W_k = W_k + \alpha \times \text{Error}$
    - Calculate the accuracy as
      - Accuracy = (# of correctly classified examples / # of examples in N_k) X 100

Instance Weighted K-NN using Gradient Descent Continued…

- Train the weights on the whole training data set
  - For every training example x_i
    - Find the K nearest neighbors based on the Euclidean distance
    - Calculate the class value as
      - $\sum w_k \times x_{j,k}$ where j is the class attribute
    - If actual class != predicted class then apply gradient descent
      - Error = Actual Class – Predicted Class
      - For every W_k
        - $W_k = W_k + \alpha \times \text{Error}$
    - Calculate the accuracy as
      - Accuracy = (# of correctly classified examples / # of training examples) X 100
  - Repeat the process till desired accuracy is reached

Instance Weighted K-NN using Gradient Descent Continued…

- For each testing example in the testing set
  - Find the K nearest neighbors based on the Euclidean distance
  - Calculate the class value as
    - $\sum w_k \times x_{j,k}$ where j is the class attribute
- Calculate the accuracy as
  - Accuracy = (# of correctly classified examples / # of testing examples) X 100

Example with Gradient Descent

- Consider K = 3, α = 0.2, and the 3 nearest neighbors to x_q are x_1, x_2, x_3

K nearest neighbors	Euclidean Distance	Class	Random Weights
X_1	12	1	$W_1 = 0.2$
X_2	14	2	$W_2 = 0.1$
X_3	16	2	$W_3 = 0.005$

- Class of x_q = 0.2 X 1 + 0.1 X 2 + 0.005 X 2 = 0.41 => 0
- Correct Class of x_q = 1
- Applying Gradient Descent
- W_1 = 0.2 + 0.2 X (1 - 0) = 0.4
- W_2 = 0.1 + 0.2 X (1 - 0) = 0.3
- W_3 = 0.005 + 0.2 X (1 - 0) = 0.205
- Class of x_q = 0.4 X 1 + 0.3 X 2 + 0.205 X 2 = 1.41
- Class of x_q => 1
- Simple K-NN would have predicted the class as 2

Attribute Weighted KNN

- Read the training data from a file <x, f(x)>
- Read the testing data from a file <x, f(x)>
- Set K to some value
- Set the learning rate α
- Set the value of N for number of folds in the cross validation
- Normalize the attribute values by standard deviation
- Assign random weight wi to each attribute Ai
- Divide the number of training examples into N sets

Attribute Weighted KNN continued

- Train the weights by cross validation
  - For every set Nk in N, do
    - Set Nk = Validation Set
    - For every example xi in N such that xi does not belong to Nk do
      - Find the K nearest neighbors based on the Euclidean distance

$$d(x, y) = \sqrt{\sum_{j=1}^{d} w_j^2 (x_i - y_i)^2}$$

      - Return the class that represents the maximum of the k instances
      - If actual class != predicted class then apply gradient descent
        - Error = Actual Class – Predicted Class
        - For every Wk
          - Wk = Wk + α * Error * Vk (where Vk is the query attribute value)
    - Calculate the accuracy as
      - Accuracy = (# of correctly classified examples / # of examples in Nk) X 100

Attribute Weighted KNN continued

- Train the weights on the whole training data set
  - For every training example xi
    - Find the K nearest neighbors based on the Euclidean distance
    - Return the class that represents the maximum of the k instances
    - If actual class != predicted class then apply gradient descent
      - Error = Actual Class – Predicted Class
      - For every Wk
        - Wk = Wk + α * Error * Vk (where Vk is the query attribute value)
  - Calculate the accuracy as Accuracy = (# of correctly classified examples / # of training examples) X 100
  - Repeat the process till desired accuracy is reached
- For each testing example in the testing set
  - Find the K nearest neighbors based on the Euclidean distance
  - Return the class that represents the maximum of the k instances
  - Calculate the accuracy as
    - Accuracy = (# of correctly classified examples / # of testing examples) X 100

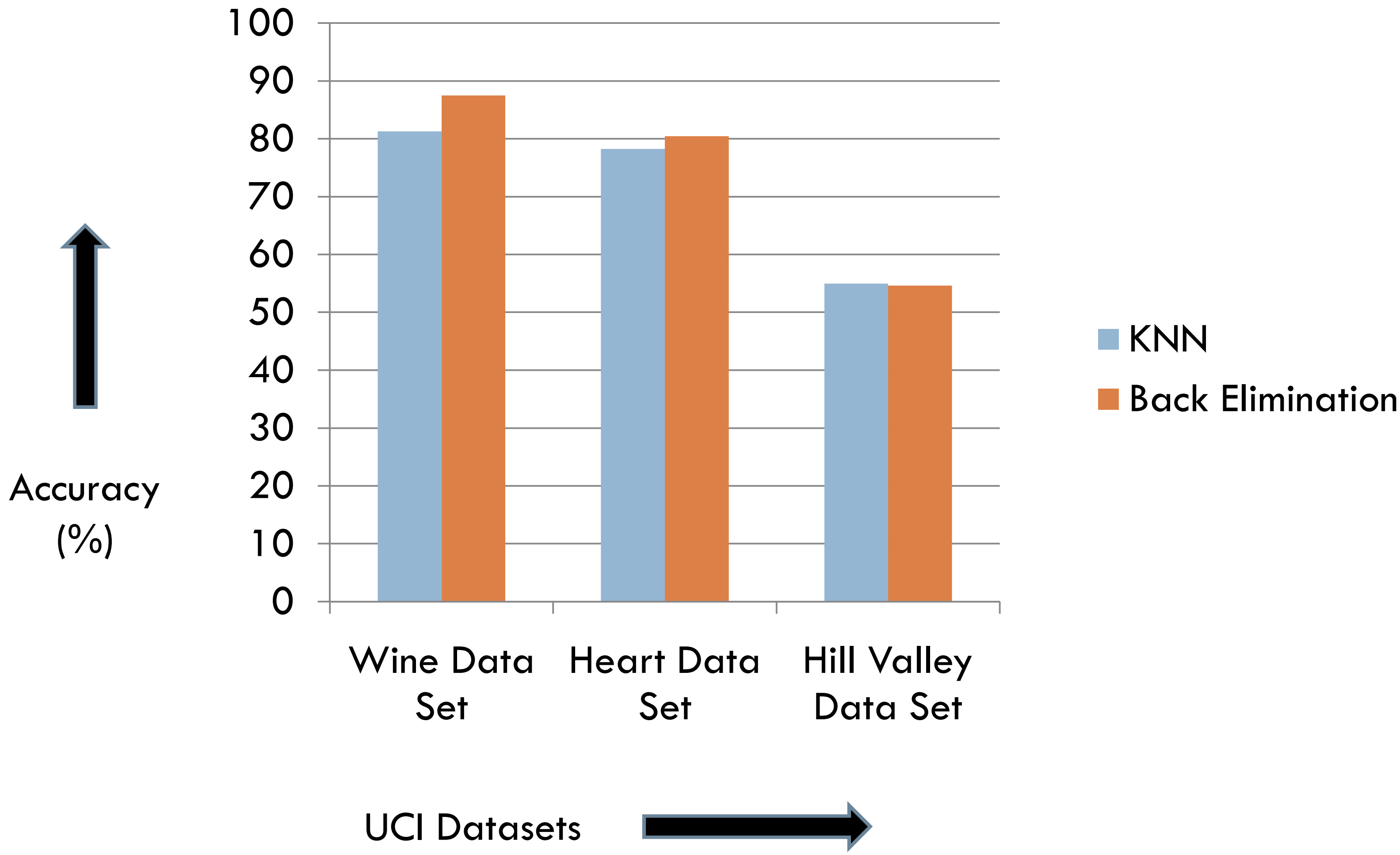
Results (KNN Vs Back Elimination)

Heart Data Set	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	2	NA	270	224	46	13	2	78.26
Back Elimination	2	NA	270	224	46	9	2	80.44

Wine Data Set	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	2	NA	178	146	32	13	3	78.26
Back Elimination	2	NA	178	146	32	4	3	80.44

Hill Valley Data Set	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	2	NA	1212	606	606	100	2	54.95
Back Elimination	2	NA	1212	606	606	94	2	54.62

Results (KNN Vs Back Elimination)

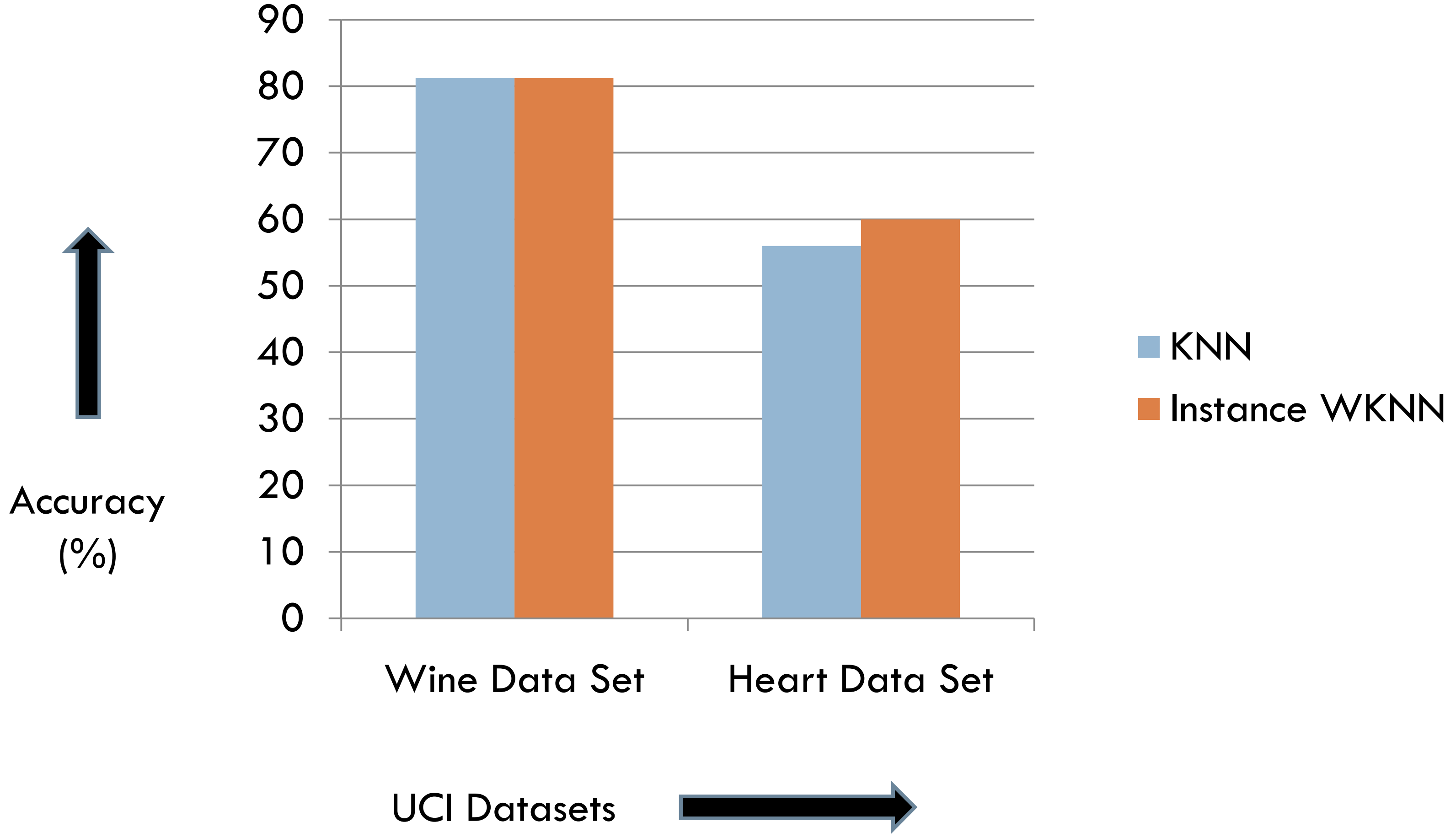


Results (KNN Vs Instance WKNN)

Heart Data Set - 1	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	2	NA	303	203	100	13	4	56
Instance WKNN	2	0.001	303	203	100	13	4	60

Wine Data Set	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	2	NA	178	146	32	13	3	81.25
Instance WKNN	2	0.005	178	146	32	13	3	81.25

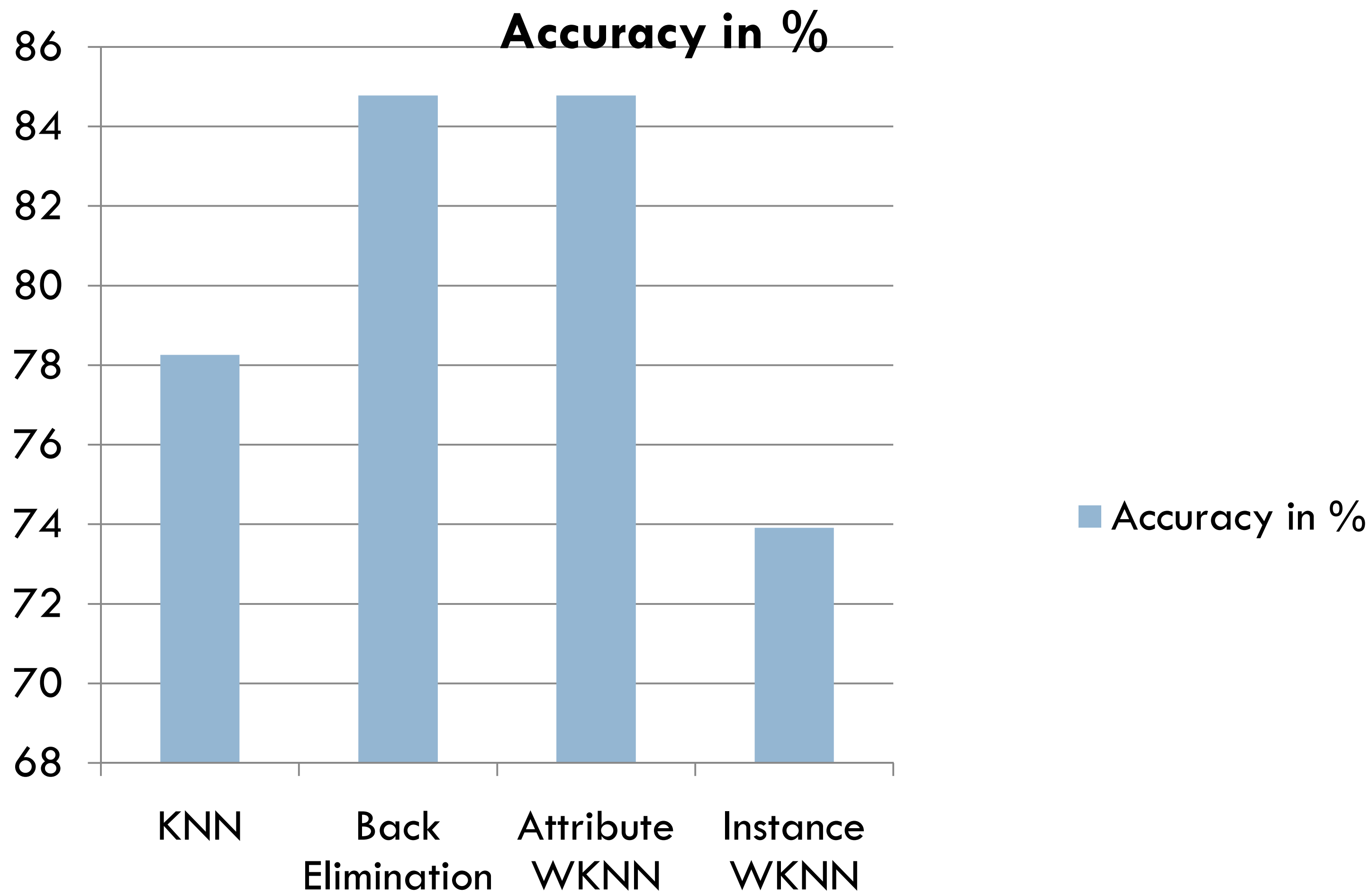
Results (KNN Vs Instance WKNN)



Results (Heart Data Set)

Heart Data Set	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	3	NA	270	224	46	13	2	78.26
Back Elimination	3	NA	270	224	46	11	2	84.78
Attribute WKNN	3	0.005	270	224	46	13	2	84.78
Instance WKNN	3	0.001	270	224	46	13	2	73.91

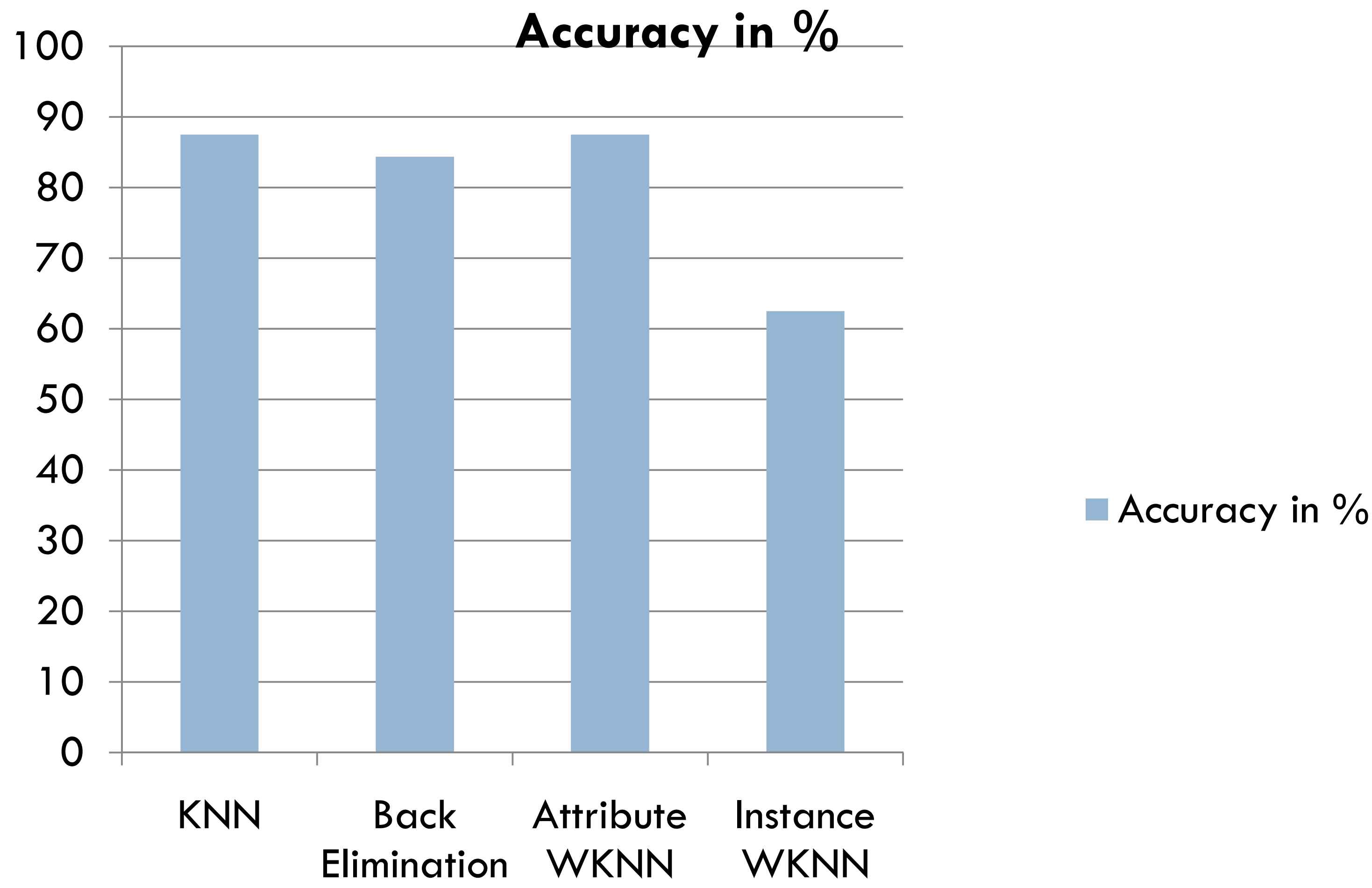
Results (Heart Data Set)



Results (Wine Data Set)

Wine Data Set	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	3	NA	178	146	32	13	3	87.5
Back Elimination	3	NA	178	146	32	10	3	84.38
Attribute WKNN	3	0.005	178	146	32	13	3	87.5
Instance WKNN	3	0.005	178	146	32	13	3	62.5

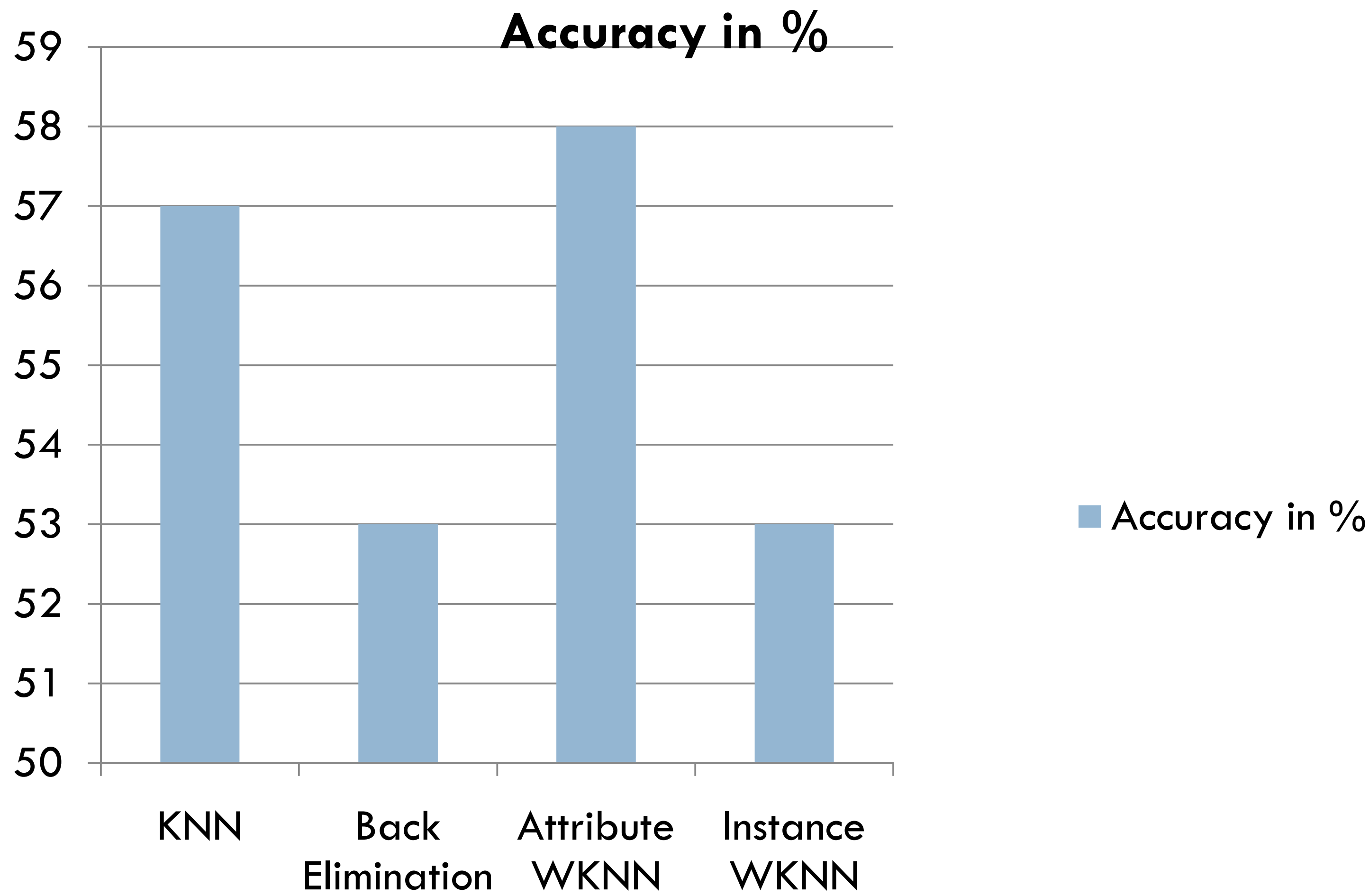
Results (Wine Data Set)



Results (Heart-1 Data Set)

Heart-1 Data Set	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	3	NA	303	203	100	13	4	57
Back Elimination	3	NA	303	203	100	8	4	53
Attribute WKNN	3	0.005	303	203	100	13	4	58
Instance WKNN	3	0.005	303	203	100	13	4	53

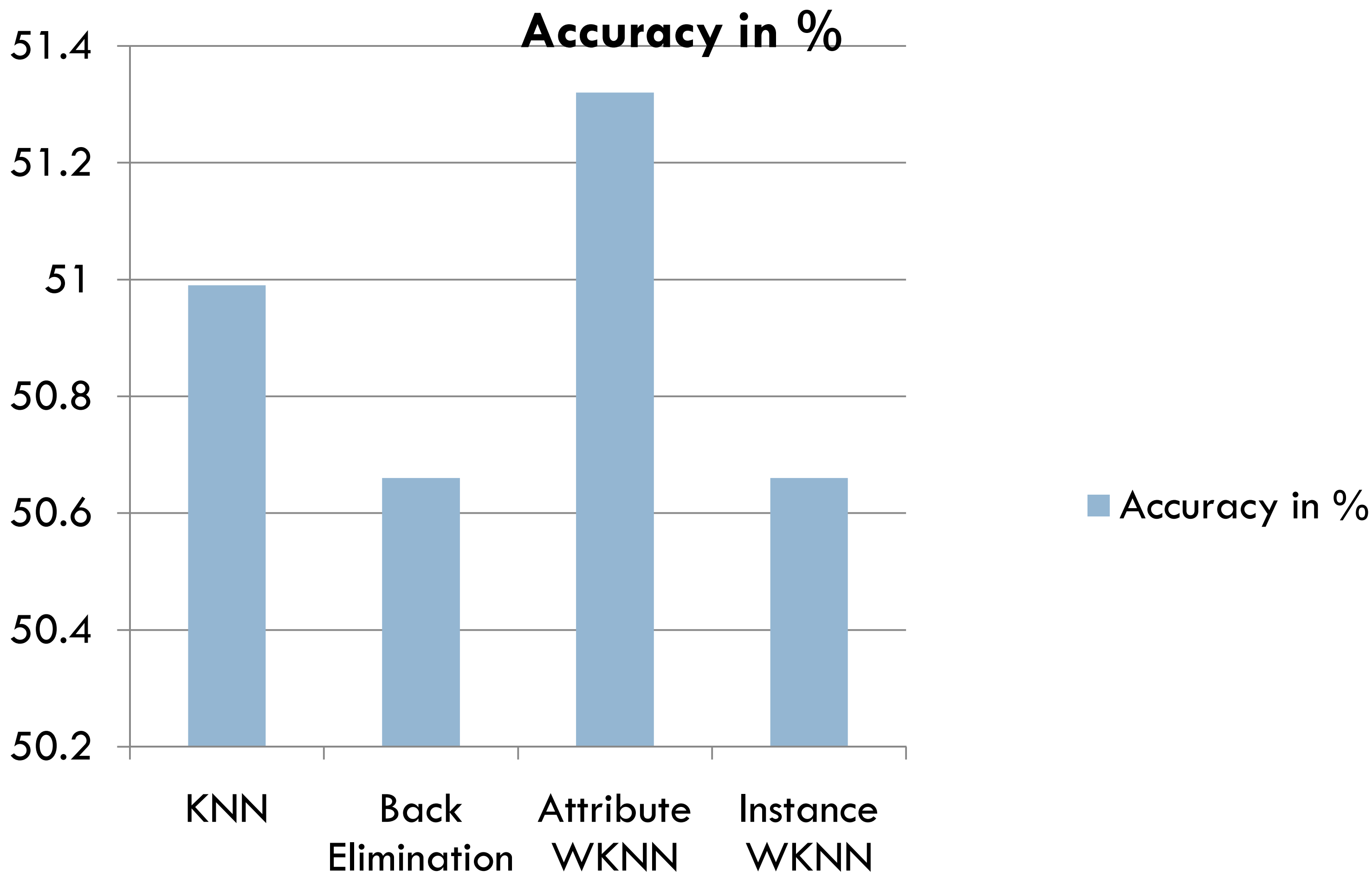
Results (Heart-1 Data Set)



Results (Hill Valley Data Set)

Hill Valley Data Set	K	Learning Rate	# of examples	# of training examples	# of testing examples	# of attributes	# of classes	Accuracy
KNN	3	NA	1212	606	606	100	2	50.99
Back Elimination	3	NA	1212	606	606	94	2	50.66
Attribute WKNN	3	0.005	1212	606	606	100	2	51.32
Instance WKNN	3	0.005	1212	606	606	100	2	

Results (Hill Valley Data Set)



Implementation

- Implemented in C++
- Implemented following algorithms
  - Simple K-NN
  - Weighted K-NN with backward elimination
  - Weighted K-NN with cross validation and gradient descent
    - Instance Weighted KNN
    - Attribute Weighted KNN
- Assumptions made while implementation
  - All the attribute values are numerical
  - Class attribute values are distinct integer values
    - For example: 0,1,2…..
  - Euclidean Distance used for similarity measure
  - For N fold cross validation, N = 3
  - A training example which is not near any instance is removed from the training set
  - For K = 1, do not consider the nearest with distance = 0 (nearest is the same as queried)
- Details will be available on my website in a couple of days
  - http://www.d.umn.edu/~deoka001/index.html

DIET

- Outline
  - What is DIET ?
  - DIET Algorithm
  - Wrapper Model
  - Results

DIET

- DIET is an algorithm which uses a simple wrapper approach to heuristically search through a set of weights used for nearest neighbor classification.

- DIET sometimes causes features to lose weight, sometimes to gain weight and sometimes to remain the same.

DIET Algorithm

- In the DIET algorithm we have a discrete, finite set of weights instead of continuous weights.
- If we choose k number of weights then the set of weights will be: $\{0, 1/k, 2/k, \ldots, (k-1)/k, 1\}$
- If $k = 2$, then the set of weights would be $\{0, 1\}$ which means that we either give weight $= 0$ or 1 to an attribute.
- When $k = 1$, we have only one weight which is taken as 0. This translates into simply ignoring all the weights and predicting the most frequent class.
- Generally when we have k weights, we start with the assignment closest to the middle weight.

DIET Algorithm Continued…

- For each attribute we move through the weight space in search of the weight which minimizes the error until minimum or maximum of the weight is reached.
- The number of neighbors used in the classification is 1 since the goal is to investigate feature weighting rather than the number of neighbors
- Error is calculated every time using tenfold cross validation over the training data with KNN algorithm.
- A halting criterion is used where in we stop the search when five consecutive nodes have children with no better results than their parents. (0.1%)

Wrapper Model

- We search through the weight space heuristically using the wrapper model.
- We search the space for feature subsets till we reach some threshold accuracy.
- The paper mentions about using the wrapper model, but the authors have not mentioned how they have adapted the model for DIET.
- The approaches used for feature subset selection are backward elimination where you start with all the features and greedily remove the one that most improves performance and another is forward selection which starts with a empty set of features and greedily adds features.

DIET Results

- For data sets that contain few or no irrelevant features, DIET performs comparably to simple KNN or slightly worse due to the increased size of the hypothesis space.
- For domains in which relevant features have equal importance, DIET with few weights outperforms DIET with many weights.
- DIET with one non zero weight, which means that either a feature is relevant or irrelevant, outperforms DIET with many weights on most of real world data sets tested.

References

- Machine Learning – Tom Mitchell
- The Utility of Feature Weighting in Nearest-Neighbor Algorithms - Ron Kohavi, Pat Langley, Yeogirl Yun
- Irrelevant Features and the Subset Selection Problem – George John, Ron Kohavi, Karl Pfleger