Report

HW2

Yasaman Mirmohammad | Data Mining | Fall_2018

Task 1:KNN Algorithm

The intuition behind the KNN algorithm is one of the simplest of all the supervised machine learning algorithms.

It simply calculates the distance of a new data point to all other training data points. The distance can be of any type e.g Euclidean or Manhattan etc. It then selects the K-nearest data points, where K can be any integer. Finally it assigns the data point to the class to which the majority of the K data points belong.

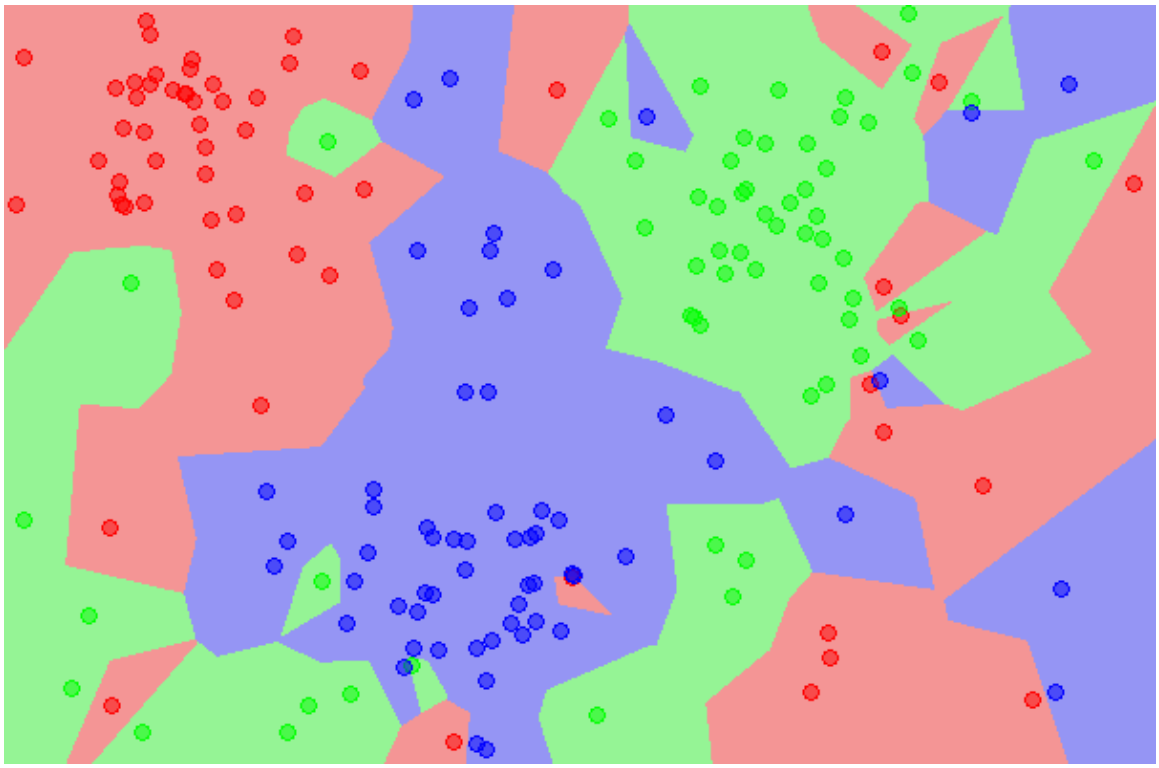
The kNN algorithm is belongs to the family of instance-based, competitive learning and lazy learning algorithms.

Instance-based algorithms are those algorithms that model the problem using data instances (or rows) in order to make predictive decisions. The kNN algorithm is an extreme form of instance-based methods because all training observations are retained as part of the model.

It is a competitive learning algorithm, because it internally uses competition between model elements (data instances) in order to make a predictive decision. The objective similarity measure between data instances causes each data instance to compete to "win" or be most similar to a given unseen data instance and contribute to a prediction.

Lazy learning refers to the fact that the algorithm does not build a model until the time that a prediction is required. It is lazy because it only does work at the last second. This has the benefit of only including data relevant to the unseen data, called a localized model. A disadvantage is that it can be computationally expensive to repeat the same or similar searches over larger training datasets.

Finally, kNN is powerful because it does not assume anything about the data, other than a distance measure can be calculated consistently between any two instances. As such, it is called non-parametric or non-linear as it does not assume a functional form.



k-Nearest Neighbors algorithm
Image from [Wikipedia]

Train Test Split:

80% train , 20% test

```
y=data['Win/Loss'] X_train, X_test, y_train, y_test =
train_test_split(data, y, test_size=0.2)
```

Normalization:

Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

```
scaler = StandardScaler()
scaler.fit(X_train) X_train = scaler.transform(X_train) X_test = scaler.transform(X_test)
```

a) Implement NN1 and calculate the error:

```
classifier = KNeighborsClassifier(n_neighbors=1)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

print('Classification Error:',np.mean(y_pred != y_test))
```

1. Calculate "d(x, x_i)" i =1, 2, ….., **n**; where **d** denotes the Euclidean or Minkowski distance between the points.
2. Arrange the calculated **n** Euclidean distances in non-decreasing order.
3. Let **k** be a +ve integer, take the first **k** distances from this sorted list.
4. Find those **k**-points corresponding to these **k**-distances.
5. Let $\mathbf{k}_i$ denotes the number of points belonging to the i^{th} class among **k** points i.e. k ≥ 0
6. If $k_i > k_j \; \forall \; i \neq j$ then put x in class i.

Classification Error: 0.25245901639344265

Confusion Matrix:
[[81 42]
[35 147]]

	precision	recall	f1-score	support
0	0.70	0.66	0.68	123
1	0.78	0.81	0.79	182
avg / total	0.75	0.75	0.75	305

Mean accuracy: 0.7475409836065574

Error on 100 iteration:

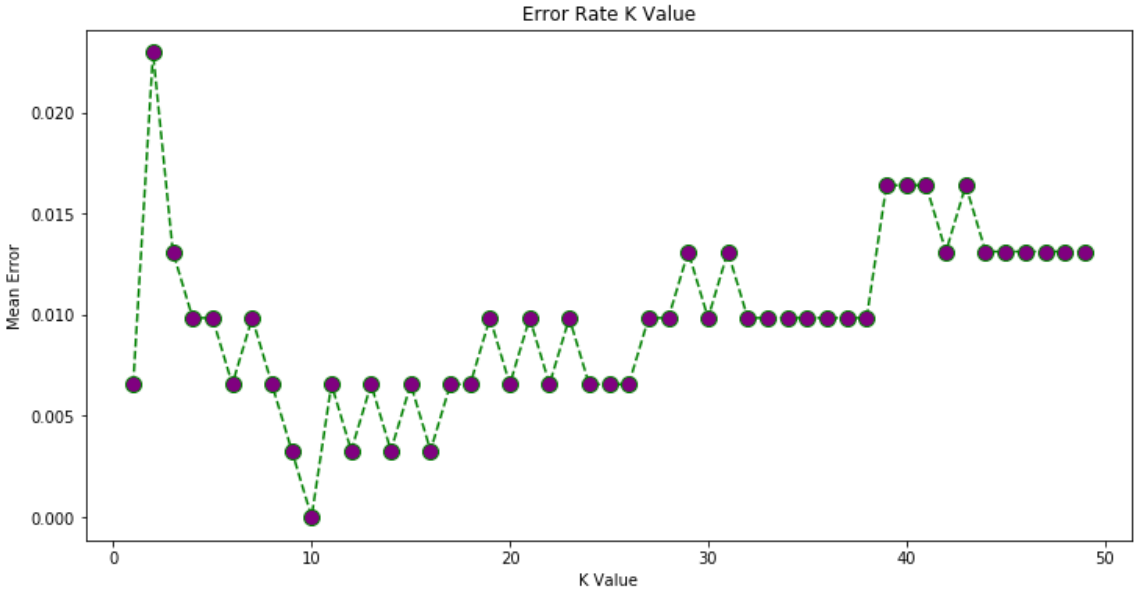
[0.006557377049180328,
0.006557377049180328,
0.006557377049180328,
0.009836065573770493,
0.009836065573770493,
0.013114754098360656,
0.006557377049180328,
0.013114754098360656,
0.006557377049180328,
0.009836065573770493,

0.01639344262295082,
0.013114754098360656,
0.022950819672131147,
0.013114754098360656,
0.013114754098360656,
0.01639344262295082,
0.009836065573770493,
0.01639344262295082,
0.013114754098360656,
0.01639344262295082,
0.013114754098360656,
0.019672131147540985,
0.01639344262295082,
0.006557377049180328,
0.013114754098360656,
0.013114754098360656,
0.009836065573770493,
0.01639344262295082,
0.019672131147540985,
0.019672131147540985,
0.009836065573770493,
0.009836065573770493,
0.01639344262295082,
0.019672131147540985,
0.009836065573770493,
0.003278688524590164,
0.009836065573770493,
0.003278688524590164,
0.009836065573770493,
0.006557377049180328,
0.006557377049180328,
0.003278688524590164,
0.006557377049180328,
0.013114754098360656,
0.003278688524590164,
0.01639344262295082,
0.022950819672131147,
0.0,
0.01639344262295082,
0.009836065573770493,
0.019672131147540985,
0.01639344262295082,
0.01639344262295082,
0.01639344262295082,
0.01639344262295082,
0.022950819672131147,
0.013114754098360656,
0.01639344262295082,
0.013114754098360656,
0.019672131147540985,
0.01639344262295082,
0.009836065573770493,
0.01639344262295082,
0.013114754098360656,
0.009836065573770493,

0.02622950819672131,
0.006557377049180328,
0.006557377049180328,
0.006557377049180328,
0.0,
0.003278688524590164,
0.029508196721311476,
0.009836065573770493,
0.006557377049180328,
0.013114754098360656,
0.013114754098360656,
0.009836065573770493,
0.0,
0.003278688524590164,
0.003278688524590164,
0.009836065573770493,
0.003278688524590164,
0.009836065573770493,
0.006557377049180328,
0.009836065573770493,
0.009836065573770493,
0.009836065573770493,
0.006557377049180328,
0.013114754098360656,
0.003278688524590164,
0.009836065573770493,
0.013114754098360656,
0.019672131147540985,
0.01639344262295082,
0.003278688524590164,
0.013114754098360656,
0.019672131147540985,
0.009836065573770493,
0.029508196721311475]

b)Error Calculation Based on K Value

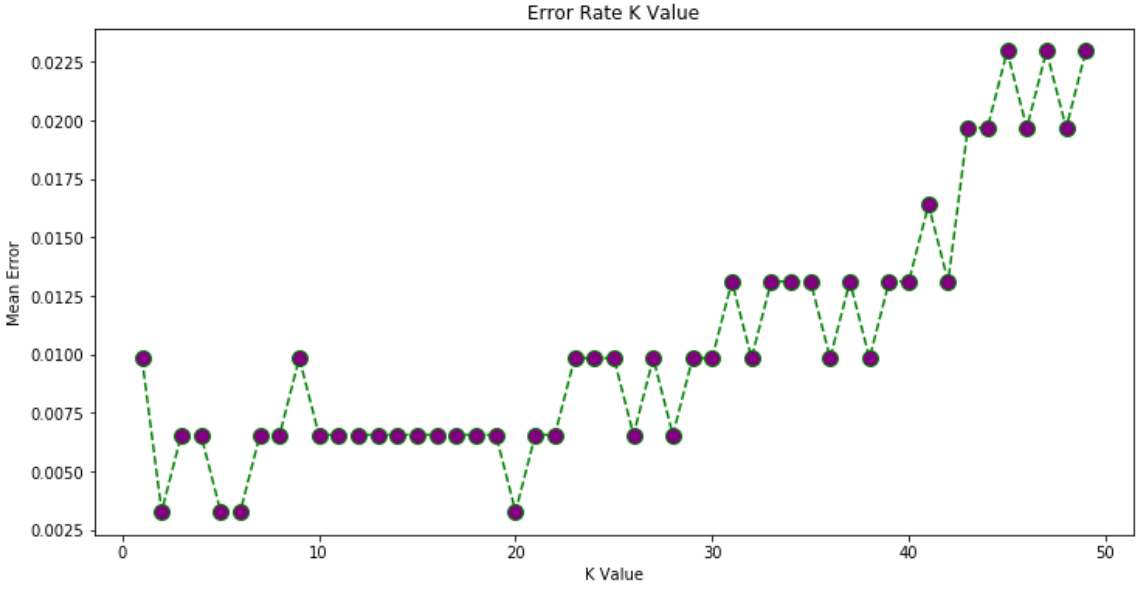
For finding the best value for k we plot the graph of K value and the corresponding error rate for the dataset:

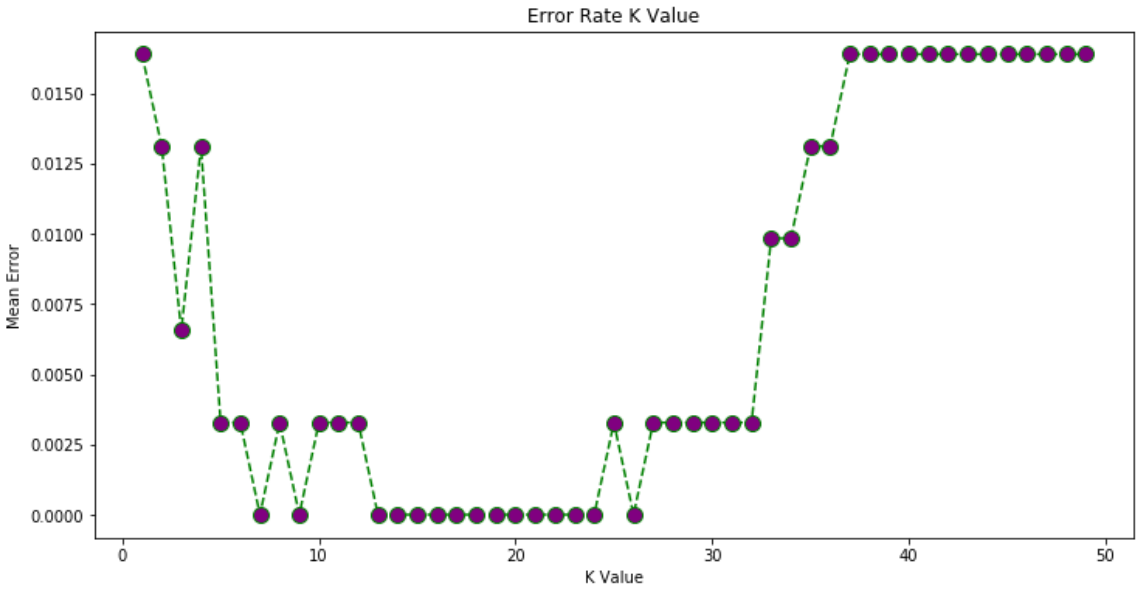


We can observe that for k=10 error is near 0 . and After that error increases.

Caution: Because we shuffle data before implementation each time, results wouldn't be alike.

Look:





If the k in too small, it will start overfitting and will be susceptible to noise and outlier data, and cause overfitting.

If it is too big, the result will be unreliable and the because each class will be containing data from other classes too.

So the appropriate value for k will obtained in trial_and_error.

Task 2:Hunt's Algorithm

Building a decision tree based on information gain.

Definition: Suppose S is a set of instances, A is an attribute, S_v is the subset of s with A = v and Values(A) is the set of all possible of A, then

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

$|S|$ denotes the size of set S

The goal is let the tree grows and then cut off the unreliable ones.

Algorithm

1. If Dt contains records that belong the same class yt, then t is a leaf node labeled as yt
2. If Dt is an empty set, then t is a leaf node labeled by the default class, yd
3. If Dt contains records that belong to more than one class, use an attribute test to split the data into smaller subsets.

It recursively applies the procedure to each subset until all the records in the subset belong to the same class. The Hunt's algirithm assumes that each combination of attribute sets has a unique class label during the procedure. If all the records associated with Dt have identical attribute values except for the class label, then it is not possible to split these records any future. In this case, the node is decalred a leaf node with the same class label as the majority class of training records associated with this node.

Decision Tree Training

Now we fit Decision tree algorithm on training data, predicting labels for validation dataset and printing the accuracy of the model using various parameters.

DecisionTreeClassifier(): This is the classifier function for DecisionTree. It is the main function for implementing the algorithms. Some important parameters are:

- **criterion:** It defines the function to measure the quality of a split. Sklearn supports "gini" criteria for Gini Index & "entropy" for Information Gain. By default, it takes "gini" value.
- **splitter:** It defines the strategy to choose the split at each node. Supports "best" value to choose the best split & "random" to choose the best random split. By default, it takes "best" value.
- **max_features:** It defines the no. of features to consider when looking for the best split. We can input integer, float, string & None value.

    - If an integer is inputted then it considers that value as max features at each split.
    - If float value is taken then it shows the percentage of features at each split.
    - If "auto" or "sqrt" is taken then max_features=sqrt(n_features).
    - If "log2" is taken then max_features= log2(n_features).
    - If None, then max_features=n_features. By default, it takes "None" value.
- **max_depth:** The max_depth parameter denotes maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. By default, it takes "None" value.
- **min_samples_split:** This tells above the minimum no. of samples reqd. to split an internal node. If an integer value is taken then consider min_samples_split as the minimum no. If float, then it shows percentage. By default, it takes "2" value.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node. If an integer value is taken then consider min_samples_leaf as the minimum no. If float, then it shows percentage. By default, it takes "1" value.
- **max_leaf_nodes:** It defines the maximum number of possible leaf nodes. If None then it takes an unlimited number of leaf nodes. By default, it takes "None" value.
- **min_impurity_split:** It defines the threshold for early stopping tree growth. A node will split if its impurity is above the threshold otherwise it is a leaf.

Algorithm:

```
def buildtree(rows,scoref=entropy):

if len(rows)==0: return decisionnode( )

current_score=scoref(rows)

# Set up some variables to track the best criteria

best_gain=0.0

best_criteria=None

best_sets=None
```

```
column_count=len(rows[0])-1

for col in range(0,column_count):

# Generate the list of different values in

# this column

column_values={}

for row in rows:

column_values[row[col]]=1

# Now try dividing the rows up for each value

# in this column

for value in column_values.keys( ):

(set1,set2)=divideset(rows,col,value)

# Information gain

p=float(len(set1))/len(rows)

gain=current_score-p*scoref(set1)-(1-p)*scoref(set2)

if gain>best_gain and len(set1)>0 and len(set2)>0:

best_gain=gain

best_criteria=(col,value)

best_sets=(set1,set2)

# Create the subbranches

if best_gain>0:

trueBranch=buildtree(best_sets[0])

falseBranch=buildtree(best_sets[1])
```

```
return decisionnode(col=best_criteria[0],value=best_criteria[1],

tb=trueBranch,fb=falseBranch)

else:

return decisionnode(results=uniquecounts(rows))
```

The next example function [2] is pruning the built decision tree. Pruning involves checking pairs of nodes that have a common parent to see if merging them would increase the entropy by less than a specified threshold. If so, the leaves are merged into a single node with all the possible outcomes. This helps avoid overfitting and stops the tree from making predictions that are more confident than what can really be gleaned from the data.

When prune function is called on the root node, it will traverse all the way down the tree to the nodes that only have leaf nodes as children. It will create a combined list of results from both of the leaves and will test the entropy. If the change in entropy is less than the mingain parameter, the leaves will be deleted and all their results moved to their parent node. The combined node then becomes a possible candidate for deletion and merging with another node.

```
def prune(tree,mingain):

# If the branches aren't leaves, then prune them

if tree.tb.results==None:

prune(tree.tb,mingain)

if tree.fb.results==None:

prune(tree.fb,mingain)

# If both the subbranches are now leaves, see if they

# should merged

if tree.tb.results!=None and tree.fb.results!=None:

# Build a combined dataset

tb,fb=[],[]

for v,c in tree.tb.results.items( ):

tb+=[[v]]*c
```

```
for v,c in tree.fb.results.items( ):

fb+=[[v]]*c

# Test the reduction in entropy

delta=entropy(tb+fb)-(entropy(tb)+entropy(fb)/2)

if delta<="" pre="">
```

CONCLUSION

Finding an optimal decision tree is an NP-complete problem. Many decision tree algorithms employ a heuristic-based approach or greedy strategy to guide their search in the vast hypothesis space. The constructing decision tree techniques are generally computationally inexpensive, making it possible to quickly construct models even when the training set size is very large. Furthermore, once a decision tree has been built, classifying a test record is extremely fast.

REFERENCES

[1] Introduction to Data Mining, Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Published by Addison Wesley.

[2] Programming Collective Intelligence, Toby Segaran, First Edition, Published by O' Reilly Media, Inc.

The End