Assignment 2 (ELL409)

Abhishek Yadav (2015MT10585)

Feature Selection Methods

In Machine learning the use of feature selection is to select the best attributes from set of attributes which are most relevant for our purpose.

Basically by feature selection we reduce the computation cost and also we are able to avoid curse of dimensionality to a much more extent and also sometimes it's not possible to use all the features of the data set because of our limitations in computation.

It also reduce the chances of over fitting.

There are basically three methods for feature selection-:

1. Filter Methods:

In this method, they are usually used for preprocessing step and they don't use any machine learning classification algorithm for feature selection, instead they use some statistical test for these purposes.
The drawback of filter methods is that the feature produced by them are not tuned in predictive model.

Different techniques for feature selection in Filter Methods.

LDA:-Linear Discriminant Analysis, in this method it is used to find a linear combination of features that separates, two or more classes.
It makes some assumptions about data that it is Gaussian and each attribute has same variance.
Then it estimates the mean and variance for each classes.
Then this method is used for selecting the most relevant features from all set of features.

Chi-Square: Chi- square test is one of the most relevant test in Statistics, Chi-square is a distribution, as some of square of random variable distributed normally. It is then used to groups of categorical features to get

the likelihood of association or correlation amongst them by the frequency distribution of them.

Formula for Chi-Square test-:

$$X^2=\sum(O_i-E_i)^2/(E_i)$$

Relief: - Features are given some weights and then, these weights are used to select best features based on some threshold. We usually find the distance of nearest neighbor of two points selected at randomly in which one belongs to same class and the other belong to other class then in the weight i we will add the distance for that particular class which belong to other one and subtract for the other one, by this way we do the n iterations and then in last we is the weights are greater than a threshold we choose them as the set of selected features.

$$W_i=W_i-(x_i-newhit)^2+(x_i-newmiss)^2$$

Results for filter method

```
>>>
 RESTART: C:\Users\dell.DESKTOP-5VND8PJ\Desktop\Semester 6\ELL 409\Assignments\A
ssignment2\Final\relief_filter.py
[1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0]
>>>
```

(relief_result on small dataset)

```
 RESTART: C:\Users\dell.DESKTOP-5VND8PJ\Desktop\Semester 6\ELL 409\Assignments\A
ssignment2\Final\relief_filter.py
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0
, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0
, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0
, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0
, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0
, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
>>>
```

(relief_result on large dataset)

```
>>> X,y = readfile('data.csv')
>>> X_new = SelectKBest(chi2, k=2).fit_transform(X, y)
>>> X_new.shape
(66, 2)
>>>
```

chi-square(best two feature) my dataset

```
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X.shape
(150, 4)
>>> X_new = SelectKBest(chi2, k=2).fit_transform(X, y)
>>> X_new.shape
(150, 2)
```

chi-square(best two feature) iris dataset

2. **Wrapper Method:** In this type of methods we choose a subsets of features and then we try to train a model using them and based on our previous model we try to one of the best features from them, which improves our accuracy to a best extent. They usually take more computation time as compare to other methods.

   Few of the wrapper methods-:

   1. **Sequential Forward Selection**: In this method we start with an empty set and then one by one we start adding most relevant feature which improves accuracy to a best extent, till we reach our threshold of d features or till we got the improve in accuracy less than a threshold.

      Drawback: Once a feature added can't be removed.

   2. **Sequential Backward Selection:** In this method we start with set of all features then we repeatedly deletes the most insignificant feature, till we get the set of d features or till our decrease is more than a threshold.

      Drawback: Once a feature removed can't be added.

   3. **Recursive Feature Elimination:** In this method, it is a greedy optimization algorithm, with each iteration, it creates a model and select the best and worst performing features and then the left features are used for creating the next model and this process is repeated till all the features

are exhausted. Then the features are ranked based on order of elimination.

Results for Wrapper method

```
 RESTART: C:\Users\dell.DESKTOP-5VND8PJ\Desktop\Semester 6\ELL 409\Assignments\A
ssignment2\Final\forward_selection_wrapper.py
{0, 2, 3, 5, 6, 7, 14}
>>>
```

(forward_selection on small dataset)

```
>>> selector = RFE(estimator, 5, step=1)
>>> selector = selector.fit(X, y)
>>> selector.support_
array([False, False,  True, False, False,  True, False,  True, False,
        True, False, False, False,  True, False, False])
>>>
```

(recursive_feature_elimination on small dataset)

```
>>> from sklearn.datasets import make_friedman1
>>> X, y = make_friedman1(n_samples=50, n_features=10, random_state=0)
>>> selector = selector.fit(X, y)
>>> selector.support_
array([ True,  True,  True,  True,  True, False, False, False, False,
       False])
>>>
```

(recursive_feature_elimination on differnt dataset)

3. **Embedded Methods:** They basically are a combinations of Filter and Wrapper methods, they are basically implemented by methods which have their own feature selection criteria or algorithm.

   Few of the Embedded Methods are -:

   1. **Decision Tree:** Decision tree have their own feature selection methods, as we can see while making a decision tree at each step a feature is selected and then the datasets are divided based on that feature, and

then at next level these are again used to do that till we get a fixed level of the tree or till all train datasets are classified.

Now to check how decision tree are used for feature selection, we will choose those attributes from tree which are used to making the tree and those will be our best features.

2. **Forward Selection with least square:** The classical forward selection for least square finds greedily the components which minimize the residual errors, the algorithm is as follows:
   We first start with subset S=null then we choose a projection matrix Ps, then take Y residual vectors, and these are given by
   P_sY with $P_s:=I-X_s'(X_s X_s')^{-1}X_s$
   Here X_s is the sub matrix of when only S features are included.
   As the algorithm proceeds we find component 'I' such that
   Norm of P_iY is minimum, and then add 'I' to S, and we repeat this process till threshold is reached.

Results for Embedded method

```
 RESTART: C:\Users\dell.DESKTOP-5VND8PJ\Desktop\Semester 6\ELL 409\Assignments\A
ssignment2\Final\decision_tree_embedded.py
<sklearn.tree._tree.Tree object at 0x00000190219D72A0>
[0.         0.         0.         0.         0.23561732 0.
 0.05656109 0.         0.         0.         0.         0.27149321
 0.         0.         0.43632838 0.        ]
{11, 4, 14, 6}
>>>
```

(decision_tree on small dataset)

```
 RESTART: C:\Users\dell.DESKTOP-5VND8PJ\Desktop\Semester 6\ELL 409\Assignments
ssignment2\Final\decision_tree_embedded.py
<sklearn.tree._tree.Tree object at 0x0000028E7B0C0238>
{451, 48, 338, 153, 378, 475}
>>>
```

(decision_tree on large dataset)

```
 RESTART: C:\Users\dell.DESKTOP-5VND8PJ\Desktop\Semester 6\ELL 409\Assignments\A
ssignment2\Final\tree.py
[0.0533978  0.04612036 0.0530116  0.04511248 0.0323558  0.08584208
 0.10517317 0.07929411 0.06778197 0.03138007 0.0926299  0.06792849
 0.06422563 0.03633486 0.08344945 0.05596221]
(66, 8)
>>>
```

(tree based (library) on small dataset)

```
 0.00282877 0.00129477 0.0023673  0.00247056 0.00211031 0.00240131
 0.00194491 0.00144579 0.00071953 0.0015721  0.00280015 0.00318214
 0.00098998 0.00139605 0.00275117 0.00197813 0.00212126 0.00146294
 0.0024979  0.0009808  0.01117164 0.0013409  0.000871   0.00125061
 0.0011298  0.0023737  0.0023255  0.00113553 0.00096852 0.00087096
 0.00213186 0.00255192 0.00182703 0.00253705 0.00181816 0.00191998
 0.00196461 0.00140551 0.00119656 0.00120059 0.00136167 0.00092425
 0.00142216 0.00095873 0.00162341 0.00215776 0.00189553 0.00156805
 0.00247073 0.00181097 0.00217564 0.00103952 0.00230421 0.00233228
 0.00161416 0.00261221 0.00193874 0.00155659 0.00246857 0.00202179
 0.00517214 0.00130759 0.0019632  0.00312083 0.00192832 0.0013701
 0.00223258 0.00147476 0.00118157 0.00130999 0.00152291 0.00291322
 0.00199316 0.00292988 0.00120801 0.0019797  0.00123357 0.0014383
 0.00180271 0.00123999 0.00170707 0.00175645 0.0017039  0.00137059
 0.00197997 0.00280907 0.0020364  0.00140209 0.00217838 0.003513
 0.00078805 0.00236146 0.00090856 0.00209078 0.00282234 0.0009877
 0.0023595  0.00149521 0.00144177 0.00128409 0.00198429 0.00163169
 0.00173237 0.00091497 0.00163552 0.00160017 0.00120437 0.00216487
 0.00149226 0.00198645 0.00165835 0.00189293 0.00202484 0.00179228
 0.00098999 0.00411061 0.00282534 0.00239642 0.00090832 0.00173873
 0.00128181 0.00214735 0.00273592 0.001218   0.00678256 0.00216246
 0.00231577 0.00174521 0.00179223 0.00155492 0.00165023 0.00156843
 0.00105754 0.00918943 0.00142402 0.00315866 0.00078456 0.00277472
 0.00154224 0.0023952  0.00123016 0.00358915 0.00213226 0.00129437
 0.00101655 0.00164946 0.00089701 0.00163251 0.00092458 0.00142879
 0.00246476 0.00107216 0.00135814 0.00235029 0.00409703 0.00171835
 0.0025621  0.01128185 0.00202558 0.0020675  0.00232986 0.00112378
 0.00127724 0.00195644 0.00140131 0.00158227 0.00124159 0.00190736
 0.00192487 0.00214103 0.00143068 0.00218208 0.00159194 0.00198664
 0.00188578 0.00550193 0.00263758 0.0021997  0.00299434 0.0024224
 0.00091078 0.00218614]
(1999, 201)
>>>
```

(tree based (library) on large dataset, with importance some of features)

Codes

1. Relief Feature Selection:

```
import csv
import numpy as np
import matplotlib.pyplot as plt
import scipy
import scipy.spatial
import random
import math
#It is the function for the loading of the csv file into a numpy array
def readfile(data,k):
    with open(data,"r") as csvfile:
        x = [[] for i in range(k)]
        item  = list(csv.reader(csvfile))
        for i in range(len(item)):
            item[i] = [int(x) for x in item[i]]
            kl=item[i][len(item[i])-1]
            del item[i][-1]
            x[kl].append(item[i])
    return x,len(item)

if __name__ == "__main__":
  x,m = readfile('madelon_data.csv',2)
  xl=list()
  for i in range(0,2):
   temp=scipy.spatial.KDTree(x[i])
   xl.append(temp)
  a=len(x[0][0])
  w = [0] * a
  wl= [0] * a
  k=2
  n=66
  alpha=0.000000005
  thres=1/math.sqrt(alpha*m)
  for i in range(0,n):
    classl=random.randint(0, k-1)
    sample=random.randint(0, len(x[classl])-1)
    temp=x[classl][sample]
    #print(temp)
    dist2=math.inf
    distl=math.inf
    indexl=0
```

```
    dist1=math.inf
    index1=0
    index2=0
    class2=0
    for j in range(0,k):
      if(j==class1):
        dist1,d1=x1[class1].query(temp, k=2, eps=0, p=2, distance_upper_bound=ma
        index1=d1[1]
      else:
        dist21,d2=x1[j].query(temp, k=1, eps=0, p=2, distance_upper_bound=math.i
        if dist2<dist21:
          dist2=dist21
          index2=d2[0]
          class2=j
    for j in range(0,a):
      point=x[class1][sample][j]
      w[j]+=math.pow((x[class2][index2][j]-point),2)-math.pow((x[class1][index1]
  for j in range(0,a):
    w[j]/=n
    if w[j]>=thres:
        w1[j]=1
        #print(w[j])
print(w1)
```

2. Forward-Selection

```
import csv
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
#It is the function for the loading of the csv file into a numpy array
def readfile(data):
    with open(data,"r") as csvfile:
        item  = list(csv.reader(csvfile))
        x = [[] for i in range(len(item))]
        for i in range(len(item)):
            item[i] = [int(x) for x in item[i]]
            kl=item[i][len(item[i])-1]
            del item[i][-1]
            x[i].append(kl)
    return item,x
def f(x,y,xl,yl,s,k):
    u = [[] for i in range(len(x))]
    v=y
    ul= [[] for i in range(len(xl))]
    vl=yl
    for i in range(len(x)):
        u[i].append(x[i][k])
        for j in s:
            u[i].append(x[i][j])

    for i in range(len(xl)):
        ul[i].append(xl[i][k])
        for j in s:
            ul[i].append(xl[i][j])
    neigh.fit(u, np.array(v).ravel())
    acc=0
    for i in range(len(ul)):
       pre=neigh.predict([ul[i]])
       if pre==vl[i]:
           acc+=1
    return acc
if __name__ == "__main__":
  x,y = readfile('data.csv')
  xl,yl = readfile('data.csv')
  temp=list()
```

Feature Select
DC

```
    for i in range(len(ul)):
        pre=neigh.predict([ul[i]])
        if pre==vl[i]:
            acc+=1
    return acc
if __name__ == "__main__":
  x,y = readfile('data.csv')
  xl,yl = readfile('data.csv')
  temp=list()
  a=len(x[0])
  w = [0] * a
  d=7
  s=set()
  for i in range(0,d):
      accr=-1
      index=-1
      for k in range(0,a):
          if w[k]==0:
              accrl=f(x,y,xl,yl,s,k)
              if accrl>accr:
                  index=k
                  accr=accrl
      w[index]=1
      s.add(index)
  print(s)
```

3. Decision Tree

```
import csv
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier

def readfile(data):
    with open(data,"r") as csvfile:
        item  = list(csv.reader(csvfile))
        y=list()
        for i in range(len(item)):
            item[i] = [int(x) for x in item[i]]
            kl=item[i][len(item[i])-1]
            del item[i][-1]
            y.append(kl)
    return item,y
# Function to perform training with giniIndex.
def decision_tree(X_train, y_train):

    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
    features = clf_gini.n_features_
    tree=clf_gini.tree_
    print(tree)
    n_nodes = clf_gini.tree_.node_count
    children_left = clf_gini.tree_.children_left
    children_right = clf_gini.tree_.children_right
    feature = clf_gini.tree_.feature
    threshold = clf_gini.tree_.threshold

    temp=set()
    node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
    is_leaves = np.zeros(shape=n_nodes, dtype=bool)
    stack = [(0, -1)]  # seed is the root node id and its parent depth
    while len(stack) > 0:
```

```
    feature = clf_gini.tree_.feature
    threshold = clf_gini.tree_.threshold

    temp=set()
    node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
    is_leaves = np.zeros(shape=n_nodes, dtype=bool)
    stack = [(0, -1)]  # seed is the root node id and its parent depth
    while len(stack) > 0:
        node_id, parent_depth = stack.pop()
        node_depth[node_id] = parent_depth + 1

    # If we have a test node
        if (children_left[node_id] != children_right[node_id]):
            stack.append((children_left[node_id], parent_depth + 1))
            stack.append((children_right[node_id], parent_depth + 1))
        else:
            is_leaves[node_id] = True
    for i in range(n_nodes):
        if is_leaves[i]:
            continue
        f=feature[i]
        if f<0:
            f+=features
        temp.add(f)

    importances = clf_gini.feature_importances_
    #print(importances)
    return clf_gini,temp

if __name__ == "__main__":
  x,y = readfile('madelon_data.csv')
  c,temp=decision_tree(x,y)
  print(temp)
```