Untitled

January 17, 2019

0.0.1 1. Python modules used

```
<li>Numpy for storing and manipulating data.</li>
<li>Pandas for reading csv data and organizing into numpy arrays</li>
<li>Scikit learn for comparison with its version of decision tree</li>
<li>Matplotlib for crating graphs</li>
```

```
In [20]: import numpy as np
         import pandas as pd
         from sklearn import tree as dtree
         from sklearn import preprocessing
         import matplotlib.patches as mpatches
         import matplotlib.pyplot as plt
```

0.0.2 2. Code Structure

The code for the decision tree is written as a set of functions which forms the core part of the decision trees. The core part forms the set of functions and parameters which are then called to do the 6 different questions. The following sections details the working of the core functions of the code

0.0.3 3. Core Functions

The core functions is a set of functions for creating the decision tree based on some impurity metric function. This inolves

```
<li>Loading the data</li>
<li>Separting the various attributes into either numerical or categorical</li>
<li>Finding a way of spliting data into subsets based on maximum information gain</li>
<li>Separating the dataset into train and validation sets</li>
<li>Creating a decision tree based on the training data greedily using the impurity metric used<
<li>Predicting the label of an unknown data using the created tree</li>
```

3.1 Impuriy metrics used Three different impurity metrics are used

```
<li>Entropy : $$E(t) = t * log_2(t) + (1 - t) log_2(1-t)$$</li>
<li>Gini Index :  $$G(t) = 2t(t - 1)$$</li>
<li>Misclassification : $$M(t) = min(t,1-t)$$
```

In [3]:
```
#define entropy function
def entropy(q):
    #print(q)
    if q <= 0 or q >= 1:
        return 0
    return -1 * (q * np.log2(q) + (1- q) * np.log2(1-q))
#define gini index
def gini_index(q):
    if q <= 0 or q >= 1:
        return 0
    return 2 * q * (1 - q)
#define misclassification rate
def miss_rate(q):
    if q <= 0 or q >= 1:
        return 0
    return min(q,1 - q)
#define function to determin which type was used
def ftype(func):
    if func == entropy:
        return "entropy"
    else:
        return "gini"
```

In [4]:
```
#calculate gain ratio of a categorical data
def cat_gain_ratio(data_label,func):

    attr_cat = np.unique(data_label[:,0],return_counts = True)
    if attr_cat[0].shape[0] == 1:
        return (-1,-1)
    attr_countn = data_label.shape[0]
    prev_res = func(np.sum(data_label[:,1])/attr_countn)
    if prev_res == 0:
        return (-1,-1)

    intr = 0
    weight_entropy = 0

    for attr,attr_count in list(zip(attr_cat[0],attr_cat[1])):
        intr += (attr_count/attr_countn) * np.log2(attr_count/attr_countn)
        q = np.sum((data_label[:,0] == attr) & (data_label[:,1] == 1)) / attr_count
        weight_entropy += (attr_count/attr_countn) * func(q)

    intr = -1 * intr
    if intr:
        return ((prev_res - weight_entropy),0)
    return (0.0,0)
```

In [5]:
```
#calculate gain ratio with numerical data
```

```
def num_gain_split(data_label,func):

    attr_countn = data_label.shape[0]
    left_countn = np.sum(data_label[:,1] == 1)
    prev_res = func(left_countn/attr_countn)
    if prev_res == 0:
        return -1,-1

    data_label = sorted(data_label,key = lambda x:x[0])
    csum = data_label[0][1]
    pos = 0
    max_ratio = 0
    split_pos = 0

    for i in data_label[1:attr_countn-1]:
        if i[0] == data_label[pos][0]:
            csum += i[1]
            pos += 1
            continue
        intr = -1 * (((pos + 1)/attr_countn) * np.log2((pos + 1)/attr_countn) + \
                ((attr_countn - pos - 1)/attr_countn) * np.log2((attr_countn - pos - 1)/
        q = (prev_res - (((pos + 1)/attr_countn) * func(csum/(pos + 1)) + \
                ((attr_countn - pos - 1)/attr_countn) * \
                func((left_countn - csum)/(attr_countn - pos - 1)) \
                )#/intr
        if q > max_ratio:
                max_ratio = q
                split_pos = pos
        csum += i[1]
        pos += 1

    return (max_ratio,data_label[split_pos][0])
```

```
In [6]: def split_attribute(data,attr_type = 1,sp = 0):

            if attr_type:
                #split on categorical attribute
                attr_cat = np.unique(data)
                attr_pos = []
                for attr in attr_cat:
                    attr_pos += [(attr,np.where(data == attr))]

                return attr_pos
            else:
                #split on numerical attribute
                return [(sp,np.where(data <= sp)),(sp+1,np.where(data > sp))]
```

In [7]:
```
#create the tree
def create_tree(dataset,attr_type,label,func):

    i = 0
    max_ent = 0
    res = []
    while i < len(attr_type):
        if i == label:
            res += [(-1,-1)]
        elif attr_type[i]:
            res += [cat_gain_ratio(dataset[:,[i,label]],func)]
        else:
            res += [num_gain_split(dataset[:,[i,label]],func)]
        if res[i][0] > res[max_ent][0]:
            max_ent = i
        i += 1

    onec = np.sum(dataset[:,label])
    zeroc = dataset.shape[0] - onec
    if res[max_ent][0] <= 0.0:
        return (-1,onec,zeroc)
    branch = {}
    split_vec = split_attribute(dataset[:,max_ent],attr_type[max_ent],res[max_ent][1])
    for attr,arr in split_vec:
        branch[attr] = create_tree(dataset[arr],attr_type,label,func)
    return (max_ent,onec,zeroc,branch)
```

In [8]:
```
#prediction algorithm
#max_rec parameter is used to limit the depth upto which prediction will recurse into
#used for question no 5
def predict(testdata,attr_type,dec_tree,max_rec = 2000000):

    if max_rec  == 0 or dec_tree[0] == -1:
        return [dec_tree[1],dec_tree[2]]
    elif attr_type[dec_tree[0]]:
        if testdata[dec_tree[0]] in dec_tree[3].keys():
            return predict(testdata,attr_type,dec_tree[3][testdata[dec_tree[0]]],max_rec
        #else:
         #   return [dec_tree[1],dec_tree[2]]
        count = np.array([0,0])
        for val in dec_tree[3].values():
            count += np.array(predict(testdata,attr_type,val,max_rec-1))
        return [count[0],count[1]]
    else:
        sp = sorted(list(dec_tree[3].keys()))
        if testdata[dec_tree[0]] <= sp[0]:
            return predict(testdata,attr_type,dec_tree[3][sp[0]],max_rec-1)
```

```
            else:
                return predict(testdata,attr_type,dec_tree[3][sp[1]],max_rec-1)
```

```
In [9]: #test data and show stats
        def testtree(tree,test,label,attr_type,showstats=1,max_rec = 2000000):
            if test.shape[0] == 0:
                print("No validation data remaining")
                return

            tp,tn = 0,0
            fp,fn = 0,0

            for d in test:
                p = predict(d,attr_type,tree,max_rec)
                p = p[0] > p[1]
                if int(p) == int(d[label]):
                    if int(p) == 1:
                        tp += 1
                    else:
                        tn += 1
                else:
                    if int(p)== 1:
                        fp += 1
                    else:
                        fn += 1

            acc = ((tp + tn)/test.shape[0])*100
            rec = tp/(tp + fn)
            pre = tp/(tp + fp)
            f1s = 2/(1/rec + 1/pre)

            if showstats:
                print("My Implementation \n==================================")
                print("Validation data prediction:")
                print("Total dataset size",test.shape[0])
                print("True positive",tp,"True Negetive",tn,"Correct Predictions",tn + tp)
                print("False positive",fp,"False Negetive",fn,"Incorrect Predictions",fn + fp)
                print("Accuracy:",acc,"Recall:",rec,"Precision:",pre,"F1 score:",f1s)
                print("-------------------------------")
            return (acc,rec,pre,f1s)
```

```
In [10]: #scikit - learn implementation
         def sklearn_dec(data,tlim,label,test):
             i = 0
             tdata = np.zeros([data.shape[0],1])
             if test is not None:
                 pdata = np.zeros([test.shape[0],1])
```

```python
le = preprocessing.LabelEncoder()
while i < data.shape[1]:
    if i != label:
        if attr_type[i]:
            le.fit(np.unique(data[:,i]))
            tdata = np.hstack((tdata,le.transform(data[:,i]).reshape(-1,1)))
            if test is not None:
                pdata = np.hstack((pdata,le.transform(test[:,i]).reshape(-1,1)))
        else:
            tdata = np.hstack((tdata,data[:,i].reshape(-1,1)))
            if test is not None:
                pdata = np.hstack((pdata,test[:,i].reshape(-1,1)))
    i += 1

tdata = np.delete(tdata,0,1)
if test is not None:
    pdata = np.delete(pdata,0,1)
traindata = tdata[:tlim]
valdata = tdata[tlim:]
trainlabel = data[:tlim,label].astype(int)
validationlabel = data[tlim:,label]
clf = dtree.DecisionTreeClassifier(criterion=ftype(imp_measure))
clf.fit(traindata,trainlabel)

if valdata.shape[0] != 0:
    tp,tn = 0,0
    fp,fn = 0,0
    for i,j in list(zip(valdata,validationlabel)):
        p = int(clf.predict([i])[0])
        if p == int(j):
            if p == 1:
                tp += 1
            else:
                tn += 1
        else:
            if p == 1:
                fp += 1
            else:
                fn += 1

    print("Validation data prediction:")
    print("Total dataset size",valdata.shape[0])
    print("True positive",tp,"True Negetive",tn,"Correct Predictions",tn + tp)
    print("False positive",fp,"False Negetive",fn,"Incorrect Predictions",fn + fp)
    acc = ((tp + tn)/valdata.shape[0])*100
    rec = tp/(tp + fn)
    pre = tp/(tp + fp)
    f1s = 2/(1/rec + 1/pre)
```

```
            print("Accuracy:",acc,"Recall:",rec,"Precision:",pre,"F1 score:",f1s)
            print("-------------------------------")
        else:
            print("No validation data remaining")

        if test is not None:
            print("Test data predictions:")
            for i,j in list(zip(test,pdata)):
                print(i,"result =",int(clf.predict([j])[0]))
```

In [11]:
```
#parameters for the questions

#data location
data_loc = 'decision_Tree/train.csv'

#split percentage of training set
#by default the program uses the remaining data for validation
split = 0.9

#attribute type : 1 for categorical data, 0 for numerical data
#represented as a boolean vector of same dimensions as the data including label attribu
attr_type = [0,0,0,0,0,1,1,1,1,1]
'''
    example for the given data
    0 : 'satisfaction_level',
    1 : 'last_evaluation',
    2 : 'number_project',
    3 : 'average_montly_hours',
    4 : 'time_spend_company',
    5 : 'Work_accident',
    6 : 'left',
    7 : 'promotion_last_5years',
    8 : 'sales',
    9 : 'salary'

    for satisfacation level to be treated as categorical set attr_type[0] = 1
    similarly set attr_type[9] = 1 for salary to be considered categorical
    otherwise set attr_type[0] = 0 for it to be a numerical data
'''

#attribute name on which the classification will take place
attr_name = "left"

#test data location, will not be evaluated if left as None
#test data is assumed to have all attributes as there are in training data except the c
test_data_loc = "decision_Tree/sample_test.csv"
```

```
        #default impurity measure func
        imp_measure = entropy

        #function to set parameters with default params
        def setParams(dl,sp = 0.9,atp = [0,0,0,0,0,1,1,1,1,1],anm = "left",ipm = entropy,tdl =
            global data_loc,split,attr_name,attr_type,test_data_loc,imp_measure
            data_loc = dl
            split = sp
            attr_type = atp
            attr_name = anm
            imp_measure = ipm
            test_data_loc = tdl
```

```
In [12]: #run decision tree
         #shows stats if showstats is true (default)
         #comparing with scikit learns implementation if schk is true
         #max recursion or the depth to which a prediction will traverse the tree
         #optional parameter for a previously computed tree
         def runDecisionTree(showstats=1,schk=1,max_rec=2000000,tree=None):
             pdata = pd.read_csv(data_loc)
             data = pdata.values
             label = pdata.columns.get_loc(attr_name)

             tlim = int(split*data.shape[0])
             train = data[:tlim]
             vdata = data[tlim:]
             if tree is None:
                 tree = create_tree(train,attr_type,label,imp_measure)

             ret = testtree(tree,vdata,label,attr_type,showstats,max_rec)

             if schk:
                 tdata = None
                 if test_data_loc is not None:
                     ptdata = pd.read_csv(test_data_loc)
                     ptdata[attr_name] = np.zeros([ptdata.shape[0],1])
                     tdata = ptdata[pdata.columns.values].values
                     print("Test data predictions:")
                     for i,j in list(zip(tdata,ptdata.values)):
                         p = predict(i,attr_type,tree)
                         print(j, " result =",int(p[0] > p[1]))

                 print("--------------------------------")
                 print("Scikit learn \n==================================")
                 sklearn_dec(data,tlim,label,tdata)
             return ret,tree
```

```
In [13]: #set up the parameters for our implementation
```

```
#look at the parameters section for more details
#in this configuration all the attributes are considered categorical
setParams('decision_Tree/train.csv',0.90,[1,1,1,1,1,1,1,1,1,1],"left",entropy,'decision
res = runDecisionTree()
```

```
My Implementation
==================================
Validation data prediction:
Total dataset size 1124
True positive 254 True Negetive 795 Correct Predictions 1049
False positive 46 False Negetive 29 Incorrect Predictions 75
Accuracy: 93.32740213523132 Recall: 0.8975265017667845 Precision: 0.8466666666666667 F1 score: 0
-------------------------------
Test data predictions:
[0.69 0.69 3 236 4 0 0 'product_mng' 'medium' 0.0]  result = 0
[0.36 0.54 2 153 3 1 0 'accounting' 'medium' 0.0]  result = 1
-------------------------------
Scikit learn
==================================
Validation data prediction:
Total dataset size 1124
True positive 274 True Negetive 826 Correct Predictions 1100
False positive 15 False Negetive 9 Incorrect Predictions 24
Accuracy: 97.86476868327402 Recall: 0.9681978798586572 Precision: 0.9480968858131488 F1 score: 0
-------------------------------
Test data predictions:
[0.69 0.69 3 236 4 0 0.0 0 'product_mng' 'medium'] result = 0
[0.36 0.54 2 153 3 1 0.0 0 'accounting' 'medium'] result = 1
```

In [14]:
```
#parameters, first 5 attributes are numerical remaining are categorical
#although it is necessary to include the label vector in attribute type the processing
#and consequently its index value in attr_type is inconsqeuntial
setParams('decision_Tree/train.csv',0.90,[0,0,0,0,0,1,1,1,1,1],"left",entropy,'decision
res = runDecisionTree()
#print(res[1])
```

```
My Implementation
==================================
Validation data prediction:
Total dataset size 1124
True positive 272 True Negetive 828 Correct Predictions 1100
False positive 13 False Negetive 11 Incorrect Predictions 24
Accuracy: 97.86476868327402 Recall: 0.9611307420494699 Precision: 0.9543859649122807 F1 score: 0
-------------------------------
Test data predictions:
[0.69 0.69 3 236 4 0 0 'product_mng' 'medium' 0.0]  result = 0
[0.36 0.54 2 153 3 1 0 'accounting' 'medium' 0.0]  result = 1
```

```
-------------------------------
Scikit learn
==================================
Validation data prediction:
Total dataset size 1124
True positive 274 True Negetive 827 Correct Predictions 1101
False positive 14 False Negetive 9 Incorrect Predictions 23
Accuracy: 97.95373665480427 Recall: 0.9681978798586572 Precision: 0.9513888888888888 F1 score: 0
-------------------------------
Test data predictions:
[0.69 0.69 3 236 4 0 0.0 0 'product_mng' 'medium'] result = 0
[0.36 0.54 2 153 3 1 0.0 0 'accounting' 'medium'] result = 1
```

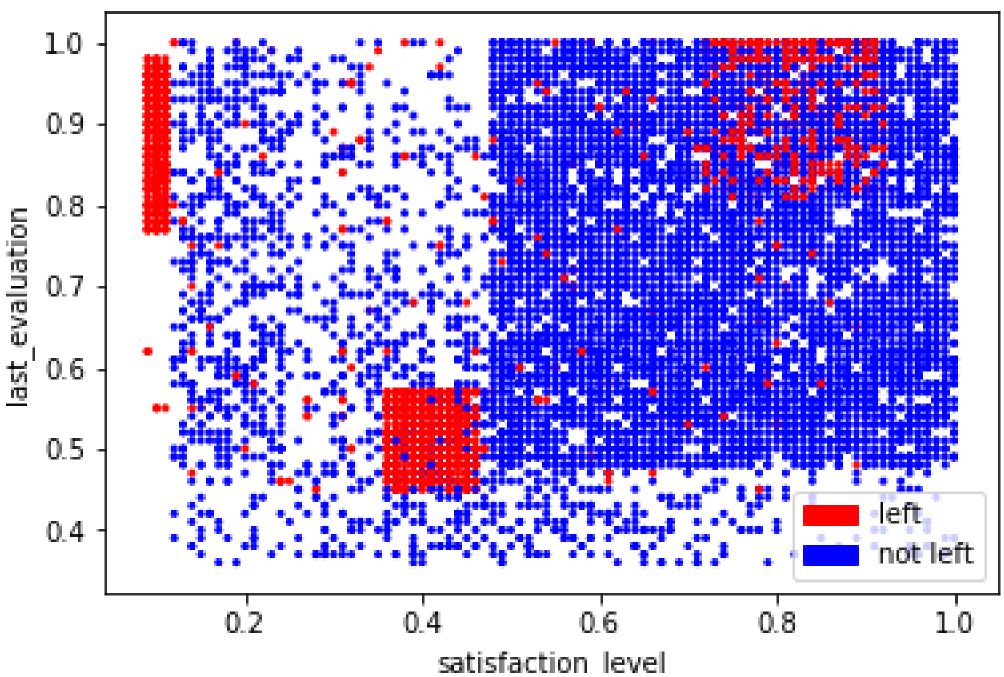
```
In [15]: #train and test with entropy
         print("Using Entropy\n++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
         setParams('decision_Tree/train.csv',0.90,[0,0,0,0,0,1,1,1,1,1],"left",entropy,'decision
         res = runDecisionTree()
         #train and test using gini
         print("\nUsing Gini Index\n+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
         setParams('decision_Tree/train.csv',0.90,[0,0,0,0,0,1,1,1,1,1],"left",gini_index,'decis
         res = runDecisionTree()
         #train and test using miss-classification rate
         #scikit learn does not have miss-classification rate and hence is skipped
         print("\nUsing Miss classification rate\n+++++++++++++++++++++++++++++++++++++++++++++++
         setParams('decision_Tree/train.csv',0.90,[0,0,0,0,0,1,1,1,1,1],"left",miss_rate,'decisi
         res = runDecisionTree(schk=0)
```

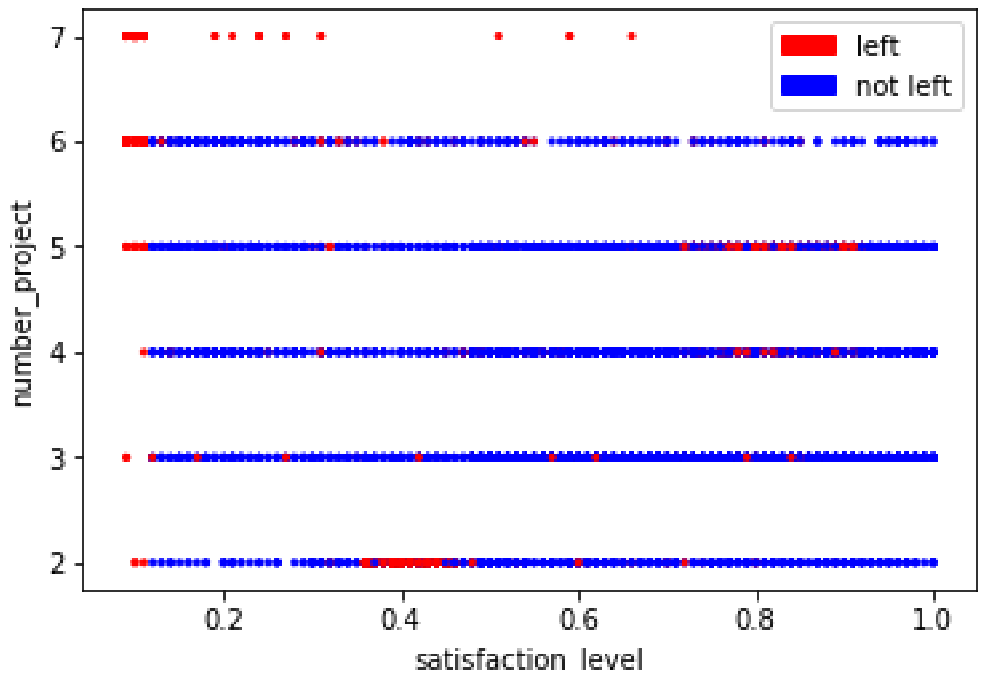
```
Using Entropy
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
My Implementation
==================================
Validation data prediction:
Total dataset size 1124
True positive 272 True Negetive 828 Correct Predictions 1100
False positive 13 False Negetive 11 Incorrect Predictions 24
Accuracy: 97.86476868327402 Recall: 0.9611307420494699 Precision: 0.9543859649122807 F1 score: 0
-------------------------------
Test data predictions:
[0.69 0.69 3 236 4 0 0 'product_mng' 'medium' 0.0]  result = 0
[0.36 0.54 2 153 3 1 0 'accounting' 'medium' 0.0]  result = 1
-------------------------------
Scikit learn
==================================
Validation data prediction:
Total dataset size 1124
True positive 274 True Negetive 826 Correct Predictions 1100
False positive 15 False Negetive 9 Incorrect Predictions 24
```

```
Accuracy: 97.86476868327402 Recall: 0.9681978798586572 Precision: 0.9480968858131488 F1 score: 0
-------------------------------
Test data predictions:
[0.69 0.69 3 236 4 0 0.0 0 'product_mng' 'medium'] result = 0
[0.36 0.54 2 153 3 1 0.0 0 'accounting' 'medium'] result = 1

Using Gini Index
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
My Implementation
==================================
Validation data prediction:
Total dataset size 1124
True positive 274 True Negetive 823 Correct Predictions 1097
False positive 18 False Negetive 9 Incorrect Predictions 27
Accuracy: 97.59786476868328 Recall: 0.9681978798586572 Precision: 0.9383561643835616 F1 score: 0
-------------------------------
Test data predictions:
[0.69 0.69 3 236 4 0 0 'product_mng' 'medium' 0.0]  result = 0
[0.36 0.54 2 153 3 1 0 'accounting' 'medium' 0.0]  result = 1
-------------------------------
Scikit learn
==================================
Validation data prediction:
Total dataset size 1124
True positive 274 True Negetive 817 Correct Predictions 1091
False positive 24 False Negetive 9 Incorrect Predictions 33
Accuracy: 97.06405693950177 Recall: 0.9681978798586572 Precision: 0.9194630872483222 F1 score: 0
-------------------------------
Test data predictions:
[0.69 0.69 3 236 4 0 0.0 0 'product_mng' 'medium'] result = 0
[0.36 0.54 2 153 3 1 0.0 0 'accounting' 'medium'] result = 1

Using Miss classification rate
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
My Implementation
==================================
Validation data prediction:
Total dataset size 1124
True positive 265 True Negetive 838 Correct Predictions 1103
False positive 3 False Negetive 18 Incorrect Predictions 21
Accuracy: 98.13167259786478 Recall: 0.9363957597173145 Precision: 0.9888059701492538 F1 score: 0
-------------------------------
```

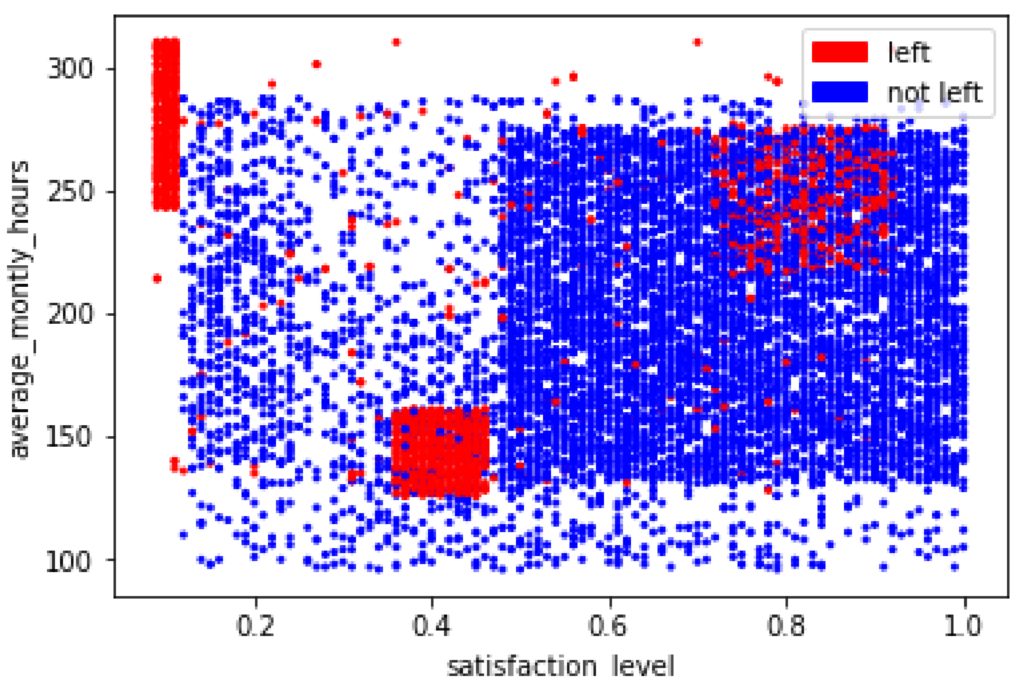
In [16]:
```
#function to plot graph
def print_graph(data,attr1,attr2):
    col = []
    for i in data[attr_name]:
```

```
        if i == 1:
            col += ['#ff0000'] #red color for left entries
        else:
            col += ['#0000ff'] #blue color for non-left entries

    plt.scatter(data[attr1],data[attr2],c=col,s=4)
    plt.xlabel(attr1)
    plt.ylabel(attr2)
    red_patch = mpatches.Patch(color = 'red', label='left')
    blue_patch = mpatches.Patch(color = 'blue', label='not left')
    plt.legend(handles=[red_patch,blue_patch])
    plt.show()


pdata = pd.read_csv(data_loc)
for i in pdata.columns.values:
    for j in pdata.columns.values:
        if i != j and i != attr_name and j != attr_name and pdata.columns.get_loc(i) <
            print_graph(pdata,i,j)
```

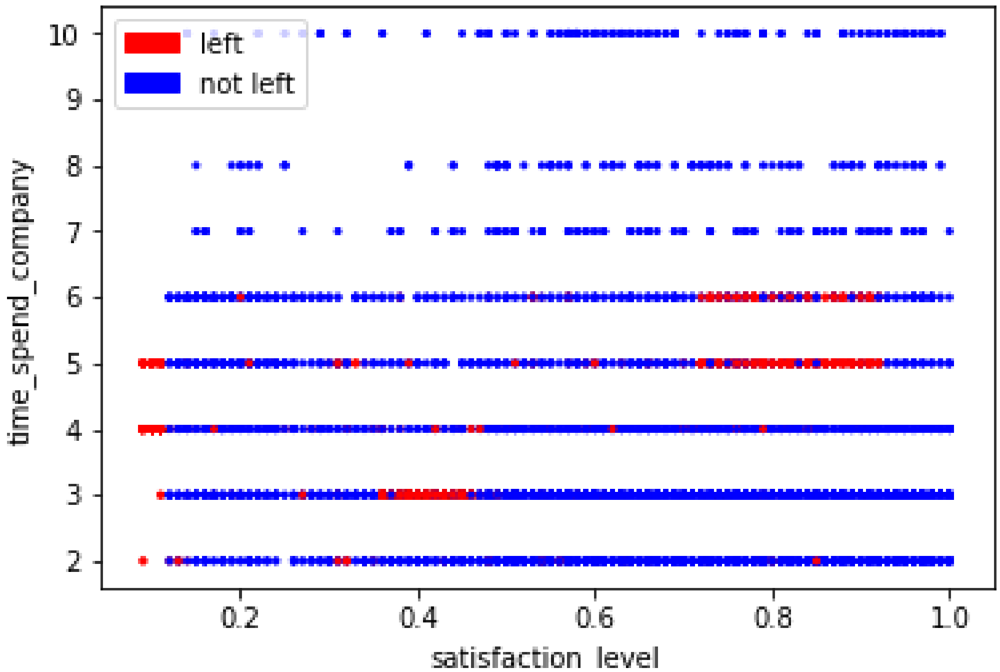


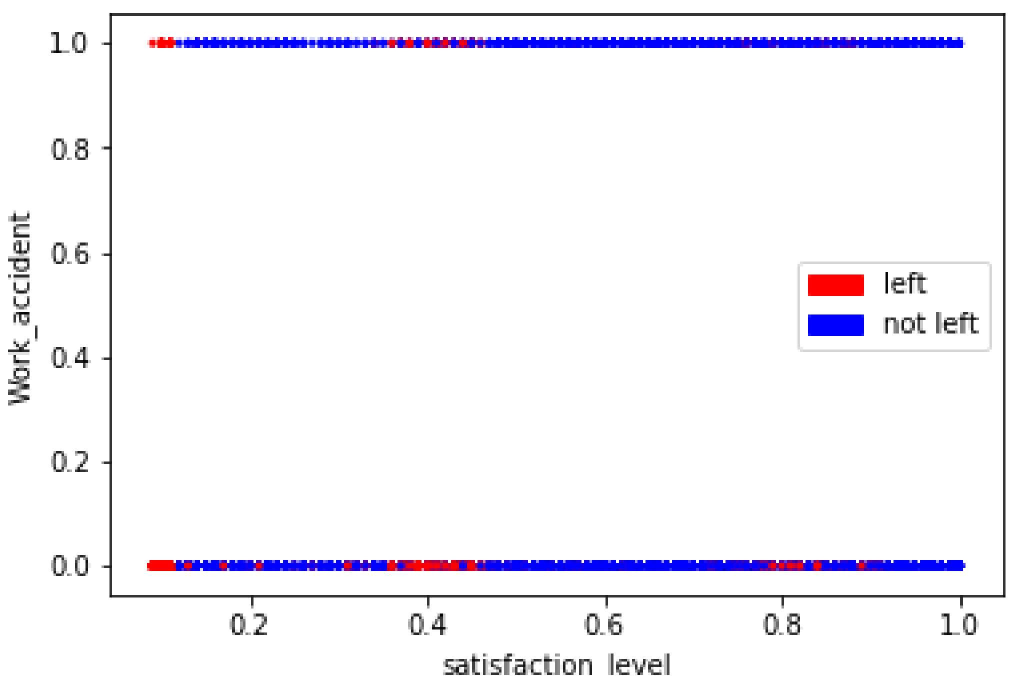


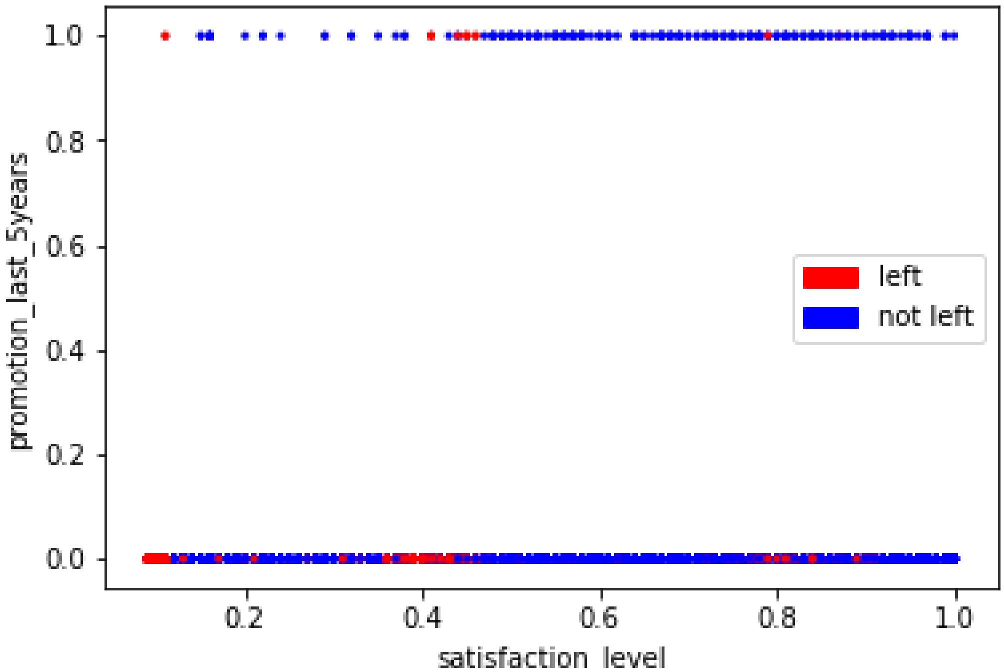
left
not left
number_project
satisfaction_level

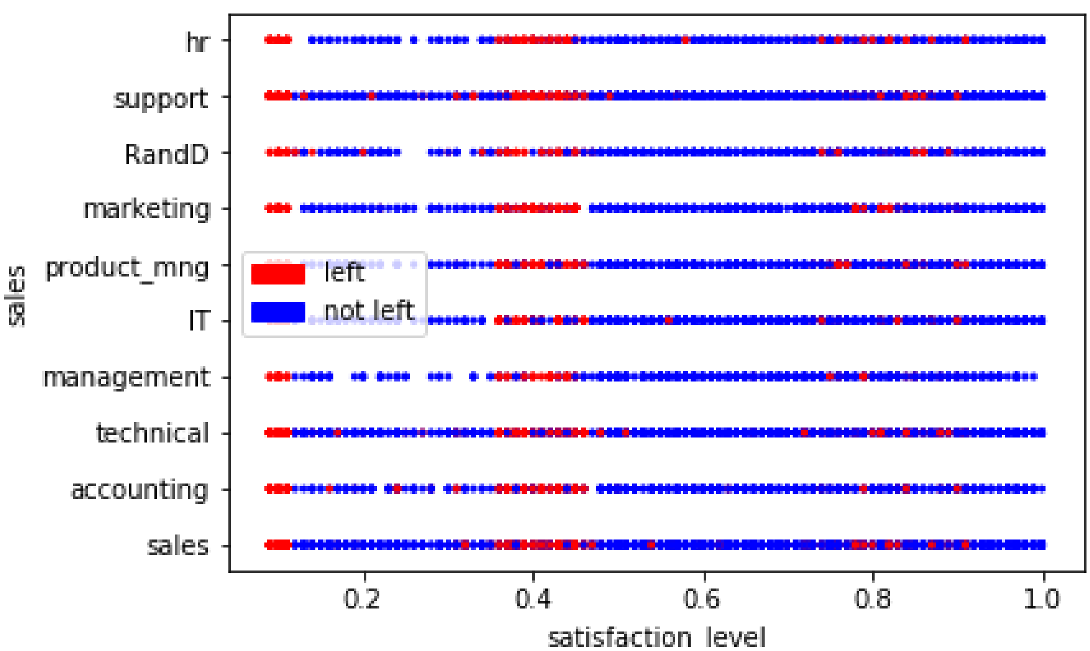


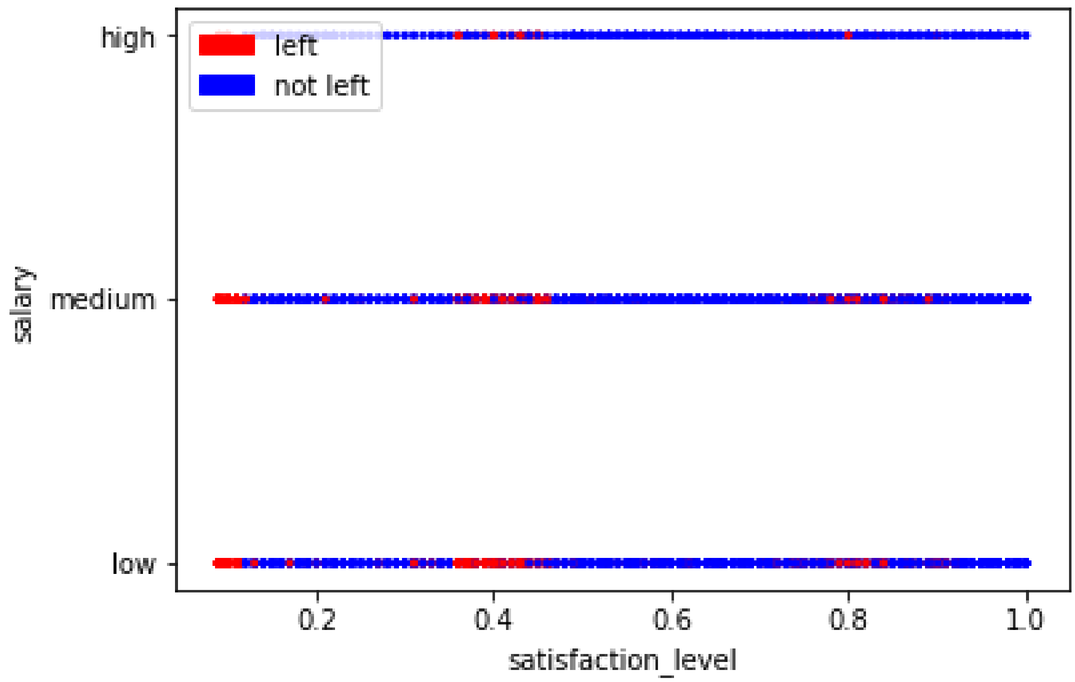
left
not left
average_montly_hours
satisfaction_level

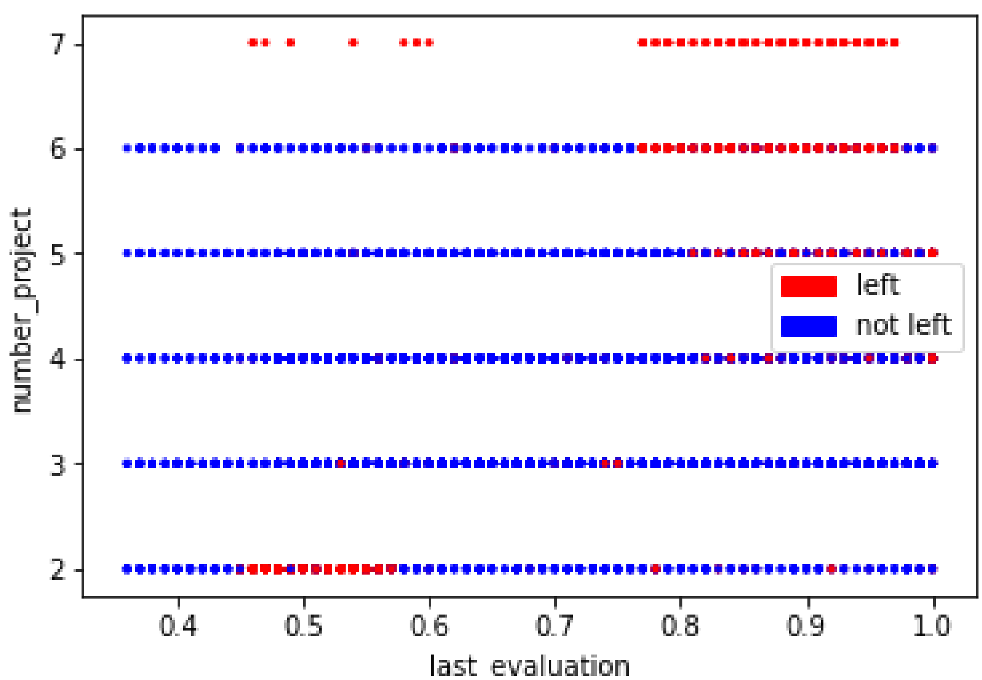

left
not left
time_spend_company
satisfaction_level
10
9
8
7
6
5
4
3
2
0.2
0.4
0.6
0.8
1.0

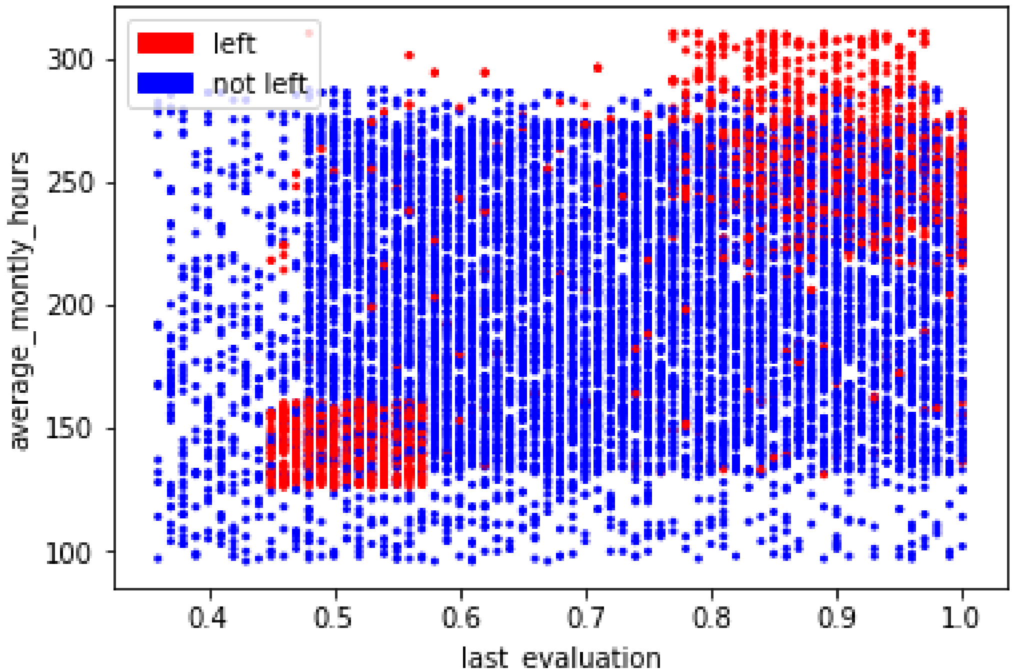

left
not left
Work_accident
satisfaction_level
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
promotion_last_5years
left
not left
0.2
0.4
0.6
0.8
1.0
satisfaction_level

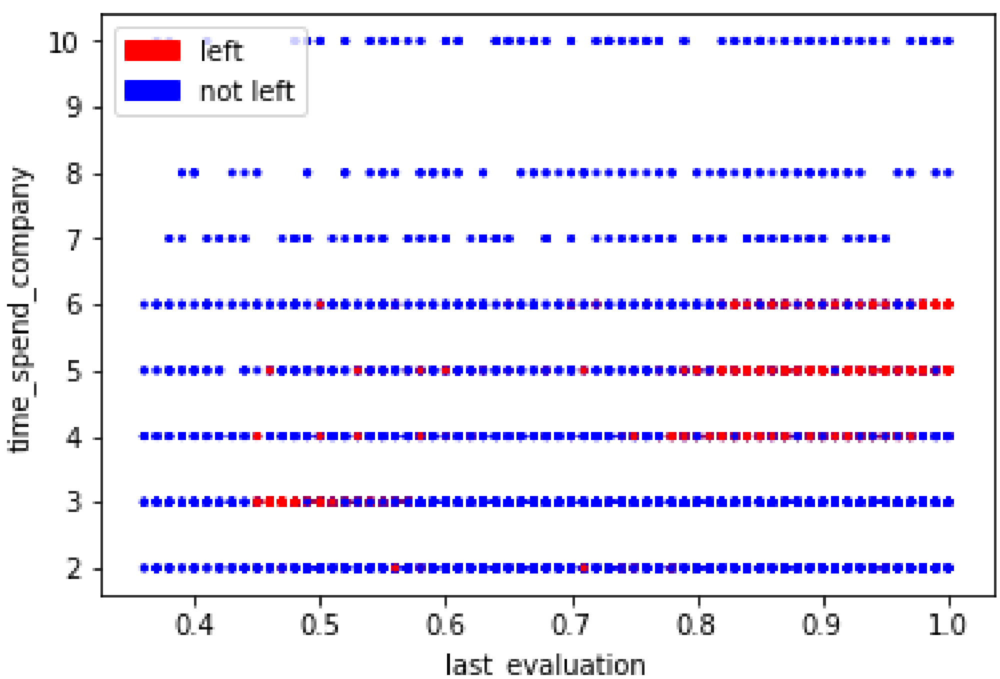

hr
support
RandD
marketing
product_mng
IT
management
technical
accounting
sales
sales
left
not left
0.2
0.4
0.6
0.8
1.0
satisfaction_level


high
medium
low
salary
left
not left
0.2
0.4
0.6
0.8
1.0
satisfaction_level

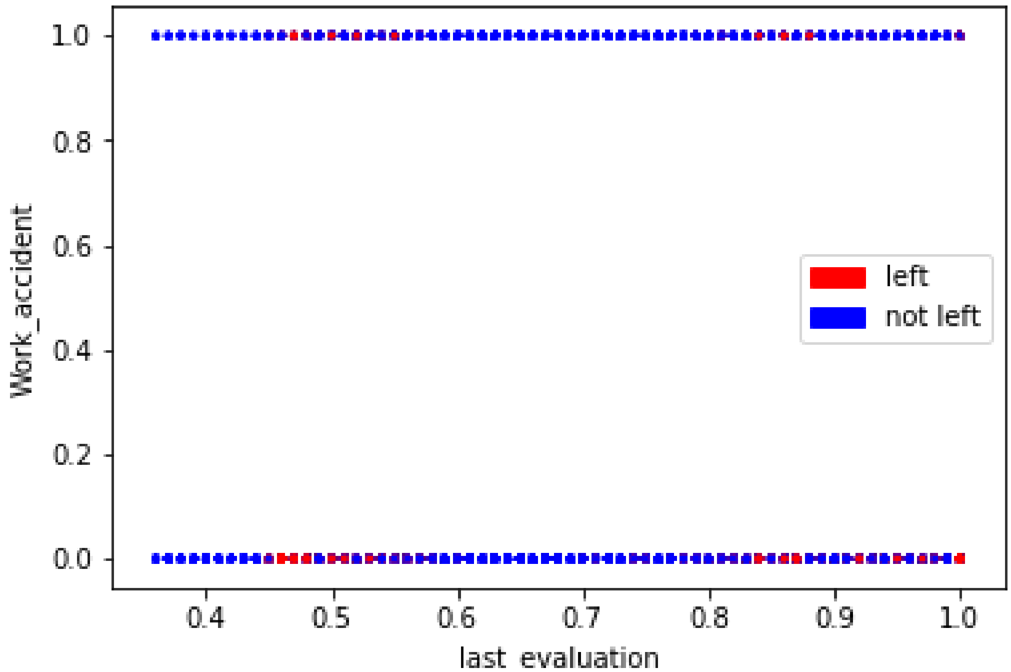

7
6
5
4
3
2
number_project
left
not left
0.4
0.5
0.6
0.7
0.8
0.9
1.0
last_evaluation



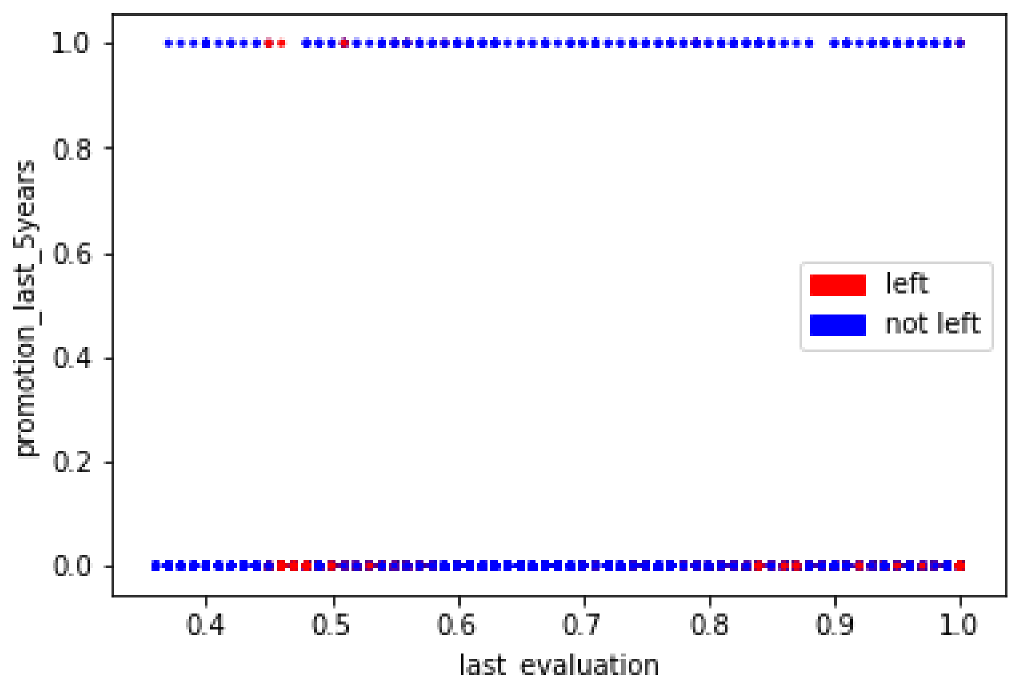
left
not left
300
250
200
150
100
average_montly_hours
0.4
0.5
0.6
0.7
0.8
0.9
1.0
last_evaluation



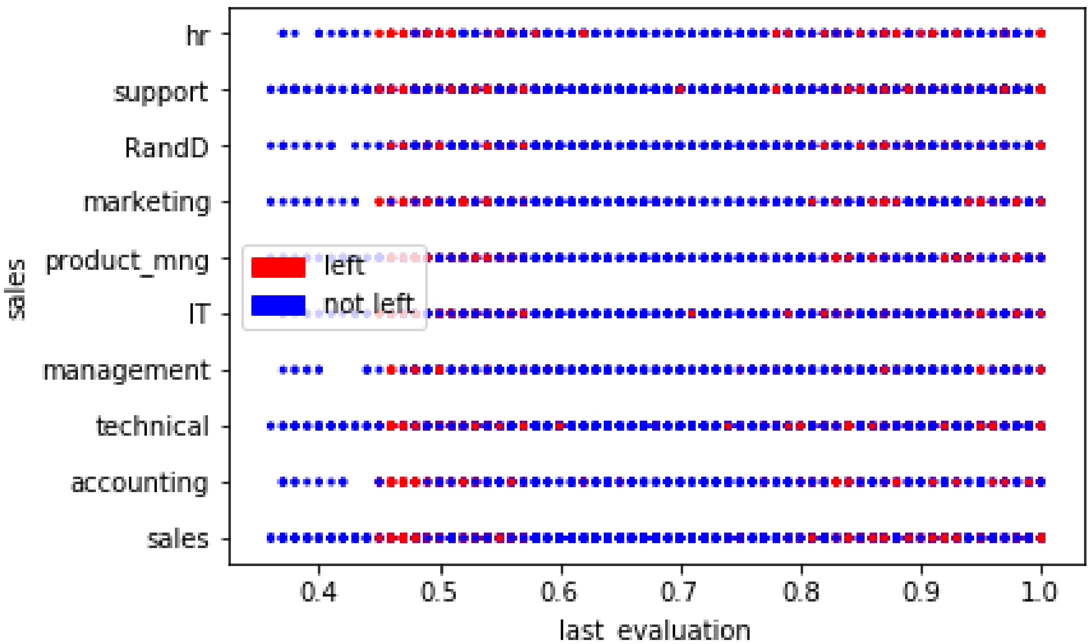
left
not left
10
9
8
7
6
5
4
3
2
time_spend_company
0.4
0.5
0.6
0.7
0.8
0.9
1.0
last_evaluation

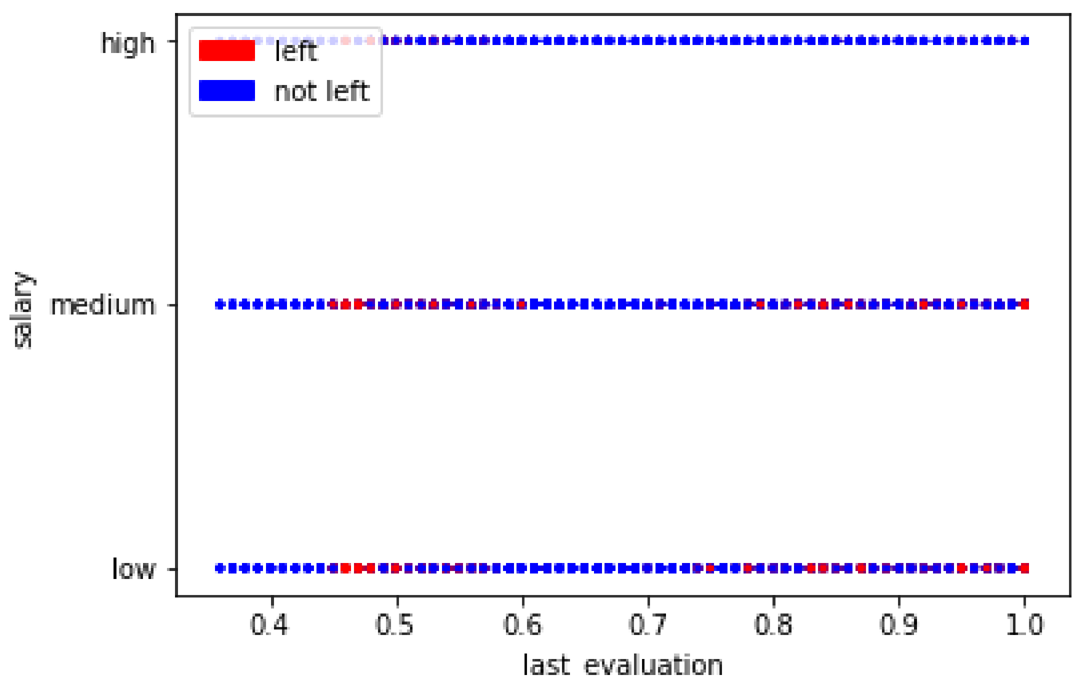


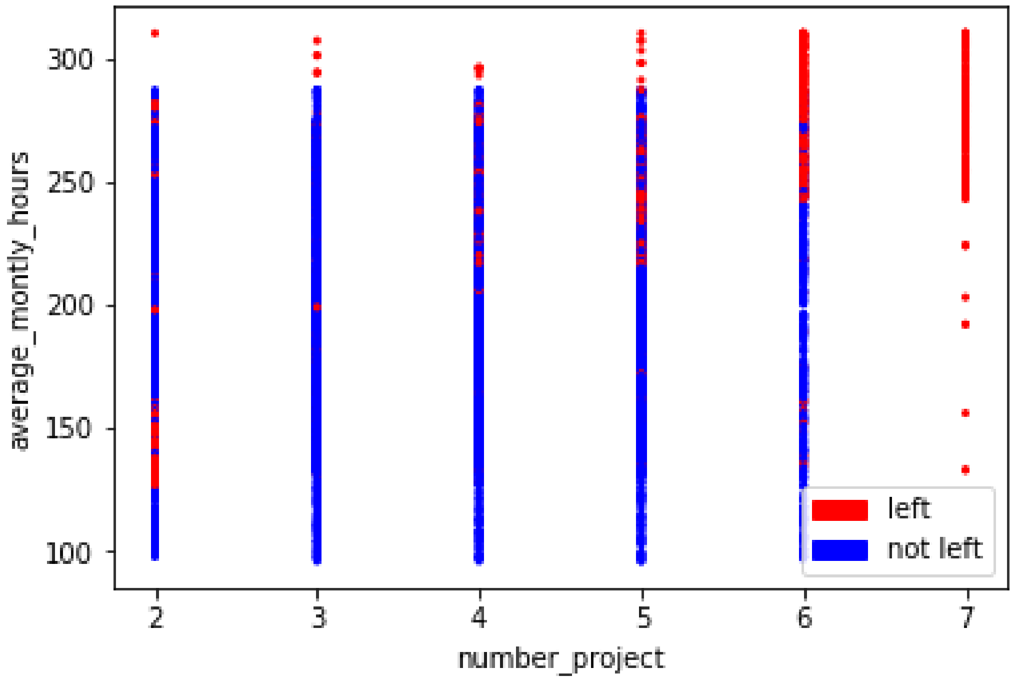
1.0
0.8
0.6
0.4
0.2
0.0
Work_accident
left
not left
0.4
0.5
0.6
0.7
0.8
0.9
1.0
last_evaluation



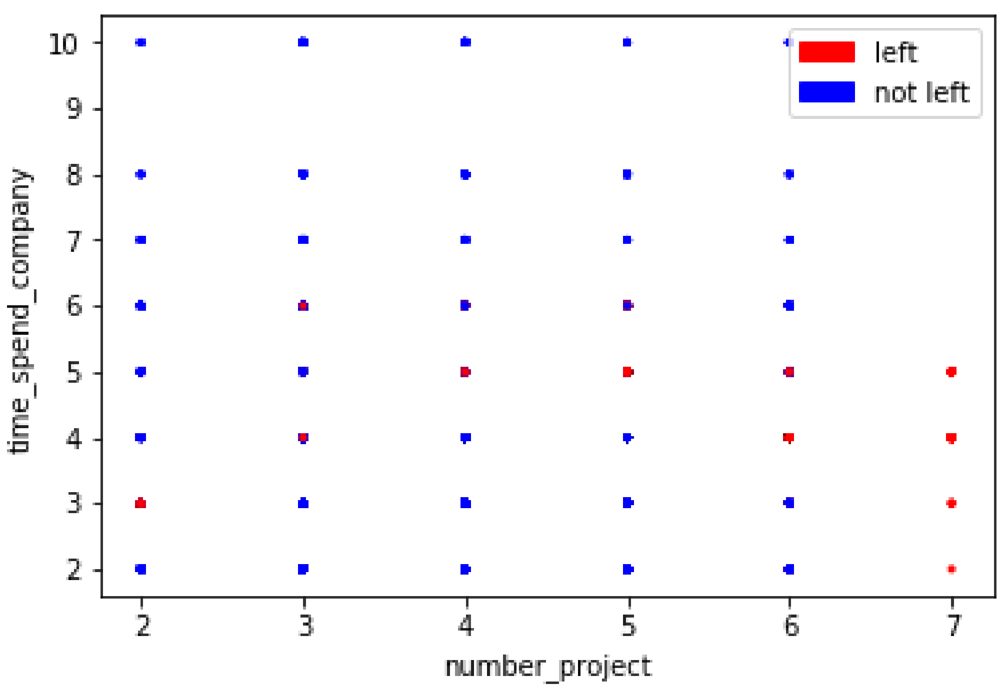
1.0
0.8
0.6
0.4
0.2
0.0
promotion_last_5years
left
not left
0.4
0.5
0.6
0.7
0.8
0.9
1.0
last_evaluation


hr
support
RandD
marketing
product_mng
IT
management
technical
accounting
sales
sales
left
not left
0.4
0.5
0.6
0.7
0.8
0.9
1.0
last_evaluation

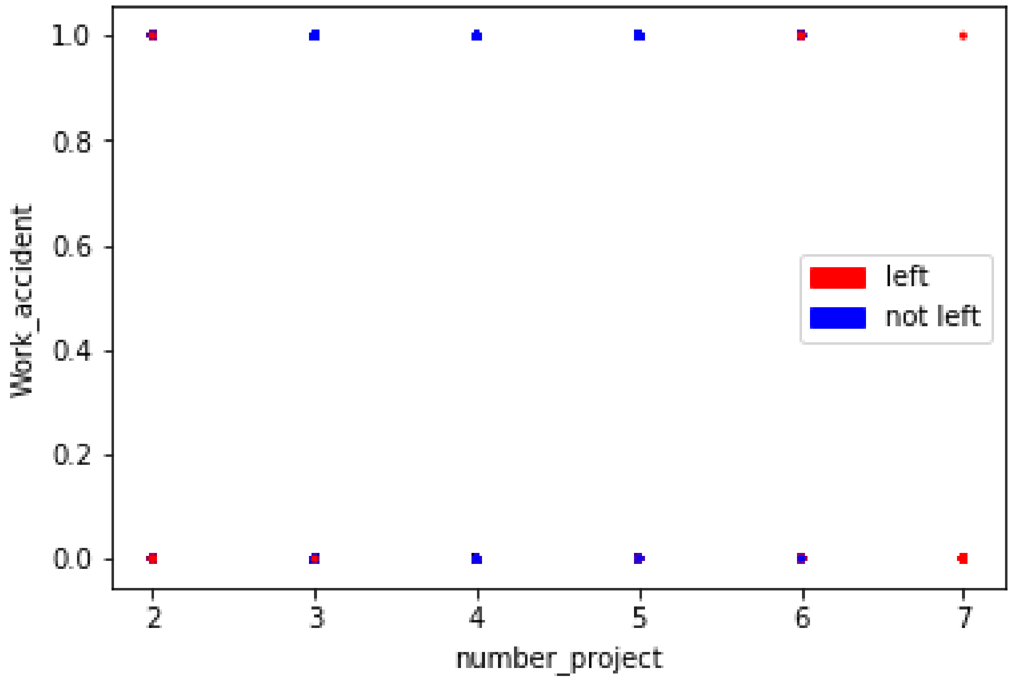

high
medium
low
salary
left
not left
0.4
0.5
0.6
0.7
0.8
0.9
1.0
last_evaluation



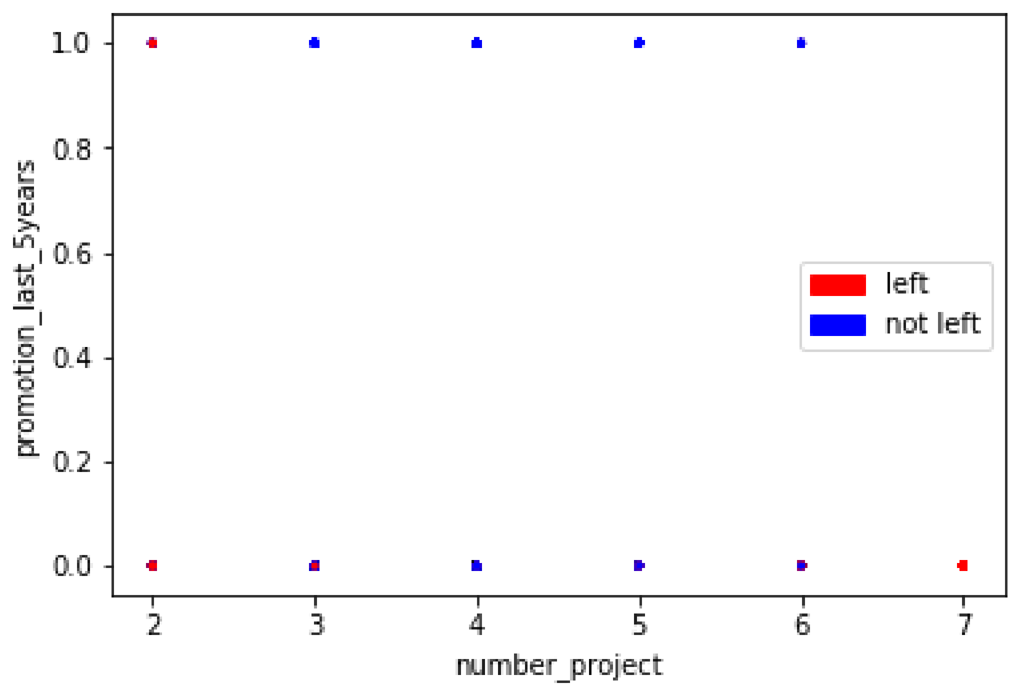
average_montly_hours
300
250
200
150
100
2
3
4
5
6
7
number_project
left
not left



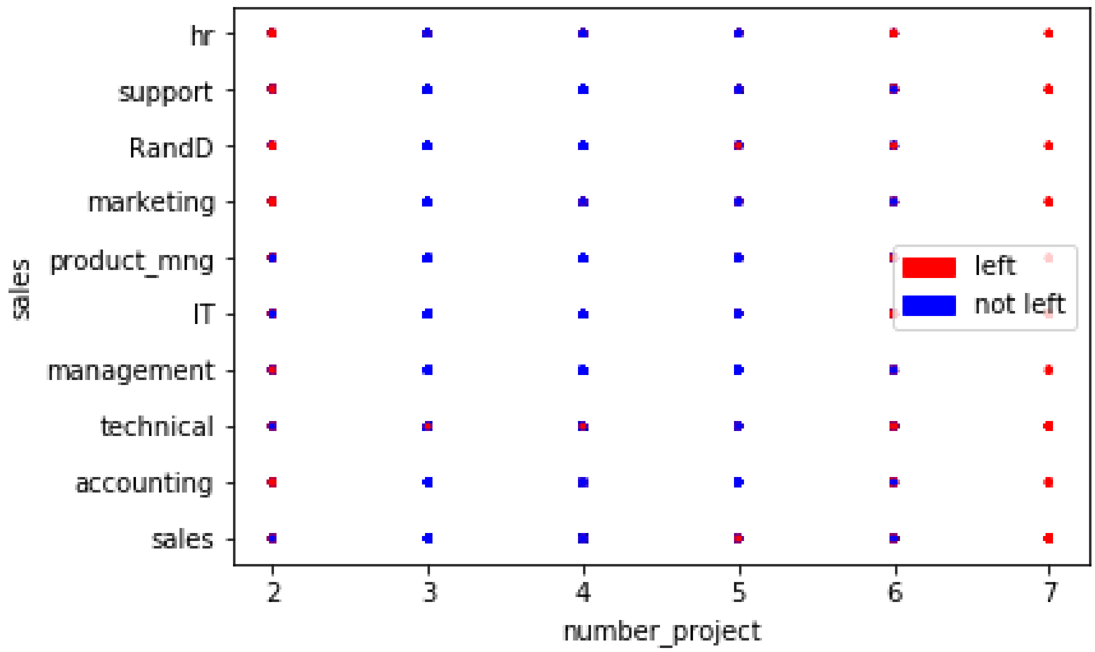
time_spend_company
10
9
8
7
6
5
4
3
2
2
3
4
5
6
7
number_project
left
not left



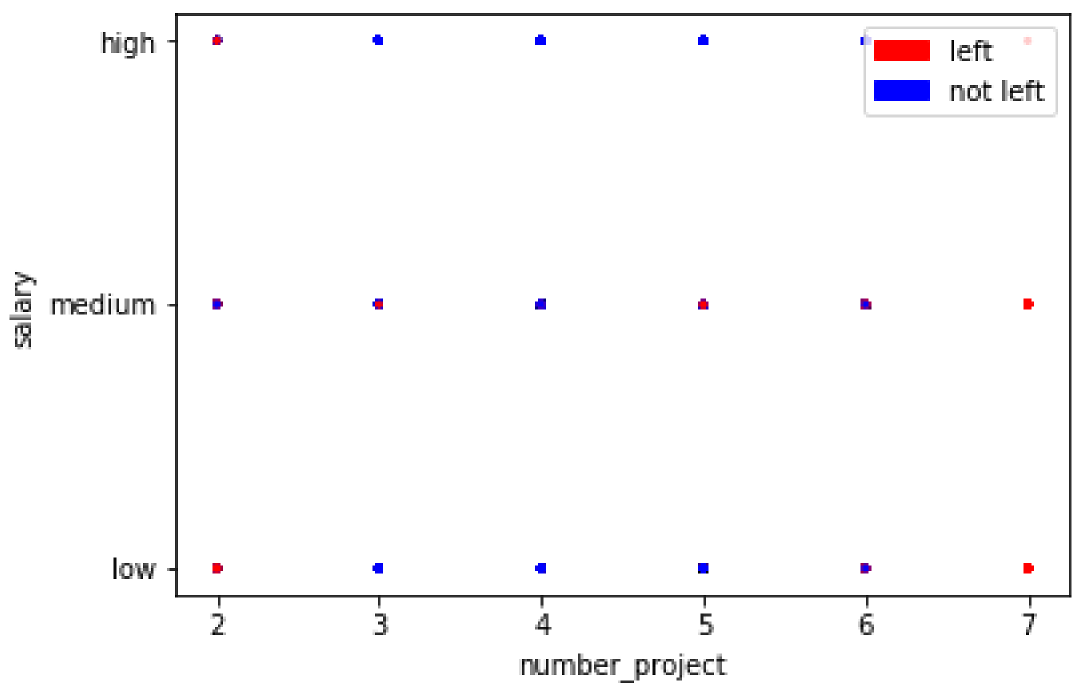
Work_accident
number_project
left
not left



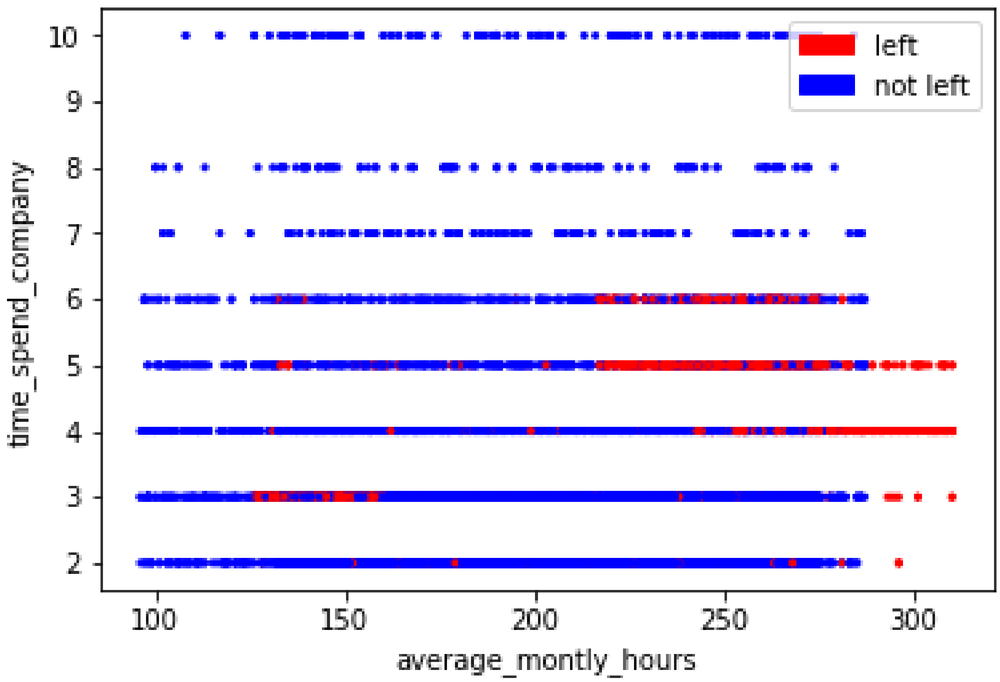
promotion_last_5years
number_project
left
not left

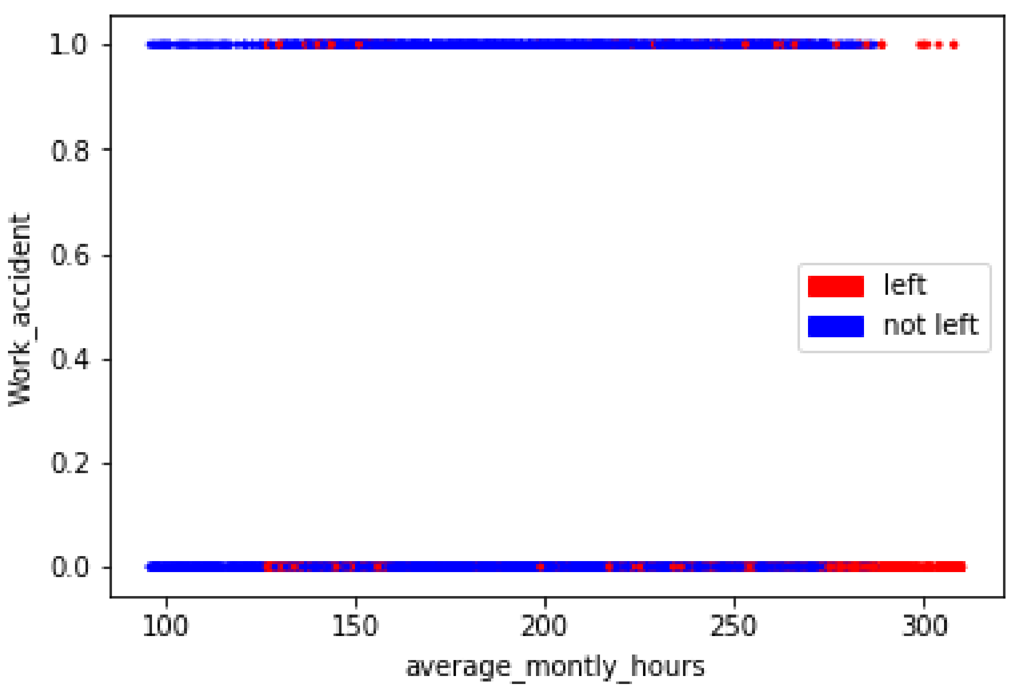


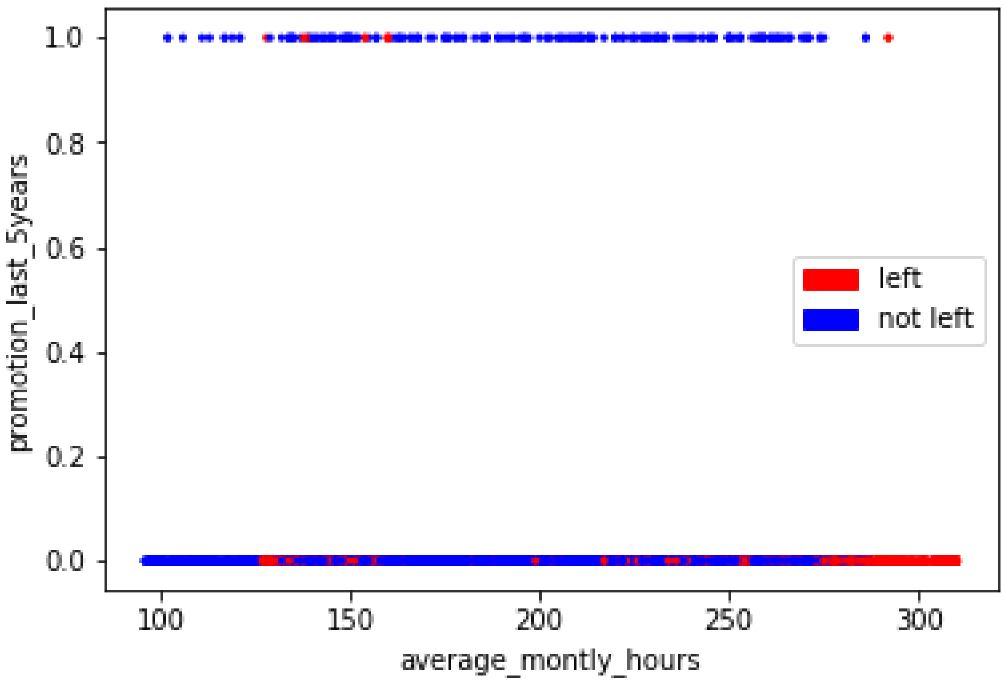
hr
support
RandD
marketing
product_mng
IT
management
technical
accounting
sales
sales
2
3
4
5
6
7
number_project
left
not left

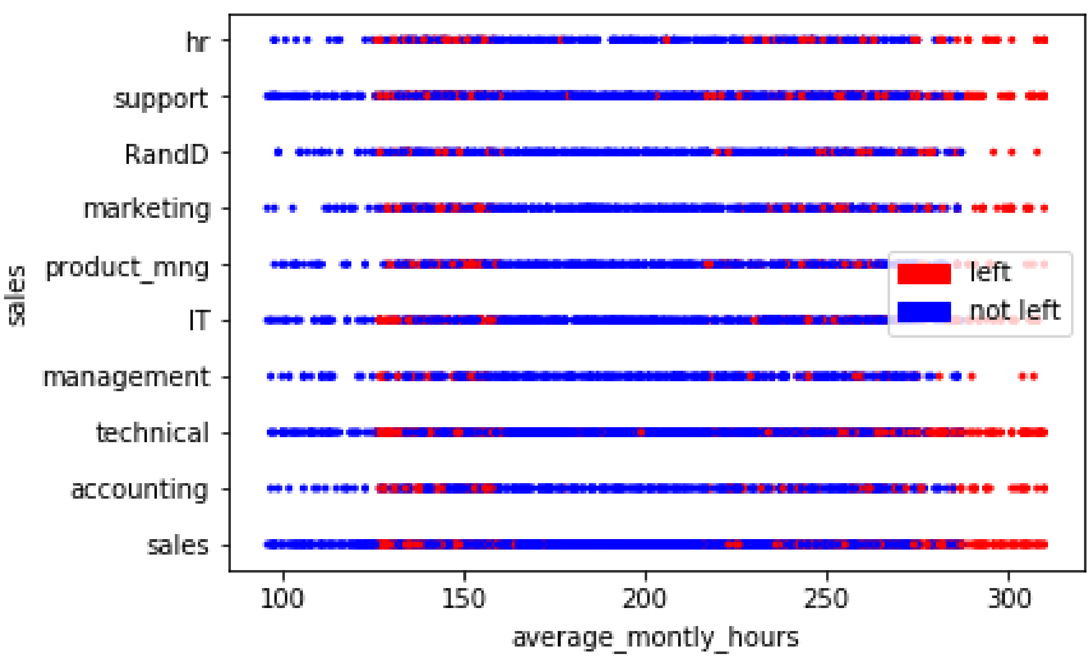


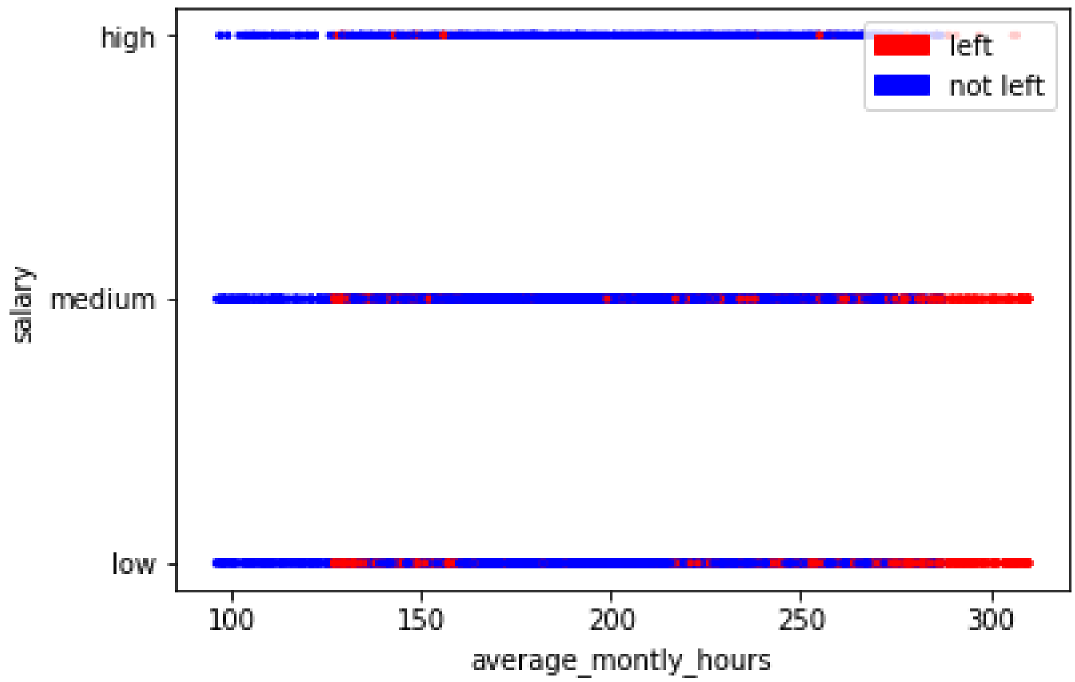
high
medium
low
salary
2
3
4
5
6
7
number_project
left
not left

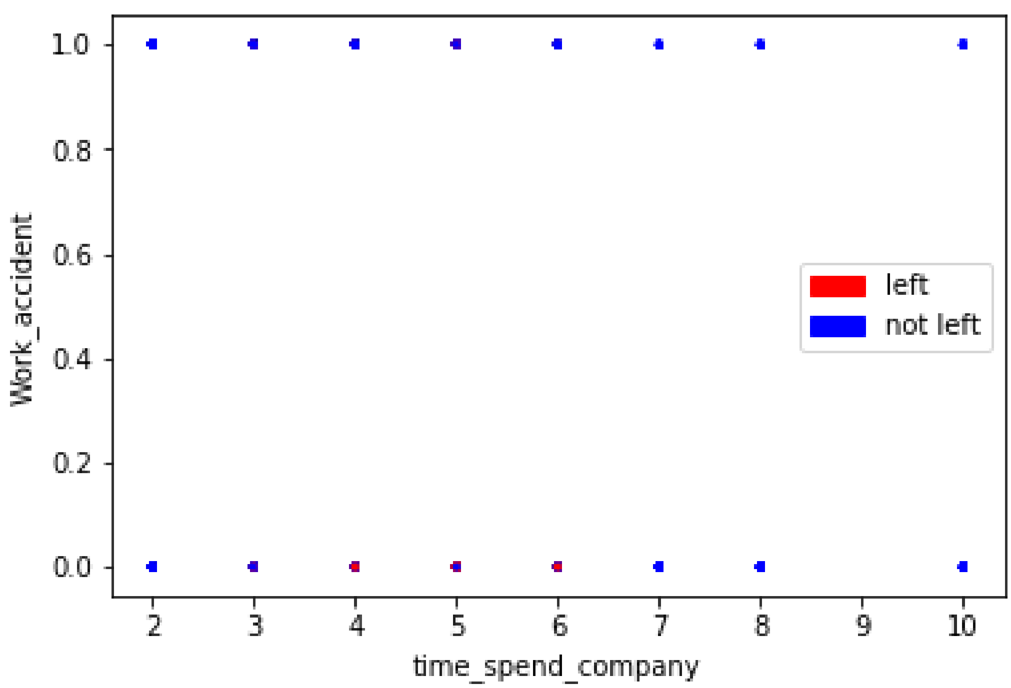

left
not left
time_spend_company
average_montly_hours

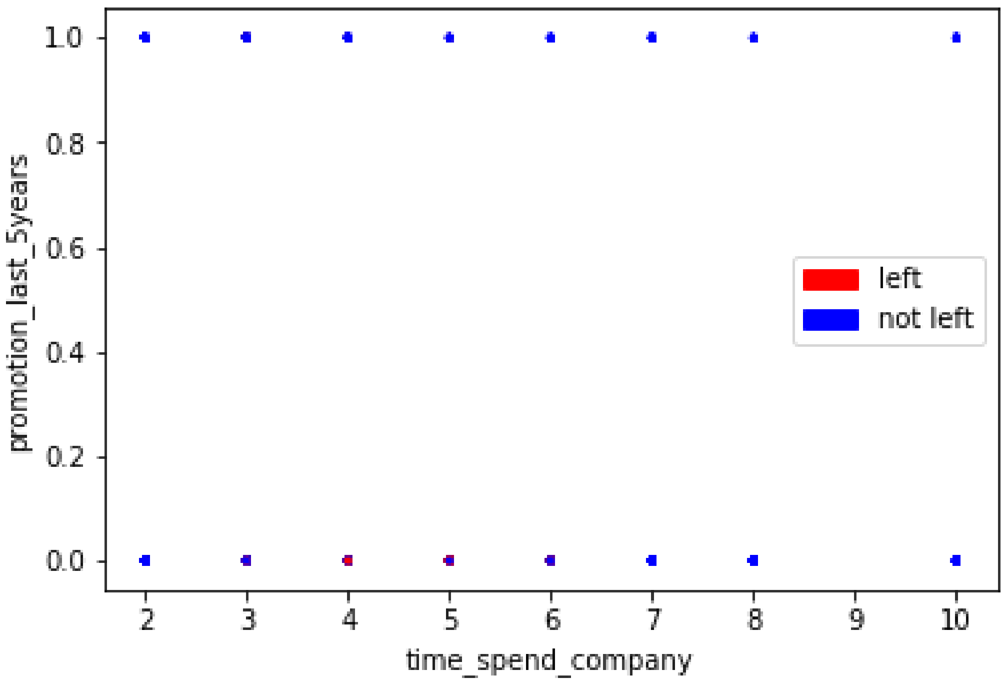

left
not left
Work_accident
average_montly_hours

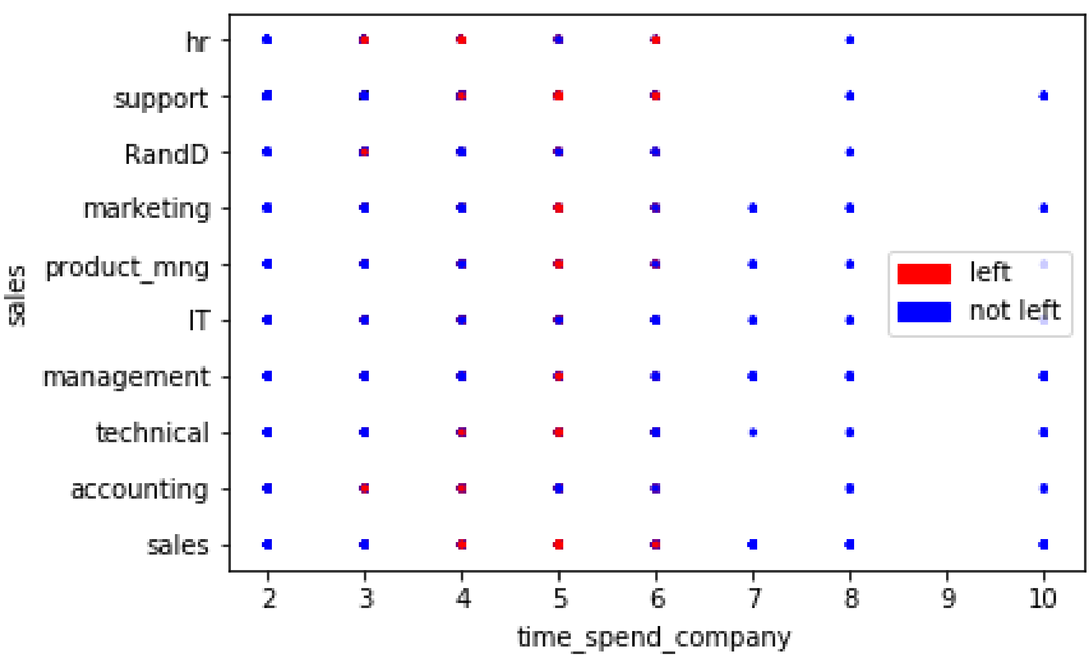

1.0
0.8
0.6
0.4
0.2
0.0
promotion_last_5years
100
150
200
250
300
average_montly_hours
left
not left

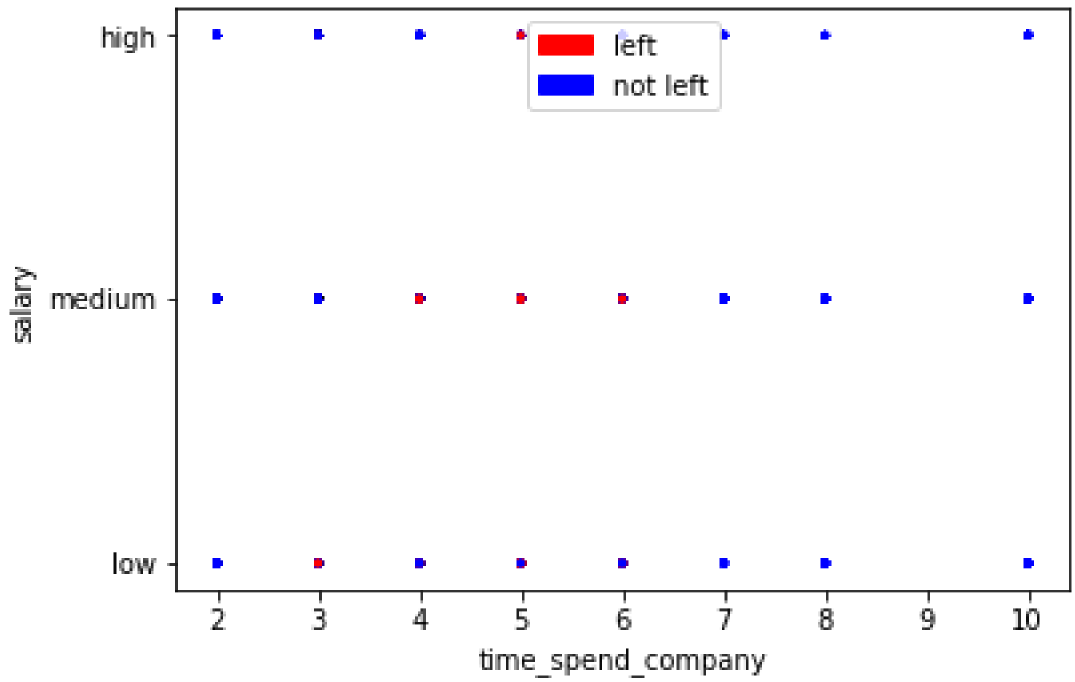

hr
support
RandD
marketing
product_mng
IT
management
technical
accounting
sales
sales
100
150
200
250
300
average_montly_hours
left
not left

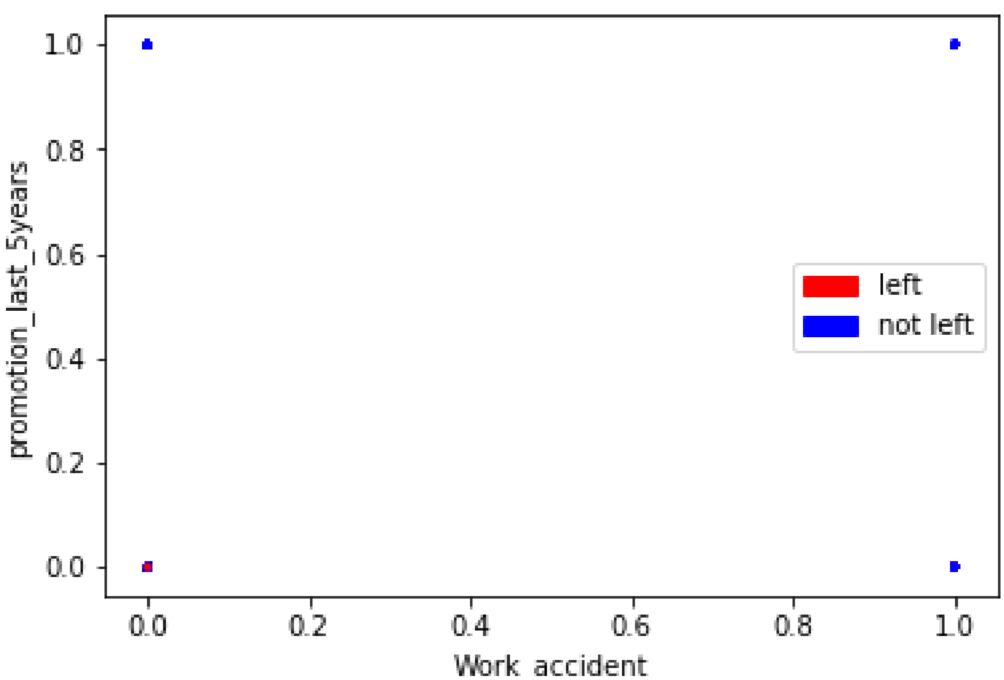

high
medium
low
salary
100
150
200
250
300
average_montly_hours
left
not left

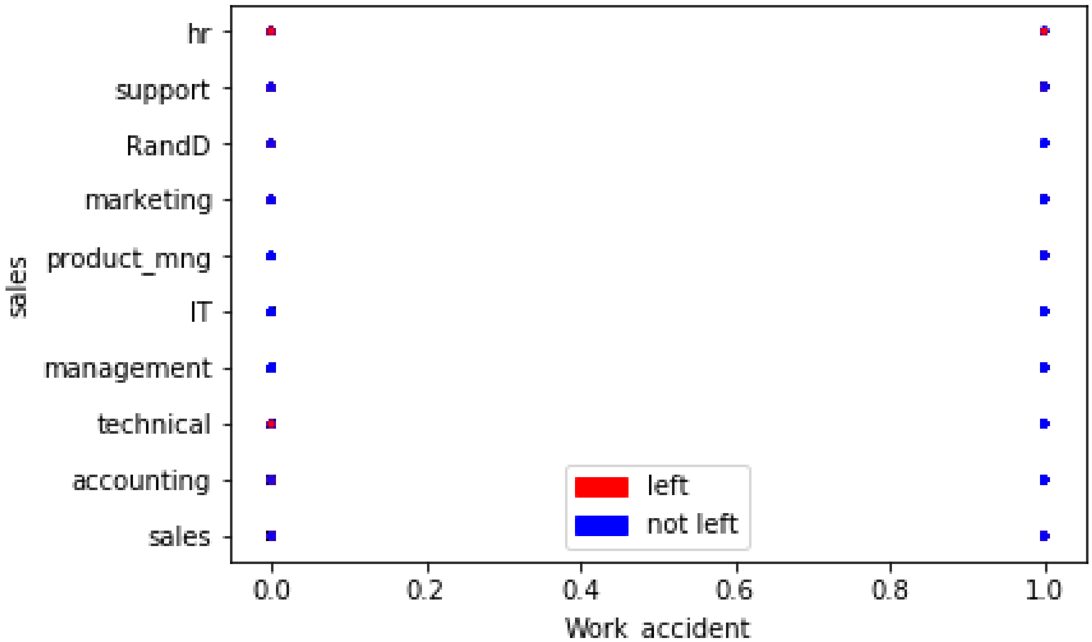

1.0
0.8
0.6
0.4
0.2
0.0
Work_accident
2
3
4
5
6
7
8
9
10
time_spend_company
left
not left

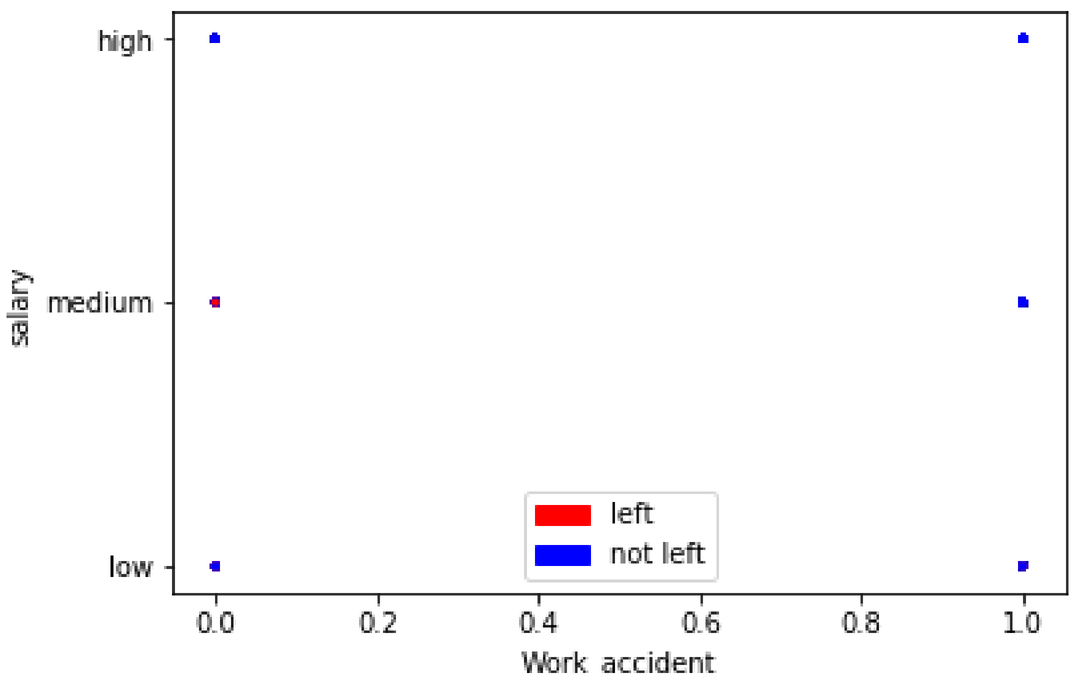

1.0
0.8
0.6
0.4
0.2
0.0
promotion_last_5years
left
not left
2
3
4
5
6
7
8
9
10
time_spend_company

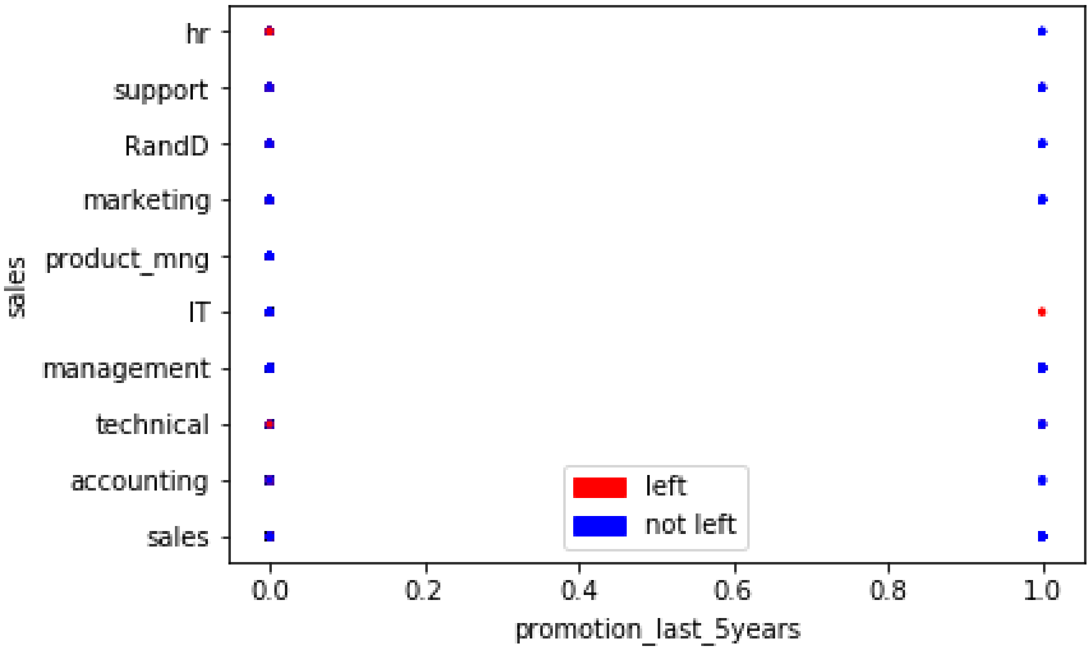

hr
support
RandD
marketing
product_mng
IT
management
technical
accounting
sales
sales
left
not left
2
3
4
5
6
7
8
9
10
time_spend_company

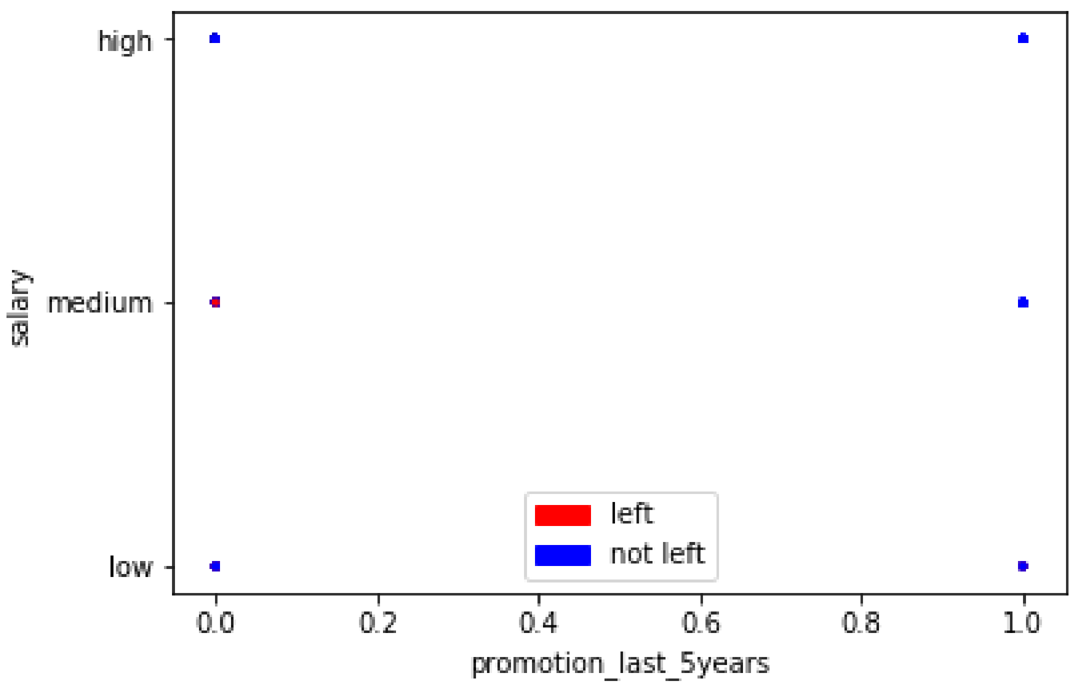

high
medium
low
salary
2
3
4
5
6
7
8
9
10
time_spend_company
left
not left

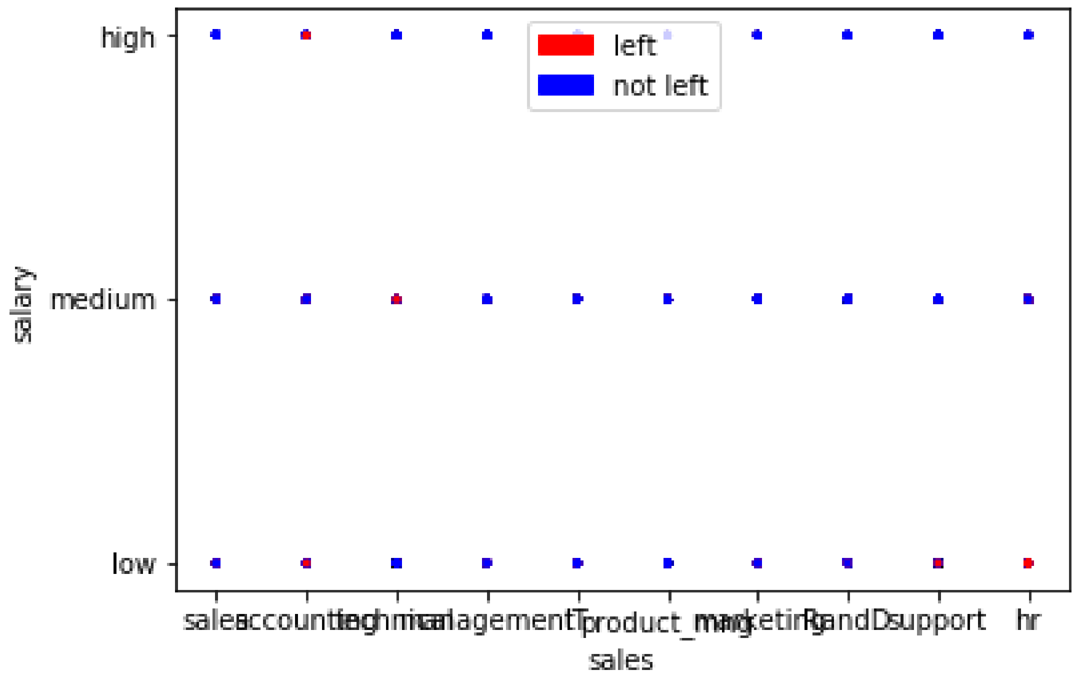

1.0
0.8
0.6
0.4
0.2
0.0
promotion_last_5years
0.0
0.2
0.4
0.6
0.8
1.0
Work_accident
left
not left


hr
support
RandD
marketing
product_mng
IT
management
technical
accounting
sales
0.0
0.2
0.4
0.6
0.8
1.0
Work_accident
left
not left


salary
high
medium
low
0.0
0.2
0.4
0.6
0.8
1.0
Work_accident
left
not left

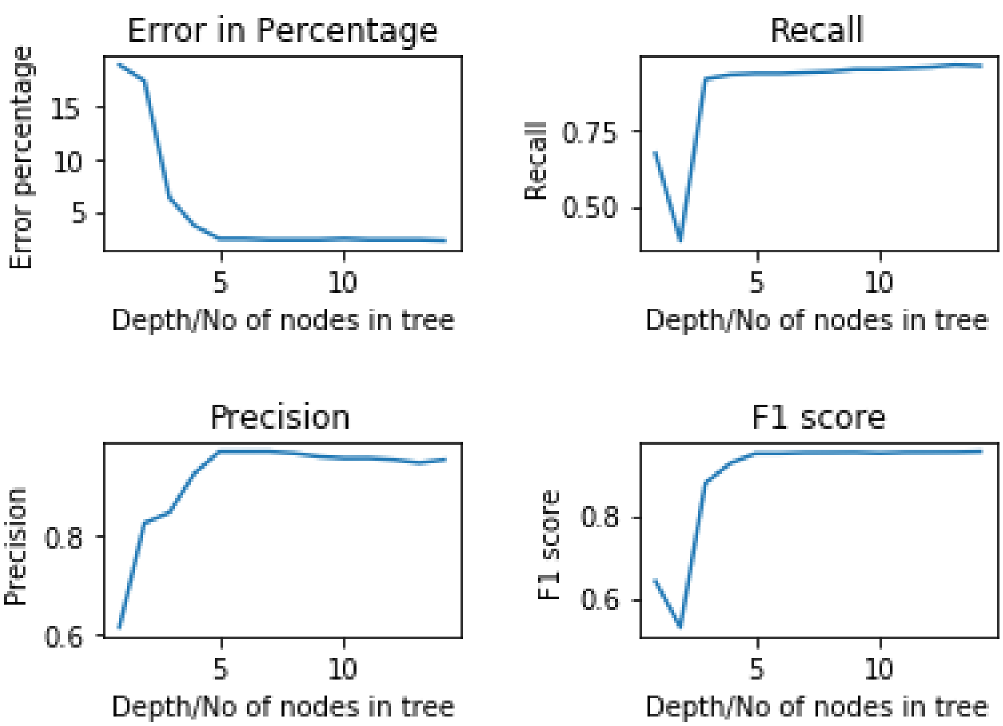

hr
support
RandD
marketing
product_mng
IT
management
technical
accounting
sales
sales
left
not left
0.0
0.2
0.4
0.6
0.8
1.0
promotion_last_5years


high
medium
low
salary
left
not left
0.0
0.2
0.4
0.6
0.8
1.0
promotion_last_5years



In [17]:
```
#set parameters for training on a numerical + categorical data
setParams('decision_Tree/train.csv',0.90,[0,0,0,0,0,1,1,1,1,1],"left",entropy)
rec_max = 14
x = [x for x in range(1,rec_max+1)]
res = runDecisionTree(showstats=0,schk=0,max_rec=1)
y = [res[0]]
for i in x[1:]:
    y += [runDecisionTree(showstats=0,schk=0,max_rec=i,tree=res[1])[0]]
y = np.array(y)

plt.subplot(2,2,1)
plt.subplots_adjust(wspace=0.5,hspace=1)
ye = np.ones([1,rec_max])*100 - y[:,0]
plt.plot(x,ye.reshape(-1,1))
plt.title("Error in Percentage")
plt.xlabel("Depth/No of nodes in tree")
plt.ylabel("Error percentage")

plt.subplot(2,2,2)
yr = y[:,1]
plt.plot(x,yr.reshape(-1,1))
plt.title("Recall")
plt.xlabel("Depth/No of nodes in tree")
plt.ylabel("Recall")
```

```
plt.subplot(2,2,3)
yp = y[:,2]
plt.plot(x,yp.reshape(-1,1))
plt.title("Precision")
plt.xlabel("Depth/No of nodes in tree")
plt.ylabel("Precision")

plt.subplot(2,2,4)
yf = y[:,3]
plt.plot(x,yf.reshape(-1,1))
plt.title("F1 score")
plt.xlabel("Depth/No of nodes in tree")
plt.ylabel("F1 score")
plt.show()
```




```
In [18]: setParams('decision_Tree/train.csv',1/100,[0,0,0,0,0,1,1,1,1,1],"left",entropy,'decisio
         ret,tree = runDecisionTree(showstats=0,schk=0)
         x = [0.01]
         y = [ret[0]]
         for i in range(2,100):
             x += [i/100]
             setParams('decision_Tree/train.csv',i/100,[0,0,0,0,0,1,1,1,1,1],"left",entropy,'dec
             res = (runDecisionTree(showstats=0,schk=0)[0])
```

```
        y += [res[0]]

    plt.plot(x,y)
```

Out[18]: [<matplotlib.lines.Line2D at 0x7f6080cff1d0>]