LAB 3 CARAVAN INSURANCE PROBLEM

October 7, 2020

1 Lab 3

[1]:
```
import pandas as pd
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.metrics.pairwise import pairwise_distances
import matplotlib.pyplot as plt
import csv
import seaborn as sns
import collections, numpy


from sklearn import metrics
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Load dataset
train = pd.read_csv('caravan.csv')
```

```
X_train = train.iloc[:,0:-1]
y_train = train.iloc[:,-1]
train.head()

test = pd.read_csv('caravanTest.csv')

X_test = train.iloc[:,0:-1]
y_test = train.iloc[:,-1]
test.head()

print('Training set shape:    ', X_train.shape, y_train.shape)
print('Test set shape:        ', X_test.shape, y_test.shape)
```

```
Training set shape:    (5822, 85) (5822,)
Test set shape:        (5822, 85) (5822,)
```

[3]:
```
X = train.iloc[:,0:-1]
y = train.iloc[:,-1]
```

1.1 Assignment 1

The focus of this assignment is to anwer the question: "Can you describe a potential customer interested in buyng a caravan insurance?"

In order to answer this question we need to perfom feature seleciton. That is we need to identify a set of features that is able to suggest the main characteristics a customer may have in order for him to be interested in buying an insurance.

We'll apply different methods and check out the one that gives the best results in terms of predictive accuracy (in predicting the class). This way we hopefully will identify the characteristics that best describe a potential customer.

[4]:
```
print(train.shape)
print(test.shape)
```

```
(5822, 86)
(4000, 86)
```

[5]:
```
train.describe()
```

[5]:
```
       Customer Subtype  Number of houses  Avg size household      Avg Age  \
count       5822.000000       5822.000000         5822.000000  5822.000000
mean          24.253349          1.110615            2.678805     2.991240
std           12.846706          0.405842            0.789835     0.814589
min            1.000000          1.000000            1.000000     1.000000
25%           10.000000          1.000000            2.000000     2.000000
50%           30.000000          1.000000            3.000000     3.000000
```

```
75%          35.000000          1.000000            3.000000     3.000000
max          41.000000         10.000000            5.000000     6.000000

       Customer main type  Roman catholic   Protestant  Other religion  \
count         5822.000000     5822.000000  5822.000000     5822.000000
mean             5.773617        0.696496     4.626932        1.069907
std              2.856760        1.003234     1.715843        1.017503
min              1.000000        0.000000     0.000000        0.000000
25%              3.000000        0.000000     4.000000        0.000000
50%              7.000000        0.000000     5.000000        1.000000
75%              8.000000        1.000000     6.000000        2.000000
max             10.000000        9.000000     9.000000        5.000000

       No religion      Married  ...  \
count  5822.000000  5822.000000  ...
mean      3.258502     6.183442  ...
std       1.597647     1.909482  ...
min       0.000000     0.000000  ...
25%       2.000000     5.000000  ...
50%       3.000000     6.000000  ...
75%       4.000000     7.000000  ...
max       9.000000     9.000000  ...

       Number of private accident insurance policies  \
count                                    5822.000000
mean                                        0.005325
std                                         0.072782
min                                         0.000000
25%                                         0.000000
50%                                         0.000000
75%                                         0.000000
max                                         1.000000

       Number of family accidents insurance policies  \
count                                    5822.000000
mean                                        0.006527
std                                         0.080532
min                                         0.000000
25%                                         0.000000
50%                                         0.000000
75%                                         0.000000
max                                         1.000000

       Number of disability insurance policies  Number of re policies  \
count                              5822.000000            5822.000000
mean                                  0.004638               0.570079
std                                   0.077403               0.562058
```

```
min                                       0.000000              0.000000
25%                                       0.000000              0.000000
50%                                       0.000000              1.000000
75%                                       0.000000              1.000000
max                                       2.000000              7.000000

       Number of surfboard policies  Number of boat policies  \
count                   5822.000000              5822.000000
mean                       0.000515                 0.006012
std                        0.022696                 0.081632
min                        0.000000                 0.000000
25%                        0.000000                 0.000000
50%                        0.000000                 0.000000
75%                        0.000000                 0.000000
max                        1.000000                 2.000000

       Number of bicycle policies  Number of property insurance policies  \
count                 5822.000000                            5822.000000
mean                     0.031776                               0.007901
std                      0.210986                               0.090463
min                      0.000000                               0.000000
25%                      0.000000                               0.000000
50%                      0.000000                               0.000000
75%                      0.000000                               0.000000
max                      3.000000                               2.000000

       Number of social security insurance policies  CARAVAN POLICY
count                                   5822.000000     5822.000000
mean                                       0.014256        0.059773
std                                        0.119996        0.237087
min                                        0.000000        0.000000
25%                                        0.000000        0.000000
50%                                        0.000000        0.000000
75%                                        0.000000        0.000000
max                                        2.000000        1.000000

[8 rows x 86 columns]
```

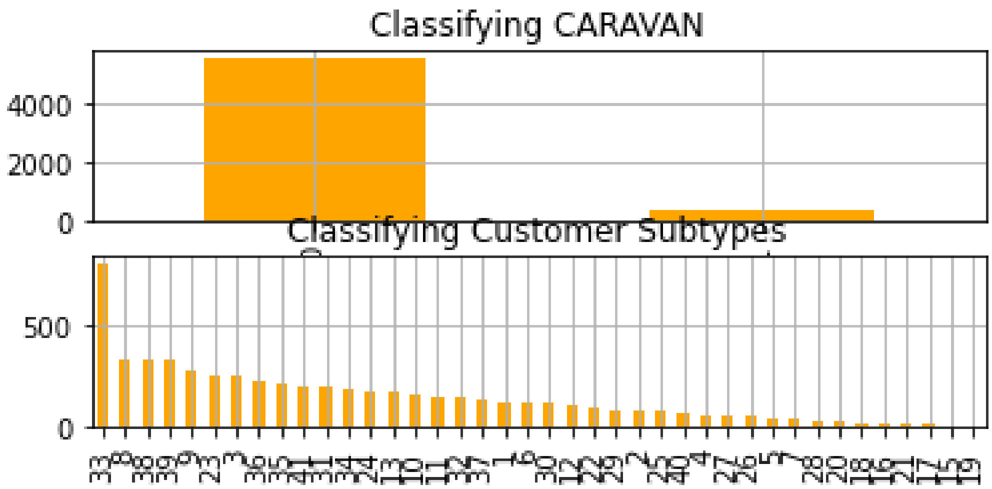
Training data set has 5822 instances each with 86 attributes. The features are divided into sociodemographic data (attribute 1-43) which is derived from zip codes which means that these attributes are categorical without order (Nominal). Features from 44 to 86 refer to product ownership, these can be ranked and can be considered ordinal features.

Similarly we have 4000 customer records in testing set.

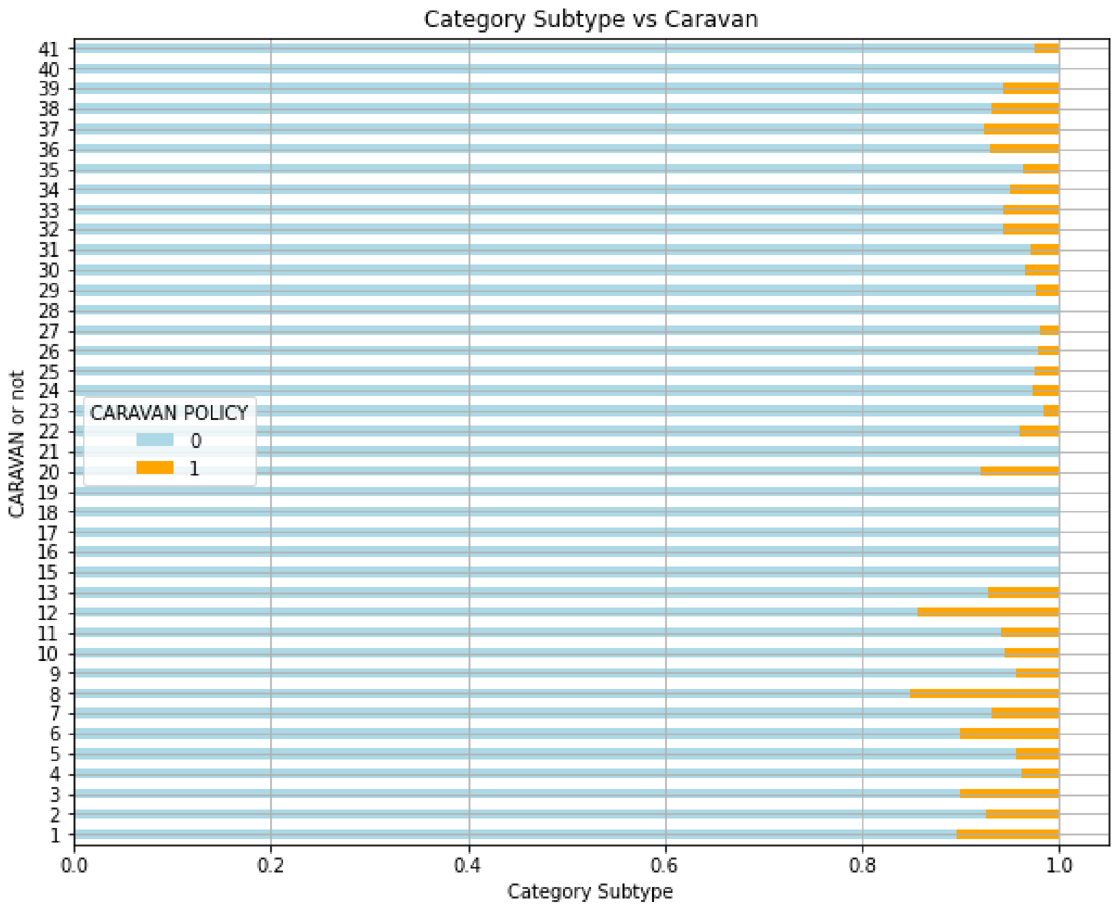
[6]:
```
train = pd.read_csv('caravan.csv')
```

```python
# Plot Telling the total count of different values in CARAVAN
plt.subplot(3,1,1);
train['CARAVAN POLICY'].value_counts().plot(kind='bar', title='Classifying CARAVAN', color='orange', grid=True);

# Plot Telling the total count of different values in customer subtype
plt.subplot(3,1, 2);
train['Customer Subtype'].value_counts().plot(kind='bar', align='center', title='Classifying Customer Subtypes', color='orange', grid=True);
```



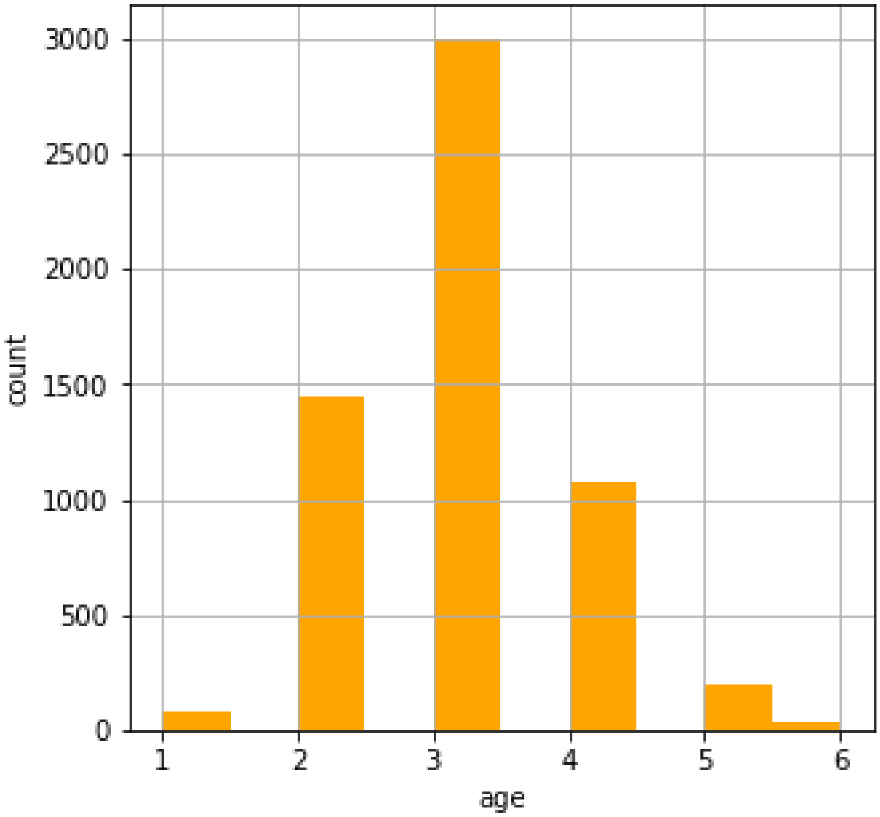
[7]:
```python
categorysubtype_caravan = pd.crosstab(train['Customer Subtype'], train['CARAVAN POLICY']);
categorysubtype_caravan_percentage = categorysubtype_caravan.div(categorysubtype_caravan.sum(1).astype(float), axis=0);
categorysubtype_caravan_percentage.plot(figsize= (10,8), kind='barh', stacked=True, color=['lightblue', 'orange'], title='Category Subtype vs Caravan', grid=True);
plt.xlabel('Category Subtype');
plt.ylabel('CARAVAN or not');
```



Form the appendinx of Lab 3 sheet we can distinguish teh correspondences of the encoding of variable "Customer subtyppe". The graph above clearly shows how: - Senior cosmopolitans, - Students in apartments, - Fresh masters in the city, - Single youth, - Suburban youth, - Large family farms don't have a single Caravan Policy. On the other hand,
- Middle class families, - Affluent young families have most of the Caravan Policies

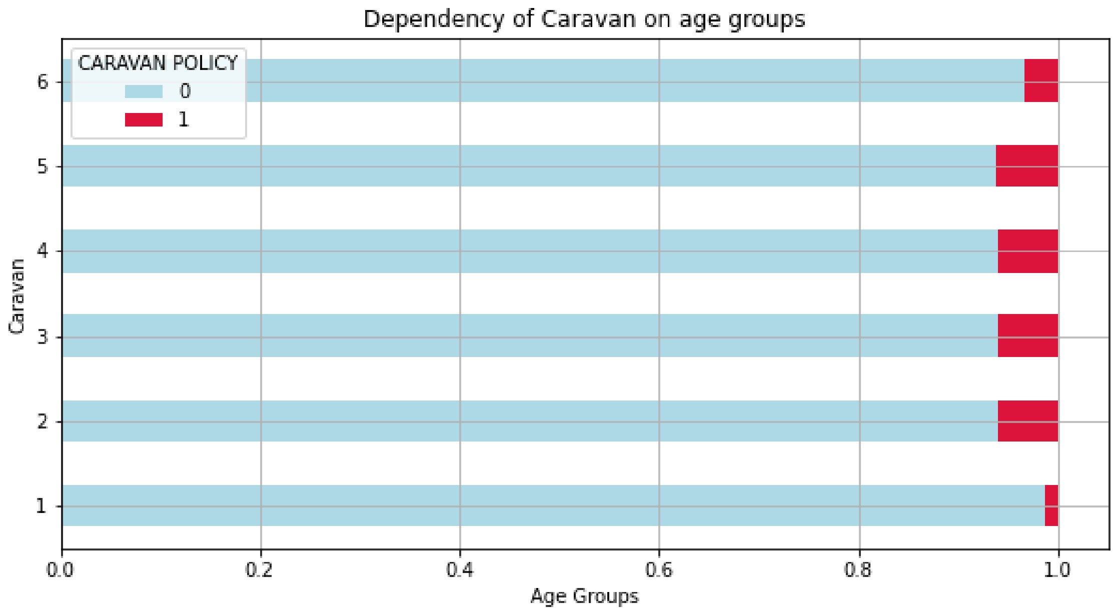
1.1.1 Visualization: Plotting the dependency of preferring caravan policy based on age

[8]:
```
train['Avg Age'].hist(figsize=(5,5), fc='orange', grid=True);
plt.xlabel('age');
plt.ylabel('count');
```



Average age is encoded according to L1. The above barplot shows how the age group 3 corresponding to the 40-50 Yrs group bought most policies. On the other hand, group 1 and 6 corresponding to 20-30 70-80 Age groups respectively, didn't buy any Policies

[9]:
```
age_caravan = pd.crosstab(train['Avg Age'], train['CARAVAN POLICY']);
age_caravan_percentage = age_caravan.div(age_caravan.sum(1).
 ↪astype(float),axis=0);
age_caravan_percentage.plot(figsize=(10,5), kind='barh', stacked=True,␣
 ↪color=['lightblue', 'Crimson'], title='Dependency of Caravan on age groups',␣
 ↪grid=True);
plt.xlabel('Age Groups');
plt.ylabel('Caravan');
```

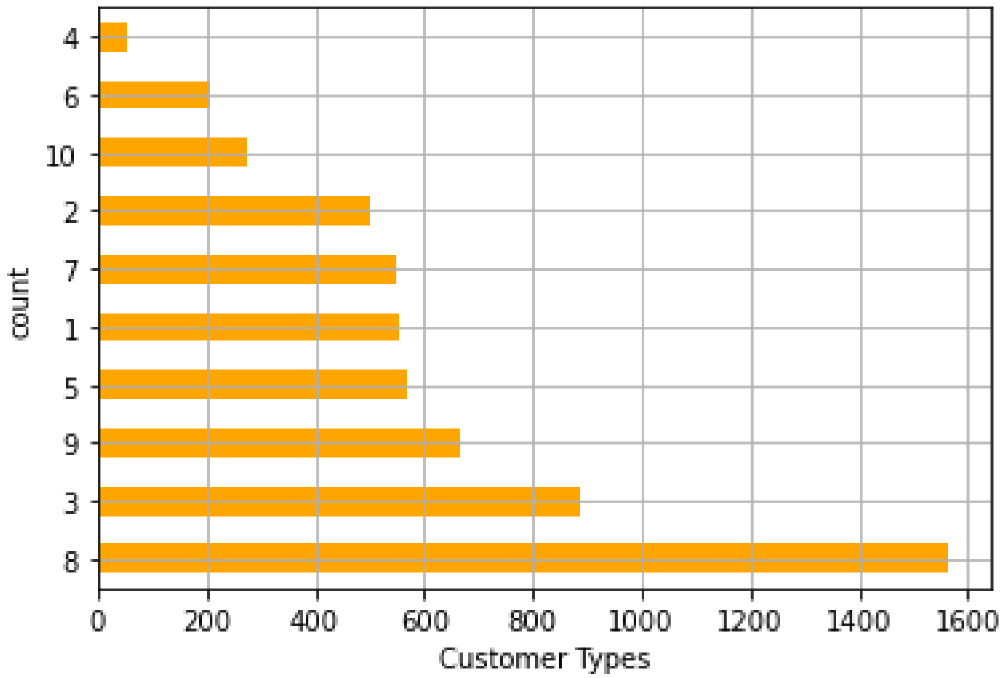


Here we verify that almost no one in age group 1(20-30yrs) has a caravan policy.

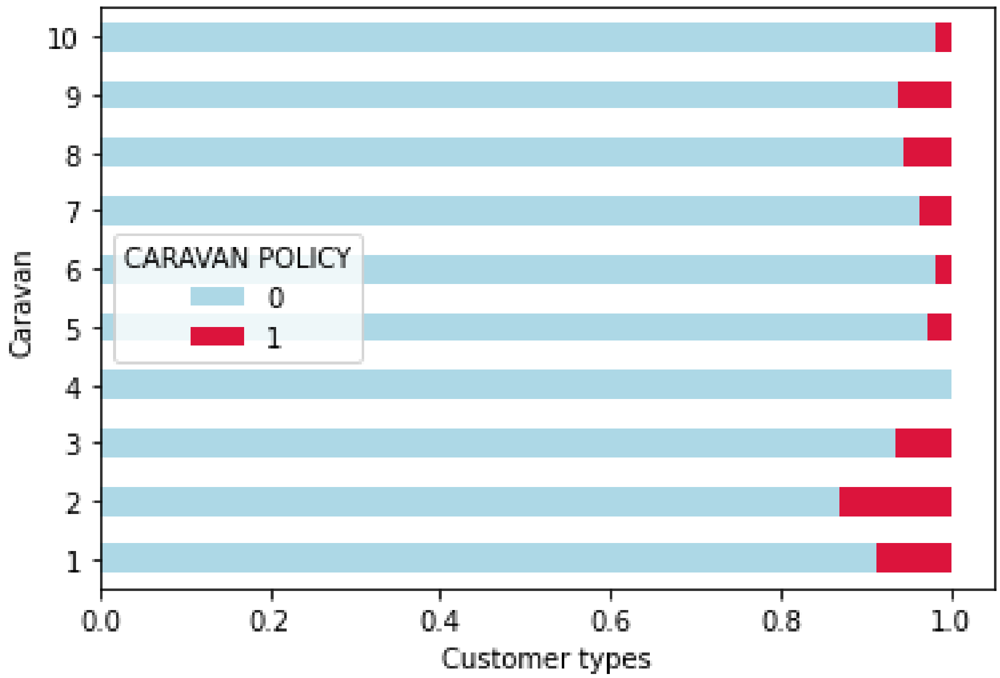
Thus Age and Subtype can be important features for correct classification.

1.1.2 Visualization: Dependency of Caravan policy based on Customer type

[10]:
```
train['Customer main type'].value_counts().plot(kind='barh', color='orange',␣
 ↪grid=True);
plt.xlabel('Customer Types');
plt.ylabel('count');
```



[11]:
```
cust_type_caravan = pd.crosstab(train['Customer main type'], train['CARAVAN␣
 ↪POLICY']);
cust_type_caravan_percentage = cust_type_caravan.div(cust_type_caravan.sum(1).
 ↪astype(float), axis=0);
cust_type_caravan_percentage.plot(kind='barh', stacked=True, color =␣
 ↪['lightblue', 'Crimson']);
plt.xlabel('Customer types');
plt.ylabel('Caravan');
```



The Customer main type variable is encoded according to the L2 domain. From the above plot we can see that tyoes such as 'Family with grown ups' and 'Driven Growers' are the categories containing the most caravan insurance owners.

[ ]:

```

```

[12]:

```
train = pd.read_csv('caravan.csv')

X_train = train.iloc[:,0:-1]
y_train = train.iloc[:,-1]
train.head()
```

[12]:

```
   Customer Subtype  Number of houses  Avg size household  Avg Age  \
0                33                 1                   3        2
1                37                 1                   2        2
2                37                 1                   2        2
3                 9                 1                   3        3
4                40                 1                   4        2

   Customer main type  Roman catholic  Protestant  Other religion  \
0                   8               0           5               1
1                   8               1           4               1
2                   8               0           4               2
3                   3               2           3               2
4                  10               1           4               1
```

```
   No religion  Married  ...  Number of private accident insurance policies  \
0            3        7  ...                                              0
1            4        6  ...                                              0
2            4        3  ...                                              0
3            4        5  ...                                              0
4            4        7  ...                                              0

   Number of family accidents insurance policies  \
0                                              0
1                                              0
2                                              0
3                                              0
4                                              0

   Number of disability insurance policies  Number of re policies  \
0                                        0                      1
1                                        0                      1
2                                        0                      1
3                                        0                      1
4                                        0                      1

   Number of surfboard policies  Number of boat policies  \
0                             0                        0
1                             0                        0
2                             0                        0
3                             0                        0
4                             0                        0

   Number of bicycle policies  Number of property insurance policies  \
0                           0                                      0
1                           0                                      0
2                           0                                      0
3                           0                                      0
4                           0                                      0

   Number of social security insurance policies  CARAVAN POLICY
0                                             0               0
1                                             0               0
2                                             0               0
3                                             0               0
4                                             0               0

[5 rows x 86 columns]
```

1.1.3 Feature Selection using Random Forest

[13]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.25)
```

[14]:
```
clf= RandomForestClassifier(oob_score = True,bootstrap=True,n_estimators=100)
forest_fit=clf.fit(X_train, y_train);
forest_fit;
predict = clf.predict(X_test)
accuracy = accuracy_score(y_test, predict)
print('Accuracy is :'+"{:.2f}".format(accuracy*100),"%")
```

```
Accuracy is :93.68 %
```

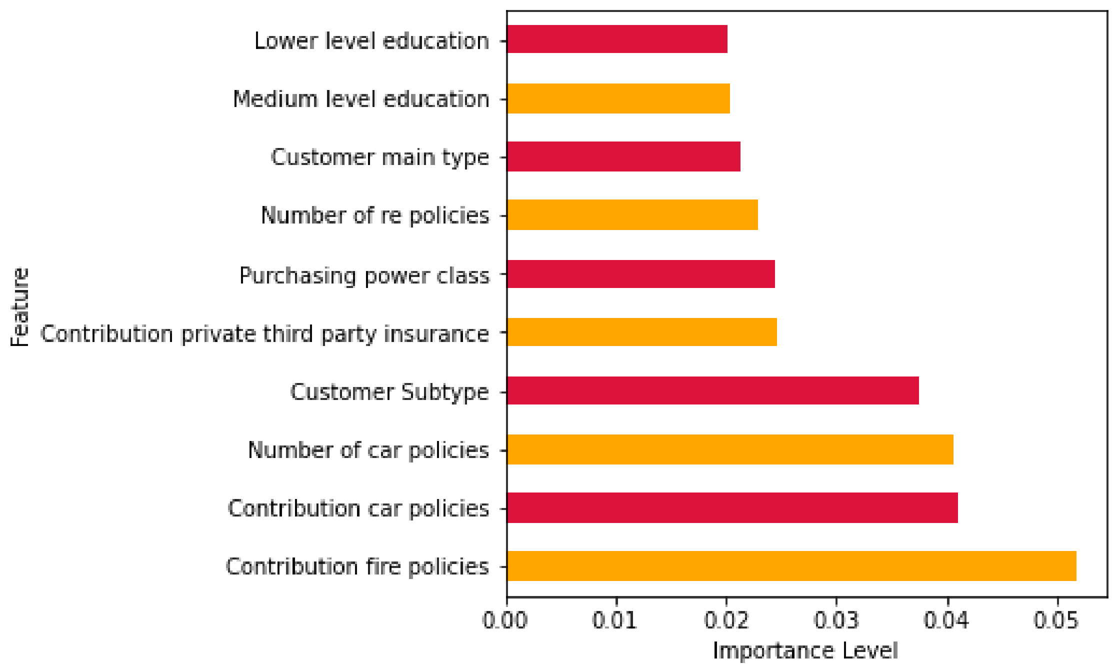
[15]:
```
feature_list=X.columns
importances = list(clf.feature_importances_)
feature_importance=pd.DataFrame({'Feature':feature_list, 'Importance':
 ↪importances})
feature_importance.sort_values(by ='Importance',ascending=False)
```

[15]:

	Feature	Importance
58	Contribution fire policies	5.185444e-02
46	Contribution car policies	4.100131e-02
67	Number of car policies	4.056470e-02
0	Customer Subtype	3.746879e-02
43	Contribution private third party insurance	2.468719e-02
..	...	...
59	Contribution surfboard policies	6.403492e-06
49	Contribution lorry policies	1.647410e-06
70	Number of lorry policies	6.866169e-07
73	Number of agricultural machines policies	0.000000e+00
52	Contribution agricultural machines policies	0.000000e+00

[85 rows x 2 columns]

[16]:
```
plt.figure(figsize=(5,5))
feat_importances = pd.Series(clf.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh', stacked=True, color = ['orange',␣
 ↪'crimson']);
plt.xlabel('Importance Level');
plt.ylabel('Feature');
```



1.1.4 Feature selection using SelectKbest and RFE

Almost all "simple" classifiers as well as ensambles of classifiers studied in the course were trained and tested with the exeption of kNN as it is known how it doesn't perfomr well on high dimensional data or in large datasets.

Decision trees shall be excluded for the same reason as kNN but it will still be trained and tested in order to compare it with random forest. For now the training dataset will be used both fro training and testing performin a 10-fold cross validation.

[17]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import␣
 ↪confusion_matrix,accuracy_score,precision_recall_curve, auc, roc_auc_score,␣
 ↪roc_curve, recall_score, classification_report
from sklearn.model_selection import cross_val_predict

dt = DecisionTreeClassifier()
gaus = GaussianNB()
log = LogisticRegression(solver='liblinear')
svc = SVC(gamma='scale', kernel='linear')
# ensembles:
rf = RandomForestClassifier(n_estimators=1000, criterion='gini', max_depth=40,
                                min_samples_split=2, min_samples_leaf=1,␣
 ↪min_weight_fraction_leaf=0.0,
```

```
                                max_features='auto', bootstrap=True, n_jobs=1,
                                random_state=42, warm_start=False)
adaB = AdaBoostClassifier(n_estimators=100)

def build_confuision_matrix(classifier, data, clas):
    y_pred = cross_val_predict(classifier, data, clas, cv = 10) #10-fold␣
 ↪cross-validation
    conf_mat = confusion_matrix(clas, y_pred)
    return conf_mat

def generic_classifier(classifier, data):
    clas = train.iloc[:,-1]
    data =  train.iloc[:,0:-1]
    cv_scores = cross_val_score(classifier, data, clas, cv=10)
    conf = build_confuision_matrix(classifier, data, clas)
    acc = round(np.mean(cv_scores)*100,2)
    return acc, conf


acc, conf = generic_classifier(dt, train)
print(' Decision Trees with accuracy:{} %'.format(acc))
print(conf)
print('')


acc, conf = generic_classifier(gaus, train)
print(' Naive Bayes with accuracy:{} %'.format(acc))
print(conf)
print('')

acc, conf = generic_classifier(log, train)
print(' Logistic Regression with accuracy:{} %'.format(acc))
print(conf)
print('')

acc, conf = generic_classifier(svc, train)
print(' Support Vector Machines with accuracy:{} %'.format(acc))
print(conf)

acc, conf = generic_classifier(rf, train)
print(' Random Forest with accuracy:{} %'.format(acc))
print(conf)

acc, conf = generic_classifier(adaB, train)
print(' AdaBoost with accuracy:{} %'.format(acc))
print(conf)
```

```
 Decision Trees with accuracy:89.28 %
[[5137  337]
 [ 302   46]]

 Naive Bayes with accuracy:18.28 %
[[ 733 4741]
 [  17  331]]

 Logistic Regression with accuracy:93.83 %
[[5461   13]
 [ 346    2]]

 Support Vector Machines with accuracy:94.02 %
[[5474    0]
 [ 348    0]]
 Random Forest with accuracy:92.73 %
[[5381   93]
 [ 330   18]]
 AdaBoost with accuracy:93.52 %
[[5436   38]
 [ 339    9]]
```

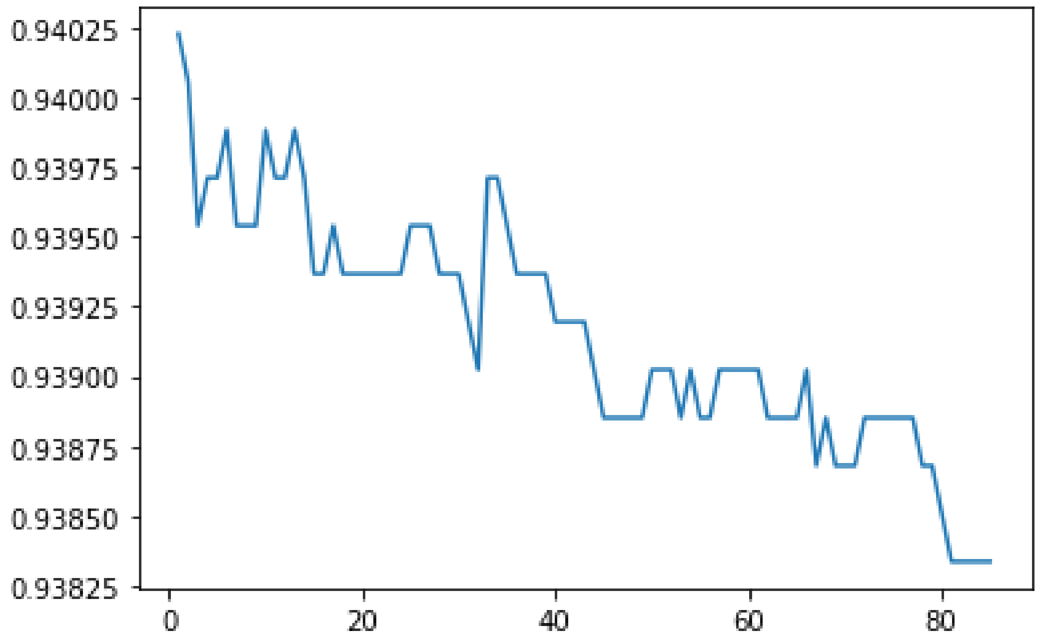
Logistic Regression, SVM, Random Forest and boosting (AdaBoost) were the models performing better. For sake of computational simplicity Logistic regression will be used for the remainder of this assignment.

Feature selection using K best

[18]:
```
classifier = log
feature_name_dict = {}
accuracy_list = []
for i in range(1, len(train.columns)):
    selector = SelectKBest(chi2, k=i)
    selector.fit(X, y)
    feature_list = selector.get_support(indices=True)
    pipeClassifier = make_pipeline(selector, classifier)
    pipeClassifier.fit(X,y)
    feature_name_dict[i] = train.columns[selector.get_support(indices=True)]
    a = cross_val_score(pipeClassifier, X, y, cv=10)
    accuracy_list.append(a.mean())
```

[19]:
```
#Accuracy plot for k
a_x = np.arange(1,86,1)
plt.plot(a_x, accuracy_list)
```

[19]:
```
[<matplotlib.lines.Line2D at 0x1a39cde0d00>]
```



[20]:
```
max(accuracy_list)
sorted(range(len(accuracy_list)), key=lambda k: accuracy_list[k])[-5:]
```

[20]:
```
[5, 9, 12, 1, 0]
```

[21]:
```
max(accuracy_list)
```

[21]:
```
0.9402268159124801
```

[22]:
```
feature_name_dict[12]
```

[22]:
```
Index(['Customer Subtype', 'High level education', 'Rented house',
       'Home owners', 'Income < 30.000',
       'Contribution private third party insurance',
       'Contribution car policies', 'Contribution fire policies',
       'Contribution boat policies',
       'Contribution social security insurance policies',
       'Number of car policies', 'Number of boat policies'],
      dtype='object')
```

As the sorted cross validation accuracies indicate, the pipeline shows the best performance for selecting 12 variables for classification.

Recursive Feature Elimination

RFE feature selection was also tryed out. However, it was computationally hard to iterate through all the possible number of variables to extract.

RFE using 12 selection variables:

[23]:
```
classifier = LogisticRegression(solver = "liblinear")
selector = RFE(classifier, 12, 1)
selector = selector.fit(X, y)
feature_name_dict_RFE = train.columns[np.nonzero(selector.support_)]
a = cross_val_score(selector, X, y, cv=10)
a.mean()
```

[23]:
```
0.9402271106317011
```

2 Assignment 2

2.0.1 Via Logistic Regression

[25]:
```
X_test = test.iloc[:,0:-1]
y_test = test.iloc[:,-1]
```

[26]:
```
selector = SelectKBest(k=35)
selector.fit(X, y)
pipeClassifier = make_pipeline(selector, classifier)
pipeClassifier.fit(X,y)
a = cross_val_score(pipeClassifier, X_test, y_test, cv=10)
prediction = pipeClassifier.predict(X_test)
a.mean()
```

[26]:
```
0.9397499999999999
```

[27]:
```
np.nonzero(prediction)
```

[27]:
```
(array([ 291,  575, 1995, 2621, 2862, 3138, 3499], dtype=int64),)
```

The training model was unable to find 800 customers from the test data.

[ ]:
```
```

[ ]:
```
```