Auto credit card approval - case study description

This case study focuses on building an automatic credit card approval predictor using machine learning. The Credit Approval Data Set from the [UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets/credit+approval)](http://archive.ics.uci.edu/ml/datasets/credit+approval)is used as an example dataset to demonstrate the methodology. Although the features labels are sanitized to maintain anonymity, expert opinions suggest that the feature labels may be: `Gender`, `Age`, `Debt`, `Married_status`, `BankCustomer`, `EducationLevel`, `Ethnicity`, `YearsEmployed`, `NoPriorDefault`, `Employed`, `CreditScore`, `DriversLicense`, `Citizen`, `ZipCode`, `Income` and `ApprovalStatus`.

Credit card approval is a perfect case study for applied machine learning since the application approval process can be easily framed as a classification problem. The underlying pattern that differentiates between trustworthy customers and unreliable customers can be ascertained through the customer's credit and personal details. The conventional system for approvals were subjective and based on the bank manager's experience. Using machine learning, this subjective judgement can be supplemented with quantitave metrics that can lead to faster and more accurate approval processes.

This analysis will involve data pre-processing and cleaning followed by an exploratory analysis. Pre-processing is reqired to deal with the missing values and prepping the dataset for use in machine learning libraries. After some exploratory analysis, we'll build a pipeline that will test several machine learning models and their predictors for te credit card applications.

Sources:

Data: Credit Approval Data Set, UCI Machine Learning Repository

Project: Sayak Paul, Predicting Credit Card Approvals, Datacamp

In [387]:

```
#from IPython.core.debugger import set_trace
#Original dataset
import pandas as pd
import numpy as np

df = pd.read_csv('../dat/cc_approvals.data',header=None,na_values='?')
cc_col_details=['Gender', 'Age', 'Debt', 'Married_status', 'BankCustomer
', 'EducationLevel', 'Ethnicity', 'YearsEmployed', 'NoPriorDefault', 'Emp
loyed', 'CreditScore', 'DriversLicense', 'Citizen', 'ZipCode', 'Income' ,
'ApprovalStatus']
df.columns = cc_col_details
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
Gender            678 non-null object
Age               678 non-null float64
Debt              690 non-null float64
Married_status    684 non-null object
BankCustomer      684 non-null object
EducationLevel    681 non-null object
Ethnicity         681 non-null object
YearsEmployed     690 non-null float64
NoPriorDefault    690 non-null object
Employed          690 non-null object
CreditScore       690 non-null int64
DriversLicense    690 non-null object
Citizen           690 non-null object
ZipCode           677 non-null float64
Income            690 non-null int64
ApprovalStatus    690 non-null object
dtypes: float64(4), int64(2), object(10)
memory usage: 86.4+ KB
```

Out[387]:

	Gender	Age	Debt	Married_status	BankCustomer	EducationLevel	Ethnicity	YearsEmployed	NoPriorDefaul
0	b	30.83	0.000	u	g	w	v	1.25	
1	a	58.67	4.460	u	g	q	h	3.04	
2	a	24.50	0.500	u	g	q	h	1.50	
3	b	27.83	1.540	u	g	w	v	3.75	
4	b	20.17	5.625	u	g	w	v	1.71	

Data cleaning and pre-processing

As can be seen, the dataset requires some cleaning before it can be used for any exploratory analysis.

In [388]:
```
def convert_cat_cols(df,cat_var_limit=10,verbose=False):
    """
    Converts columns with a small amount of unique values that are of
    type Object into categorical variables
    """
    temp_var = df.apply(lambda x: len(x.value_counts())<cat_var_limit)
    temp_var2 = df.apply(lambda x: x.value_counts().index.dtype == 'O')
    df[temp_var[temp_var2].index] = df[temp_var[temp_var2].index].astype
('category')
    if verbose:
        print(df[temp_var[temp_var2].index].describe())
    return df

def impute_most_freq(df):
    """
    Imputes the most frequent value in place of NaN's
    """
    temp_var = df.apply(lambda x: x.value_counts().index[0])
    return df.fillna(temp_var)


df = pd.read_csv('../dat/cc_approvals.data',header=None,na_values='?')
df.columns = cc_col_details
df = convert_cat_cols(df,10).fillna(df.median())
df = impute_most_freq(df)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
Gender            690 non-null category
Age               690 non-null float64
Debt              690 non-null float64
Married_status    690 non-null category
BankCustomer      690 non-null category
EducationLevel    690 non-null category
Ethnicity         690 non-null category
YearsEmployed     690 non-null float64
NoPriorDefault    690 non-null category
Employed          690 non-null category
CreditScore       690 non-null int64
DriversLicense    690 non-null category
Citizen           690 non-null category
ZipCode           690 non-null float64
Income            690 non-null int64
ApprovalStatus    690 non-null category
dtypes: category(10), float64(4), int64(2)
memory usage: 41.1 KB
```

All data cleaning is done. Next step is to perform preprocessing of the dataset for insertion into the SciKit library functions which require numeric values. We'll convert our dataset into a binary integer representation using pd.get_dummies as well as a 0 - n_class-1 integer representation using the Scikit Transformer LabelEncoder

In [389]:
```
from sklearn.preprocessing import LabelEncoder

def convert_str_int_labels(df):
    """
    Converts columns with strings (class labels) into integer representat
ion
    """
    for col in df.columns:
        le = LabelEncoder()
        df.loc[:,col]=le.fit_transform(df.loc[:,col])
    return df

df_bin = pd.get_dummies(df,drop_first=True)
df.loc[:,df.columns[df.dtypes=='category']]=\
convert_str_int_labels(df.loc[:,df.columns[df.dtypes=='category']])

print(df_bin.head())
df.head()


#for i in df.columns:
#    print(df[i].value_counts(dropna=False))
#    print('\n')
#df.head()
```

```
     Age   Debt  YearsEmployed  CreditScore  ZipCode  Income  Gender_b  \
0  30.83  0.000           1.25            1    202.0       0         1
1  58.67  4.460           3.04            6     43.0     560         0
2  24.50  0.500           1.50            0    280.0     824         0
3  27.83  1.540           3.75            5    100.0       3         1
4  20.17  5.625           1.71            0    120.0       0         1

   Married_status_u  Married_status_y  BankCustomer_gg  ...  Ethnicity_n  \
0                 1                 0                0  ...            0
1                 1                 0                0  ...            0
2                 1                 0                0  ...            0
3                 1                 0                0  ...            0
4                 1                 0                0  ...            0

   Ethnicity_o  Ethnicity_v  Ethnicity_z  NoPriorDefault_t  Employed_t  \
0            0            1            0                 1           1
1            0            0            0                 1           1
2            0            0            0                 1           0
3            0            1            0                 1           1
4            0            1            0                 1           0

   DriversLicense_t  Citizen_p  Citizen_s  ApprovalStatus_-
0                 0          0          0                 0
1                 0          0          0                 0
2                 0          0          0                 0
3                 1          0          0                 0
4                 0          0          1                 0

[5 rows x 38 columns]
```

Out[389]:

	Gender	Age	Debt	Married_status	BankCustomer	EducationLevel	Ethnicity	YearsEmployed	NoPriorDefaul
0	1	30.83	0.000	1	0	12	7	1.25	
1	0	58.67	4.460	1	0	10	3	3.04	
2	0	24.50	0.500	1	0	10	3	1.50	
3	1	27.83	1.540	1	0	12	7	3.75	
4	1	20.17	5.625	1	0	12	7	1.71	

Splitting into train and test sets with auto feature selection

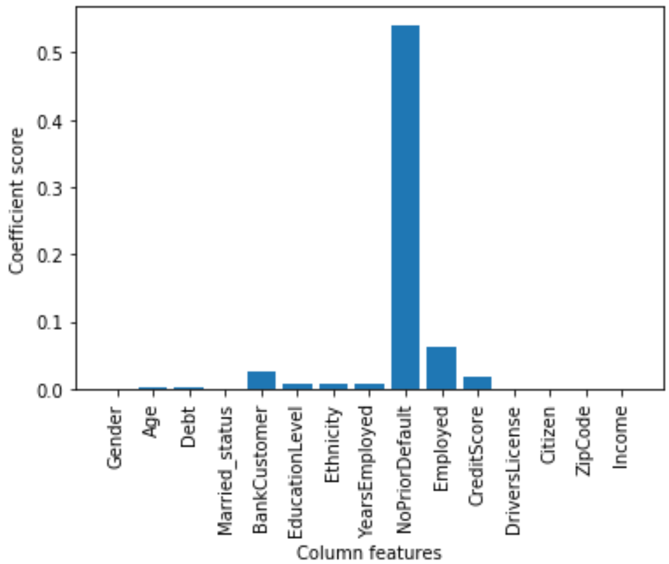
With all numeric data, the next data-prep step will be to split the data into a training set and testing set. Ideally, no information from the test data should be used to scale the training data or should be used to direct the training process of a machine learning model. Hence, we first split the data and then apply the scaling.

In [390]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
import matplotlib.pyplot as plt
from IPython.core.debugger import set_trace

def feat_select(df,test_size_var=0.3,alpha_val=0.08,random_state_var=21,u
se_feat_select=True,plot=True):
    """
    Performs feature selection on a dataframe with a single target variab
le and n features.
    Test train split is also performed and only splits of selected featur
es are returned.
    Feature selection performed using LASSO weight shrinking
    """
    x_train, x_test, y_train, y_test = train_test_split(df.iloc[:,:-1], d
f.iloc[:,-1]\
                                                        ,test_size=test_s
ize_var, random_state=random_state_var\
                                                        ,stratify=df.iloc
[:,-1])
    if use_feat_select:
        param_grid = {'alpha': np.linspace(0.01,0.02,20)}
        lasso_gcv = GridSearchCV(Lasso(normalize=False),param_grid,cv=5,n
_jobs=-1,iid=True)
        lasso_coeffs = lasso_gcv.fit(x_train, y_train).best_estimator_.co
ef_
        if plot:
            plt.bar(x=range(len(df.columns[:-1])),height=np.abs(lasso_coe
ffs)\
                    ,tick_label=df.columns[:-1].values,)
            plt.xlabel('Column features')
            plt.ylabel('Coefficient score')
            plt.xticks(rotation=90)
            plt.show()
        select_feats = df.columns[:-1][np.abs(lasso_coeffs) > 0].values
        #set_trace()
        x_train = x_train.loc[:,select_feats]
        x_test = x_test.loc[:,select_feats]
    return x_train.values, x_test.values, y_train.values, y_test.values


#x_train, x_test, y_train, y_test = feat_select(df,use_feat_select=True,p
lot=True)
```

Creating a transformation and analysis pipeline

The dataset can now be rescaled so that no feature can artificially bias the analysis. In this case, no specialized feature engineering is performed and all the feature variables will be rescaled broadly.
Both the binary integer representation and the non-binary integer representation will be tested.
Note 1: better scores can be expected from intelligently rescaling the data. For example, age can be standardized while credit scores can be rescaled between 0 and -1. This will be implemented in the future.
Note 2: The above preprocessing functions can be integrated into the pipeline ibject using FunctionTransformer. This will be implemented in the future.

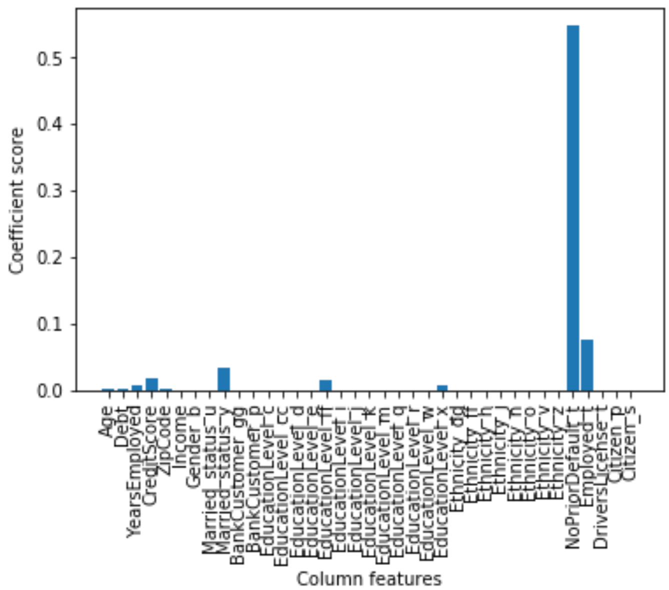
In [397]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC

x_train, x_test, y_train, y_test = feat_select(df,use_feat_select=True,pl
ot=True)

steps = ('Scaler', MinMaxScaler(feature_range=(-1, 1)))
steps_norm = ('Scaler', Normalizer())
steps_stand = ('Scaler', StandardScaler())


classifiers = [('logreg',LogisticRegression(solver='lbfgs')),\
        ('knnstep',KNeighborsClassifier()),\
        ('svcstep',SVC(gamma='scale')),\
         ('gradbooststep',GradientBoostingClassifier(subsample=.8))      ]

parameters = { 'logreg':{'logreg__C' : [0.8,1,1.2,1.4]} ,\
              'knnstep':{'knnstep__n_neighbors':np.arange(3,16)},\
              'svcstep':{'svcstep__C': [0.5,1,1.5,2,2.5,2.6]},\
             'gradbooststep':{'gradbooststep__max_depth': [2,3,4,5],'grad
booststep__n_estimators': [40,60,80,100]}}

trained_models = []

for clf in classifiers:
    pipeline = Pipeline([steps,clf])
    print('\nAnalysis for : '+clf[0])
    gcv = GridSearchCV(pipeline,param_grid = parameters[clf[0]],cv=5,iid=
True)
    gcv.fit(x_train,y_train)
    trained_models.append(gcv)
    print(gcv.best_params_)
    print(pd.DataFrame(gcv.cv_results_)[['mean_test_score','params']])
    print('The score for '+ clf[0] +' is '+ str(gcv.score(x_test,y_tes
t)))

#trained_models[0].predict(x_test)   #testing using the logreg model
```


Coefficient score
0.5
0.4
0.3
0.2
0.1
0.0
Gender
Age
Debt
Married_status
BankCustomer
EducationLevel
Ethnicity
YearsEmployed
NoPriorDefault
Employed
CreditScore
DriversLicense
Citizen
ZipCode
Income
Column features

```
Analysis for : logreg
{'logreg__C': 0.8}
   mean_test_score              params
0         0.859213  {'logreg__C': 0.8}
1         0.859213    {'logreg__C': 1}
2         0.857143  {'logreg__C': 1.2}
3         0.857143  {'logreg__C': 1.4}
The score for logreg is 0.8743961352657005

Analysis for : knnstep
{'knnstep__n_neighbors': 8}
    mean_test_score                        params
0          0.842650   {'knnstep__n_neighbors': 3}
1          0.836439   {'knnstep__n_neighbors': 4}
2          0.846791   {'knnstep__n_neighbors': 5}
3          0.853002   {'knnstep__n_neighbors': 6}
4          0.855072   {'knnstep__n_neighbors': 7}
5          0.857143   {'knnstep__n_neighbors': 8}
6          0.857143   {'knnstep__n_neighbors': 9}
7          0.846791  {'knnstep__n_neighbors': 10}
8          0.850932  {'knnstep__n_neighbors': 11}
9          0.844720  {'knnstep__n_neighbors': 12}
10         0.846791  {'knnstep__n_neighbors': 13}
11         0.850932  {'knnstep__n_neighbors': 14}
12         0.853002  {'knnstep__n_neighbors': 15}
The score for knnstep is 0.8888888888888888

Analysis for : svcstep
{'svcstep__C': 0.5}
   mean_test_score               params
0         0.859213  {'svcstep__C': 0.5}
1         0.857143    {'svcstep__C': 1}
2         0.853002  {'svcstep__C': 1.5}
3         0.857143    {'svcstep__C': 2}
4         0.857143  {'svcstep__C': 2.5}
5         0.857143  {'svcstep__C': 2.6}
The score for svcstep is 0.8888888888888888

Analysis for : gradboostingstep
{'gradboosstep__max_depth': 2, 'gradboosstep__n_estimators': 60}
    mean_test_score                                             params
0          0.867495  {'gradbooststep__max_depth': 2, 'gradbooststep...
1          0.884058  {'gradbooststep__max_depth': 2, 'gradbooststep...
2          0.873706  {'gradbooststep__max_depth': 2, 'gradbooststep...
3          0.873706  {'gradbooststep__max_depth': 2, 'gradbooststep...
4          0.875776  {'gradbooststep__max_depth': 3, 'gradbooststep...
5          0.873706  {'gradbooststep__max_depth': 3, 'gradbooststep...
6          0.875776  {'gradbooststep__max_depth': 3, 'gradbooststep...
7          0.869565  {'gradbooststep__max_depth': 3, 'gradbooststep...
8          0.877847  {'gradbooststep__max_depth': 4, 'gradbooststep...
9          0.877847  {'gradbooststep__max_depth': 4, 'gradbooststep...
10         0.861284  {'gradbooststep__max_depth': 4, 'gradbooststep...
11         0.863354  {'gradbooststep__max_depth': 4, 'gradbooststep...
12         0.873706  {'gradbooststep__max_depth': 5, 'gradbooststep...
13         0.869565  {'gradbooststep__max_depth': 5, 'gradbooststep...
14         0.873706  {'gradbooststep__max_depth': 5, 'gradbooststep...
15         0.867495  {'gradbooststep__max_depth': 5, 'gradbooststep...
The score for gradbooststep is 0.8792270531400966
```

In [396]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC

x_train, x_test, y_train, y_test = feat_select(df_bin,use_feat_select=Tru
e,plot=True)

steps = ('Scaler', MinMaxScaler(feature_range=(-1, 1)))
steps_norm = ('Scaler', Normalizer())
steps_stand = ('Scaler', StandardScaler())


classifiers = [('logreg',LogisticRegression(solver='lbfgs')),\
        ('knnstep',KNeighborsClassifier()),\
        ('svcstep',SVC(gamma='scale')),\
         ('gradbooststep',GradientBoostingClassifier(subsample=.8))      ]

parameters = { 'logreg':{'logreg__C' : [0.8,1,1.2,1.4]} ,\
              'knnstep':{'knnstep__n_neighbors':np.arange(3,16)},\
              'svcstep':{'svcstep__C': [0.5,1,1.5,2,2.5,2.6]},\
             'gradbooststep':{'gradbooststep__max_depth': [2,3,4,5],'grad
booststep__n_estimators': [40,60,80,100]}}

trained_models = []

for clf in classifiers:
    pipeline = Pipeline([steps,clf])
    print('\nAnalysis for : '+clf[0])
    gcv = GridSearchCV(pipeline,param_grid = parameters[clf[0]],cv=5,iid=
True)
    gcv.fit(x_train,y_train)
    trained_models.append(gcv)
    print(gcv.best_params_)
    print(pd.DataFrame(gcv.cv_results_)[['mean_test_score','params']])
    print('The score for '+ clf[0] +' is '+ str(gcv.score(x_test,y_tes
t)))

#trained_models[0].predict(x_test)   #testing using the logreg model
```


Coefficient score
0.0
0.1
0.2
0.3
0.4
0.5
Age
Debt
YearsEmployed
CreditScore
ZipCode
Income
Gender_b
Married_status_u
Married_status_y
BankCustomer_gg
BankCustomer_p
EducationLevel_c
EducationLevel_cc
EducationLevel_d
EducationLevel_e
EducationLevel_ff
EducationLevel_i
EducationLevel_j
EducationLevel_k
EducationLevel_m
EducationLevel_q
EducationLevel_r
EducationLevel_w
EducationLevel_x
Ethnicity_dd
Ethnicity_ff
Ethnicity_h
Ethnicity_j
Ethnicity_n
Ethnicity_o
Ethnicity_v
Ethnicity_z
NoPriorDefault_t
Employed_t
DriversLicense_t
Citizen_p
Citizen_s
Column features

```
Analysis for : logreg
{'logreg__C': 1.4}
   mean_test_score              params
0         0.857143  {'logreg__C': 0.8}
1         0.859213    {'logreg__C': 1}
2         0.859213  {'logreg__C': 1.2}
3         0.861284  {'logreg__C': 1.4}
The score for logreg is 0.8888888888888888

Analysis for : knnstep
{'knnstep__n_neighbors': 9}
    mean_test_score                        params
0          0.840580   {'knnstep__n_neighbors': 3}
1          0.832298   {'knnstep__n_neighbors': 4}
2          0.846791   {'knnstep__n_neighbors': 5}
3          0.838509   {'knnstep__n_neighbors': 6}
4          0.848861   {'knnstep__n_neighbors': 7}
5          0.840580   {'knnstep__n_neighbors': 8}
6          0.859213   {'knnstep__n_neighbors': 9}
7          0.848861  {'knnstep__n_neighbors': 10}
8          0.850932  {'knnstep__n_neighbors': 11}
9          0.844720  {'knnstep__n_neighbors': 12}
10         0.855072  {'knnstep__n_neighbors': 13}
11         0.850932  {'knnstep__n_neighbors': 14}
12         0.850932  {'knnstep__n_neighbors': 15}
The score for knnstep is 0.8454106280193237

Analysis for : svcstep
{'svcstep__C': 2.5}
   mean_test_score               params
0         0.846791  {'svcstep__C': 0.5}
1         0.853002    {'svcstep__C': 1}
2         0.855072  {'svcstep__C': 1.5}
3         0.855072    {'svcstep__C': 2}
4         0.859213  {'svcstep__C': 2.5}
5         0.855072  {'svcstep__C': 2.6}
The score for svcstep is 0.8647342995169082

Analysis for : gradbooststep
{'gradbooststep__max_depth': 2, 'gradbooststep__n_estimators': 80}
    mean_test_score                                             params
0          0.867495  {'gradbooststep__max_depth': 2, 'gradbooststep...
1          0.875776  {'gradbooststep__max_depth': 2, 'gradbooststep...
2          0.892340  {'gradbooststep__max_depth': 2, 'gradbooststep...
3          0.886128  {'gradbooststep__max_depth': 2, 'gradbooststep...
4          0.867495  {'gradbooststep__max_depth': 3, 'gradbooststep...
5          0.877847  {'gradbooststep__max_depth': 3, 'gradbooststep...
6          0.890269  {'gradbooststep__max_depth': 3, 'gradbooststep...
7          0.873706  {'gradbooststep__max_depth': 3, 'gradbooststep...
8          0.877847  {'gradbooststep__max_depth': 4, 'gradbooststep...
9          0.877847  {'gradbooststep__max_depth': 4, 'gradbooststep...
10         0.879917  {'gradbooststep__max_depth': 4, 'gradbooststep...
11         0.873706  {'gradbooststep__max_depth': 4, 'gradbooststep...
12         0.871636  {'gradbooststep__max_depth': 5, 'gradbooststep...
13         0.875776  {'gradbooststep__max_depth': 5, 'gradbooststep...
14         0.871636  {'gradbooststep__max_depth': 5, 'gradbooststep...
15         0.873706  {'gradbooststep__max_depth': 5, 'gradbooststep...
The score for gradbooststep is 0.8454106280193237
```

Conclusion

This notebook shows the development of an analysis and machine learning pipeline for credit card application approval. The notebook also shows data cleaning and pre-processing of the data set prior to the analysis. Four classifiers were tested with their hyperparameters tuned using a 5-fold cross-validated gridsearch. It is observed that the logistic regression classifier outperformed the support vector, K-neighbours, and gradient boosted classifiers and exhibited a classification accuracy of around 89% on the test set.