PORTFOLIO

DATA ANALYTICS | BIG DATA | MACHINE LEARNING| ARTIFICIAL INTELLIGENCE | DATA MINING| DATA ENGINEERING

A set of projects made by

Oleksandr Kim

Table of CONTENTS

Machine learning

2 - 17	Predict selling price of houses in Ames, Iowa by using machine learning techniques (Multiple Linear Regression, SVR, Decision Tree, Decision Forest)
18 - 60	Predict whether student will pass a course or not (K-NN, SVM, Bayes, Decision Tree, Decision Forest)
61 - 67	Extract information from Zomato API and from Zomato website with BeautifulSoup to Categorize restaurants in Ontario, Canada by prices and ratings (KMEans Clustering)
68 - 90	Predict whether student passes math and Portuguese course in school or not (Logistic Regression)

2 - 17

1.1. Predict selling price of houses in Ames, Iowa by using machine learning techniques (Multiple Linear Regression, SVR, Decision Tree, Decision Forest)

Predict selling price of houses in Ames, Iowa by using machine learning techniques (Multiple Linear Regression, SVR, Decision Tree, Decision Forest)

Information about the dataset

- Number of inputs: **1461**
- Number of variables: **79**
- Dataset: https://www.kaggle.com/c/house-prices-advanced-regression-techniques
- Data fields description: can be found here "data_description.txt"

Importing main libraries

In [1]:
```
import numpy as np
import pandas as pd
```

Importing the dataset

In [2]:
```
df = pd.read_csv('ames_1.csv')
df = df.drop(df.columns[0], axis=1) #deleting the id column
df = df.fillna(0) # replacing NaN with zeros, needed for onehotencoder,
NaN is not accepted
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].astype(float).values #last column are prices
```

Creating a list of categorical variables and encoding them with LabelEncoder

In [3]:
```
col_list = [0,1,4,5,6,7,8,9,10,11,12,13,14,15,16,17,20,
            21,22,23,24,26,27,28,29,30,31,32,34,
            38,39,40,41,52,54,56,57,59,62,63,64,71,72,73,77,78]
#selection was done manually because some categorical variables are nume
rical, some are strings

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder = LabelEncoder()
for i in col_list:
    df.iloc[:, i] = labelencoder.fit_transform(df.iloc[:, i].astype(str)
)
```

Creating a list of continious variables

In [4]:
```
no_cat_var = []
for el in range(len(df.columns)-1):         #excluding variable that we
are predicting
    if el in col_list:
```

```
        continue
    else:
        no_cat_var.append(el)
```

Creating a reference dictionary to find corresponding variables after OneHotEncoding in the initial dataframe

This dictionary can be used to find corresponding variables that were chosen by "Forward Selection" and "Backward Elimination" further below

In [5]:
```
ref_dict = {}
dict_iter=0
```

Encoding categorical variables with OneHotEncoder

The first categorical column will be encoded, result will be added separately in a ndarray, ecluding first dummy column. All other categorical columns will be encoded and added to this ndarray afterwards via loop. That allows to use OneHotEncoder on range of categorical variables without manually encoding one variable after another

In [6]:
```
df_categorical = df.iloc[:, col_list]                                  #df with
 categorical variables

X_cat = df_categorical.iloc[:, :].values                               #categor
ical ndarray
X_cat[:, 0] = labelencoder.fit_transform(X_cat[:, 0])
onehotencoder = OneHotEncoder(categorical_features = [0])              #encodin
g 1st column
X_cc = onehotencoder.fit_transform(X_cat).toarray()
dummy_col = df_categorical.iloc[:, 0].nunique()                        #finding
 out number of dummy colummns created

X_cc_2 = X_cc[:, 1:dummy_col]                                          #moving
to a separate ndarray excluding first dummy column

df_cat_no_one = df_categorical.iloc[:, 1:]                             #first c
olumn was preprocessed so it was excuded from further loop
X_cat_no_one = df_cat_no_one.iloc[:, :].values

ref_dict[0] = list(range(dict_iter, dict_iter+dummy_col))            #adding id
 of original column as key, all corresponding dummy columns as list
dict_iter = dict_iter + dummy_col
```

Now the first column was encoded in dummy variables and they were added to separate ndarray. Other encoded variables will be added to this ndarray via loop below

Adding other categorical variables to ndarray via loop

In [7]:
```
dict_iter=0
for c in range(len(col_list)-1):
    X_cat_no_one[:, c] = labelencoder.fit_transform(X_cat_no_one[:, c])

    onehotencoder = OneHotEncoder(categorical_features = [c])
    X_cc = onehotencoder.fit_transform(X_cat_no_one).toarray()
```

```
    dummy_col = df_categorical.iloc[:, c+1].nunique()
          #+1 because of reffering to df with all categorical variables,
 including the first one
    X_cc2_2 = X_cc[:, 1:dummy_col]
          #excluding first dummy column
    X_cc_2 = np.concatenate((X_cc_2, X_cc2_2), axis=1)
          #merge 2 ndarrays
    ref_dict[c+1] = list(range(dict_iter, dict_iter+dummy_col))
          #adding id of original column as key, all corresponding dummy
columns as list
    dict_iter = dict_iter + dummy_col
```

After that all continious variables are added to a ndarray with encoded categorical variables

In [8]:
```
df_non_categorical = df.iloc[:, no_cat_var]
X_no_cat_var = df_non_categorical.iloc[:, :].values
merged_dataset = np.concatenate((X_cc_2, X_no_cat_var), axis=1)
```

Final dataset to work with

296 columns

Selecting columns to work with

At this point there are 296 columns to choose from for a machine learning model. A subjective selection is not appropriate so two approaches will be used to select a required range of variables for machine learning algorithm. These approaches are "Backward Elimination" and "Forward selection": https://en.wikipedia.org/wiki/Stepwise_regression

Lets start with **Backward Elimination:**

In [9]:
```
import statsmodels.formula.api as sm

p=0.05

#imputs for def are: dataset and p-value
def BackwardElimination(merged_dataset, p):
    merged_dataset = np.append(arr = np.ones((np.size(merged_dataset,0),
1)).astype(int), values=merged_dataset, axis=1) #np.size(merged_categ,0)
 - number of rows in numpy array
    #this adds our dataset to a column of one so ones are in the first c
olumn

    #number of columns
    len_list = []                                          #list of indexes of al
l columns
    for i in range(np.size(merged_dataset,1)+1):
        len_list.append(i)


    p = p #p-value for; can be adjusted depending on desired result (def
ault - 0.05)
```

```
    end = False
    while end==False:
        regressor_OLS = sm.OLS(endog = y, exog = merged_dataset).fit()
        p_values = regressor_OLS.pvalues
        #enable these prints to see a process of selection in a real tim
e
        #print("P values are: "+str(['%.3f' % i for i in p_values.tolist
()]))
        #print("Max p value: "+str(max(p_values)))
        #print("==============================================")
        if max(p_values)<p:
            end = True
            return merged_dataset
        elif max(p_values)>=p:
            p_max_pos = p_values.tolist().index(max(p_values))
            merged_dataset = np.delete(merged_dataset, [p_max_pos], axis
=1)

X = BackwardElimination(merged_dataset, p)
```

LinearRegression with cross-validation (Backward Elimination)

In [10]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
2, random_state = 0)

from sklearn.linear_model import LinearRegression
regressor_back = LinearRegression()
regressor_back.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor_back.predict(X_test)

r2_scores = cross_val_score(regressor_back, X_train, y_train, scoring='r
2', cv=3)
print('Cross-validation score for r^2={}'.format(r2_scores))
```

```
Cross-validation score for r^2=[0.78636832 0.91921861 0.9122908 ]
```

R-squared of Linear Regression

In [11]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[11]: 0.570934689084265

MSE of Linear Regression

In [12]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[12]: 2963060661.942607

MAE of Linear Regression

In [13]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[13]: 19686.99444786974

Adding results to a table for summarization in the end

In [14]:
```
model_name=[]
mse=[]
r2=[]
mae=[]

model_name.append("Backward/MLR")
mae.append(mean_absolute_error(y_test, y_pred))
r2.append(r2_score(y_test, y_pred))
mse.append(mean_squared_error(y_test, y_pred))
```

SVR (RBF kernel) (Backward Elimination)

Train test split and Feature Scaling

In [15]:
```
from sklearn.svm import SVR
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train) #X_train.reshape(-1, 1) is added b
ecause there is only one column
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train.reshape(-1, 1))
y_test = sc_y.fit_transform(y_test.reshape(-1, 1))
```

Using GridSearch to find the best combination of C and gamma

In [16]:
```
#parameters
Cs = [0.0001, 0.001, 0.01, 0.1, 1, 10]
gammas = [0.0001, 0.001, 0.01, 0.1, 1, 2]
param_grid = dict(gamma=gammas, C=Cs)

#model
from sklearn.model_selection import GridSearchCV
svr = SVR(kernel='rbf')
grid_search = GridSearchCV(svr, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train.ravel()) #ravel is needed to convert in
t to float

y_pred = grid_search.predict(X_test)
```

In [17]:
```
print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'C': 10, 'gamma': 0.001}
```

In [18]:
```python
print('Grid best score (accuracy): ', grid_search.best_score_) #train da
ta
```

```
Grid best score (accuracy):  0.9094093728337792
```

R-squared of SVR

In [19]:
```python
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[19]:
```
0.7680587755446528
```

MSE of SVR

In [20]:
```python
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[20]:
```
0.2319412244553471
```

MAE of SVR

In [21]:
```python
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[21]:
```
0.21544819542982774
```

Adding results to a table for summarization in the end

In [22]:
```python
model_name.append("Backward/SVR")
mae.append(mean_absolute_error(y_test, y_pred))
r2.append(r2_score(y_test, y_pred))
mse.append(mean_squared_error(y_test, y_pred))
```

Decision Tree with cross-validation and GridSearch (Backward Elimination)

Train test split and Feature Scaling

In [23]:
```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train) #X_train.reshape(-1, 1) is added b
ecause there is only one column
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train.reshape(-1, 1))
y_test = sc_y.fit_transform(y_test.reshape(-1, 1))
```

Using GridSearch to find the best combination of parameters

In [24]:
```
max_depth = np.linspace(1, 40, 40, endpoint=True)

param_grid = dict(max_depth=max_depth)

#model
from sklearn.model_selection import GridSearchCV
dec_tree = DecisionTreeRegressor()
grid_search = GridSearchCV(dec_tree, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train.ravel()) #ravel is needed to convert in
t to float

y_pred = grid_search.predict(X_test)
```

In [25]:
```
print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'max_depth': 9.0}
```

In [26]:
```
print('Grid best score (accuracy): ', grid_search.best_score_) #train da
ta
```

```
Grid best score (accuracy):  0.7326437511733014
```

R-squared of Decision Tree

In [27]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[27]:
```
0.7646343334387243
```

MSE of Decision Tree

In [28]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[28]:
```
0.23536566656127567
```

MAE of Decision Tree

In [29]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[29]:
```
0.3132771151905251
```

Adding results to a table for summarization in the end

In [30]:
```
model_name.append("Backward/Decision Tree")
mae.append(mean_absolute_error(y_test, y_pred))
r2.append(r2_score(y_test, y_pred))
mse.append(mean_squared_error(y_test, y_pred))
```

Random Forest

Train test split and Feature Scaling

In [31]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train) #X_train.reshape(-1, 1) is added because there is only one column
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train.reshape(-1, 1))
y_test = sc_y.fit_transform(y_test.reshape(-1, 1))
```

Using GridSearch to find the best combination of parameters

In [32]:
```
from sklearn.ensemble import RandomForestRegressor

max_depth = np.linspace(1, 40, 40, endpoint=True)
n_estimators = [5,10,15,20,30]

param_grid = dict(max_depth=max_depth, n_estimators = n_estimators)

#model
from sklearn.model_selection import GridSearchCV
forest = RandomForestRegressor()
grid_search = GridSearchCV(forest, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train.ravel())

y_pred = grid_search.predict(X_test)
```

In [33]:
```
print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'max_depth': 32.0, 'n_estimators': 30}
```

In [34]:
```
print('Grid best score (accuracy): ', grid_search.best_score_) #train data
```

```
Grid best score (accuracy):  0.857532073745342
```

R-squared of Decision Forest

In [35]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[35]:
```
0.8256867368203639
```

MSE of Decision Forest

In [36]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[36]: 0.1743132631796361

MAE of Decision Forest

In [37]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[37]: 0.23992240987487665

Adding results to a table for summarization in the end

In [38]:
```
model_name.append("Backward/Random Forest")
mae.append(mean_absolute_error(y_test, y_pred))
r2.append(r2_score(y_test, y_pred))
mse.append(mean_squared_error(y_test, y_pred))
```

Lets apply Forward Selection:

In [39]:
```
import statsmodels.formula.api as sm
y = df.iloc[:, -1].astype(float).values

def ForwardSelection(merged_dataset, p):
    unknown_variables = []                  #a list of variables that ar
e not included as "good" ones; after each iteration some variable dissap
ears from "unknown" and becomes "good"
    for i in range(merged_dataset.shape[1]):
        unknown_variables.append(i)

    #adding b0 variable from formula
    merged_dataset = np.append(arr = np.ones((np.size(merged_dataset,0),
1)).astype(int), values=merged_dataset, axis=1) #np.size(merged_categ,0)
 - number of rows in numpy array
    p = p

    ###first iteration is added separately, others in a loop below
    p_values_list=[]
    good_variables=[]
    for i in range(merged_dataset.shape[1]):
        X_opt = merged_dataset[:, i]
        regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()   #finding
 p value of every variable and y(the variable to predict)
        p_value = regressor_OLS.pvalues
        p_values_list.extend(p_value.tolist())
    min_p_value = min(p_values_list)                            #finding
 the minimum p value
    min_index = p_values_list.index(min_p_value)                #variabl
e with the smallest p value
    good_variables.append(min_index)                            #add a v
ariable to a "good" list
    unknown_variables.remove(min_index)                         #remove
index from a list of "bad" variables

    end=False
    while end==False:
        comb_list = []
        p_values_list=[]
```

```
        #this loop exists to make combinations of "good" variables with
every "unknown" to find p value of every combination
        for i in unknown_variables:
            temp_list = []
            for t in good_variables:
                temp_list.append(t)
            temp_list.append(i)
            comb_list.append([temp_list])
            #print(temp_list)
        for el in comb_list:
            X_opt = merged_dataset[:, el[0]]
            regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
            p_value = regressor_OLS.pvalues
            pvalue_lst = p_value.tolist()
            p_values_list.append(pvalue_lst[-1])
        #finding combination with min p value
        min_p_value = min(p_values_list)
        min_index = p_values_list.index(min_p_value)
        good_variables.append(comb_list[min_index][-1][-1])
        unknown_variables.remove(comb_list[min_index][-1][-1])
        #uncomment to see every step
        #print("Min p value: "+str(min_p_value))
        #print("List of variables: "+str(good_variables))
        #print("####################################")
        if min_p_value>p:
             end=True

    #print("UN: "+str(unknown_variables))
    print("GN: "+str(good_variables))
    return merged_dataset[:, good_variables]
```

In [40]:
```
p = 0.05
X = ForwardSelection(merged_dataset, p)
```

```
GN: [0, 269, 259, 159, 265, 279, 275, 93, 92, 85, 70, 235, 40, 168, 91,
264, 1, 249, 49, 99, 258, 100, 98, 101, 56, 50, 286, 234, 233, 243, 255,
 206, 113, 55, 116, 60, 277, 77, 260, 261, 37, 285, 107, 97, 76, 173, 29
, 19, 268, 41, 112, 119, 170, 22, 108, 158, 284, 276, 274, 270, 272, 228
, 210, 186, 17]
```

LinearRegression via cross-validation (Forward Selection)

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
2, random_state = 0)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)

r2_scores = cross_val_score(regressor, X_train, y_train, scoring='r2', c
v=3)
print('Cross-validation score for^2={}'.format(r2_scores))
```

```
Cross-validation score for^2=[0.78110129 0.91666683 0.89260419]
```

R-squared of Linear Regression

In [42]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[42]: 0.6605208927634301

MSE of Linear Regression

In [43]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[43]: 2344391780.489629

MAE of Linear Regression

In [44]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[44]: 19278.979224686384

Adding results to a table for summarization in the end

In [45]:
```
model_name.append("Forward/MLR")
mae.append(mean_absolute_error(y_test, y_pred))
r2.append(r2_score(y_test, y_pred))
mse.append(mean_squared_error(y_test, y_pred))
```

SVR (RBF kernel) (Forward Selection)

Train test split and Feature Scaling

In [46]:
```
from sklearn.svm import SVR
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train) #X_train.reshape(-1, 1) is added b
ecause there is only one column
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train.reshape(-1, 1))
y_test = sc_y.fit_transform(y_test.reshape(-1, 1))
```

Using GridSearch to find the best combination of C and gamma

In [47]:
```
#parameters
Cs = [0.0001, 0.001, 0.01, 0.1, 1, 10]
gammas = [0.0001, 0.001, 0.01, 0.1, 1, 2]
param_grid = dict(gamma=gammas, C=Cs)

#model
from sklearn.model_selection import GridSearchCV
```

```
svr = SVR(kernel='rbf')
grid_search = GridSearchCV(svr, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train.ravel()) #ravel is needed to convert in
t to float

y_pred = grid_search.predict(X_test)
```

In [48]:
```
print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'C': 10, 'gamma': 0.001}
```

In [49]:
```
print('Grid best score (accuracy): ', grid_search.best_score_) #train da
ta
```

```
Grid best score (accuracy):  0.9012570709150719
```

R-squared of SVR

In [50]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[50]: 0.759773373113279

MSE of SVR

In [51]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[51]: 0.24022662688672103

MAE of SVR

In [52]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[52]: 0.21734930595806085

Adding results to a table for summarization in the end

In [53]:
```
model_name.append("Forward/SVR")
mae.append(mean_absolute_error(y_test, y_pred))
r2.append(r2_score(y_test, y_pred))
mse.append(mean_squared_error(y_test, y_pred))
```

Decision Tree with cross-validation and GridSearch (Forward Selection)

Train test split and Feature Scaling

In [54]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
2, random_state = 0)
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train) #X_train.reshape(-1, 1) is added b
ecause there is only one column
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train.reshape(-1, 1))
y_test = sc_y.fit_transform(y_test.reshape(-1, 1))
```

Using GridSearch to find the best combination of parameters

In [55]:
```
max_depth = np.linspace(1, 40, 40, endpoint=True)

param_grid = dict(max_depth=max_depth)

#model
from sklearn.model_selection import GridSearchCV
dec_tree = DecisionTreeRegressor()
grid_search = GridSearchCV(dec_tree, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train.ravel()) #ravel is needed to convert in
t to float

y_pred = grid_search.predict(X_test)
```

In [56]:
```
print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'max_depth': 8.0}
```

In [57]:
```
print('Grid best score (accuracy): ', grid_search.best_score_) #train da
ta
```

```
Grid best score (accuracy):  0.7363753120652549
```

R-squared of Decision tree

In [58]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[58]:
```
0.7534796402230192
```

MSE of Decision tree

In [59]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[59]:
```
0.24652035977698084
```

MAE of Decision Tree

In [60]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[60]:
```
0.33652024310605166
```

Adding results to a table for summarization in the end

In [61]:
```
model_name.append("Forward/Decision Tree")
mae.append(mean_absolute_error(y_test, y_pred))
r2.append(r2_score(y_test, y_pred))
mse.append(mean_squared_error(y_test, y_pred))
```

Random Forest

Train test split and Feature Scaling

In [62]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train) #X_train.reshape(-1, 1) is added b
ecause there is only one column
X_test = sc_X.transform(X_test)
sc_y = StandardScaler()
y_train = sc_y.fit_transform(y_train.reshape(-1, 1))
y_test = sc_y.fit_transform(y_test.reshape(-1, 1))
```

Using GridSearch to find the best combination of parameters

In [63]:
```
from sklearn.ensemble import RandomForestRegressor

max_depth = np.linspace(1, 40, 40, endpoint=True)
n_estimators = [5,10,15,20,30]

param_grid = dict(max_depth=max_depth, n_estimators = n_estimators)

#model
from sklearn.model_selection import GridSearchCV
forest = RandomForestRegressor()
grid_search = GridSearchCV(forest, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train.ravel())

y_pred = grid_search.predict(X_test)
```

In [64]:
```
print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'max_depth': 20.0, 'n_estimators'
: 30}
```

In [65]:
```
print('Grid best score (accuracy): ', grid_search.best_score_) #train da
ta
```

```
Grid best score (accuracy):  0.8610798031811986
```

R-squared

In [66]:
```python
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[66]: 0.8449653911665643

MSE

In [67]:
```python
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[67]: 0.15503460883343567

MAE of Decision Tree

In [68]:
```python
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[68]: 0.23708490356253453

Adding results to a table for summarization in the end

In [69]:
```python
model_name.append("Forward/Random Forest")
mae.append(mean_absolute_error(y_test, y_pred))
r2.append(r2_score(y_test, y_pred))
mse.append(mean_squared_error(y_test, y_pred))
```

In [75]:
```python
d = {'model_name': model_name, 'mse': mse, 'r2': r2,
'mae': mae}
df = pd.DataFrame(data=d)
df.round(3)
```

Out[75]:

	model_name	mse	r2	mae
0	Backward/MLR	2.963061e+09	0.571	19686.994
1	Backward/SVR	2.320000e-01	0.768	0.215
2	Backward/Decision Tree	2.350000e-01	0.765	0.313
3	Backward/Random Forest	1.740000e-01	0.826	0.240
4	Forward/MLR	2.344392e+09	0.661	19278.979
5	Forward/SVR	2.400000e-01	0.760	0.217
6	Forward/Decision Tree	2.470000e-01	0.753	0.337
7	Forward/Random Forest	1.550000e-01	0.845	0.237

Summary

In summary, Random Forest achieved the best r-squared among all models as well as the best MSE. As of the feature selection approach, Forward Selection did better for Random Forst, but worse for SVR and Decision Tree. The worst model in linear regression

18 - 67

1.2. Predict whether student will pass a course or not (K-NN, SVM, Bayes, Decision Tree, Decision Forest)

Predict whether student will pass a course or not (K-NN, SVM, Bayes, Decision Tree, Decision Forest)

Information about the dataset

- Number of inputs: **480**
- Number of variables: **17**
- Dataset: https://www.kaggle.com/aljarah/xAPI-Edu-Data
- Data fields description: https://www.kaggle.com/aljarah/xAPI-Edu-Data
  Dataset has a range of features of a student that allow to predict a performance of a student

Importing main libraries

In [41]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
from mpl_toolkits.mplot3d import Axes3D
import warnings; warnings.simplefilter('ignore')
```

Importing the dataset

In [42]:
```
dataset = pd.read_csv('xAPI-Edu-Data.csv')
```

Changing last column that corresponds to a mark. High and Middle - passing marks, Low is a fail

In [43]:
```
dataset.iloc[:, -1] = dataset.iloc[:, -1].map({'H': 1, 'M': 1, 'L': 0})
```

9,10,11,12 - are continious columns that will be used to build k-nn, svm, decision tree
9 Raised hand- how many times the student raises his/her hand on classroom (numeric:0-100)
10 Visited resources- how many times the student visits a course content(numeric:0-100)
11 Viewing announcements-how many times the student checks the new announcements(numeric:0-100)
12 Discussion groups- how many times the student participate on discussion groups (numeric:0-100)

In [44]:
```
X = dataset.iloc[:, [9, 10 , 11, 12]].values#.astype(float)
y = dataset.iloc[:, 16].values#.astype(float)
```

Selecting columns to work with

Two approaches will be used to select a required range of variables for machine learning algorithm. These approaches are "Backward Elimination" and "Forward selection":

https://en.wikipedia.org/wiki/Stepwise_regression

Lets start with **Backward Elimination:**

In [45]:
```
import statsmodels.formula.api as sm

p=0.05

#imputs for def are: X, y and p-value
def BackwardElimination(merged_dataset, y, p):
    #merged_dataset = np.append(arr = np.ones((np.size(merged_dataset,0)
,1)).astype(int), values=merged_dataset, axis=1) #np.size(merged_categ,0
) - number of rows in numpy array
    #this adds our dataset to a column of one so ones are in the first c
olumn (for linear regression)

    #number of columns
    len_list = []                                   #list of indexes of al
l columns
    for i in range(np.size(merged_dataset,1)+1):
        len_list.append(i)


    p = p #p-value for; can be adjusted depending on desired result (def
ault - 0.05)

    end = False
    while end==False:
        regressor_OLS = sm.OLS(endog = y, exog = merged_dataset).fit()
        p_values = regressor_OLS.pvalues
        #enable these prints to see a process of selection in a real tim
e
        #print("P values are: "+str(['%.3f' % i for i in p_values.tolist
()]))
        #print("Max p value: "+str(max(p_values)))
        #print("==============================================")
        if max(p_values)<p:
            end = True
            return merged_dataset
        elif max(p_values)>=p:
            p_max_pos = p_values.tolist().index(max(p_values))
            merged_dataset = np.delete(merged_dataset, [p_max_pos], axis
=1)

X = BackwardElimination(X, y, p)
```

Building a K-NN model using PramGrid (Backward Elimination)

In [46]:
```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
25, random_state = 0)
```

Feature Scaling

In [47]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
#sc_y = StandardScaler()
#y_train = sc_y.fit_transform(y_train.reshape(-1, 1))
#y_test = sc_y.fit_transform(y_test.reshape(-1, 1))
```

Fitting a model

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier

n_neighbors = range(1,10)

param_grid = dict(n_neighbors=n_neighbors)

from sklearn.model_selection import GridSearchCV
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)
```

In [49]:
```
print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'n_neighbors': 7}
```

Train score

In [50]:
```
grid_search.score(X_train, y_train)
```

Out[50]:
```
0.8944444444444445
```

Test score

In [51]:
```
grid_search.score(X_test, y_test)
```

Out[51]:
```
0.9083333333333333
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [52]:
```
from sklearn.model_selection import cross_val_score
print('Accuracy: '+ str(cross_val_score(grid_search, X_train, y_train.ra
vel(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(grid_search, X_train, y_train.r
avel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(grid_search, X_train, y_train.rave
l(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(grid_search, X_train, y_train.ravel(),
 scoring='f1', cv=3)))
```

```
Accuracy: [0.85123967 0.88333333 0.85714286]
Precision: [0.9382716  0.89473684 0.8988764 ]
Recall: [0.85393258 0.95505618 0.90909091]
F1: [0.89411765 0.92391304 0.9039548 ]
```

Building a Confusion Metrics

In [53]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[53]:
```
array([[27,  6],
       [ 5, 82]], dtype=int64)
```

109 were predicted right, while 11 were predicted wrong. 109/120 = 0.91

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [54]:
```
from sklearn.metrics import precision_score
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```

```
False Positive Rate is: 0.07
```

AUC score

In [55]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

```
Area under the curve score: 0.88
```

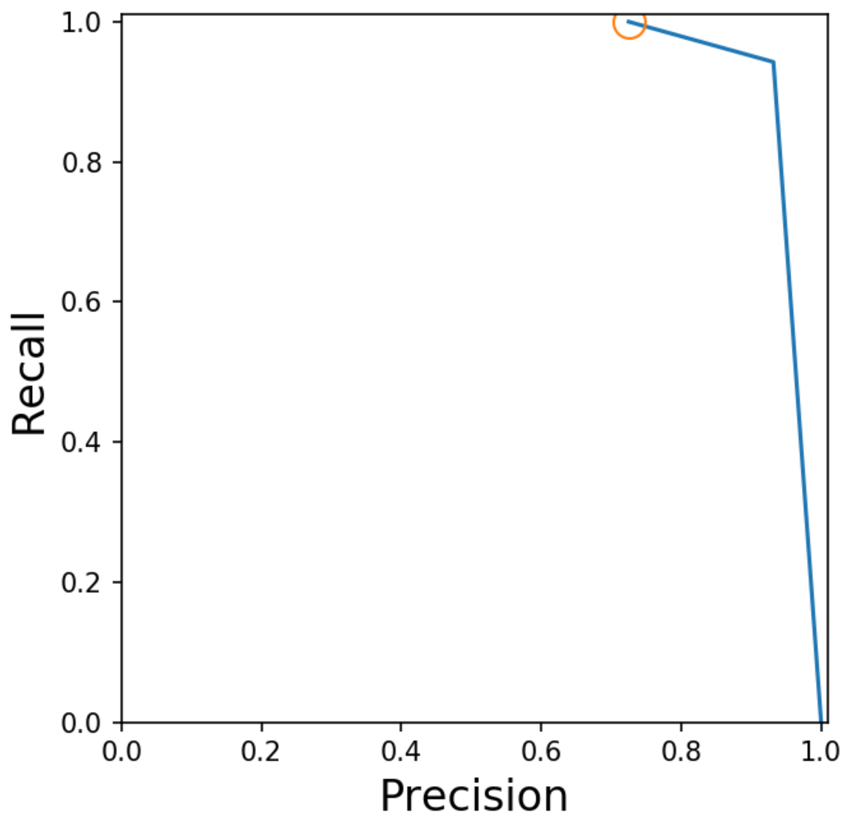
Adding results to a table for summarization in the end

In [56]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score



model_name=[]
accuracy_col=[]
precision_col=[]
recall_col=[]
f1_col=[]
auc_col=[]

model_name.append("Backward/KNN")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

Building a precision-recall curve

In [57]:
```
from sklearn.metrics import precision_recall_curve
```

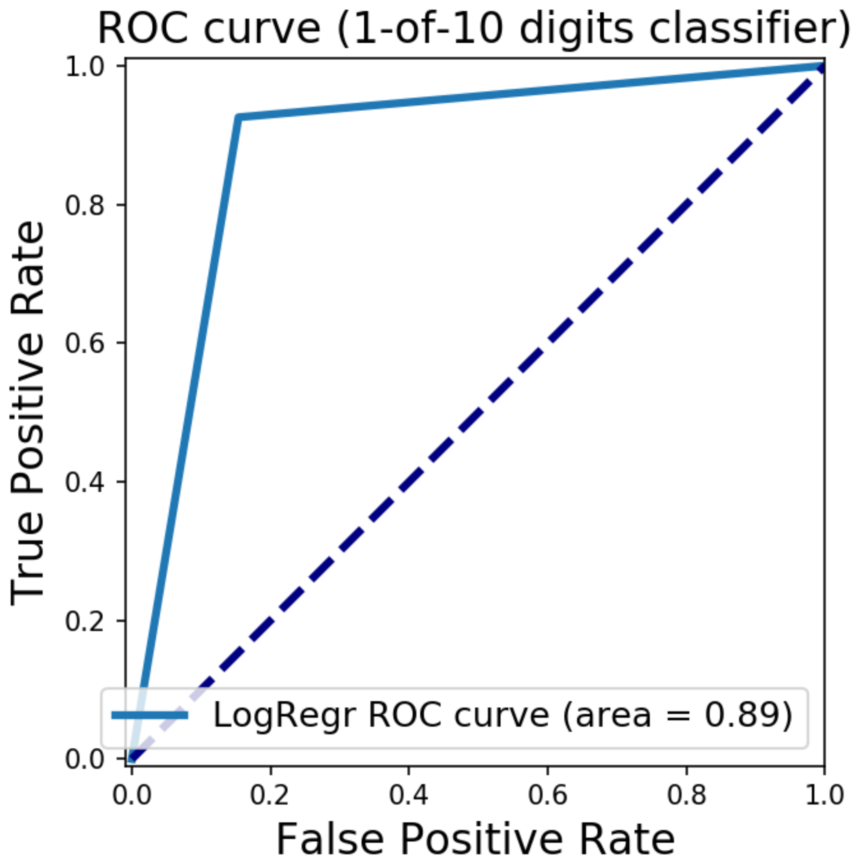
```
y_scores_lr = grid_search.fit(X_train, y_train).predict(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [58]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).predict(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
```

```
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Building a Naive Bayes model using PramGrid (Backward Elimination)

Fitting a model

In [59]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

Train score

In [60]:
```
classifier.score(X_train, y_train)
```

Out[60]: 0.8555555555555555

Test score

In [61]:
```
classifier.score(X_test, y_test)
```
Out[61]:
```
0.8833333333333333
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [62]:
```
from sklearn.model_selection import cross_val_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: '+ str(cross_val_score(classifier, X_train, y_train.rav
el(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(classifier, X_train, y_train.ra
vel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(classifier, X_train, y_train.ravel
(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(classifier, X_train, y_train.ravel(),
scoring='f1', cv=3)))
```
```
Accuracy: [0.85950413 0.85833333 0.8487395 ]
Precision: [0.9375     0.94805195 0.925     ]
Recall: [0.86206897 0.84883721 0.86046512]
F1: [0.89820359 0.89570552 0.89156627]
```

Building a Confusion Metrics

In [63]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```
Out[63]:
```
array([[25,  1],
       [13, 81]], dtype=int64)
```

106 were predicted right, while 14 were predicted wrong. 106/120 = 0.88

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [64]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```
```
False Positive Rate is: 0.01
```

AUC score

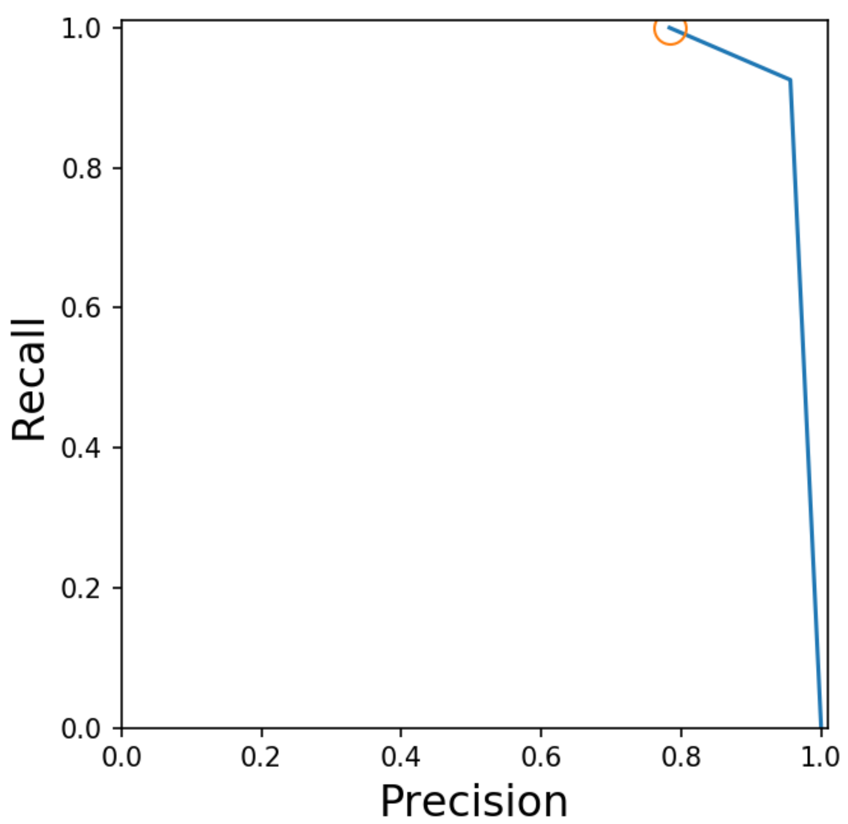
In [65]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```
```
Area under the curve score: 0.91
```

Adding results to a table for summarization in the end

In [66]:
```
model_name.append("Backward/Bayes")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

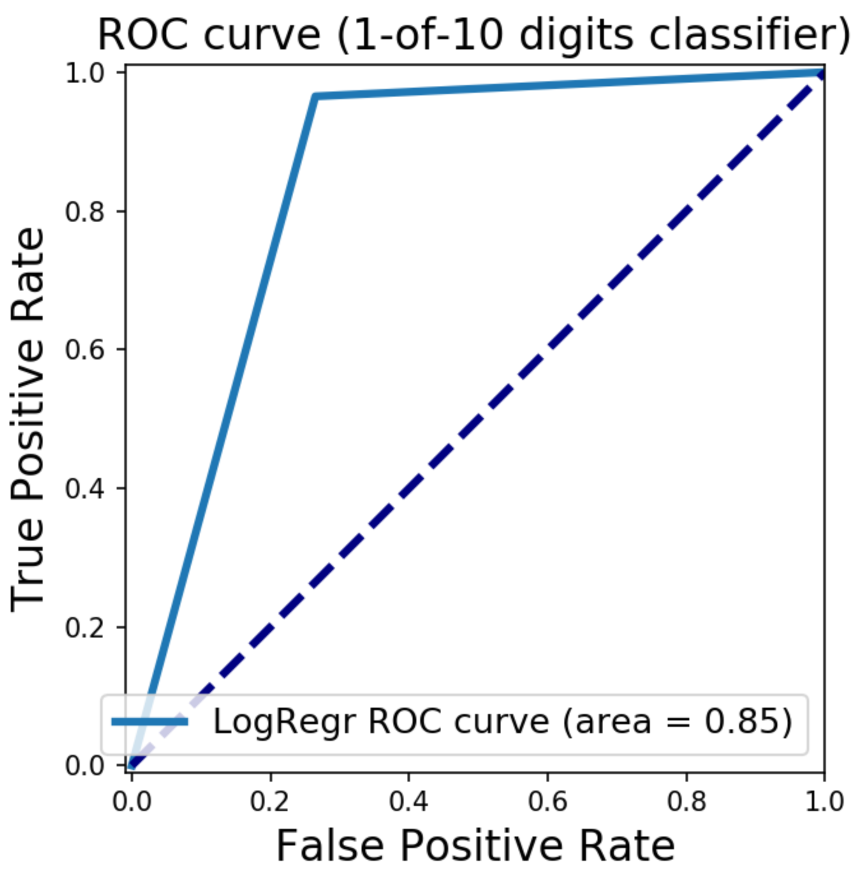
Building a precision-recall curve

In [67]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).predict(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [68]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).predict(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Building a SVC model using PramGrid (Backward Elimination)

Fitting a model

In [69]:
```
from sklearn.svm import SVC
```

```
Cs = [0.0001, 0.001, 0.01, 0.1, 1, 10]
gammas = [0.0001, 0.001, 0.01, 0.1, 1, 2]
param_grid = dict(gamma=gammas, C=Cs)

from sklearn.model_selection import GridSearchCV
svc = SVC(kernel = 'rbf', random_state = 0)
grid_search = GridSearchCV(svc, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)
```

In [70]:
```
print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'C': 10, 'gamma': 0.001}
```

Train score

In [71]:
```
grid_search.score(X_train, y_train)
```

Out[71]:
```
0.9055555555555556
```

Test score

In [72]:
```
grid_search.score(X_test, y_test)
```

Out[72]:
```
0.9
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [73]:
```
from sklearn.model_selection import cross_val_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: '+ str(cross_val_score(grid_search, X_train, y_train.ra
vel(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(grid_search, X_train, y_train.r
avel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(grid_search, X_train, y_train.rave
l(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(grid_search, X_train, y_train.ravel(),
 scoring='f1', cv=3)))
```

```
Accuracy: [0.8        0.9        0.81666667]
Precision: [0.90123457 0.91397849 0.89411765]
Recall: [0.82022472 0.95505618 0.85393258]
F1: [0.85882353 0.93406593 0.87356322]
```

Building a Confusion Metrics

In [74]:
```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, y_pred)
```

Out[74]:
```
array([[25,  9],
       [ 3, 83]], dtype=int64)
```

108 were predicted right, while 14 were predicted wrong. 108/120 = 0.86

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [75]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```

```
False Positive Rate is: 0.10
```

AUC score

In [76]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

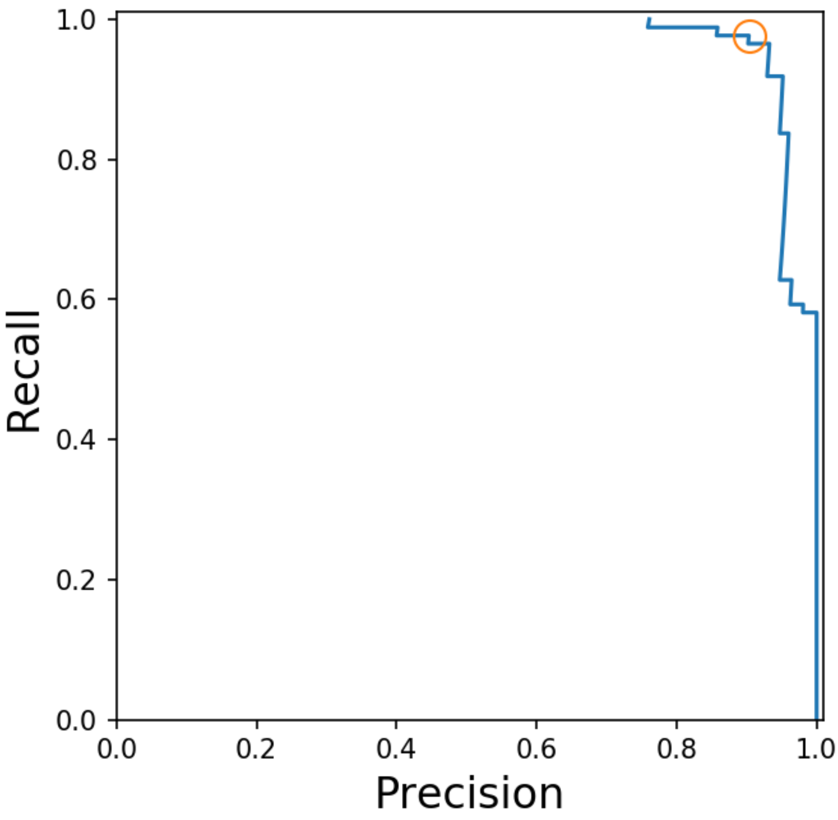
```
Area under the curve score: 0.85
```

Adding results to a table for summarization in the end

In [77]:
```
model_name.append("Backward/SVC")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

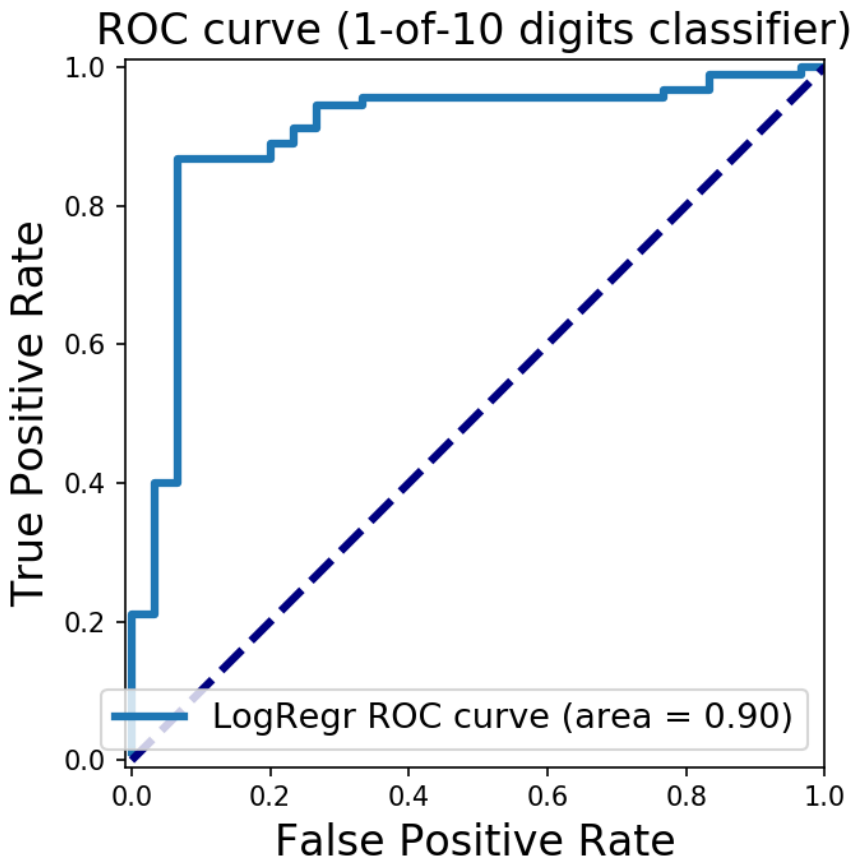
Building a precision-recall curve

In [78]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).decision_function(X_test
)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [79]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).decision_function(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Building a Decision Tree model using PramGrid (Backward Elimination)

In [80]:
```
max_depth = np.linspace(1, 40, 40, endpoint=True)
min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint=True)
criterion = ['entropy', 'gini']

param_grid = dict(max_depth=max_depth,
                  min_samples_split = min_samples_splits,
                  min_samples_leaf = min_samples_leafs,
                  criterion=criterion)

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
grid_search = GridSearchCV(classifier, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train)

grid_search.predict(X_test)


print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
y_pred = grid_search.predict(X_test)
```

```
Grid best parameter (max. accuracy):  {'criterion': 'gini', 'max_depth':
 1.0, 'min_samples_leaf': 0.1, 'min_samples_split': 0.1}
```

Train score

In [81]:
```
grid_search.score(X_train, y_train)
```
Out[81]:
```
0.8638888888888889
```

Test score

In [82]:
```
grid_search.score(X_test, y_test)
```
Out[82]:
```
0.8166666666666667
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [83]:
```
from sklearn.model_selection import cross_val_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: '+ str(cross_val_score(grid_search, X_train, y_train.ra
vel(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(grid_search, X_train, y_train.r
avel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(grid_search, X_train, y_train.rave
l(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(grid_search, X_train, y_train.ravel(),
 scoring='f1', cv=3)))
```
```
Accuracy: [0.85950413 0.84166667 0.86554622]
Precision: [0.89010989 0.91566265 0.9382716 ]
Recall: [0.92045455 0.86363636 0.87356322]
F1: [0.90502793 0.88888889 0.9047619 ]
```

Building a Confusion Metrics

In [84]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```
Out[84]:
```
array([[23,  7],
       [15, 75]], dtype=int64)
```

100 were predicted right, while 22 were predicted wrong. 100/120 = 0.825

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [90]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```
```
False Positive Rate is: 0.26
```

AUC score

In [86]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

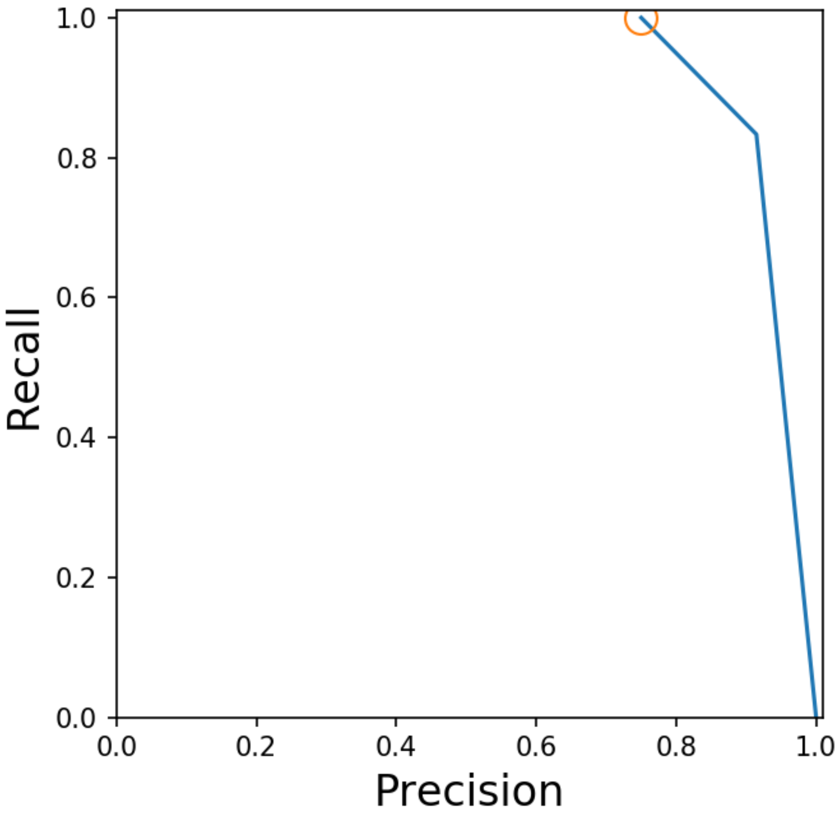
```
Area under the curve score: 0.80
```

Adding results to a table for summarization in the end

In [87]:
```
model_name.append("Backward/Decision Tree")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

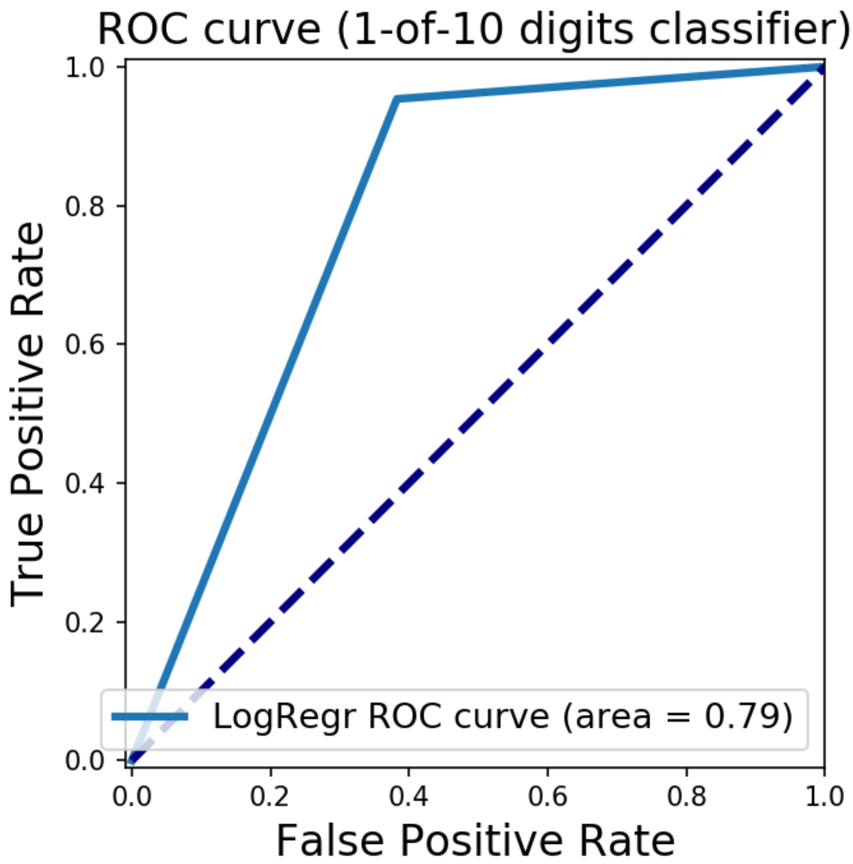
Building a precision-recall curve

In [88]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).predict(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [89]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).predict(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Building a Random Forest model using PramGrid (Backward Elimination)

In [91]:
```
from sklearn.ensemble import RandomForestClassifier

max_depth = np.linspace(1, 40, 40, endpoint=True)
n_estimators = [5,10,15,20,30]
criterion = ['entropy', 'gini']

param_grid = dict(max_depth=max_depth,
                  n_estimators = n_estimators,
                 criterion=criterion)

#model
from sklearn.model_selection import GridSearchCV
forest = RandomForestClassifier()
grid_search = GridSearchCV(forest, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train.ravel())

y_pred = grid_search.predict(X_test)
```

Train score

In [92]:
```
grid_search.score(X_train, y_train)
```

Out[92]: 0.8888888888888888

Test score

In [93]:
```
grid_search.score(X_test, y_test)
```

Out[93]:
```
0.8666666666666667
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [95]:
```
from sklearn.model_selection import cross_val_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: '+ str(cross_val_score(grid_search, X_train, y_train.ra
vel(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(grid_search, X_train, y_train.r
avel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(grid_search, X_train, y_train.rave
l(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(grid_search, X_train, y_train.ravel(),
 scoring='f1', cv=3)))
```

```
Accuracy: [0.85833333 0.875      0.84166667]
Precision: [0.87234043 0.875      0.91860465]
Recall: [0.95505618 0.94382022 0.87640449]
F1: [0.8972973  0.91891892 0.89772727]
```

Building a Confusion Metrics

In [96]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[96]:
```
array([[23, 11],
       [ 5, 81]], dtype=int64)
```

104 were predicted right, while 16 were predicted wrong. 104/120 = 0.85

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [97]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```

```
False Positive Rate is: 0.12
```

AUC score

In [98]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

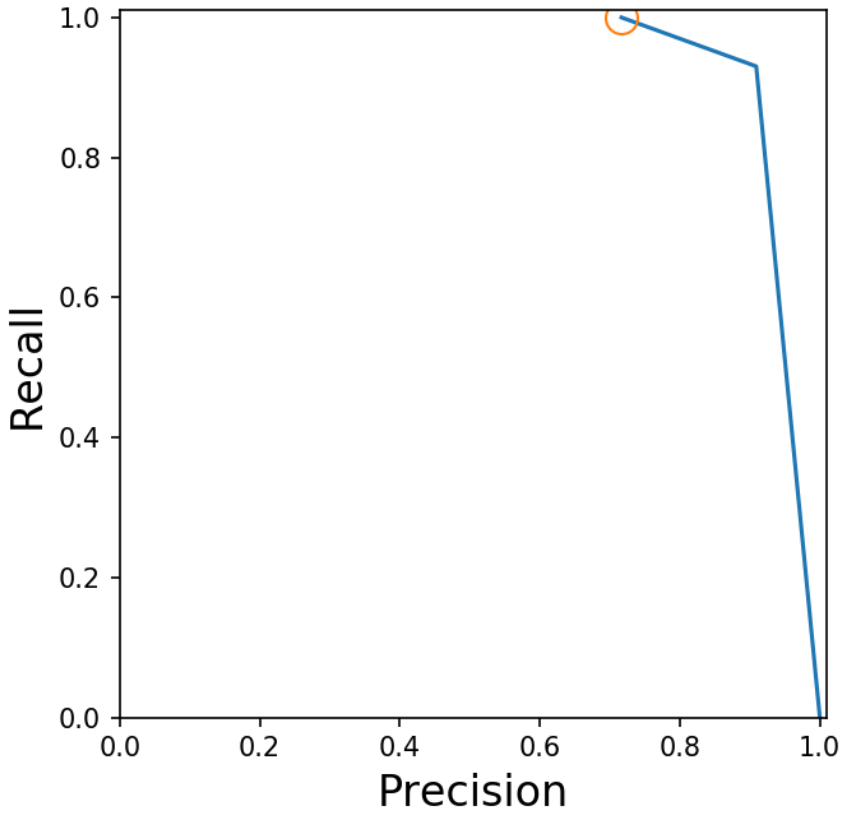
```
Area under the curve score: 0.81
```

Adding results to a table for summarization in the end

In [99]:
```
model_name.append("Backward/Random Forest")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

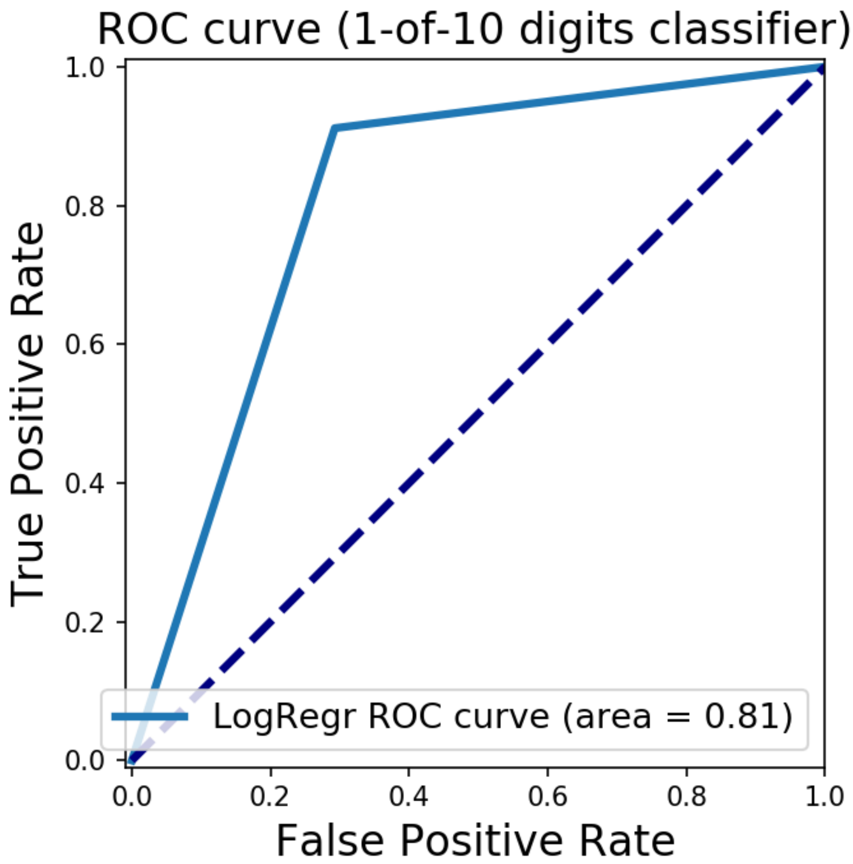
Building a precision-recall curve

In [100]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).predict(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [101]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).predict(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Lets apply Forward Selection now:

In [102]:
```
import statsmodels.formula.api as sm

def ForwardSelection(merged_dataset, y, p):
    unknown_variables = []                  #a list of variables that ar
e not included as "good" ones; after each iteration some variable dissap
ears from "unknown" and becomes "good"
    for i in range(merged_dataset.shape[1]):
        unknown_variables.append(i)

    #adding b0 variable from formula
    #merged_dataset = np.append(arr = np.ones((np.size(merged_dataset,0)
,1)).astype(int), values=merged_dataset, axis=1) #np.size(merged_categ,0
) - number of rows in numpy array
    p = p

    ###first iteration is added separately, others in a loop below
    p_values_list=[]
    good_variables=[]
    for i in range(merged_dataset.shape[1]):
        X_opt = merged_dataset[:, i]
        regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()   #finding
 p value of every variable and y(the variable to predict)
        p_value = regressor_OLS.pvalues
        p_values_list.extend(p_value.tolist())
    min_p_value = min(p_values_list)                            #finding
 the minimum p value
    min_index = p_values_list.index(min_p_value)                #variabl
e with the smallest p value
    good_variables.append(min_index)                            #add a v
```

```
ariable to a "good" list
    unknown_variables.remove(min_index)                         #remove
index from a list of "bad" variables

    end=False
    while end==False:
        comb_list = []
        p_values_list=[]

        #this loop exists to make combinations of "good" variables with
every "unknown" to find p value of every combination
        for i in unknown_variables:
            temp_list = []
            for t in good_variables:
                temp_list.append(t)
            temp_list.append(i)
            comb_list.append([temp_list])
            #print(temp_list)
        for el in comb_list:
            X_opt = merged_dataset[:, el[0]]
            regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
            p_value = regressor_OLS.pvalues
            pvalue_lst = p_value.tolist()
            p_values_list.append(pvalue_lst[-1])
        #finding combination with min p value
        min_p_value = min(p_values_list)
        min_index = p_values_list.index(min_p_value)
        good_variables.append(comb_list[min_index][-1][-1])
        unknown_variables.remove(comb_list[min_index][-1][-1])
        #uncomment to see every step
        #print("Min p value: "+str(min_p_value))
        #print("List of variables: "+str(good_variables))
        #print("####################################")
        if min_p_value>p:
             end=True

    #print("UN: "+str(unknown_variables))
    print("GN: "+str(good_variables))
    return merged_dataset[:, good_variables]
```

In [103]:
```
p = 0.05
X = dataset.iloc[:, [9, 10 , 11, 12]].values
y = dataset.iloc[:, 16].values
X = ForwardSelection(X, y, p)
```

```
GN: [1, 3, 0, 2]
```

Building a K-NN model using PramGrid (Forward Selection)

In [104]:
```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
25, random_state = 0)
```

Feature Scaling

In [105]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
#sc_y = StandardScaler()
#y_train = sc_y.fit_transform(y_train.reshape(-1, 1))
#y_test = sc_y.fit_transform(y_test.reshape(-1, 1))
```

Fitting a model

In [106]:
```
from sklearn.neighbors import KNeighborsClassifier

n_neighbors = range(1,10)

param_grid = dict(n_neighbors=n_neighbors)

from sklearn.model_selection import GridSearchCV
knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train)

y_pred = grid_search.predict(X_test)

print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'n_neighbors': 7}
```

Train score

In [107]:
```
grid_search.score(X_train, y_train)
```

Out[107]:
```
0.8833333333333333
```

Test score

In [108]:
```
grid_search.score(X_test, y_test)
```

Out[108]:
```
0.875
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [109]:
```
from sklearn.model_selection import cross_val_score
print('Accuracy: '+ str(cross_val_score(grid_search, X_train, y_train.ra
vel(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(grid_search, X_train, y_train.r
avel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(grid_search, X_train, y_train.rave
l(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(grid_search, X_train, y_train.ravel(),
 scoring='f1', cv=3)))
```

```
Accuracy: [0.82644628 0.85       0.87394958]
Precision: [0.89534884 0.86597938 0.93975904]
Recall: [0.86516854 0.94382022 0.88636364]
F1: [0.88       0.90322581 0.9122807 ]
```

Building a Confusion Metrics

In [110]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[110]:
```
array([[23, 10],
       [ 5, 82]], dtype=int64)
```

105 were predicted right, while 15 were predicted wrong. 105/120 = 0.875

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [111]:
```
from sklearn.metrics import precision_score
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```

```
False Positive Rate is: 0.11
```

AUC score

In [112]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

```
Area under the curve score: 0.82
```

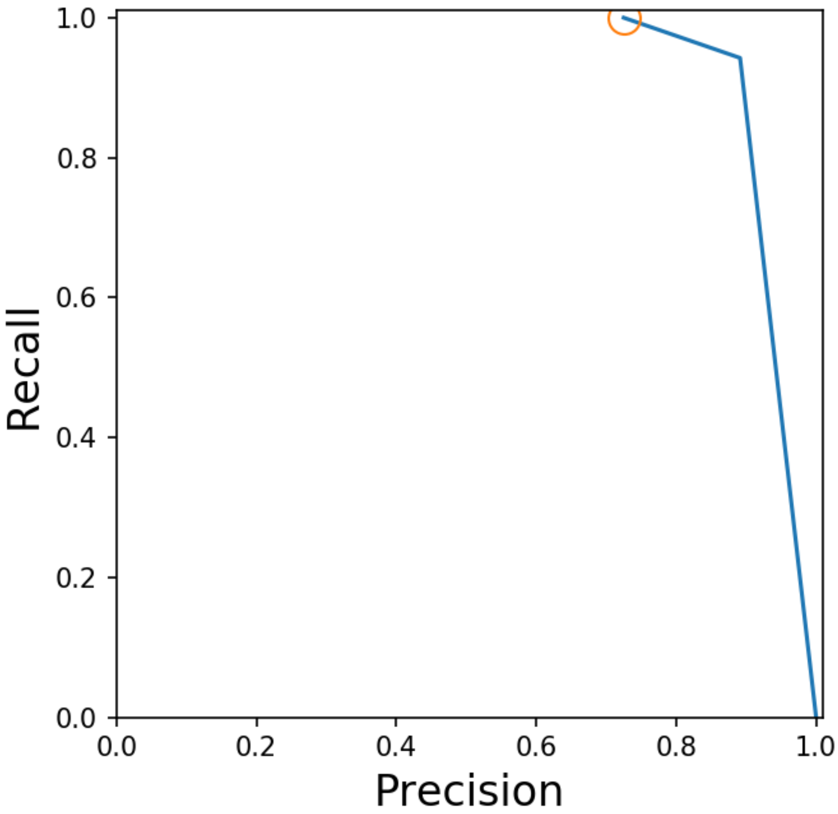
Adding results to a table for summarization in the end

In [113]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

model_name.append("Forward/KNN")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

Building a precision-recall curve

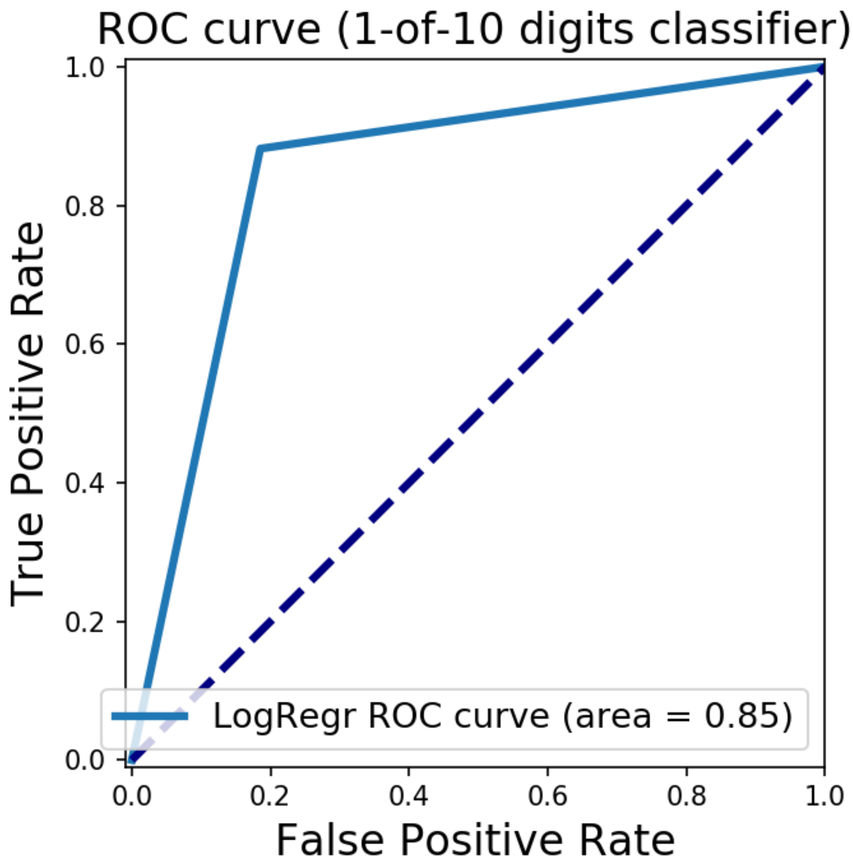
In [114]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).predict(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
```

```
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [115]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).predict(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Building a Naive Bayes model using PramGrid (Forward Selection)

Fitting a model

In [116]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

Train score

In [117]:
```
classifier.score(X_train, y_train)
```

Out[117]: 0.8555555555555555

Test score

In [118]:
```
classifier.score(X_test, y_test)
```

Out[118]: 0.875

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [119]:
```
from sklearn.model_selection import cross_val_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: '+ str(cross_val_score(classifier, X_train, y_train.rav
el(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(classifier, X_train, y_train.ra
vel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(classifier, X_train, y_train.ravel
(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(classifier, X_train, y_train.ravel(),
scoring='f1', cv=3)))
```

```
Accuracy: [0.82644628 0.83333333 0.87394958]
Precision: [0.875      0.90361446 1.        ]
Recall: [0.88505747 0.86206897 0.8255814 ]
F1: [0.88       0.88235294 0.9044586 ]
```

Building a Confusion Metrics

In [120]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[120]:
```
array([[27,  0],
       [15, 78]], dtype=int64)
```

105 were predicted right, while 15 were predicted wrong. 102/120 = 0.86

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [121]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```

```
False Positive Rate is: 0.00
```

AUC score

In [122]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

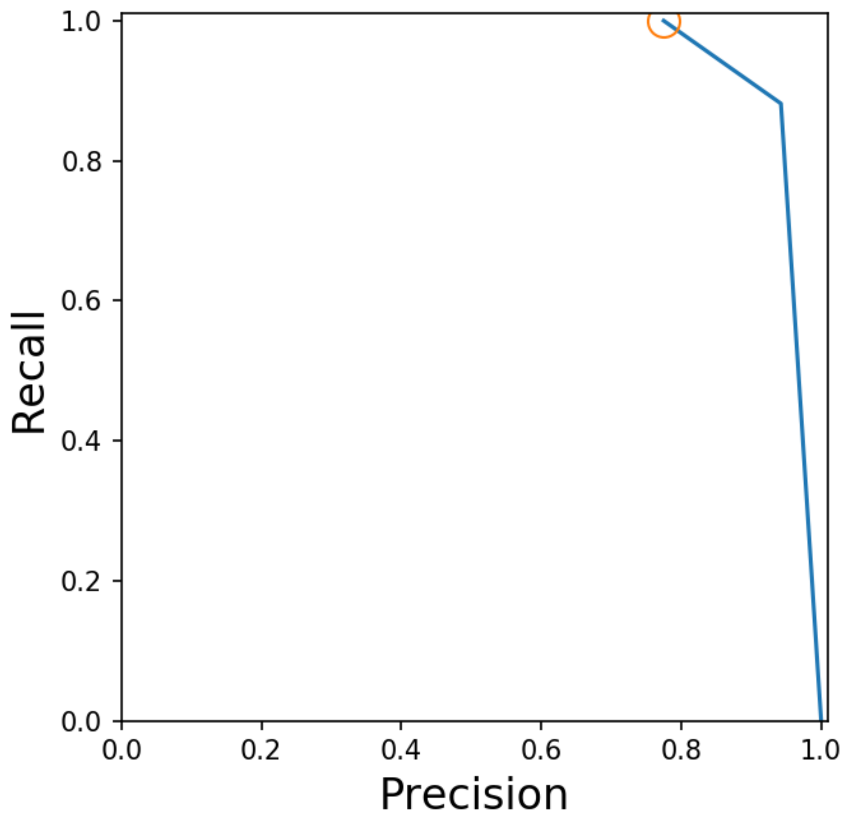
```
Area under the curve score: 0.92
```

Adding results to a table for summarization in the end

In [123]:
```
model_name.append("Forward/Bayes")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

Building a precision-recall curve

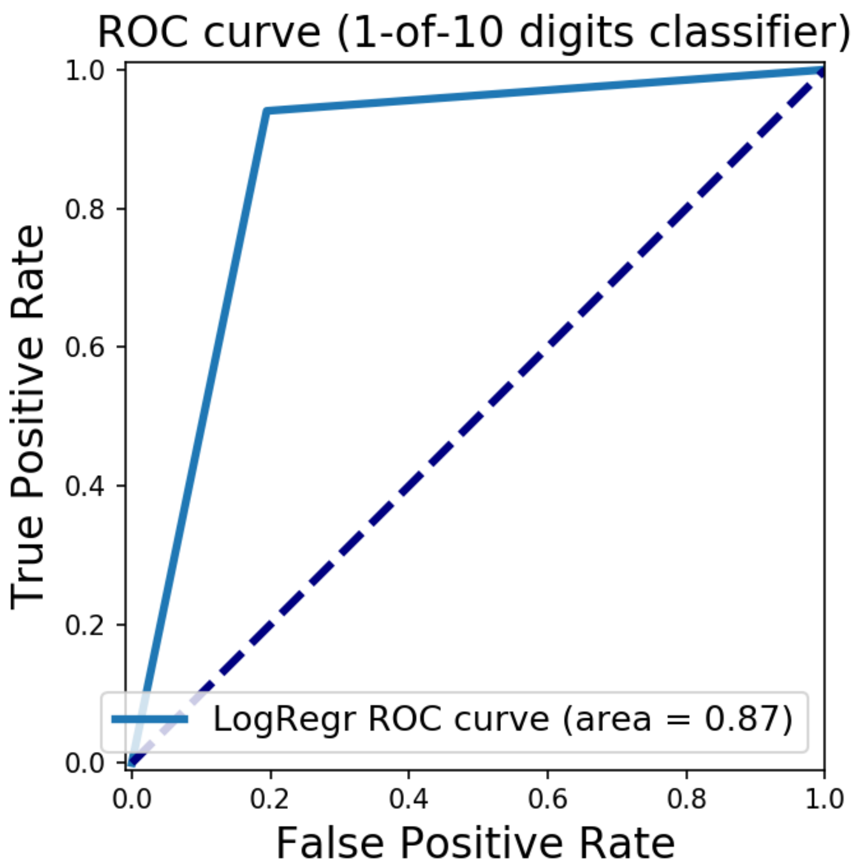
In [124]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).predict(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [125]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).predict(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)
```

```
plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Building a SVC model using PramGrid (Forward Selection)

Fitting a model

In [126]:
```
from sklearn.svm import SVC

Cs = [0.0001, 0.001, 0.01, 0.1, 1, 10]
gammas = [0.0001, 0.001, 0.01, 0.1, 1, 2]
param_grid = dict(gamma=gammas, C=Cs)

from sklearn.model_selection import GridSearchCV
svc = SVC(kernel = 'rbf', random_state = 0)
grid_search = GridSearchCV(svc, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train)
```

```
y_pred = grid_search.predict(X_test)

print('Grid best parameter (max. accuracy): ', grid_search.best_params_)
```

```
Grid best parameter (max. accuracy):  {'C': 1, 'gamma': 0.0001}
```

Train score

In [127]:
```
grid_search.score(X_train, y_train)
```

Out[127]:
```
0.8666666666666667
```

Test score

In [128]:
```
grid_search.score(X_test, y_test)
```

Out[128]:
```
0.875
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [129]:
```
from sklearn.model_selection import cross_val_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: '+ str(cross_val_score(grid_search, X_train, y_train.ra
vel(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(grid_search, X_train, y_train.r
avel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(grid_search, X_train, y_train.rave
l(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(grid_search, X_train, y_train.ravel(),
 scoring='f1', cv=3)))
```

```
Accuracy: [0.85950413 0.85833333 0.88235294]
Precision: [0.86868687 0.93975904 0.91208791]
Recall: [0.95555556 0.86666667 0.93258427]
F1: [0.91005291 0.9017341  0.92222222]
```

Building a Confusion Metrics

In [130]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[130]:
```
array([[26, 10],
       [ 5, 79]], dtype=int64)
```

105 were predicted right, while 15 were predicted wrong. 101/120 = 0.85

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to

minimize FPR or to increse Precision

In [131]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```

```
False Positive Rate is: 0.11
```

AUC score

In [132]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

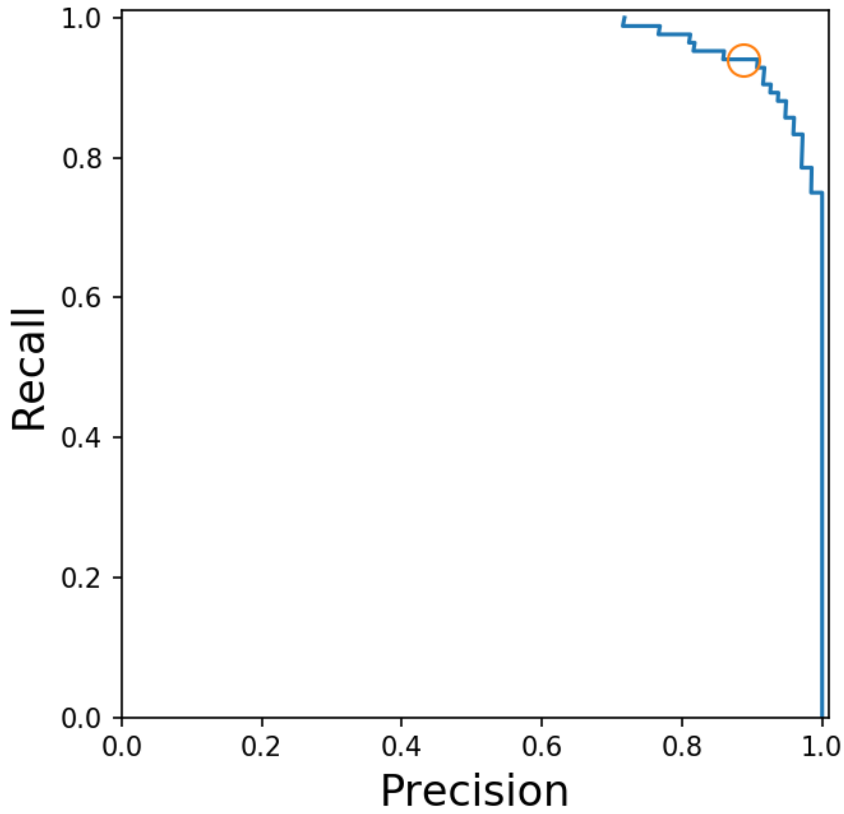
```
Area under the curve score: 0.83
```

Adding results to a table for summarization in the end

In [133]:
```
model_name.append("Forward/SVC")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

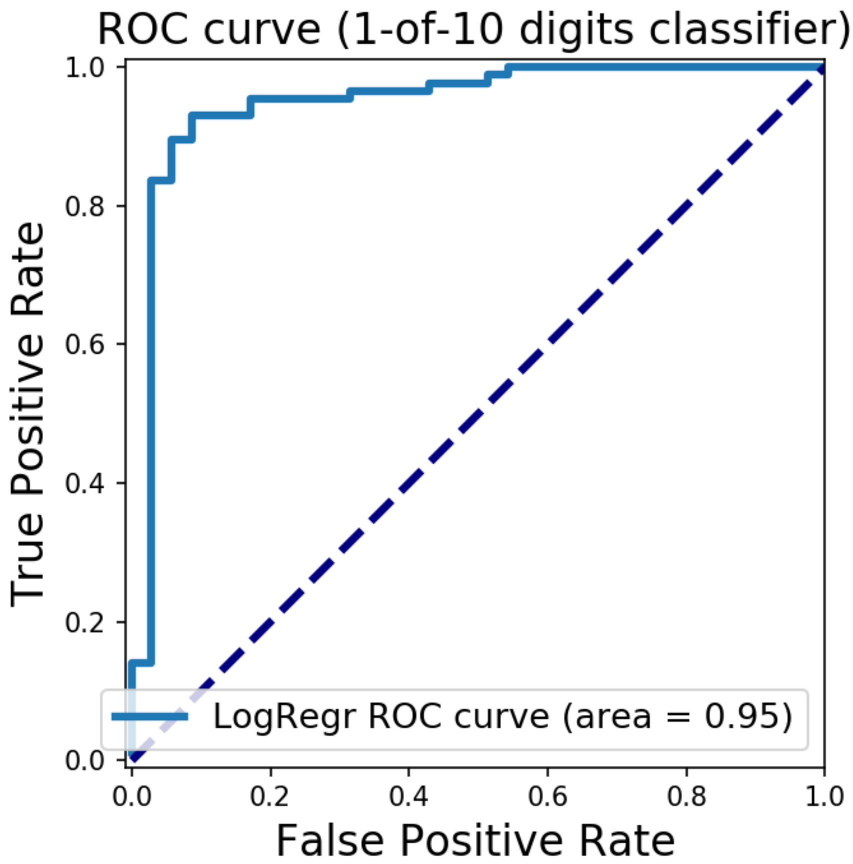
Building a precision-recall curve

In [134]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).decision_function(X_test
)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [135]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).decision_function(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Building a Decision Tree model using PramGrid (Forward Selection)

In [136]:
```
max_depth = np.linspace(1, 40, 40, endpoint=True)
min_samples_splits = np.linspace(0.1, 1.0, 10, endpoint=True)
min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint=True)
criterion = ['entropy', 'gini']

param_grid = dict(max_depth=max_depth,
                  min_samples_split = min_samples_splits,
                  min_samples_leaf = min_samples_leafs,
                  criterion=criterion)

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
grid_search = GridSearchCV(classifier, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train)

grid_search.predict(X_test)


print('Grid best parameter (max. accuracy): ', grid_search.best_params_)

y_pred = grid_search.predict(X_test)
```

```
Grid best parameter (max. accuracy):  {'criterion': 'gini', 'max_depth':
 1.0, 'min_samples_leaf': 0.30000000000000004, 'min_samples_split': 0.1}
```

Train score

In [137]:
```
grid_search.score(X_train, y_train)
```
Out[137]:
```
0.8444444444444444
```

Test score

In [138]:
```
grid_search.score(X_test, y_test)
```
Out[138]:
```
0.85
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [139]:
```
from sklearn.model_selection import cross_val_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: '+ str(cross_val_score(grid_search, X_train, y_train.ra
vel(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(grid_search, X_train, y_train.r
avel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(grid_search, X_train, y_train.rave
l(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(grid_search, X_train, y_train.ravel(),
 scoring='f1', cv=3)))
```
```
Accuracy: [0.84297521 0.85       0.80672269]
Precision: [0.89010989 0.90804598 0.94594595]
Recall: [0.9        0.88764045 0.78651685]
F1: [0.89502762 0.89772727 0.85889571]
```

Building a Confusion Metrics

In [140]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```
Out[140]:
```
array([[28,  7],
       [11, 74]], dtype=int64)
```

102 were predicted right, while 18 were predicted wrong. 105/120 = 0.85

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [141]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```
```
False Positive Rate is: 0.09
```

AUC score

In [142]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

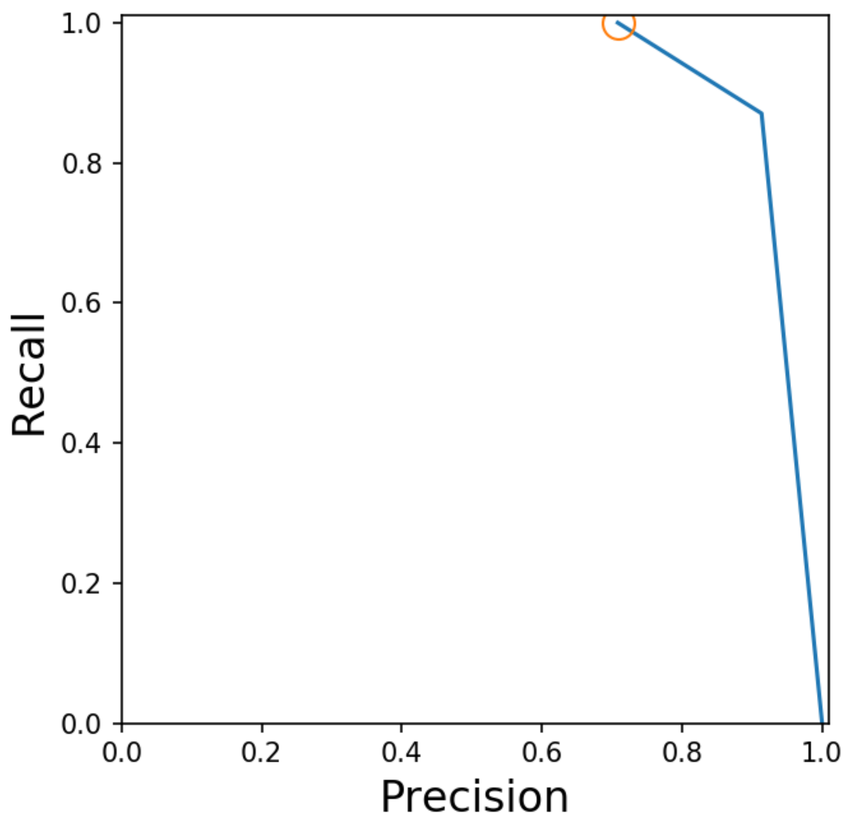
```
Area under the curve score: 0.84
```

Adding results to a table for summarization in the end

In [143]:
```
model_name.append("Forward/Decision Tree")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

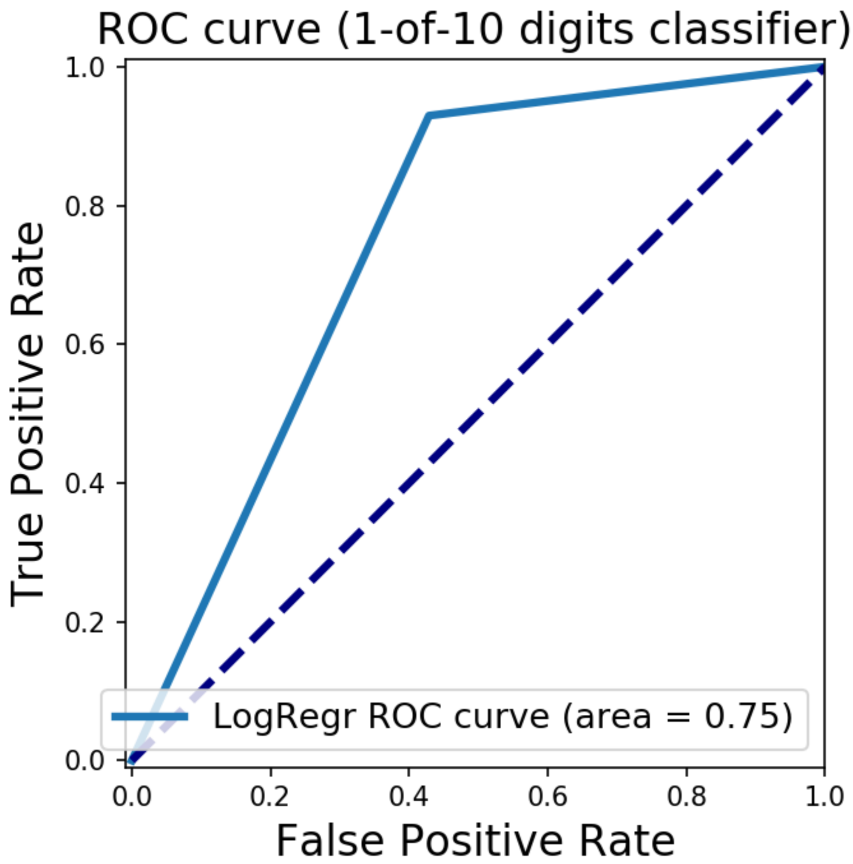
Building a precision-recall curve

In [144]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).predict(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [145]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).predict(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Building a Random Forest model using PramGrid (Forward Selection)

In [146]:
```
from sklearn.ensemble import RandomForestClassifier

max_depth = np.linspace(1, 40, 40, endpoint=True)
n_estimators = [5,10,15,20,30]
criterion = ['entropy', 'gini']

param_grid = dict(max_depth=max_depth,
                  n_estimators = n_estimators,
                 criterion=criterion)

#model
from sklearn.model_selection import GridSearchCV
forest = RandomForestClassifier()
grid_search = GridSearchCV(forest, param_grid)

#fit best combination of parameters
grid_search.fit(X_train, y_train.ravel())

y_pred = grid_search.predict(X_test)
```

Train score

In [147]:
```
grid_search.score(X_train, y_train)
```

Out[147]: 0.9888888888888889

Test score

In [148]:
```
grid_search.score(X_test, y_test)
```
Out[148]:
```
0.85
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [149]:
```
from sklearn.model_selection import cross_val_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: '+ str(cross_val_score(grid_search, X_train, y_train.ra
vel(), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(grid_search, X_train, y_train.r
avel(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(grid_search, X_train, y_train.rave
l(), scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(grid_search, X_train, y_train.ravel(),
 scoring='f1', cv=3)))
```
```
Accuracy: [0.85950413 0.83333333 0.86554622]
Precision: [0.93975904 0.90909091 0.87628866]
Recall: [0.86666667 0.91011236 0.94382022]
F1: [0.88095238 0.90607735 0.93406593]
```

Building a Confusion Metrics

In [150]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```
Out[150]:
```
array([[23, 12],
       [ 6, 79]], dtype=int64)
```

105 were predicted right, while 15 were predicted wrong. 105/120 = 0.875

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [151]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```
```
False Positive Rate is: 0.13
```

AUC score

In [152]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

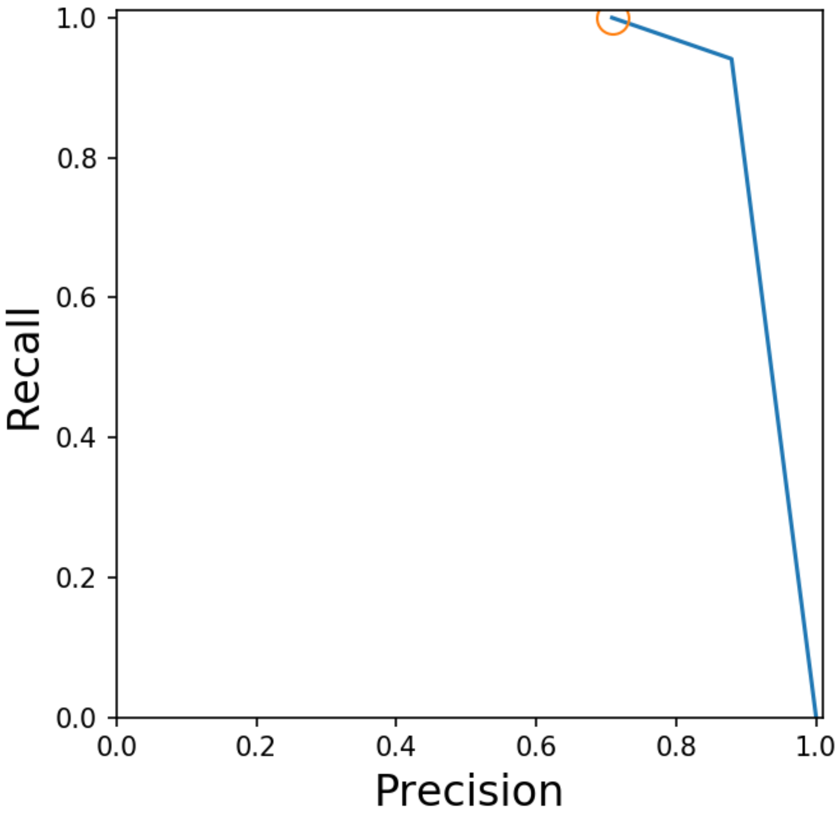
```
Area under the curve score: 0.79
```

Adding results to a table for summarization in the end

In [153]:
```
model_name.append("Forward/Random Forest")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

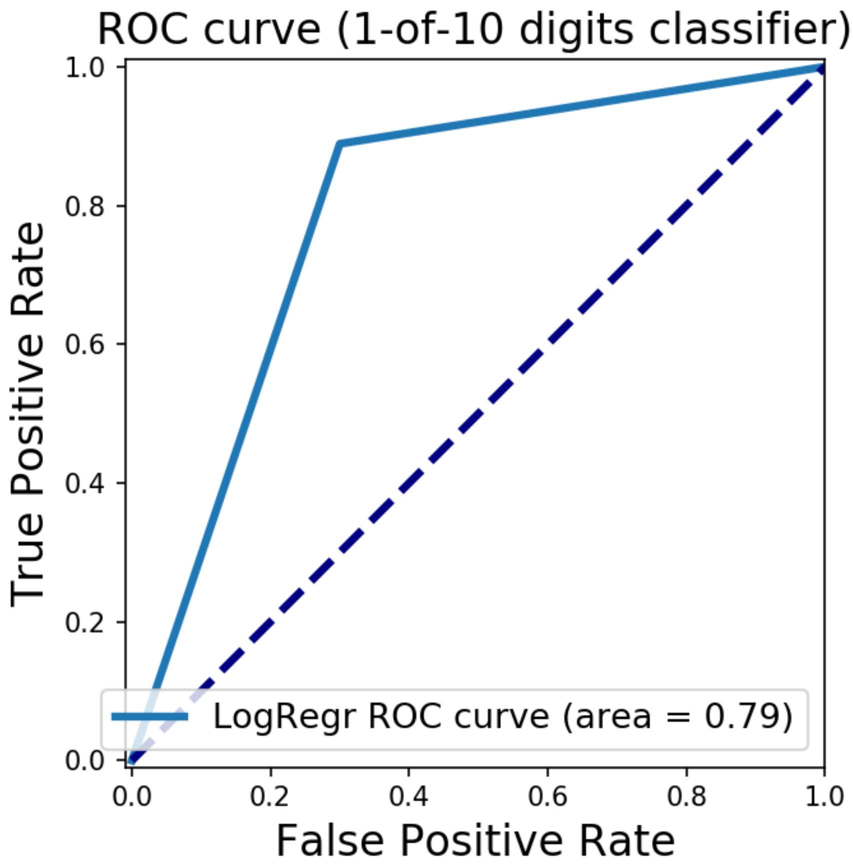
Building a precision-recall curve

In [154]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = grid_search.fit(X_train, y_train).predict(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [155]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y)

y_pred_lr = grid_search.fit(X_train, y_train).predict(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



A table to compare results

In [156]:
```
d = {'model_name': model_name, 'accuracy_col': accuracy_col, 'precision_
col': precision_col,
'recall_col': recall_col, 'f1_col': f1_col, 'auc_col': auc_col}
df = pd.DataFrame(data=d)
df
```

Out[156]:

	model_name	accuracy_col	precision_col	recall_col	f1_col	auc_col
0	Backward/KNN	0.908333	0.931818	0.942529	0.937143	0.880355
1	Backward/Bayes	0.883333	0.987805	0.861702	0.920455	0.911620
2	Backward/SVC	0.900000	0.902174	0.965116	0.932584	0.850205
3	Backward/Decision Tree	0.816667	0.914634	0.833333	0.872093	0.800000
4	Backward/Random Forest	0.866667	0.880435	0.941860	0.910112	0.809166
5	Forward/KNN	0.875000	0.891304	0.942529	0.916201	0.819749
6	Forward/Bayes	0.875000	1.000000	0.838710	0.912281	0.919355
7	Forward/SVC	0.875000	0.887640	0.940476	0.913295	0.831349
8	Forward/Decision Tree	0.850000	0.913580	0.870588	0.891566	0.835294
9	Forward/Random Forest	0.850000	0.868132	0.929412	0.897727	0.793277

Summary

Considering that in this case precision is more important, bayes model does the best job in achieving high precision score. Moreover, AUC with bayes is the highest. As of feature selection approach, Backward Elimination selected better set of features. The worst model in terms of precision rate is Random Forest

61 - 67

1.3. Extract information from Zomato API and from Zomato website with BeautifulSoup to Categorize restaurants in Ontario, Canada by prices and ratings (KMEans Clustering)

Extract information from Zomato API and from website with BeautifulSoup to Categorize restaurants in Ontario, Canada by prices and ratings (KMEans Clustering)

Information about the dataset

- Number of inputs: **1800**
- Number of variables: **2**
- Dataset: **Zomato API** - https://developers.zomato.com/api#headline1
- Goal: Categorize restaurants by price and rating: low price-high rating, middle price-high rating, low price-low rating etc.

Data extraction from API

Some information like API key was removed so the code is not going to work, it is just for a reference

Extraction of city names in Canada from a website using BeautifulSoup

In [ ]:
```
import requests
import re
url = "https://www.zomato.com/canada"
# in case you need a session
headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6
) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537
.36'}


r = requests.get(url, headers=headers)
# or without a session: r = requests.get(url)

html_doc = r.content

from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')

elements = soup.findAll('a', attrs={'style': 'flex-grow: 1;'})

cities = []
for el in elements:
    words = el.text
    end = re.search("Restauracje", words).start()
    city =  words[:int(end)-1]
    cities.append(city)
```

Usage of API to extract 100 (api limit) restaurants of cities in Canada. Cities in Ontario were separated manually after

```
In [ ]: import requests
        import time

        import json

        #get_cities_.py for cities

        import os

        #get cities ids
        cities_ids = []
        cities_names=[]
        for city in cities:
            headers = {
                "Accept": "application/json",
                "user-key": "api key of zomato",
            }

            params = (
                ("q", city),
                ("count", '1'),
            )

            response = requests.get("https://developers.zomato.com/api/v2.1/citi
        es", headers=headers, params=params)
            data = response.json()
            data_j = json.dumps(data)
            cities_ids.append(data["location_suggestions"][0]["id"])
            cities_names.append(data["location_suggestions"][0]["name"])



        #create an empty file with all cities in it
        fu= open("[folder with a file]","w+")

        for id in cities_ids:

            f= open("folder with a file/"+str(id)+".txt","w+")
            print("city id: "+str(id))



            headers = {
                'Accept': 'application/json',
                'user-key': 'api key of zomato',
            }

            start=0
            #overcome limit
            for i in range(5):
                print("start: "+str(start))
                params = (
                    ('entity_id', id),
                    ('entity_type', 'city'),
                    ('start', start),
                    ('count', '20'),
                 )

                response = requests.get('https://developers.zomato.com/api/v2.1/
        search', headers=headers, params=params)
                data = response.json()
```

```
        #extracting the whole data outputs an error of converting utf-8
        """
        try:
            fu.write(str(data))
            f.write(str(data))
            f.write("\n")
            fu.write("\n")
        except Exception:
            pass

        """
        for r in range(len(data['restaurants'])):
            try:
                fu.write(str(json.dumps(data['restaurants'][r])))
                f.write(str(json.dumps(data['restaurants'][r])))
                fu.write("\n")
                f.write("\n")
            except Exception:
                pass

        start+=20
    time.sleep(5)       #making a pause for 3 seconds every 5th iteratio
n

    f.close()
fu.close()
```

Importing main libraries

In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import json
import warnings; warnings.simplefilter('ignore')
```

Importing the dataset

Extracting only required data from json

In [6]:
```
prices=[]
ratings=[]
file = open('ontario_zomato.json', 'r')
for l in file:
    j_line = json.loads(l)
    res_id = j_line['restaurant']['R']['res_id']
    prices.append(j_line['restaurant']['average_cost_for_two'])
    ratings.append(j_line['restaurant']['user_rating']['aggregate_rating
'])
file.close()
```

Putting extracted data in a dataframe

In [7]:
```
d = {'prices': prices, 'ratings': ratings}
df = pd.DataFrame(data=d)
df= df.convert_objects(convert_numeric=True) #conversion is required for
```

```
second variable
```

Remove restaurants with rating 0

In [8]:
```
df.drop(df[df.iloc[:, 1] <= 0].index, inplace=True)
```

In [9]:
```
print("Number of inputs without rating = 0: "+str(len(df)))
```

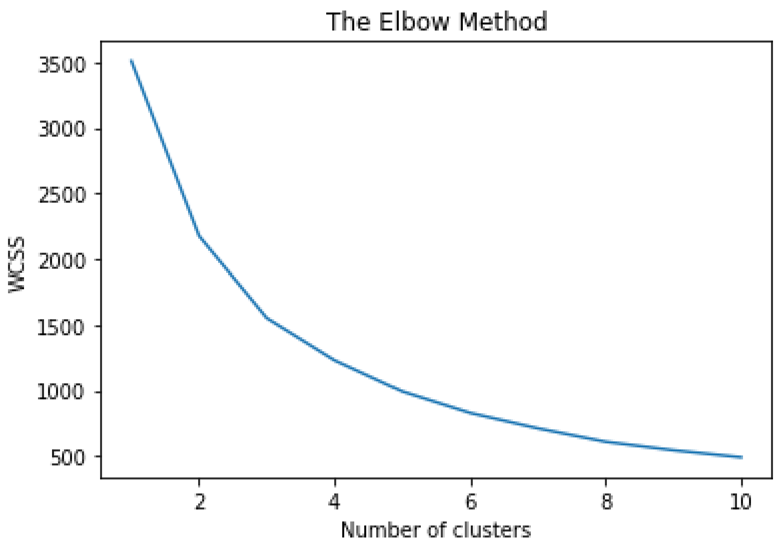
```
Number of inputs without rating = 0: 1754
```

Feature Scaling

In [10]:
```
from sklearn.preprocessing import StandardScaler
X = df.iloc[:, [0, 1]].values
sc_X = StandardScaler()
X = sc_X.fit_transform(X)
```

Using the elbow method to find the optimal number of clusters

In [11]:
```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 4
2)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



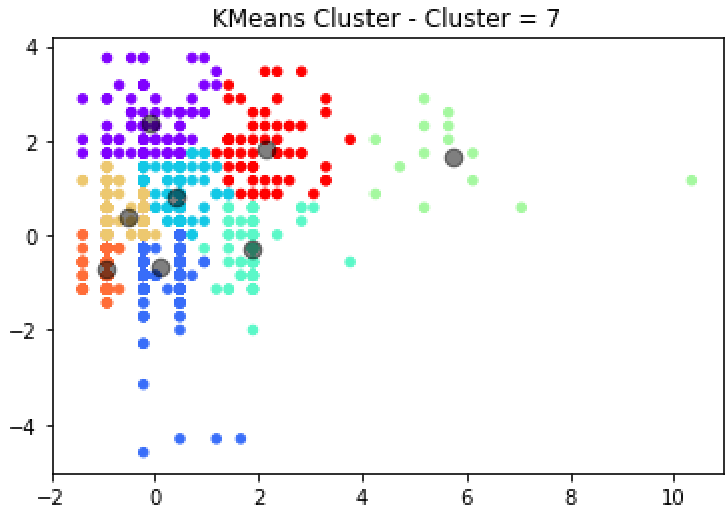
The optimal number of clusters is around 6. However, in this particaular situation depends on the number of possible clusters:

```
- low price/low rating
- low price/middle rating
- low price/high rating
- middle price/low rating
```

```
    - middle price/middle rating
    - middle price/high rating
    - high price/low rating
    - high price/high rating
```

Lets assume that optimal number of clusters is 8

In [12]:
```
model = KMeans(n_clusters=8)
model.fit(X)
y_test = model.labels_
centers = model.cluster_centers_ #centers of each clusters are extracted
 to plot them later

plt.scatter(X[:, 0], X[:, 1], c = y_test, cmap='rainbow' , s = 20)
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=75, alpha=0.5);

#Compare Target vs Cluster
titl=str(7)
plt.title('KMeans Cluster - Cluster = '+ titl) #name of the graph
plt.show()
print("Model strength:", model.inertia_) #prints "objective function" of
 each graph
```

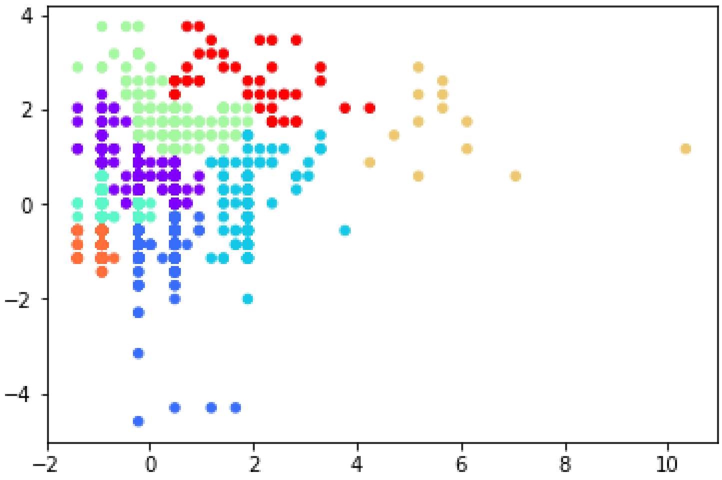



```
Model strength: 621.5795690308071
```

Agglomerative clustering

In [13]:
```
from sklearn.cluster import AgglomerativeClustering
cls = AgglomerativeClustering(n_clusters = 8)
cls_assignment = cls.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c = cls_assignment, cmap='rainbow' , s = 2
0)
```

Out[13]:
```
<matplotlib.collections.PathCollection at 0x1970e859a20>
```

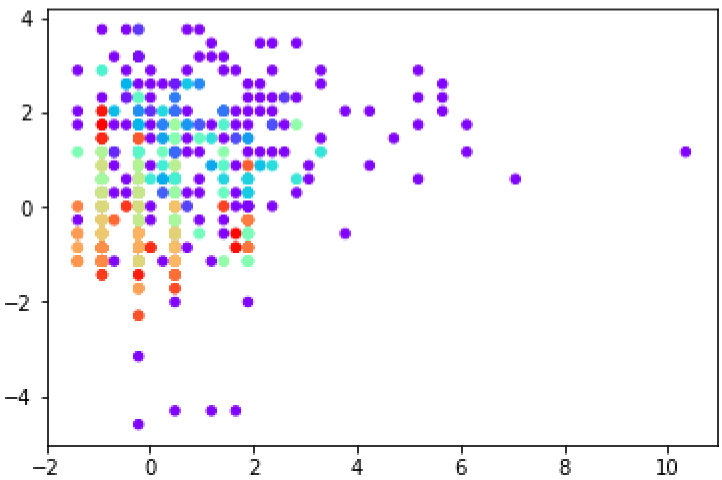


DBSCAN clustering

In [14]:
```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs

dbscan = DBSCAN(eps = 0.2, min_samples = 2)

cls = dbscan.fit_predict(X)
print("Cluster membership values:\n{}".format(cls))


plt.scatter(X[:, 0], X[:, 1], c = cls + 1, cmap='rainbow' , s = 20)
```

```
Cluster membership values:
[-1 -1 -1 ... 76 88 84]
```

Out[14]: <matplotlib.collections.PathCollection at 0x1970ea19d30>



Summary

From my own perspective, the first approach, **K-means clustering**, is the best in this case because clusters are groupped more consistent and shapes of clusters can be considerd as having low, middle, high price. While the second and the third approaches create clusters that are more chaotic and unstable

68 - 90

1.4. Predict whether student passes math and Portuguese course in school or not (Logistic Regression)

Predict whether student passes math and portuguese course in school or not (Logistic Regression)

Information about the dataset

- Number of inputs: **649**
- Number of variables: **30**
- Dataset: https://archive.ics.uci.edu/ml/datasets/student+performance
- Data fields description: https://archive.ics.uci.edu/ml/datasets/student+performance

Importing main libraries

In [41]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import warnings; warnings.simplefilter('ignore')
```

Importing the dataset

In [42]:
```
data_por = pd.read_csv('student-por.csv')
data_math = pd.read_csv('student-mat.csv')
```

Creates a variable that shows that students passed or did not pass a course (grade>=10 == passed)

In [43]:
```
#create a variable 'pass'
data_por["Pass"] = [1 if ele >=10 else 0 for ele in data_por["G3"]]
data_math["Pass"] = [1 if ele >=10 else 0 for ele in data_math["G3"]]
```

y - a variable to predict; creating a list of columns indexes that are categorical, not yet encoded into number, for Label Encoding

In [44]:
```
y_por = data_por.iloc[:, -1].values
y_mat = data_math.iloc[:, -1].values
```

Creating a list of categorical variables and encoding them with LabelEncoder

In [45]:
```
#encoding categorical variables
categorical_var = [0,1,3,4,5,8,9,10,11,15,16,17,18,19,20,21,22]

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder = LabelEncoder()
for i in categorical_var:
    data_por.iloc[:, i] = labelencoder.fit_transform(data_por.iloc[:, i]
)
```

```
for i in categorical_var:
    data_math.iloc[:, i] = labelencoder.fit_transform(data_math.iloc[:,
i])
```

Creating a reference dictionary to find corresponding variables after OneHotEncoding in the initial dataframe

This dictionary can be used to find corresponding variables that were chosen by "Forward Selection" and "Backward Elimination" further below

In [46]:
```
ref_dict_por = {}
dict_iter_por = 0

ref_dict_mat = {}
dict_iter_mat = 0
```

Encoding categorical variables with OneHotEncoder

The first categorical column will be encoded, result will be added separately in a ndarray, ecluding first dummy column. All other categorical columns will be encoded and added to this ndarray afterwards via loop. That allows to use OneHotEncoder on range of categorical variables without manually encoding one variable after another

In [47]:
```
col_list = [0,1,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
24,25,26,27,28]
no_cat_var = [2,29]

df_cat_por = data_por.iloc[:, col_list]                              #df wi
th categorical variables
df_cat_mat = data_math.iloc[:, col_list]
```

Portuguese

In [48]:
```
X_cat = df_cat_por.iloc[:, :].values                          #categorical
 ndarray
X_cat[:, 0] = labelencoder.fit_transform(X_cat[:, 0])
onehotencoder = OneHotEncoder(categorical_features = [0])       #encodin
g 1st column
X_cc = onehotencoder.fit_transform(X_cat).toarray()
dummy_col = df_cat_por.iloc[:, 0].nunique()                   #finding out
 number of dummy colummns created

X_cc_2_por = X_cc[:, 1:dummy_col]                                     #mov
ing to a separate ndarray excluding first dummy column

df_cat_no_one = df_cat_por.iloc[:, 1:]                        #first colum
n was preprocessed so it was excuded from further loop
X_cat_no_one = df_cat_no_one.iloc[:, :].values

ref_dict_por[0] = list(range(dict_iter_por, dict_iter_por+dummy_col))
  #adding id of original column as key, all corresponding dummy columns
as list
dict_iter_por = dict_iter_por + dummy_col
```

Now the first column was encoded in dummy variables and they were added to separate ndarray.
Other encoded variables will be added to this ndarray via loop below

Adding other categorical variables to ndarray via loop

In [49]:
```
dict_iter=0
for c in range(len(col_list)-1):
    X_cat_no_one[:, c] = labelencoder.fit_transform(X_cat_no_one[:, c])

    onehotencoder = OneHotEncoder(categorical_features = [c])
    X_cc = onehotencoder.fit_transform(X_cat_no_one).toarray()

    dummy_col = df_cat_por.iloc[:, c+1].nunique()
      #+1 because of reffering to df with all categorical variables, inc
luding the first one
    X_cc2_2 = X_cc[:, 1:dummy_col]
          #excluding first dummy column
    X_cc_2_por = np.concatenate((X_cc_2_por, X_cc2_2), axis=1)
                  #merge 2 ndarrays
    ref_dict_por[c+1] = list(range(dict_iter, dict_iter+dummy_col))
              #adding id of original column as key, all corresponding du
mmy columns as list
    dict_iter_por = dict_iter_por + dummy_col
```

After that all continious variables are added to a ndarray with encoded categorical variables

In [50]:
```
df_non_categorical = data_por.iloc[:, no_cat_var]
X_no_cat_var = df_non_categorical.iloc[:, :].values
merged_dataset_por = np.concatenate((X_cc_2_por, X_no_cat_var), axis=1)
```

Math

In [51]:
```
X_cat = df_cat_mat.iloc[:, :].values                            #categorical
 ndarray
X_cat[:, 0] = labelencoder.fit_transform(X_cat[:, 0])
onehotencoder = OneHotEncoder(categorical_features = [0])       #encodin
g 1st column
X_cc = onehotencoder.fit_transform(X_cat).toarray()
dummy_col = df_cat_mat.iloc[:, 0].nunique()                     #finding out
 number of dummy colummns created

X_cc_2_mat = X_cc[:, 1:dummy_col]                                   #mov
ing to a separate ndarray excluding first dummy column

df_cat_no_one = df_cat_mat.iloc[:, 1:]                          #first colum
n was preprocessed so it was excuded from further loop
X_cat_no_one = df_cat_no_one.iloc[:, :].values

ref_dict_mat[0] = list(range(dict_iter_mat, dict_iter_mat+dummy_col))
  #adding id of original column as key, all corresponding dummy columns
as list
dict_iter_mat = dict_iter_mat + dummy_col
```

Now the first column was encoded in dummy variables and they were added to separate

ndarray.
Other encoded variables will be added to this ndarray via loop below

Adding other categorical variables to ndarray via loop

In [52]:
```
dict_iter=0
for c in range(len(col_list)-1):
    X_cat_no_one[:, c] = labelencoder.fit_transform(X_cat_no_one[:, c])

    onehotencoder = OneHotEncoder(categorical_features = [c])
    X_cc = onehotencoder.fit_transform(X_cat_no_one).toarray()

    dummy_col = df_cat_mat.iloc[:, c+1].nunique()
      #+1 because of reffering to df with all categorical variables, inc
luding the first one
    X_cc2_2 = X_cc[:, 1:dummy_col]
          #excluding first dummy column
    X_cc_2_mat = np.concatenate((X_cc_2_mat, X_cc2_2), axis=1)
              #merge 2 ndarrays
    ref_dict_mat[c+1] = list(range(dict_iter, dict_iter+dummy_col))
                #adding id of original column as key, all corresponding du
mmy columns as list
    dict_iter_mat = dict_iter_mat + dummy_col
```

After that all continious variables are added to a ndarray with encoded categorical variables

In [53]:
```
df_non_categorical = data_math.iloc[:, no_cat_var]
X_no_cat_var = df_non_categorical.iloc[:, :].values
merged_dataset_mat = np.concatenate((X_cc_2_mat, X_no_cat_var), axis=1)
```

Final dataset to work with

69 columns

Creating model to predicting passing a Portuguese course

Selecting columns to work with

At this point there are 69 columns to choose from for a machine learning model. A subjective selection is not appropriate so two approaches will be used to select a required range of variables for machine learning algorithm. These approaches are "Backward Elimination" and "Forward selection": https://en.wikipedia.org/wiki/Stepwise_regression

Lets start with Backward Elimination:

In [54]:
```
import statsmodels.formula.api as sm

p=0.05

#imputs for def are: dataset and p-value
```

```
def BackwardElimination(merged_dataset, y, p):
    #merged_dataset = np.append(arr = np.ones((np.size(merged_dataset,0)
,1)).astype(int), values=merged_dataset, axis=1) #np.size(merged_categ,0
) - number of rows in numpy array
    #this adds our dataset to a column of one so ones are in the first c
olumn (for linear regression)

    #number of columns
    len_list = []                                    #list of indexes of al
l columns
    for i in range(np.size(merged_dataset,1)+1):
        len_list.append(i)


    p = p #p-value for; can be adjusted depending on desired result (def
ault - 0.05)

    end = False
    while end==False:
        regressor_OLS = sm.OLS(endog = y, exog = merged_dataset).fit()
        p_values = regressor_OLS.pvalues
        #enable these prints to see a process of selection in a real tim
e
        #print("P values are: "+str(['%.3f' % i for i in p_values.tolist
()]))
        #print("Max p value: "+str(max(p_values)))
        #print("==============================================")
        if max(p_values)<p:
            end = True
            return merged_dataset
        elif max(p_values)>=p:
            p_max_pos = p_values.tolist().index(max(p_values))
            merged_dataset = np.delete(merged_dataset, [p_max_pos], axis
=1)

X = BackwardElimination(merged_dataset_por, y_por, p)
```

Building a Logisitc Regression model with GridSearch (Backward Elimination)

In [55]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y_por)

from sklearn.linear_model import LogisticRegression

C = [0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1, 5, 10]
penalty=['l1','l2']
param_grid = dict(C=C, penalty=penalty)

#model
from sklearn.model_selection import GridSearchCV
classifier = LogisticRegression()
log_reg = GridSearchCV(classifier, param_grid)

#fit best combination of parameters
log_reg.fit(X_train, y_train) #ravel is needed to convert int to float
```

```
y_pred = log_reg.predict(X_test)
```

In [56]:
```
print('Grid best parameter (max. accuracy): ', log_reg.best_params_)
```

```
Grid best parameter (max. accuracy):  {'C': 1, 'penalty': 'l2'}
```

Train score

In [57]:
```
log_reg.score(X_train, y_train)
```

Out[57]:
```
0.8765432098765432
```

Test score

In [58]:
```
log_reg.score(X_test, y_test)
```

Out[58]:
```
0.852760736196319
```

Evaluating a model

Building a Confusion Metrics

In [59]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[59]:
```
array([[  7,  20],
       [  4, 132]], dtype=int64)
```

144 were predicted right, while 19 were predicted wrong. 144/163 = 0.88

In [60]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_scor
e, f1_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
from sklearn.model_selection import cross_val_score
print('Accuracy: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(
), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(log_reg, X_train, y_train.ravel
(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(),
 scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(), sco
ring='f1', cv=3)))
```

```
Accuracy: [0.88343558 0.87654321 0.85093168]
Precision: [0.89932886 0.87820513 0.87417219]
Recall: [0.97101449 0.99275362 0.96350365]
F1: [0.93379791 0.93197279 0.91666667]
```

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [61]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test, y_pred)))
```

```
False Positive Rate is: 0.13
```

AUC score

In [62]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test, y_pred)))
```

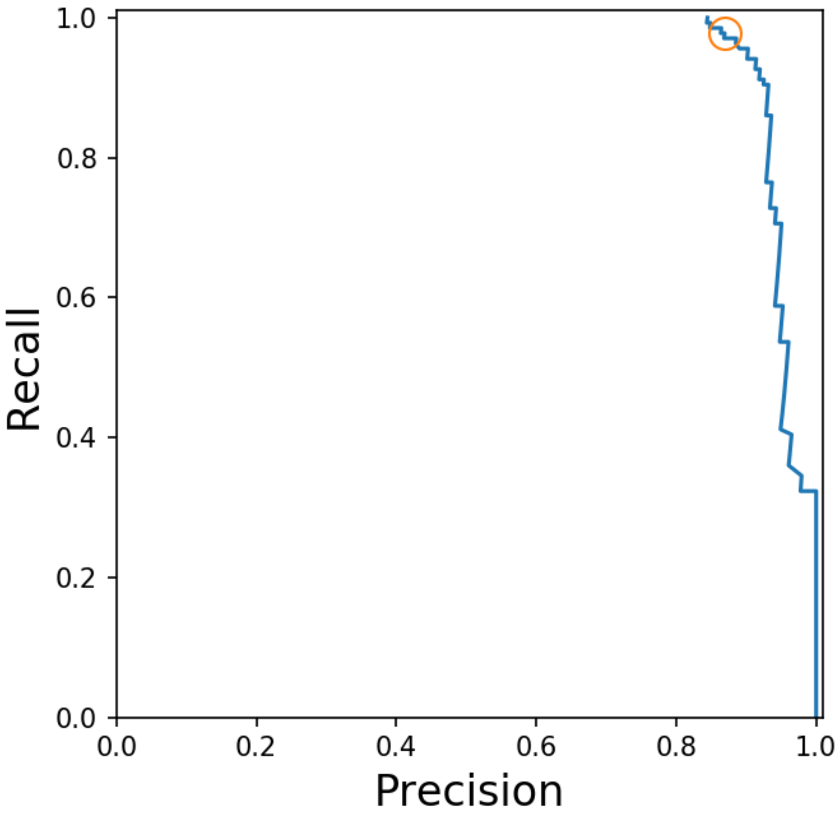
```
Area under the curve score: 0.61
```

Adding results to a table for summarization in the end

In [63]:
```
model_name=[]
accuracy_col=[]
precision_col=[]
recall_col=[]
f1_col=[]
auc_col=[]

model_name.append("Backward/Por")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

Building a precision-recall curve

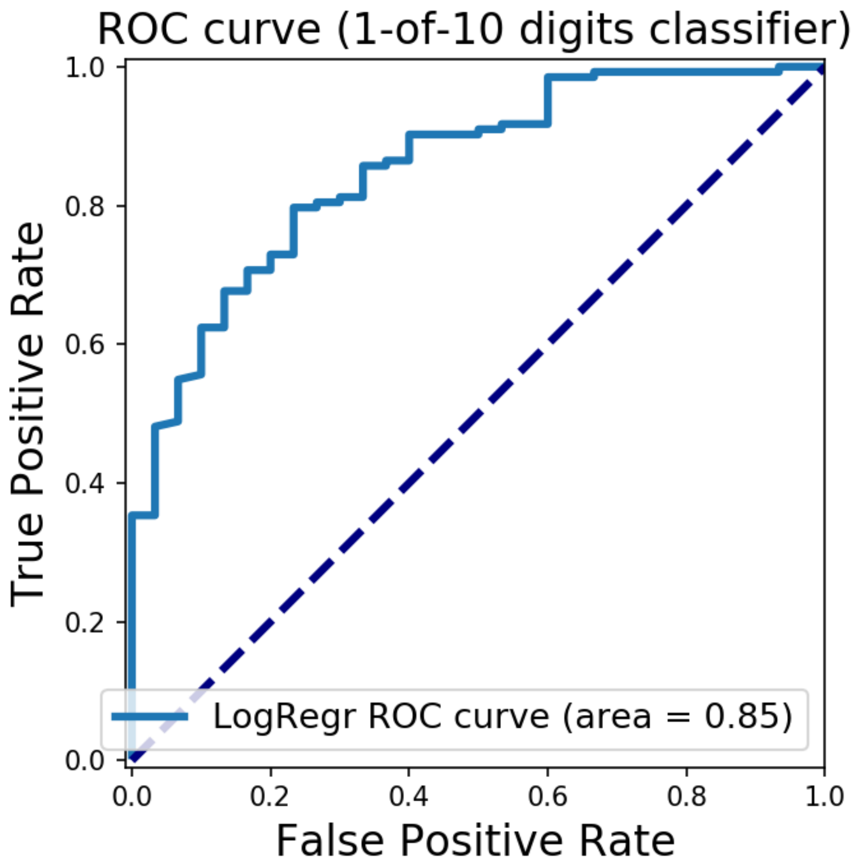
In [64]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = log_reg.fit(X_train, y_train).decision_function(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Precision rises as recall falls

Building a ROC curve

In [65]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y_por)

y_pred_lr = log_reg.fit(X_train, y_train).decision_function(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Lets apply Forward Selection:

In [66]:
```
import statsmodels.formula.api as sm

def ForwardSelection(merged_dataset, y, p):
    unknown_variables = []                          #a list of variables that ar
e not included as "good" ones; after each iteration some variable dissap
ears from "unknown" and becomes "good"
    for i in range(merged_dataset.shape[1]):
        unknown_variables.append(i)

    #adding b0 variable from formula
    #this adds our dataset to a column of one so ones are in the first c
olumn (for linear regression)
    #merged_dataset = np.append(arr = np.ones((np.size(merged_dataset,0)
,1)).astype(int), values=merged_dataset, axis=1) #np.size(merged_categ,0
) - number of rows in numpy array
    p = p

    ###first iteration is added separately, others in a loop below
    p_values_list=[]
    good_variables=[]
    for i in range(merged_dataset.shape[1]):
        X_opt = merged_dataset[:, i]
        regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()   #finding
 p value of every variable and y(the variable to predict)
        p_value = regressor_OLS.pvalues
        p_values_list.extend(p_value.tolist())
    min_p_value = min(p_values_list)                            #finding
 the minimum p value
    min_index = p_values_list.index(min_p_value)                #variabl
e with the smallest p value
```

```
    good_variables.append(min_index)                                         #add a v
ariable to a "good" list
    unknown_variables.remove(min_index)                                      #remove
index from a list of "bad" variables

    end=False
    while end==False:
        comb_list = []
        p_values_list=[]

        #this loop exists to make combinations of "good" variables with
every "unknown" to find p value of every combination
        for i in unknown_variables:
            temp_list = []
            for t in good_variables:
                temp_list.append(t)
            temp_list.append(i)
            comb_list.append([temp_list])
            #print(temp_list)
        for el in comb_list:
            X_opt = merged_dataset[:, el[0]]
            regressor_OLS = sm.OLS(endog = y, exog = X_opt).fit()
            p_value = regressor_OLS.pvalues
            pvalue_lst = p_value.tolist()
            p_values_list.append(pvalue_lst[-1])
        #finding combination with min p value
        min_p_value = min(p_values_list)
        min_index = p_values_list.index(min_p_value)
        good_variables.append(comb_list[min_index][-1][-1])
        unknown_variables.remove(comb_list[min_index][-1][-1])
        #uncomment to see every step
        #print("Min p value: "+str(min_p_value))
        #print("List of variables: "+str(good_variables))
        #print("####################################")
        if min_p_value>p:
             end=True

    #print("UN: "+str(unknown_variables))
    print("GN: "+str(good_variables))
    return merged_dataset[:, good_variables]
```

In [67]:
```
p = 0.05
X = ForwardSelection(merged_dataset_por, y_por, p)
```

```
GN: [67, 40, 0, 32, 34, 33, 45, 24, 9, 26, 68]
```

Building a Logisitc Regression model with GridSearch (Forward Selection)

In [68]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y_por)

from sklearn.linear_model import LogisticRegression

C = [0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1, 5, 10]
penalty=['l1','l2']
param_grid = dict(C=C, penalty=penalty)
```

In [69]:
```
#model
from sklearn.model_selection import GridSearchCV
classifier = LogisticRegression()
log_reg = GridSearchCV(classifier, param_grid)

#fit best combination of parameters
log_reg.fit(X_train, y_train) #ravel is needed to convert int to float

y_pred = log_reg.predict(X_test)

print('Grid best parameter (max. accuracy): ', log_reg.best_params_)
```

```
Grid best parameter (max. accuracy):  {'C': 0.9, 'penalty': 'l1'}
```

Train score

In [70]:
```
log_reg.score(X_train, y_train)
```

Out[70]:
```
0.8868312757201646
```

Test score

In [71]:
```
log_reg.score(X_test, y_test)
```

Out[71]:
```
0.8343558282208589
```

Evaluating a model

Building a Confusion Metrics

In [72]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[72]:
```
array([[  9,  23],
       [  4, 127]], dtype=int64)
```

136 were predicted right, while 27 were predicted wrong. 136/163 = 0.83

In [73]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_scor
e, f1_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Ra
te
# F1 = 2 * Precision * Recall / (Precision + Recall)
from sklearn.model_selection import cross_val_score
print('Accuracy: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(
), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(log_reg, X_train, y_train.ravel
(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(),
 scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(), sco
ring='f1', cv=3)))
```

```
Accuracy: [0.87116564 0.83333333 0.86335404]
```

```
Precision: [0.87421384 0.8943662  0.86335404]
Recall: [0.99285714 0.91366906 1.        ]
F1: [0.92976589 0.90391459 0.92666667]
```

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [74]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test, y_pred)))
```

```
False Positive Rate is: 0.15
```

AUC score

In [75]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test, y_pred)))
```

```
Area under the curve score: 0.63
```

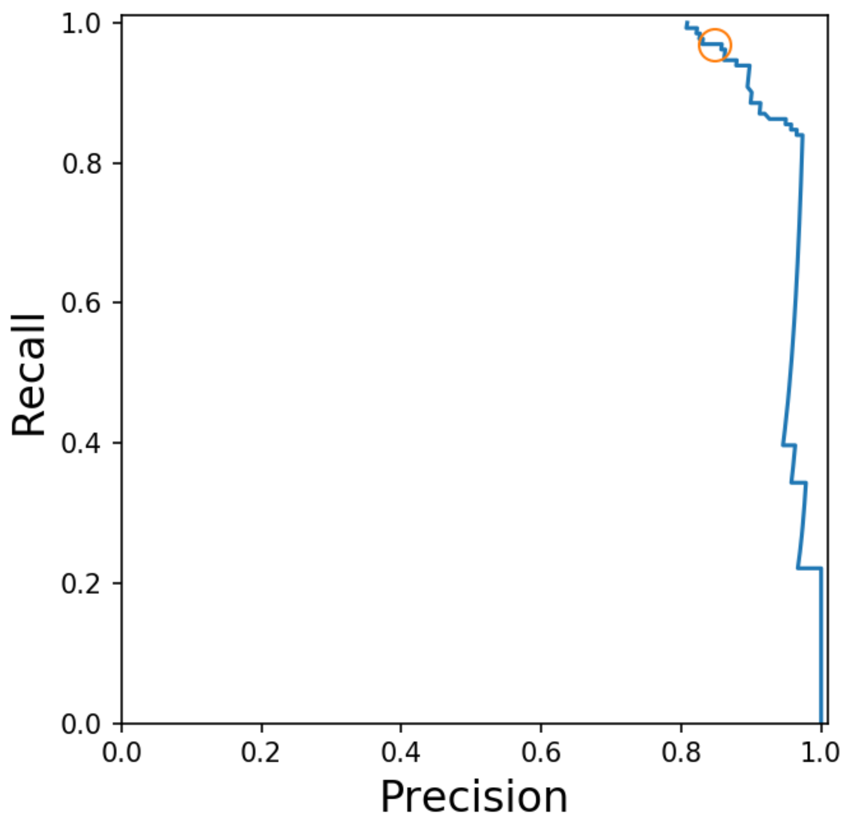
Adding results to a table for summarization in the end

In [76]:
```
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

In [77]:
```
model_name.append("Forward/Por")
```

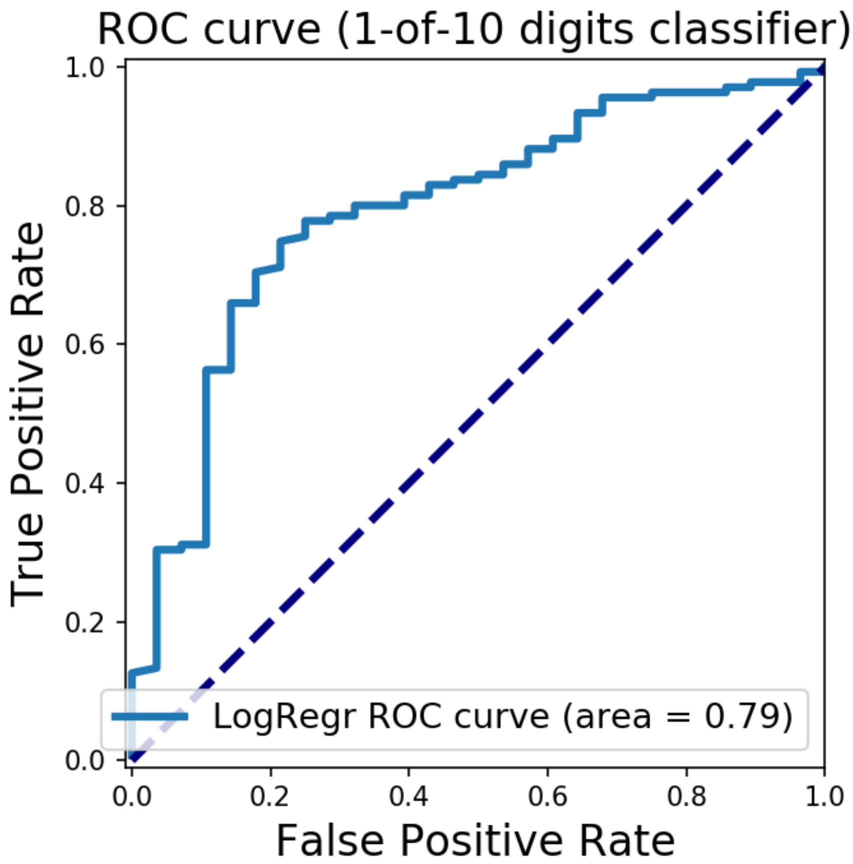
Building a precision-recall curve

In [78]:
```
from sklearn.metrics import precision_recall_curve

y_scores_lr = log_reg.fit(X_train, y_train).decision_function(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



Building a ROC curve

In [79]:
```
from sklearn.metrics import roc_curve, auc

X_train, X_test, y_train, y_test = train_test_split(X, y_por)

y_pred_lr = log_reg.fit(X_train, y_train).decision_function(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Creating model to predicting passing a Math course

Lets start with Backward Elimination

In [80]:
```
X = BackwardElimination(merged_dataset_mat, y_mat, p)
```

Building a Logisitc Regression model with GridSearch (Backward Elimination, Math)

In [81]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
X_train, X_test, y_train, y_test = train_test_split(X, y_mat)
from sklearn.linear_model import LogisticRegression
C = [0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1, 5, 10]
penalty=['l1','l2']
param_grid = dict(C=C, penalty=penalty)
#model
from sklearn.model_selection import GridSearchCV
classifier = LogisticRegression()
log_reg = GridSearchCV(classifier, param_grid)
#fit best combination of parameters
log_reg.fit(X_train, y_train) #ravel is needed to convert int to float
y_pred = log_reg.predict(X_test)
print('Grid best parameter (max. accuracy): ', log_reg.best_params_)
```

```
Grid best parameter (max. accuracy):  {'C': 5, 'penalty': 'l2'}
```

Train score

In [82]:
```python
log_reg.score(X_train, y_train)
```

Out[82]:
```
0.7601351351351351
```

Test score

In [83]:
```python
log_reg.score(X_test, y_test)
```

Out[83]:
```
0.6868686868686869
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [84]:
```python
from sklearn.model_selection import cross_val_score
print('Accuracy: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(
), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(log_reg, X_train, y_train.ravel
(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(),
 scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(), sco
ring='f1', cv=3)))
```

```
Accuracy: [0.7        0.70408163 0.65306122]
Precision: [0.70103093 0.74074074 0.71794872]
Recall: [0.98550725 0.88235294 0.82352941]
F1: [0.81927711 0.80536913 0.76712329]
```

Building a Confusion Metrics

In [85]:
```python
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[85]:
```
array([[15, 24],
       [ 7, 53]], dtype=int64)
```

71 were predicted right, while 28 were predicted wrong. 71/99 = 0.71

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [86]:
```python
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```

```
False Positive Rate is: 0.31
```

AUC score

In [87]:
```python
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

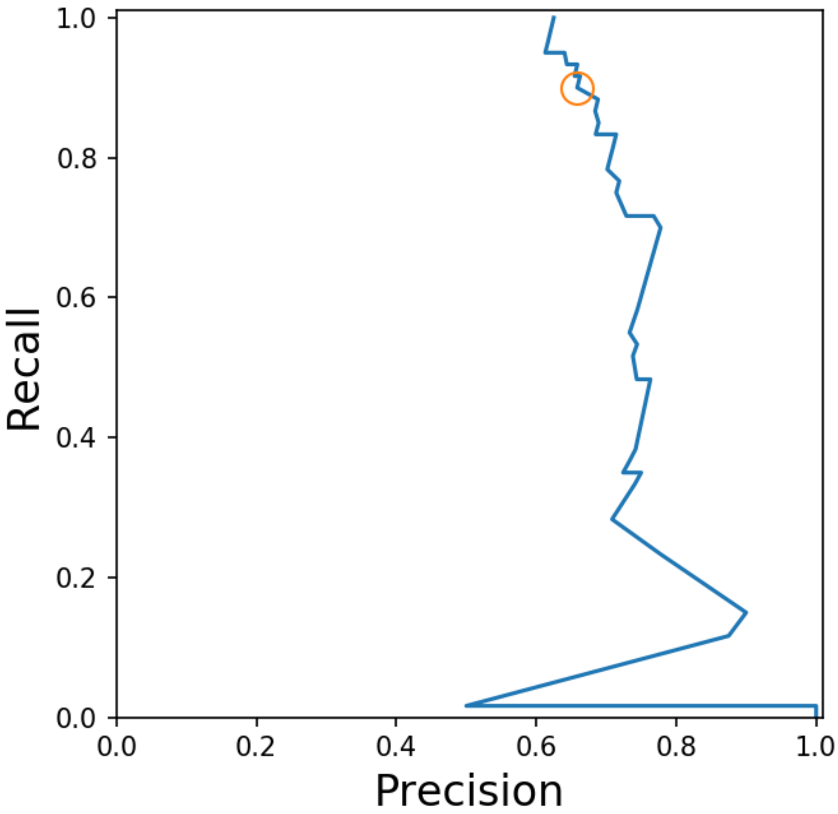
```
Area under the curve score: 0.63
```

Adding results to a table for summarization in the end

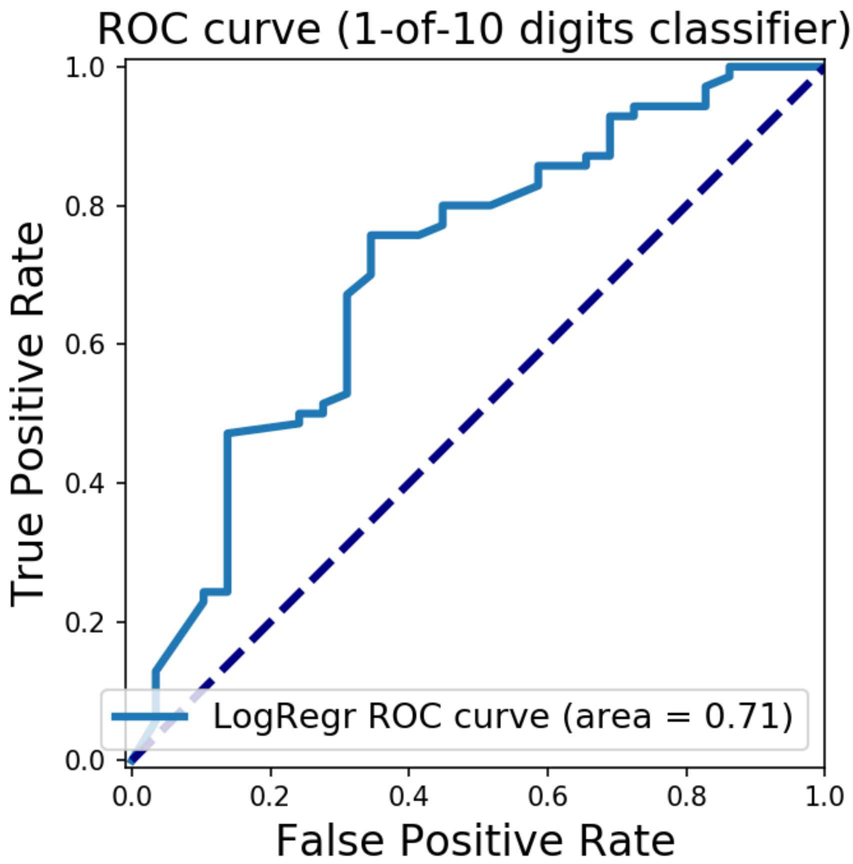
In [88]:
```
model_name.append("Backward/Mat")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

Building a precision-recall curve

In [89]:
```
from sklearn.metrics import precision_recall_curve
y_scores_lr = log_reg.fit(X_train, y_train).decision_function(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



In [180]:
```
from sklearn.metrics import roc_curve, auc
X_train, X_test, y_train, y_test = train_test_split(X, y_mat)
y_pred_lr = log_reg.fit(X_train, y_train).decision_function(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)
plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Lets apply Forward Selection:

In [90]:
```
p = 0.05
X = ForwardSelection(merged_dataset_mat, y_mat, p)
```

```
GN: [40, 33, 51, 1, 32, 52, 15, 34, 67, 13]
```

Building a Logisitc Regression model with GridSearch (Forward Selection) (Math)

In [91]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
X_train, X_test, y_train, y_test = train_test_split(X, y_mat)
from sklearn.linear_model import LogisticRegression
C = [0.001, 0.01, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1, 5, 10]
penalty=['l1','l2']
param_grid = dict(C=C, penalty=penalty)
#model
from sklearn.model_selection import GridSearchCV
classifier = LogisticRegression()
log_reg = GridSearchCV(classifier, param_grid)
#fit best combination of parameters
log_reg.fit(X_train, y_train) #ravel is needed to convert int to float
y_pred = log_reg.predict(X_test)
print('Grid best parameter (max. accuracy): ', log_reg.best_params_)
```

```
Grid best parameter (max. accuracy):  {'C': 5, 'penalty': 'l1'}
```

Train score

In [92]:
```
log_reg.score(X_train, y_train)
```
Out[92]:
```
0.7432432432432432
```

Test score

In [93]:
```
log_reg.score(X_test, y_test)
```
Out[93]:
```
0.7171717171717171
```

Evaluating a model

Cross validation: precision, accuracy, recall and f1

In [94]:
```
from sklearn.model_selection import cross_val_score
print('Accuracy: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(
), scoring='accuracy', cv=3)))
print('Precision: '+ str(cross_val_score(log_reg, X_train, y_train.ravel
(), scoring='precision', cv=3)))
print('Recall: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(),
 scoring='recall', cv=3)))
print('F1: '+ str(cross_val_score(log_reg, X_train, y_train.ravel(), sco
ring='f1', cv=3)))
```
```
Accuracy: [0.68       0.74489796 0.70408163]
Precision: [0.6875     0.7625     0.72619048]
Recall: [0.97058824 0.91044776 0.91044776]
F1: [0.80487805 0.82993197 0.80794702]
```

Building a Confusion Metrics

In [95]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```
Out[95]:
```
array([[11, 25],
       [ 3, 60]], dtype=int64)
```

76 were predicted right, while 23 were predicted wrong. 76/99 = 0.76

For this particular situation the False Positive Rate is the most important, because it is not an issue that model predicted a failure for a student while he or she passed, but it is an issue if model predicted a "pass" when student will eventually fail. It is recommended to minimize FPR or to increse Precision

In [96]:
```
print("False Positive Rate is: {:.2f}".format(1-precision_score(y_test,
y_pred)))
```
```
False Positive Rate is: 0.29
```

AUC score

In [97]:
```
from sklearn.metrics import roc_auc_score
print("Area under the curve score: {:.2f}".format(roc_auc_score(y_test,
y_pred)))
```

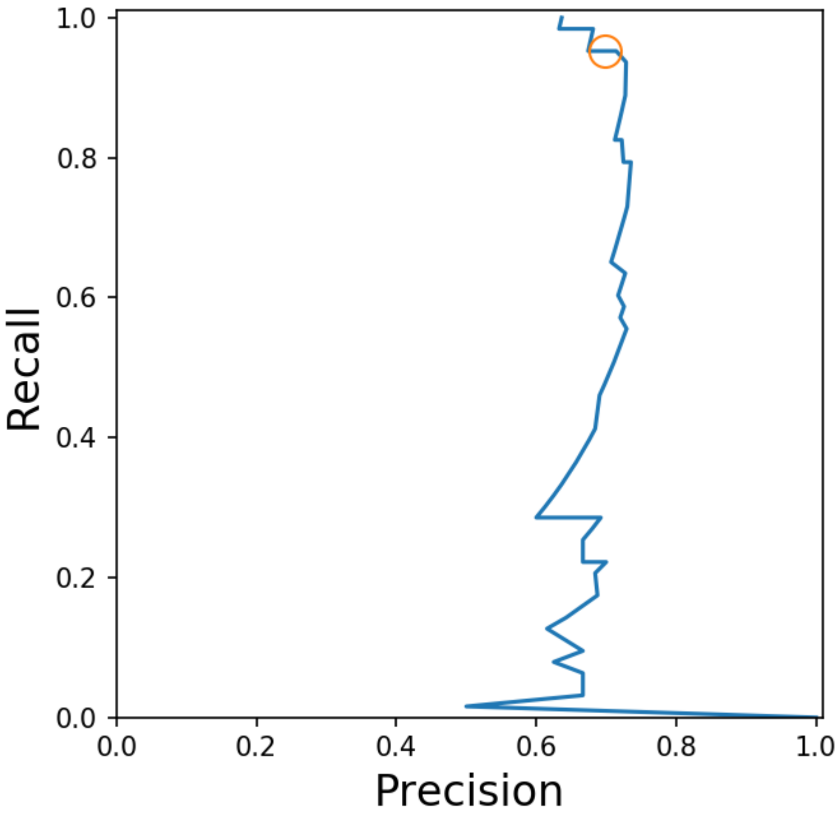
```
Area under the curve score: 0.63
```

Adding results to a table for summarization in the end

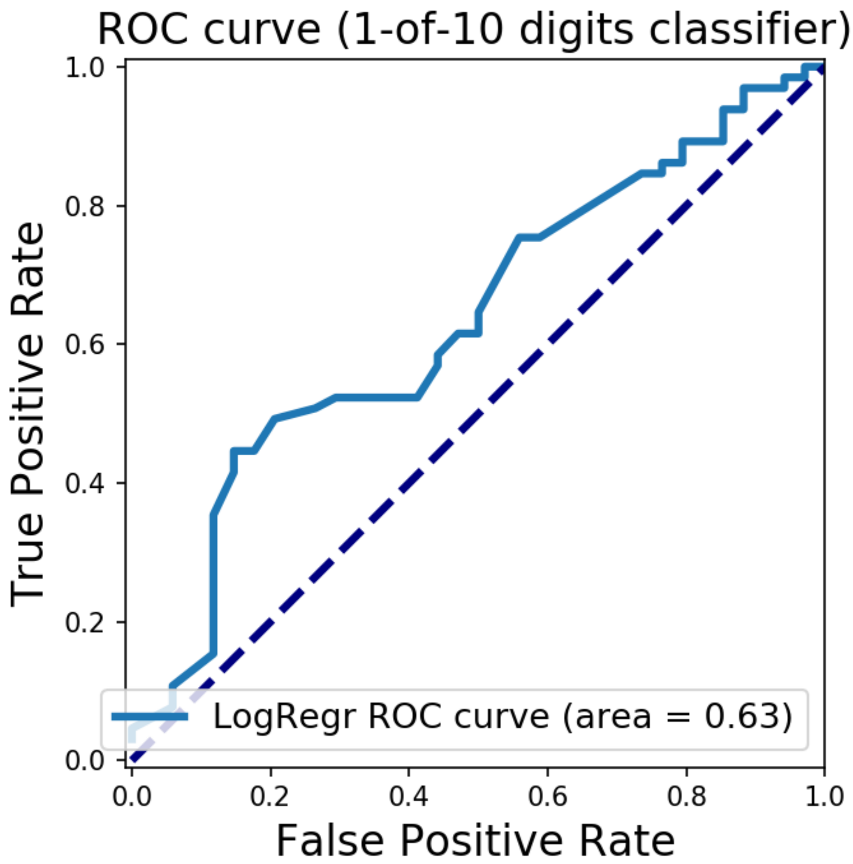
In [98]:
```
model_name.append("Forward/Mat")
accuracy_col.append(accuracy_score(y_test, y_pred))
precision_col.append(precision_score(y_test, y_pred))
recall_col.append(recall_score(y_test, y_pred))
f1_col.append(f1_score(y_test, y_pred))
auc_col.append(roc_auc_score(y_test, y_pred))
```

Building a precision-recall curve

In [99]:
```
from sklearn.metrics import precision_recall_curve
y_scores_lr = log_reg.fit(X_train, y_train).decision_function(X_test)
%matplotlib notebook
precision, recall, thresholds = precision_recall_curve(y_test, y_scores_
lr)
closest_zero = np.argmin(np.abs(thresholds))
closest_zero_p = precision[closest_zero]
closest_zero_r = recall[closest_zero]
plt.figure()
plt.xlim([0.0, 1.01])
plt.ylim([0.0, 1.01])
plt.plot(precision, recall, label='Precision-Recall Curve')
plt.plot(closest_zero_p, closest_zero_r, 'o', markersize = 12, fillstyle
 = 'none')
plt.xlabel('Precision', fontsize=16)
plt.ylabel('Recall', fontsize=16)
plt.axes().set_aspect('equal')
plt.show()
```



In [200]:
```
from sklearn.metrics import roc_curve, auc
X_train, X_test, y_train, y_test = train_test_split(X, y_mat)
y_pred_lr = log_reg.fit(X_train, y_train).decision_function(X_test)
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)
plt.figure()
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(fpr_lr, tpr_lr, lw=3, label='LogRegr ROC curve (area = {:0.2f})
'.format(roc_auc_lr))
plt.xlabel('False Positive Rate', fontsize=16)
plt.ylabel('True Positive Rate', fontsize=16)
plt.title('ROC curve (1-of-10 digits classifier)', fontsize=16)
plt.legend(loc='lower right', fontsize=13)
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.axes().set_aspect('equal')
plt.show()
```



Summary

A table to compare results

In [100]:
```
d = {'model_name': model_name, 'accuracy_col': accuracy_col, 'precision_
col': precision_col,
'recall_col': recall_col, 'f1_col': f1_col, 'auc_col': auc_col}
df = pd.DataFrame(data=d)
df
```

Out[100]:

	model_name	accuracy_col	precision_col	recall_col	f1_col	auc_col
0	Backward/Por	0.852761	0.868421	0.970588	0.916667	0.614924
1	Forward/Por	0.834356	0.846667	0.969466	0.903915	0.625358
2	Backward/Mat	0.686869	0.688312	0.883333	0.773723	0.633974
3	Forward/Mat	0.717172	0.705882	0.952381	0.810811	0.628968

Summary

In summary, it is harder to predict math score than Portuguese score. As of Portuguese, Backward Elimination creates a better set of features. In contrast to this, Forward Selection created a set of features that predict math score better than Backward Elimination.

Neural networks/AI

92-101

2.1. Predict sales and number of customers of Rossmann stores with Artificial Neural Network in Keras

Predict sales and number of customers of Rossmann stores with Artificial Neural Network in Keras (Regression)

Information about the dataset

- Number of inputs: **1 017 209**
- Number of features: **19**
- Dataset: https://www.kaggle.com/c/rossmann-store-sales
- Data fields description: can be found here https://www.kaggle.com/c/rossmann-store-sales

Importing main libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
import warnings; warnings.simplefilter('ignore')
```

Loading of the dataset. Two dataset from the website were merged sepately, a merged version is presented

In [2]:
```
df_un = pd.read_csv("exported.csv")
```

Some data preprocessing

In [3]:
```
#convert to datetime
df_un['Date'] = pd.to_datetime(df_un['Date'])

#create a month column from date column
df_un['month'] = df_un['Date'].dt.month

#create seasonal column
conditions = [
    (df_un['month'] == 1) | (df_un['month'] == 2) | (df_un['month'] == 1
2),
    (df_un['month'] == 3) | (df_un['month'] == 4) | (df_un['month'] == 5
),
    (df_un['month'] == 6) | (df_un['month'] == 7) | (df_un['month'] == 8
)
]

choices = ['Winter', 'Spring', 'Summer']
df_un['Season'] = np.select(conditions, choices, default='Autumn')
```

Removing values with competion distance = na and days when shops were closed

In [4]:
```
df_un = df_un[df_un['CompetitionDistance'].notnull()]
df_un = df_un[df_un['Open']!=0]
```

A value to predict

3rd column - sales, 4th - number of customers

In [5]:
```
y = df_un.iloc[:, 3]
y_2 = df_un.iloc[:, 4]
```

Creating a separate dataframe with categorical variables to apply get_dummies

Only some columns from a dataset will be used - DayOfWeek, Promo, StateHoliday, SchoolHoliday, StoreType, Assortment, month, Season

In [6]:
```
#indexes of columns with and without categorical variables
col_list = [1,6,7,8,9,10,19,20]
no_cat_var = [11]

df_un_cat = df_un.iloc[:, col_list]
df_un_non_cat = df_un.iloc[:, no_cat_var]
```

Convert some variables to "category" so get_dummies encodes it

In [7]:
```
#conversion so get_dummies works
df_un_cat['Promo']= df_un_cat['Promo'].astype('category')
df_un_cat['SchoolHoliday'] = df_un_cat['SchoolHoliday'].astype('category
')
df_un_cat['month']= df_un_cat['month'].astype('category')
df_un_cat['DayOfWeek']= df_un_cat['DayOfWeek'].astype('category')
```

Applying get_dummies

Dropping first dummy column is important to avoid collinearity, so drop_first is set to True

In [8]:
```
df = pd.get_dummies(df_un_cat, drop_first=True)
```

In [9]:
```
pd.options.display.max_columns = None
```

In [16]:
```
df.head()
```

Out[16]:

	DayOfWeek_2	DayOfWeek_3	...	Season_Summer	Season_Winter
1	0	0	...	1	0
2	0	1	...	0	1
3	0	0	...	0	1
4	0	0	...	0	1
5	0	0	...	0	1

5 rows × 30 columns

Adding continuos variables to encoded categorical

In [17]:
```
X = pd.merge(df, df_un_non_cat, left_index=True, right_index=True)
```

Final dataset

In [18]:
```
X.head()
```

Out[18]:

	DayOfWeek_2	DayOfWeek_3	...	Season_Winter	CompetitionDistance
1	0	0	...	0	4610.0
2	0	1	...	1	4610.0
3	0	0	...	1	4610.0
4	0	0	...	1	4610.0
5	0	0	...	1	4610.0

5 rows × 31 columns

Creation of a Neural Network

Train/test split

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size = 0
.2)
```

Feature Scaling

In [14]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Importing the Keras libraries and packages

In [15]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

```
Using TensorFlow backend.
```

Predicting sales of Rossmann shops per day

How the model will look like:
5 layers, each has 96 neurons, small dropout to prevent overfitting, relu as an activator, mean squared error as loss function, adam as an optimizer, 15 epochs.

In [16]:
```
# Initialising the ANN
model = Sequential()
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu', input_dim = 30))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation='l
inear'))
model.compile(loss='mse', optimizer='adam', metrics=['mse', 'mae', 'mape
'])
history = model.fit(X_train, y_train, batch_size = 10000, epochs = 15)
```

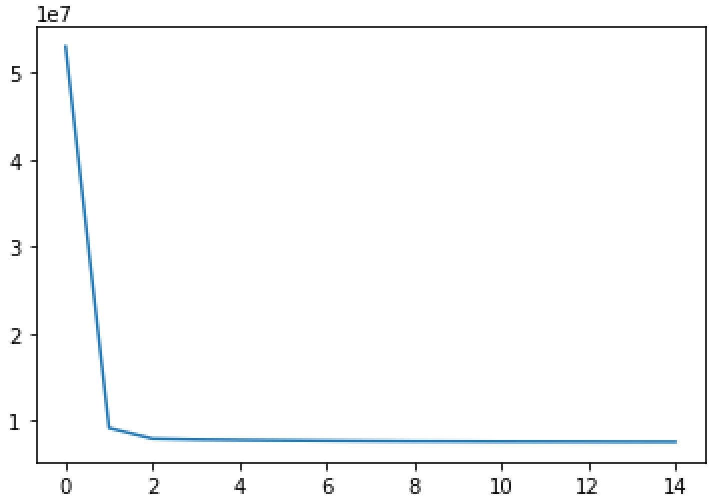
```
Epoch 1/15
673764/673764 [==============================] - 13s 19us/step - loss: 5
2987611.2091 - mean_squared_error: 52987611.2091 - mean_absolute_error:
6544.9943 - mean_absolute_percentage_error: 26674004.5298
Epoch 2/15
673764/673764 [==============================] - 14s 21us/step - loss: 9
117003.9522 - mean_squared_error: 9117003.9522 - mean_absolute_error: 21
95.7987 - mean_absolute_percentage_error: 461463964.4931
Epoch 3/15
673764/673764 [==============================] - 13s 20us/step - loss: 7
887833.1973 - mean_squared_error: 7887833.1973 - mean_absolute_error: 20
46.2621 - mean_absolute_percentage_error: 452572975.5331
Epoch 4/15
673764/673764 [==============================] - 13s 20us/step - loss: 7
785916.8853 - mean_squared_error: 7785916.8853 - mean_absolute_error: 20
30.5539 - mean_absolute_percentage_error: 443062738.4014
Epoch 5/15
673764/673764 [==============================] - 15s 23us/step - loss: 7
734546.3853 - mean_squared_error: 7734546.3853 - mean_absolute_error: 20
22.9094 - mean_absolute_percentage_error: 450890027.3082
Epoch 6/15
673764/673764 [==============================] - 13s 20us/step - loss: 7
689480.9909 - mean_squared_error: 7689480.9909 - mean_absolute_error: 20
15.6186 - mean_absolute_percentage_error: 439359591.8312
Epoch 7/15
673764/673764 [==============================] - 13s 20us/step - loss: 7
650397.4144 - mean_squared_error: 7650397.4144 - mean_absolute_error: 20
09.4008 - mean_absolute_percentage_error: 448712348.90703s - loss: 76662
64.6837 - mean_squared_error: 7666264.6837 - mean_absolute_error: 2010.8
361 - mean_absolute_percentage_error:  - ETA: 2s - loss: 7660844.6429 -
mean_squared_error: 7660844.6429 - mean_absolute_error: 2010.7851 - mean
_absolute_percenta
Epoch 8/15
673764/673764 [==============================] - 14s 21us/step - loss: 7
615763.5031 - mean_squared_error: 7615763.5031 - mean_absolute_error: 20
03.2561 - mean_absolute_percentage_error: 445686184.55488s - loss: 76667
91.7222 - mean_squared_error: 7666791.7222
Epoch 9/15
```

```
673764/673764 [==============================] - 14s 21us/step - loss: 7
588622.9197 - mean_squared_error: 7588622.9197 - mean_absolute_error: 19
98.6225 - mean_absolute_percentage_error: 448924305.2814
Epoch 10/15
673764/673764 [==============================] - 16s 24us/step - loss: 7
577212.8953 - mean_squared_error: 7577212.8953 - mean_absolute_error: 19
96.9520 - mean_absolute_percentage_error: 450400232.8444
Epoch 11/15
673764/673764 [==============================] - 15s 22us/step - loss: 7
557082.5979 - mean_squared_error: 7557082.5979 - mean_absolute_error: 19
93.4493 - mean_absolute_percentage_error: 455791863.3235
Epoch 12/15
673764/673764 [==============================] - 15s 22us/step - loss: 7
550350.7339 - mean_squared_error: 7550350.7339 - mean_absolute_error: 19
92.2672 - mean_absolute_percentage_error: 447933901.2123
Epoch 13/15
673764/673764 [==============================] - 15s 22us/step - loss: 7
540626.7996 - mean_squared_error: 7540626.7996 - mean_absolute_error: 19
90.1583 - mean_absolute_percentage_error: 455972562.0344
Epoch 14/15
673764/673764 [==============================] - 14s 20us/step - loss: 7
526775.8083 - mean_squared_error: 7526775.8083 - mean_absolute_error: 19
88.2497 - mean_absolute_percentage_error: 465239196.40311s - loss: 75421
65.0887 - mean_squared_error: 7542165.0887 - mean_absolute_error: 1989.1
682 - mean_absolute_percentage_error: 44
Epoch 15/15
673764/673764 [==============================] - 13s 20us/step - loss: 7
521704.8920 - mean_squared_error: 7521704.8920 - mean_absolute_error: 19
86.7170 - mean_absolute_percentage_error: 461258068.0457
```

MSE plot

In [17]:
```
plt.plot(history.history['mean_squared_error'])
```

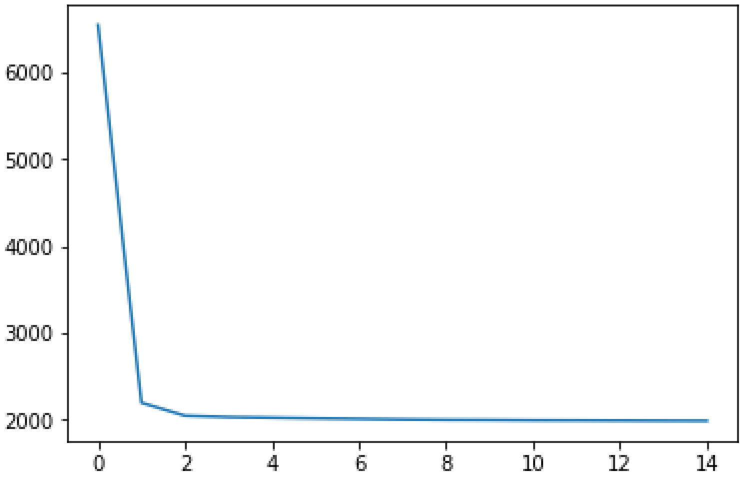
Out[17]: [<matplotlib.lines.Line2D at 0x2298e101a20>]



MAE plot

In [18]:
```
plt.plot(history.history['mean_absolute_error'])
```

Out[18]: [<matplotlib.lines.Line2D at 0x2298e1f21d0>]



R-squared of a model

In [19]:
```
y_pred = model.predict(X_test)
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Out[19]: 0.2669489140493173

MSE of a model

In [20]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[20]: 6949635.98763258

MAE of a model

In [21]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[21]: 1911.8552002898136

A table that compares real and predicted values

In [22]:
```
final_preds = []
for pred in y_pred:
    final_preds.append(pred[0])

y_test_pr = []
for pred in y_test:
    y_test_pr.append(pred)

d = {'y_test': y_test_pr, 'final_preds': final_preds}
pd.DataFrame(data=d).round(0)[:10]
```

Out[22]:

	y_test	final_preds
0	6397	6027.0
1	8123	6485.0
2	10168	7633.0

3	7270	6309.0
4	7757	5395.0
5	5626	6998.0
6	3724	5803.0
7	13033	9354.0
8	6674	6813.0
9	13137	7927.0

Predicting customers of Rossmann shops per day

In [30]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, y_2, test_size =
 0.2)
```

Feature Scaling

In [31]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [32]:
```
# Initialising the ANN
model = Sequential()
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu', input_dim = 30))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 1, kernel_initializer = 'uniform', activation='l
inear'))
model.compile(loss='mse', optimizer='adam', metrics=['mse', 'mae', 'mape
'])
history = model.fit(X_train, y_train, batch_size = 10000, epochs = 15)
```

```
Epoch 1/15
673764/673764 [==============================] - 15s 22us/step - loss: 5
94856.2430 - mean_squared_error: 594856.2430 - mean_absolute_error: 646.
6281 - mean_absolute_percentage_error: 11562664.0900 - loss: 744521.4668
 - mean_squared_error: 744521.4668 - mean_absolute_error:
Epoch 2/15
673764/673764 [==============================] - 13s 19us/step - loss: 1
28110.0609 - mean_squared_error: 128110.0609 - mean_absolute_error: 243.
9137 - mean_absolute_percentage_error: 49525179.9691
```

```
Epoch 3/15
673764/673764 [==============================] - 13s 19us/step - loss: 1
20119.1534 - mean_squared_error: 120119.1534 - mean_absolute_error: 238.
2204 - mean_absolute_percentage_error: 51319970.3562
Epoch 4/15
673764/673764 [==============================] - 13s 19us/step - loss: 1
18889.0103 - mean_squared_error: 118889.0103 - mean_absolute_error: 236.
8831 - mean_absolute_percentage_error: 51751841.86489s - loss: 119703.76
82 - mean_
Epoch 5/15
673764/673764 [==============================] - 13s 19us/step - loss: 1
18436.3699 - mean_squared_error: 118436.3699 - mean_absolute_error: 236.
1597 - mean_absolute_percentage_error: 51595504.5250
Epoch 6/15
673764/673764 [==============================] - 13s 19us/step - loss: 1
17836.2788 - mean_squared_error: 117836.2788 - mean_absolute_error: 235.
5189 - mean_absolute_percentage_error: 50133732.3387
Epoch 7/15
673764/673764 [==============================] - 15s 22us/step - loss: 1
17536.1460 - mean_squared_error: 117536.1460 - mean_absolute_error: 235.
0314 - mean_absolute_percentage_error: 50217143.56897s - loss: 116476.64
35 - mean_squared_error: 116476.6435 - m
Epoch 8/15
673764/673764 [==============================] - 19s 28us/step - loss: 1
17318.8183 - mean_squared_error: 117318.8183 - mean_absolute_error: 234.
7863 - mean_absolute_percentage_error: 51914800.3889
Epoch 9/15
673764/673764 [==============================] - 18s 26us/step - loss: 1
17227.8014 - mean_squared_error: 117227.8014 - mean_absolute_error: 234.
5484 - mean_absolute_percentage_error: 49689070.2143
Epoch 10/15
673764/673764 [==============================] - 19s 28us/step - loss: 1
16872.7239 - mean_squared_error: 116872.7239 - mean_absolute_error: 234.
2800 - mean_absolute_percentage_error: 51829955.4436
Epoch 11/15
673764/673764 [==============================] - 20s 29us/step - loss: 1
16627.2189 - mean_squared_error: 116627.2189 - mean_absolute_error: 234.
0391 - mean_absolute_percentage_error: 51162157.7671
Epoch 12/15
673764/673764 [==============================] - 17s 25us/step - loss: 1
16637.2613 - mean_squared_error: 116637.2613 - mean_absolute_error: 233.
9995 - mean_absolute_percentage_error: 52263084.1833
Epoch 13/15
673764/673764 [==============================] - 16s 23us/step - loss: 1
16421.4444 - mean_squared_error: 116421.4444 - mean_absolute_error: 233.
8766 - mean_absolute_percentage_error: 49556634.0550
Epoch 14/15
673764/673764 [==============================] - 15s 23us/step - loss: 1
16296.5473 - mean_squared_error: 116296.5473 - mean_absolute_error: 233.
7648 - mean_absolute_percentage_error: 50318545.96884s - loss: 116414.48
56 - mean_squared_error: 116414.4856 - mean_absolute_error: 233.7413 - m
Epoch 15/15
673764/673764 [==============================] - 15s 23us/step - loss: 1
16261.3080 - mean_squared_error: 116261.3080 - mean_absolute_error: 233.
5814 - mean_absolute_percentage_error: 50369694.6934
```

R-squared of a model

In [33]:
```
y_pred = model.predict(X_test)
from sklearn.metrics import r2_score
```

```
r2_score(y_test, y_pred)
```

Out[33]: 0.3175307068493741

MSE of a model

In [34]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Out[34]: 108527.08177650992

MAE of a model

In [35]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
```

Out[35]: 225.10123566703268

A table that compares real and predicted values

In [36]:
```
final_preds = []
for pred in y_pred:
    final_preds.append(pred[0])

y_test_pr = []
for pred in y_test:
    y_test_pr.append(pred)

d = {'y_test': y_test_pr, 'final_preds': final_preds}
pd.DataFrame(data=d).round(0)[:10]
```

Out[36]:

	y_test	final_preds
0	932	627.0
1	1790	846.0
2	617	695.0
3	512	569.0
4	957	812.0
5	556	577.0
6	887	869.0
7	330	613.0
8	421	801.0
9	743	576.0

Summary

Model, that predicts number of customers, does a better job than a model that predicts sales. Moreover, customers model predict with an error 200 customers on average, while sales model predicts with an error around 1900 euros.

102 -107

2.2. Predict number of customers of Rossmann stores with Artificial Neural Network in Tensorflow

Predict number of customers of Rossmann stores with Artificial Neural Network in Tensorflow

The structure of this network is literally the same as the structure of a neural network that processes the same dataset with Keras (repository: https://github.com/oleksandrkim/Predicitng-sales-and-number-of-customers-of-Rossmann-stores-with-Artificial-Neural-Network-in-Keras).
It is a "mimic" of keras model but done in tensorflow.

Information about the dataset

- Number of inputs: **1 017 209**
- Number of features: **19**
- Dataset: https://www.kaggle.com/c/rossmann-store-sales
- Data fields description: can be found here https://www.kaggle.com/c/rossmann-store-sales

Importing main libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import preprocessing
import warnings; warnings.simplefilter('ignore')
```

Loading of the dataset. Two dataset from the website were merged sepately, a merged version is presented

In [2]:
```
df_un = pd.read_csv("exported.csv")
```

Some data preprocessing

In [3]:
```
#convert to datetime
df_un['Date'] = pd.to_datetime(df_un['Date'])

#create a month column from date column
df_un['month'] = df_un['Date'].dt.month

#create seasonal column
conditions = [
    (df_un['month'] == 1) | (df_un['month'] == 2) | (df_un['month'] == 1
2),
    (df_un['month'] == 3) | (df_un['month'] == 4) | (df_un['month'] == 5
),
    (df_un['month'] == 6) | (df_un['month'] == 7) | (df_un['month'] == 8
)
]

choices = ['Winter', 'Spring', 'Summer']
```

```
df_un['Season'] = np.select(conditions, choices, default='Autumn')
```

Removing values with competion distance = na and days when shops were closed

In [4]:
```
df_un = df_un[df_un['CompetitionDistance'].notnull()]
df_un = df_un[df_un['Open']!=0]
```

A value to predict

In [5]:
```
y = df_un.iloc[:, 4] #4 for customers, 3 for sales
```

Creating a separate dataframe with categorical variables to apply get_dummies
Only some columns from a dataset will be used - DayOfWeek, Promo, StateHoliday, SchoolHoliday, StoreType, Assortment, month, Season

In [6]:
```
#indexes of columns with and without categorical variables
col_list = [1,6,7,8,9,10,19,20]
no_cat_var = [11]

df_un_cat = df_un.iloc[:, col_list]
df_un_non_cat = df_un.iloc[:, no_cat_var]
```

Convert some variables to "category" so get_dummies encodes it

In [7]:
```
#conversion so get_dummies works
df_un_cat['Promo']= df_un_cat['Promo'].astype('category')
df_un_cat['SchoolHoliday'] = df_un_cat['SchoolHoliday'].astype('category
')
df_un_cat['month']= df_un_cat['month'].astype('category')
df_un_cat['DayOfWeek']= df_un_cat['DayOfWeek'].astype('category')
```

Applying get_dummies
Dropping first dummy column is important to avoid collinearity, so drop_first is set to True

In [8]:
```
df = pd.get_dummies(df_un_cat, drop_first=True)
```

In [9]:
```
pd.options.display.max_columns = None
```

In [10]:
```
df.head()
```

Out[10]:

	DayOfWeek_2	DayOfWeek_3	...	Season_Summer	Season_Winter
1	0	0	...	1	0
2	0	1	...	0	1
3	0	0	...	0	1
4	0	0	...	0	1
5	0	0	...	0	1

5 rows × 30 columns

Adding continuos variables to encoded categorical

In [11]:
```
X = pd.merge(df, df_un_non_cat, left_index=True, right_index=True)
```

Final dataset

In [12]:
```
X.head()
```

Out[12]:

	DayOfWeek_2	DayOfWeek_3	...	Season_Winter	CompetitionDistance
1	0	0	...	0	4610.0
2	0	1	...	1	4610.0
3	0	0	...	1	4610.0
4	0	0	...	1	4610.0
5	0	0	...	1	4610.0

5 rows × 31 columns

Creation of a Neural Network

Train-test split

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size = 0
.2)


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Reshaping is needed to feed data into tensorflow

In [14]:
```
y_train = y_train.values
y_train.shape = (len(y_train), 1)

y_train = y_train.astype(float)
```

Create placeholders for x and y, layers
Biases are initialized with zeros
Kernels are initialized with glorot uniform initializer
4 hidden layers with 64 neurons (this is the only difference with keras model)
10% of data dropped to prevent overfitting

Cost is calculated wit MAE (Mean Absolute Error)
Optimizer is "adam"

In [15]:
```
import tensorflow as tf
import numpy as np
import uuid

x = tf.placeholder(shape=[None, 30], dtype=tf.float32) #number of featur
es
y = tf.placeholder(shape=[None, 1], dtype=tf.float32)


dense = tf.layers.dense(x, 30, activation = tf.nn.relu,
                        bias_initializer = tf.zeros_initializer(),
                        kernel_initializer = tf.glorot_uniform_initializ
er())
dropout = tf.layers.dropout(inputs = dense, rate = 0.1)
dense = tf.layers.dense(dropout, 64, activation = tf.nn.relu,
                        bias_initializer = tf.zeros_initializer(),
                        kernel_initializer = tf.glorot_uniform_initializ
er())
dropout = tf.layers.dropout(inputs = dense, rate = 0.1)
dense = tf.layers.dense(dropout, 64, activation = tf.nn.relu,
                        bias_initializer = tf.zeros_initializer(),
                        kernel_initializer = tf.glorot_uniform_initializ
er())
dropout = tf.layers.dropout(inputs = dense, rate = 0.1)
dense = tf.layers.dense(dropout, 64, activation = tf.nn.relu,
                        bias_initializer = tf.zeros_initializer(),
                        kernel_initializer = tf.glorot_uniform_initializ
er())
dropout = tf.layers.dropout(inputs = dense, rate = 0.1)
dense = tf.layers.dense(dropout, 64, activation = tf.nn.relu,
                        bias_initializer = tf.zeros_initializer(),
                        kernel_initializer = tf.glorot_uniform_initializ
er())
dropout = tf.layers.dropout(inputs = dense, rate = 0.1)
output = tf.layers.dense(dropout, 1, activation = tf.nn.sigmoid)

cost = tf.losses.absolute_difference(y, output) #mae
optimizer = tf.train.AdamOptimizer(learning_rate=0.0001).minimize(cost)
init = tf.global_variables_initializer()

tf.summary.scalar("cost", cost)
merged_summary_op = tf.summary.merge_all()

with tf.Session() as sess:
    sess.run(init)
    uniq_id = "/tmp/tensorboard-layers-api/" + uuid.uuid1().__str__()[:6
]
    summary_writer = tf.summary.FileWriter(uniq_id, graph=tf.get_default
_graph())
    x_vals = X_train
    y_vals = y_train
    for step in range(100):
        _, val, summary = sess.run([optimizer, cost, merged_summary_op],
                                   feed_dict={x: x_vals, y: y_vals})
        if step % 20 == 0:
            print("step: {}, value: {}".format(step, val))
            summary_writer.add_summary(summary, step)
```
```
step: 0, value: 762.8004760742188
step: 20, value: 762.5020751953125
```

```
step: 40, value: 762.4652099609375
step: 60, value: 762.3936157226562
step: 80, value: 762.3175659179688
```

Results are simiar to keras' first steps of trainig

In [ ]:
```
#TODO: batching
#input_func = tf.estimator.inputs.numpy_input_fn({'x':x_train},y_train,b
atch_size=4,num_epochs=None,shuffle=True) (???)
```

108 -118

2.3. Predict students' performance in Portuguese and Math by building a neural network with Tensorflow

Predict students' performance (grades) in Portuguese and Math by building a neural network with Tensorflow

- Number of inputs: **649**
- Number of variables: **30**
- Dataset: https://archive.ics.uci.edu/ml/datasets/student+performance
- Data fields description: https://archive.ics.uci.edu/ml/datasets/student+performance

Importing required libraries

In [9]:
```
import numpy as np
import pandas as pd
import warnings; warnings.simplefilter('ignore')
```

Importing the dataset

In [10]:
```
data_por = pd.read_csv('student-por.csv')
data_mat = pd.read_csv('student-mat.csv')
```

The model will predict a final grade

In [11]:
```
y_por = data_por.iloc[:, -1]
y_mat = data_mat.iloc[:, -1]
```

Columns with grades are dropped

In [12]:
```
data_por = data_por.drop(['G1', 'G2', 'G3'], axis=1)
data_mat = data_mat.drop(['G1', 'G2', 'G3'], axis=1)
data_por.head()
```

Out[12]:

	school	sex	age	address	famsize	...	goout	Dalc	Walc	health	absences
0	GP	F	18	U	GT3	...	4	1	1	3	4
1	GP	F	17	U	GT3	...	3	1	1	3	2
2	GP	F	15	U	LE3	...	2	2	3	3	6
3	GP	F	15	U	GT3	...	2	1	1	5	0
4	GP	F	16	U	GT3	...	2	1	2	5	0

5 rows × 30 columns

Portuguese course

Splitting the dataset in train and test parts

In [53]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_por, y_por)
```

Scaling the only continuous variable in a dataset

In [54]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train['absences'] = scaler.fit_transform(X_train['absences'].values.re
shape(-1, 1))
X_test['absences'] = scaler.fit_transform(X_test['absences'].values.resh
ape(-1, 1))
```

Importing the tensorflow

In [55]:
```
import tensorflow as tf
```

Creating feature columns

In [56]:
```
school_vocab = ['MS', 'GP']
school_column = tf.feature_column.categorical_column_with_vocabulary_lis
t(
      key="school", vocabulary_list=school_vocab)
```

In [57]:
```
sex_vocab = ['M', 'F']
sex_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="sex", vocabulary_list=sex_vocab)
```

In [58]:
```
age_vocab = [18, 16, 17, 20, 19, 15, 21, 22]
age_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="age", vocabulary_list=age_vocab)
```

In [59]:
```
address_vocab = ['R', 'U']
address_column = tf.feature_column.categorical_column_with_vocabulary_li
st(
      key="address", vocabulary_list=address_vocab)
```

In [60]:
```
famsize_vocab = ['GT3', 'LE3']
famsize_column = tf.feature_column.categorical_column_with_vocabulary_li
st(
      key="famsize", vocabulary_list=famsize_vocab)
```

In [61]:
```
Pstatus_vocab = ['T', 'A']
Pstatus_column = tf.feature_column.categorical_column_with_vocabulary_li
st(
      key="Pstatus", vocabulary_list=Pstatus_vocab)
```

In [62]:
```
Medu_vocab = [3, 4, 1, 2, 0]
Medu_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="Medu", vocabulary_list=Medu_vocab)
```

In [63]:
```
Fedu_vocab = [2, 1, 4, 3, 0]
Fedu_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="Fedu", vocabulary_list=Fedu_vocab)
```

In [64]:
```
Mjob_vocab = ['services', 'other', 'teacher', 'at_home', 'health']
Mjob_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="Mjob", vocabulary_list=Mjob_vocab)
```

In [65]:
```
Fjob_vocab = ['other', 'health', 'services', 'teacher', 'at_home']
Fjob_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="Fjob", vocabulary_list=Fjob_vocab)
```

In [66]:
```
reason_vocab = ['course', 'reputation', 'other', 'home']
reason_column = tf.feature_column.categorical_column_with_vocabulary_lis
t(
      key="reason", vocabulary_list=reason_vocab)
```

In [67]:
```
guardian_vocab = ['mother', 'father', 'other']
guardian_column = tf.feature_column.categorical_column_with_vocabulary_l
ist(
      key="guardian", vocabulary_list=guardian_vocab)
```

In [68]:
```
traveltime_vocab = [1, 2, 4, 3]
traveltime_column = tf.feature_column.categorical_column_with_vocabulary
_list(
      key="traveltime", vocabulary_list=traveltime_vocab)
```

In [69]:
```
studytime_vocab = [1, 4, 2, 3]
studytime_column = tf.feature_column.categorical_column_with_vocabulary_
list(
      key="studytime", vocabulary_list=studytime_vocab)
```

In [70]:
```
failures_vocab = [1, 4, 2, 3]
failures_column = tf.feature_column.categorical_column_with_vocabulary_l
ist(
      key="failures", vocabulary_list=failures_vocab)
```

In [71]:
```
schoolsup_vocab = ['no', 'yes']
schoolsup_column = tf.feature_column.categorical_column_with_vocabulary_
list(
      key="schoolsup", vocabulary_list=schoolsup_vocab)
```

In [72]:
```
famsup_vocab = ['yes', 'no']
famsup_column = tf.feature_column.categorical_column_with_vocabulary_lis
t(
      key="famsup", vocabulary_list=famsup_vocab)
```

In [73]:
```
paid_vocab = ['no', 'yes']
paid_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="paid", vocabulary_list=paid_vocab)
```

In [74]:
```
activities_vocab = ['no', 'yes']
activities_column = tf.feature_column.categorical_column_with_vocabulary
_list(
      key="activities", vocabulary_list=activities_vocab)
```

In [75]:
```
nursery_vocab = ['yes', 'no']
nursery_column = tf.feature_column.categorical_column_with_vocabulary_li
st(
      key="nursery", vocabulary_list=nursery_vocab)
```

In [76]:
```
higher_vocab = ['yes', 'no']
higher_column = tf.feature_column.categorical_column_with_vocabulary_lis
t(
      key="higher", vocabulary_list=higher_vocab)
```

In [77]:
```
internet_vocab = ['yes', 'no']
internet_column = tf.feature_column.categorical_column_with_vocabulary_l
ist(
      key="internet", vocabulary_list=internet_vocab)
```

In [78]:
```
romantic_vocab = ['no', 'yes']
romantic_column = tf.feature_column.categorical_column_with_vocabulary_l
ist(
      key="romantic", vocabulary_list=romantic_vocab)
```

In [79]:
```
famrel_vocab = [4, 5, 2, 1, 3]
famrel_column = tf.feature_column.categorical_column_with_vocabulary_lis
t(
      key="famrel", vocabulary_list=famrel_vocab)
```

In [80]:
```
freetime_vocab = [3, 5, 4, 2, 1]
freetime_column = tf.feature_column.categorical_column_with_vocabulary_l
ist(
      key="freetime", vocabulary_list=freetime_vocab)
```

In [81]:
```
goout_vocab = [3, 2, 5, 1, 4]
goout_column = tf.feature_column.categorical_column_with_vocabulary_list
(
      key="goout", vocabulary_list=goout_vocab)
```

In [82]:
```
Dalc_vocab = [3, 2, 5, 1, 4]
Dalc_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="Dalc", vocabulary_list=Dalc_vocab)
```

In [83]:
```
Walc_vocab = [1, 4, 2, 3, 5]
Walc_column = tf.feature_column.categorical_column_with_vocabulary_list(
      key="Walc", vocabulary_list=Walc_vocab)
```

In [84]:
```
health_vocab = [3, 5, 1, 4, 2]
health_column = tf.feature_column.categorical_column_with_vocabulary_lis
t(
      key="health", vocabulary_list=health_vocab)
```

Adding all features to a list

In [85]:
```
feature_columns = [
    tf.feature_column.indicator_column(school_column),
    tf.feature_column.indicator_column(sex_column),
    tf.feature_column.indicator_column(age_column),
    tf.feature_column.indicator_column(address_column),
    tf.feature_column.indicator_column(famsize_column),
    tf.feature_column.indicator_column(Pstatus_column),
    tf.feature_column.indicator_column(Medu_column),
    tf.feature_column.indicator_column(Fedu_column),
    tf.feature_column.indicator_column(Mjob_column),
```

```
    tf.feature_column.indicator_column(Fjob_column),
    tf.feature_column.indicator_column(reason_column),
    tf.feature_column.indicator_column(guardian_column),
    tf.feature_column.indicator_column(traveltime_column),
    tf.feature_column.indicator_column(studytime_column),
    tf.feature_column.indicator_column(failures_column),
    tf.feature_column.indicator_column(schoolsup_column),
    tf.feature_column.indicator_column(famsup_column),
    tf.feature_column.indicator_column(paid_column),
    tf.feature_column.indicator_column(activities_column),
    tf.feature_column.indicator_column(nursery_column),
    tf.feature_column.indicator_column(higher_column),
    tf.feature_column.indicator_column(internet_column),
    tf.feature_column.indicator_column(romantic_column),
    tf.feature_column.indicator_column(famrel_column),
    tf.feature_column.indicator_column(freetime_column),
    tf.feature_column.indicator_column(goout_column),
    tf.feature_column.indicator_column(Dalc_column),
    tf.feature_column.indicator_column(Walc_column),
    tf.feature_column.indicator_column(health_column),
    tf.feature_column.numeric_column('absences')
  ]
```

Creates the input function for the estimator object

In [86]:
```
input_func = tf.estimator.inputs.pandas_input_fn(x=X_train,y=y_train ,ba
tch_size=100,num_epochs=10,
                                                  shuffle=False)
```

Create the estimator model

- 3 layerrs, each has 32 neurons;
- **Adam** as an optimizer (learning rate = **0.1**)
- **Relu** as an activation function

In [87]:
```
model = tf.estimator.DNNRegressor(hidden_units=[32,32,32],feature_column
s=feature_columns,
                                  optimizer=tf.train.AdamOptimizer(learni
ng_rate=0.01),
                                  activation_fn = tf.nn.relu)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\a
lexa\AppData\Local\Temp\tmpkls4r5j_
INFO:tensorflow:Using config: {'_model_dir': 'C:\\Users\\alexa\\AppData\
\Local\\Temp\\tmpkls4r5j_', '_tf_random_seed': None, '_save_summary_step
s': 100, '_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
 '_session_config': None, '_keep_checkpoint_max': 5, '_keep_checkpoint_e
very_n_hours': 10000, '_log_step_count_steps': 100, '_train_distribute':
 None, '_service': None, '_cluster_spec': <tensorflow.python.training.se
rver_lib.ClusterSpec object at 0x000002DB1A7AD240>, '_task_type': 'worke
r', '_task_id': 0, '_global_id_in_cluster': 0, '_master': '', '_evaluati
on_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker_r
eplicas': 1}
```

Train the model for 486 steps.

In [88]:
```
model.train(input_fn=input_func,steps=10000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into C:\Users\alexa\AppData\Loc
al\Temp\tmpkls4r5j_\model.ckpt.
INFO:tensorflow:loss = 17238.445, step = 1
INFO:tensorflow:Saving checkpoints for 49 into C:\Users\alexa\AppData\Lo
cal\Temp\tmpkls4r5j_\model.ckpt.
INFO:tensorflow:Loss for final step: 419.46112.
```

Out[88]:
```
<tensorflow.python.estimator.canned.dnn.DNNRegressor at 0x2db1a7ad588>
```

Creates a prediction input function and then use the .predict method to create a list or predictions on a test data.

In [89]:
```
predict_input_func = tf.estimator.inputs.pandas_input_fn(
      x=X_test,
      batch_size=100,
      num_epochs=1,
      shuffle=False)
```

In [90]:
```
pred_gen = model.predict(predict_input_func)
```

In [91]:
```
predictions = list(pred_gen)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\alexa\AppData\Local\T
emp\tmpkls4r5j_\model.ckpt-49
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

In [92]:
```
final_preds = []
for pred in predictions:
    final_preds.append(pred['predictions'])
```

Evaluating a model

In [93]:
```
from sklearn.metrics import mean_squared_error
```

In [94]:
```
mean_squared_error(y_test,final_preds)
```

Out[94]:
```
6.269638497247284
```

In [95]:
```
from sklearn.metrics import mean_absolute_error
```

In [96]:
```
mean_absolute_error(y_test,final_preds)
```

Out[96]:
```
1.8658653914562764
```

In [97]:
```
from sklearn.metrics import r2_score
```

In [98]:
```
r2_score(y_test,final_preds)
```

Out[98]: 0.3712565761296489

Mean absolute value shows that on average model makes a mistake of around 1.8 of a mark. That considers as a good result

Compare real values to predicted

In [99]:
```
list_pred=[]
for num in final_preds:
    list_pred.append(num[0])
```

In [101]:
```
d = {'y_test': y_test, 'final_preds': list_pred}
df = pd.DataFrame(data=d)
df.round(2)[:10]
```

Out[101]:

	y_test	final_preds
562	12	12.06
466	11	10.94
556	11	9.62
573	10	11.64
325	10	11.44
66	12	12.12
289	17	15.19
359	17	13.01
390	14	12.87
48	13	13.34

Math course

Train-test split

In [138]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_mat, y_mat)
```

Scaling the only continuous variable in a dataset

In [139]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train['absences'] = scaler.fit_transform(X_train['absences'].values.reshape(-1, 1))
X_test['absences'] = scaler.fit_transform(X_test['absences'].values.reshape(-1, 1))
```

In [168]:
```
input_func = tf.estimator.inputs.pandas_input_fn(x=X_train,y=y_train ,ba
tch_size=50,num_epochs=10,
                                            shuffle=False)
```

Create the estimator model

- 3 layerrs, each has 32 neurons;
- **Adam** as an optimizer (learning rate = **0.1**)
- **Relu** as an activation function

In [224]:
```
model = tf.estimator.DNNRegressor(hidden_units=[64,64,64,64],feature_col
umns=feature_columns,
                                 optimizer=tf.train.AdamOptimizer(learni
ng_rate=0.001),
                                 activation_fn = tf.nn.relu)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\a
lexa\AppData\Local\Temp\tmpvqphj_6y
INFO:tensorflow:Using config: {'_model_dir': 'C:\\Users\\alexa\\AppData\
\Local\\Temp\\tmpvqphj_6y', '_tf_random_seed': None, '_save_summary_step
s': 100, '_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
 '_session_config': None, '_keep_checkpoint_max': 5, '_keep_checkpoint_e
very_n_hours': 10000, '_log_step_count_steps': 100, '_train_distribute':
 None, '_service': None, '_cluster_spec': <tensorflow.python.training.se
rver_lib.ClusterSpec object at 0x000002DB284C8470>, '_task_type': 'worke
r', '_task_id': 0, '_global_id_in_cluster': 0, '_master': '', '_evaluati
on_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker_r
eplicas': 1}
```

Train the model for 486 steps.

In [225]:
```
model.train(input_fn=input_func,steps=10000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into C:\Users\alexa\AppData\Loc
al\Temp\tmpvqphj_6y\model.ckpt.
INFO:tensorflow:loss = 6805.715, step = 1
INFO:tensorflow:Saving checkpoints for 60 into C:\Users\alexa\AppData\Lo
cal\Temp\tmpvqphj_6y\model.ckpt.
INFO:tensorflow:Loss for final step: 350.71356.
```

Out[225]:
```
<tensorflow.python.estimator.canned.dnn.DNNRegressor at 0x2db284c8b38>
```

Creates a prediction input function and then use the .predict method to create a list or predictions on a test data.

In [226]:
```
predict_input_func = tf.estimator.inputs.pandas_input_fn(
      x=X_test,
      batch_size=100,
      num_epochs=1,
      shuffle=False)
```

In [227]:
```
pred_gen = model.predict(predict_input_func)
predictions = list(pred_gen)

final_preds = []
for pred in predictions:
    final_preds.append(pred['predictions'])
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\alexa\AppData\Local\T
emp\tmpvqphj_6y\model.ckpt-60
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

Evaluating a model

In [228]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,final_preds)
```

Out[228]:
```
22.16076519457572
```

In [229]:
```
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test,final_preds)
```

Out[229]:
```
3.6891207646841955
```

In [230]:
```
from sklearn.metrics import r2_score
r2_score(y_test,final_preds)
```

Out[230]:
```
0.008184576135729205
```

Mean absolute value shows that on average model makes a mistake of around 3.6 of a mark. That considers as an average result

Compare real values to predicted

In [231]:
```
list_pred=[]
for num in final_preds:
    list_pred.append(num[0])

d = {'y_test': y_test, 'final_preds': list_pred}
df = pd.DataFrame(data=d)
df.round(2)[:10]
```

Out[231]:

	y_test	final_preds
59	16	11.41
72	5	10.47
357	11	10.63
184	12	11.31
340	11	10.56
292	13	11.95

314	13	10.70
1	6	9.96
329	14	11.92
19	10	11.38

Summary

There are less inputs for Math course, that led to worse results for math predicition, the error is 3.4 grage points on average. As of Portuguese, the mistake is only 1.8 on average, that can be considered as a good result.

119 - 131

2.4. Create a Neural Network that classifies employees by job satisfaction (Tensorflow/Keras) [IBM dataset]

Create a Neural Network that classifies employees by job satisfaction (Tensorflow/Keras) [IBM dataset]

Information about the dataset

- Number of inputs: **1479**
- Number of features: **33**
- Dataset: https://www.ibm.com/communities/analytics/watson-analytics-blog/hr-employee-attrition/
- Data fields description: https://www.ibm.com/communities/analytics/watson-analytics-blog/hr-employee-attrition/

In [1]:
```
import warnings; warnings.simplefilter('ignore')
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#import warnings; warnings.simplefilter('ignore')

df = pd.read_csv('Classification.csv')
```

DNN requires that y starts from 0 and continues like this 0,1,2,3...

In [2]:
```
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
le = preprocessing.LabelEncoder()
df['JobSatisfaction'] = le.fit_transform(df['JobSatisfaction'])
#df['EnvironmentSatisfaction'] = le.fit_transform(df['EnvironmentSatisfa
ction'])
```

In [3]:
```
df['JobSatisfaction'].unique()
```

Out[3]:
```
array([3, 1, 2, 0], dtype=int64)
```

Assigning values to predict

In [4]:
```
y = df['JobSatisfaction']
```

For Keras y values should be encoded as dummy variables

In [5]:
```
y_dummy= pd.get_dummies(df['JobSatisfaction'])
```

Removing unneeded columns

In [6]:
```
df = df.drop(['EmployeeCount', 'EmployeeNumber'], axis=1)

#remove columns to predict
df = df.drop(['EnvironmentSatisfaction', 'JobSatisfaction','Relationship
Satisfaction'], axis=1)
```

Building a Keras model

Dividing variables, putting them in categorical and nom-categorical dataframe to encode only categorical variables

In [13]:
```
#indexes of columns with and without categorical variables
col_list = [1,2,4,6,7,8,10,11,12,13,17,19,20,23]
no_cat_var = [0,3,5,9,14,15,16,18, 21,22,24, 25,26,27]

df_un_cat = df.iloc[:, col_list]
df_un_non_cat = df.iloc[:, no_cat_var]
```

In [23]:
```
df_un_cat.head()
```

Out[23]:

	Attrition	BusinessTravel	...	StockOptionLevel	WorkLifeBalance
0	Yes	Travel_Rarely	...	0	1
1	No	Travel_Frequently	...	1	3
2	Yes	Travel_Rarely	...	0	3
3	No	Travel_Frequently	...	0	3
4	No	Travel_Rarely	...	1	3

5 rows × 14 columns

In [24]:
```
df_un_non_cat.head()
```

Out[24]:

	Age	DailyRate	...	YearsSinceLastPromotion	YearsWithCurrManager
0	41	1102	...	0	5
1	49	279	...	1	7
2	37	1373	...	0	0
3	33	1392	...	3	0
4	27	591	...	2	2

5 rows × 14 columns

Conversion so get_dummies works as it should

In [16]:
```
df_un_cat['Education']= df_un_cat['Education'].astype('category')
df_un_cat['JobInvolvement'] = df_un_cat['JobInvolvement'].astype('catego
ry')
df_un_cat['JobLevel']= df_un_cat['JobLevel'].astype('category')
df_un_cat['PerformanceRating']= df_un_cat['PerformanceRating'].astype('c
ategory')
df_un_cat['StockOptionLevel']= df_un_cat['StockOptionLevel'].astype('cat
egory')
```

```
df_un_cat['WorkLifeBalance'] = df_un_cat['WorkLifeBalance'].astype('cate
gory')
```

In [17]:
```
df = pd.get_dummies(df_un_cat, drop_first=True)
```

Merging converted into dummies categorical variables and non-categorical variables

In [18]:
```
X = pd.merge(df, df_un_non_cat, left_index=True, right_index=True)
```

Train-test split

In [16]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y_dummy, test_siz
e = 0.2)
```

Scaling variables

In [17]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Importing Keras

In [18]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

```
Using TensorFlow backend.
```

How the model will look like:
4 hidden layers, each has 96 neurons, small dropout to prevent overfitting, relu as an activator, mean squared error as loss function, softmax as an optimizer, 200 epochs.

In [50]:
```
# Initialising the ANN
model = Sequential()
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu', input_dim = 54))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 96, kernel_initializer = 'uniform', activation =
 'relu'))
model.add(Dropout(p = 0.1))
model.add(Dense(units = 4, kernel_initializer = 'uniform', activation='s
oftmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics
=['accuracy'])
history = model.fit(X_train, y_train, batch_size = 20, epochs = 200)
```

```
Epoch 1/200
1176/1176 [==============================] - 1s 985us/step - loss: 1.372
6 - acc: 0.3053
Epoch 2/200
1176/1176 [==============================] - 0s 160us/step - loss: 1.353
8 - acc: 0.3104
Epoch 3/200
1176/1176 [==============================] - 0s 192us/step - loss: 1.343
3 - acc: 0.3078
Epoch 4/200
1176/1176 [==============================] - 0s 190us/step - loss: 1.333
4 - acc: 0.3291
Epoch 5/200
1176/1176 [==============================] - 0s 187us/step - loss: 1.309
7 - acc: 0.3461
Epoch 6/200
1176/1176 [==============================] - 0s 183us/step - loss: 1.283
0 - acc: 0.3682
Epoch 7/200
1176/1176 [==============================] - 0s 202us/step - loss: 1.245
1 - acc: 0.3861
Epoch 8/200
1176/1176 [==============================] - 0s 241us/step - loss: 1.229
1 - acc: 0.3835
Epoch 9/200
1176/1176 [==============================] - 0s 198us/step - loss: 1.191
8 - acc: 0.4005
Epoch 10/200
1176/1176 [==============================] - 0s 170us/step - loss: 1.160
1 - acc: 0.4243
Epoch 11/200
1176/1176 [==============================] - 0s 166us/step - loss: 1.112
0 - acc: 0.4609
Epoch 12/200
1176/1176 [==============================] - 0s 183us/step - loss: 1.095
6 - acc: 0.4498
Epoch 13/200
1176/1176 [==============================] - 0s 202us/step - loss: 1.089
0 - acc: 0.4566
Epoch 14/200
1176/1176 [==============================] - 0s 209us/step - loss: 1.045
8 - acc: 0.4592
Epoch 15/200
1176/1176 [==============================] - 0s 177us/step - loss: 0.994
2 - acc: 0.4864
Epoch 16/200
1176/1176 [==============================] - 0s 171us/step - loss: 0.976
3 - acc: 0.5026
Epoch 17/200
1176/1176 [==============================] - 0s 181us/step - loss: 0.932
8 - acc: 0.5340
Epoch 18/200
1176/1176 [==============================] - 0s 156us/step - loss: 0.936
4 - acc: 0.5298
Epoch 19/200
1176/1176 [==============================] - 0s 155us/step - loss: 0.913
6 - acc: 0.5757
Epoch 20/200
1176/1176 [==============================] - 0s 154us/step - loss: 0.889
7 - acc: 0.5510
Epoch 21/200
```

```
Epoch 184/200
1176/1176 [==============================] - 0s 145us/step - loss: 0.110
9 - acc: 0.9643
Epoch 185/200
1176/1176 [==============================] - 0s 147us/step - loss: 0.105
5 - acc: 0.9651
Epoch 186/200
1176/1176 [==============================] - 0s 145us/step - loss: 0.080
8 - acc: 0.9736
Epoch 187/200
1176/1176 [==============================] - 0s 158us/step - loss: 0.087
4 - acc: 0.9694
Epoch 188/200
1176/1176 [==============================] - 0s 131us/step - loss: 0.145
0 - acc: 0.9507
Epoch 189/200
1176/1176 [==============================] - 0s 152us/step - loss: 0.092
6 - acc: 0.9685
Epoch 190/200
1176/1176 [==============================] - 0s 152us/step - loss: 0.110
1 - acc: 0.9668
Epoch 191/200
1176/1176 [==============================] - 0s 157us/step - loss: 0.051
5 - acc: 0.9838
Epoch 192/200
1176/1176 [==============================] - 0s 152us/step - loss: 0.077
8 - acc: 0.9728
Epoch 193/200
1176/1176 [==============================] - 0s 242us/step - loss: 0.103
5 - acc: 0.9651
Epoch 194/200
1176/1176 [==============================] - 0s 164us/step - loss: 0.085
5 - acc: 0.9694
Epoch 195/200
1176/1176 [==============================] - 0s 160us/step - loss: 0.071
6 - acc: 0.9787
Epoch 196/200
1176/1176 [==============================] - 0s 158us/step - loss: 0.079
9 - acc: 0.9702
Epoch 197/200
1176/1176 [==============================] - 0s 159us/step - loss: 0.100
2 - acc: 0.9668
Epoch 198/200
1176/1176 [==============================] - 0s 156us/step - loss: 0.083
4 - acc: 0.9745
Epoch 199/200
1176/1176 [==============================] - 0s 147us/step - loss: 0.089
4 - acc: 0.9711
Epoch 200/200
1176/1176 [==============================] - 0s 133us/step - loss: 0.106
2 - acc: 0.9694
```

In [51]:
```
y_pred = model.predict(X_test)
```

Results of a model

In [52]:
```
import tensorflow as tf
from keras.metrics import categorical_accuracy
accuracy = categorical_accuracy(y_test, y_pred)
```

```
session = tf.Session()
session.run(accuracy)
```

Out[52]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0
.,
        0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 1
.,
        0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 1., 0., 1., 0
.,
        1., 0., 1., 0., 1., 1., 0., 1., 0., 0., 0., 0., 1., 1., 0., 1., 0
.,
        1., 0., 0., 1., 1., 0., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0
.,
        0., 1., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 0., 1
.,
        0., 1., 0., 0., 1., 1., 0., 1., 1., 0., 1., 0., 0., 0., 1., 0., 0
.,
        0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0
.,
        0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0
.,
        0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 1., 1., 0., 1., 0., 1
.,
        0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1
.,
        0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 1., 1., 0
.,
        0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0., 0
.,
        0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 1
.,
        1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 1., 1., 0., 0
.,
        0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0
.,
        0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0
.,
        1., 1., 0., 0., 1.], dtype=float32)
```

Accuracy of a model

In [53]:
```
sum(session.run(accuracy))/len(session.run(accuracy))
```

Out[53]:
```
0.32653061224489793
```

First 10 real values

In [57]:
```
y_test.round(3)[50:60]
```

Out[57]:

	0	1	2	3
1143	1	0	0	0
8	0	0	1	0
104	0	0	0	1
372	0	1	0	0
367	0	0	0	1

217	0	0	1	0
342	0	0	0	1
501	0	0	1	0
437	0	1	0	0
634	1	0	0	0

First 10 predicted values

In [58]:
```
y_pred.round(3)[50:60]
```

Out[58]:
```
array([[0.057, 0.937, 0.007, 0.   ],
       [0.   , 0.   , 0.814, 0.186],
       [0.039, 0.96 , 0.001, 0.   ],
       [0.   , 0.998, 0.   , 0.002],
       [0.   , 0.   , 1.   , 0.   ],
       [0.017, 0.   , 0.973, 0.009],
       [0.   , 0.001, 0.021, 0.977],
       [0.001, 0.   , 0.   , 0.999],
       [0.119, 0.863, 0.017, 0.001],
       [0.006, 0.001, 0.986, 0.008]], dtype=float32)
```

Building a Tensorflow model

In [59]:
```
df = pd.read_csv('Classification.csv')
y = df['JobSatisfaction']
```

DNN requires that y starts from 0 and continues like this 0,1,2,3...

In [60]:
```
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
le = preprocessing.LabelEncoder()
df['JobSatisfaction'] = le.fit_transform(df['JobSatisfaction'])
```

In [61]:
```
#remove unneeded columns
df = df.drop(['EmployeeCount', 'EmployeeNumber'], axis=1)

#remove columns to predict
df = df.drop(['EnvironmentSatisfaction', 'JobSatisfaction','Relationship
Satisfaction'], axis=1)
```

Train-test split

In [62]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, y)
```

In [ ]:
```
import tensorflow as tf
```

Creating feature columns

Categorical variables

```
In [63]: Attrition = tf.feature_column.categorical_column_with_vocabulary_list(ke
y="Attrition",
                                                                                vo
cabulary_list=['No', 'Yes'])
BusinessTravel = tf.feature_column.categorical_column_with_vocabulary_li
st(key="BusinessTravel",
    vocabulary_list=['Travel_Rarely', 'Travel_Frequently', 'Non-Travel']
)
Department = tf.feature_column.categorical_column_with_vocabulary_list(k
ey="Department",
    vocabulary_list=['Sales', 'Research & Development', 'Human Resources
'])
Education = tf.feature_column.categorical_column_with_vocabulary_list(ke
y="Education",
    vocabulary_list=[2, 3, 4, 1, 5])
EducationField = tf.feature_column.categorical_column_with_vocabulary_li
st(key="EducationField",
    vocabulary_list=['Life Sciences', 'Medical', 'Marketing', 'Technical
 Degree','Other', 'Human Resources'])
Gender = tf.feature_column.categorical_column_with_vocabulary_list(key="
Gender",
    vocabulary_list=['Female', 'Male'])
JobInvolvement = tf.feature_column.categorical_column_with_vocabulary_li
st(key="JobInvolvement",
    vocabulary_list=[3, 1, 4, 2])
JobLevel = tf.feature_column.categorical_column_with_vocabulary_list(key
="JobLevel",
    vocabulary_list=[2, 4, 1, 3, 5])
JobRole = tf.feature_column.categorical_column_with_vocabulary_list(key=
"JobRole",
    vocabulary_list=['Sales Executive', 'Research Scientist', 'Healthcar
e Representative',
                     'Sales Representative','Manufacturing Director', 'L
aboratory Technician',
                     'Manager','Research Director', 'Human Resources'])
MaritalStatus = tf.feature_column.categorical_column_with_vocabulary_lis
t(key="MaritalStatus",
    vocabulary_list=['Married', 'Divorced', 'Single'])
OverTime = tf.feature_column.categorical_column_with_vocabulary_list(key
="OverTime",
    vocabulary_list=['No', 'Yes'])
PerformanceRating = tf.feature_column.categorical_column_with_vocabulary
_list(key="PerformanceRating",
    vocabulary_list=[3, 4])
StockOptionLevel = tf.feature_column.categorical_column_with_vocabulary_
list(key="StockOptionLevel",
    vocabulary_list=[1, 0, 2, 3])
WorkLifeBalance = tf.feature_column.categorical_column_with_vocabulary_l
ist(key="WorkLifeBalance",
    vocabulary_list=[3, 1, 2, 4])
```

Continuous variables

```
In [64]: Age = tf.feature_column.numeric_column("Age")
DailyRate = tf.feature_column.numeric_column("DailyRate")
DistanceFromHome = tf.feature_column.numeric_column("DistanceFromHome")
HourlyRate = tf.feature_column.numeric_column("HourlyRate")
MonthlyIncome = tf.feature_column.numeric_column("MonthlyIncome")
MonthlyRate = tf.feature_column.numeric_column("MonthlyRate")
```

```
NumCompaniesWorked = tf.feature_column.numeric_column("NumCompaniesWorke
d")
PercentSalaryHike = tf.feature_column.numeric_column("PercentSalaryHike"
)
TotalWorkingYears = tf.feature_column.numeric_column("TotalWorkingYears"
)
TrainingTimesLastYear = tf.feature_column.numeric_column("TrainingTimesL
astYear")
YearsAtCompany = tf.feature_column.numeric_column("YearsAtCompany")
YearsInCurrentRole = tf.feature_column.numeric_column("YearsInCurrentRol
e")
YearsSinceLastPromotion = tf.feature_column.numeric_column("YearsSinceLa
stPromotion")
YearsWithCurrManager = tf.feature_column.numeric_column("YearsWithCurrMa
nager")
```

In [65]:
```
feat_cols = [tf.feature_column.indicator_column(Attrition),
             tf.feature_column.indicator_column(BusinessTravel),
             tf.feature_column.indicator_column(Department),
             tf.feature_column.indicator_column(Education),
             tf.feature_column.indicator_column(EducationField),
             tf.feature_column.indicator_column(Gender),
             tf.feature_column.indicator_column(JobInvolvement),
             tf.feature_column.indicator_column(JobLevel),
             tf.feature_column.indicator_column(JobRole),
             tf.feature_column.indicator_column(MaritalStatus),
             tf.feature_column.indicator_column(OverTime),
             tf.feature_column.indicator_column(PerformanceRating),
             tf.feature_column.indicator_column(StockOptionLevel),
             tf.feature_column.indicator_column(WorkLifeBalance),
            Age, DailyRate, DistanceFromHome, HourlyRate, MonthlyIncome,
 MonthlyRate, NumCompaniesWorked,
            PercentSalaryHike, TotalWorkingYears, TrainingTimesLastYear,
 YearsAtCompany, YearsInCurrentRole,
            YearsSinceLastPromotion, YearsWithCurrManager]
```

Scaling of continuos columns

In [66]:
```
con_col = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate', 'Monthl
yIncome', 'MonthlyRate',
         'NumCompaniesWorked', 'PercentSalaryHike', 'TotalWorkingYears',
 'TrainingTimesLastYear',
         'YearsAtCompany', 'YearsInCurrentRole',
         'YearsSinceLastPromotion', 'YearsWithCurrManager']
```

In [67]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train[con_col] = scaler.fit_transform(X_train[con_col])
X_test[con_col] = scaler.fit_transform(X_test[con_col])
```

How the model will look like:
4 hidden layers, each has 96 neurons, small dropout to prevent overfitting, relu as an activator, mean squared error as loss function, softmax as an optimizer, 200 epochs.

In [183]:
```
input_func = tf.estimator.inputs.pandas_input_fn(x=X_train,y=y_train ,ba
tch_size=25,num_epochs=200,
                                            shuffle=True)
```

In [240]:
```
#Grid - learning_rate=0.001,
#model = tf.estimator.DNNRegressor(hidden_units=[6,6,6],feature_columns=
feature_columns)

model = tf.estimator.DNNClassifier(feature_columns=feat_cols, hidden_uni
ts=[96,96,96,96],
                                   optimizer=tf.train.AdamOptimizer(lear
ning_rate=0.001),
                                   activation_fn = tf.nn.relu,
                                   n_classes=4)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\a
lexa\AppData\Local\Temp\tmpzqaa51pj
INFO:tensorflow:Using config: {'_model_dir': 'C:\\Users\\alexa\\AppData\
\Local\\Temp\\tmpzqaa51pj', '_tf_random_seed': None, '_save_summary_step
s': 100, '_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
 '_session_config': None, '_keep_checkpoint_max': 5, '_keep_checkpoint_e
very_n_hours': 10000, '_log_step_count_steps': 100, '_train_distribute':
 None, '_service': None, '_cluster_spec': <tensorflow.python.training.se
rver_lib.ClusterSpec object at 0x000002AD90C1E940>, '_task_type': 'worke
r', '_task_id': 0, '_global_id_in_cluster': 0, '_master': '', '_evaluati
on_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker_r
eplicas': 1}
```

In [241]:
```
model.train(input_fn=input_func,steps=1000000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into C:\Users\alexa\AppData\Loc
al\Temp\tmpzqaa51pj\model.ckpt.
INFO:tensorflow:loss = 34.689518, step = 1
INFO:tensorflow:global_step/sec: 151.382
INFO:tensorflow:loss = 33.234344, step = 101 (0.661 sec)
INFO:tensorflow:global_step/sec: 258.519
INFO:tensorflow:loss = 31.972576, step = 201 (0.387 sec)
INFO:tensorflow:global_step/sec: 197.958
INFO:tensorflow:loss = 32.23688, step = 301 (0.506 sec)
INFO:tensorflow:global_step/sec: 239.874
INFO:tensorflow:loss = 13.2178135, step = 401 (0.417 sec)
INFO:tensorflow:global_step/sec: 169.893
INFO:tensorflow:loss = 5.809194, step = 501 (0.595 sec)
INFO:tensorflow:global_step/sec: 141.218
INFO:tensorflow:loss = 3.5631287, step = 601 (0.706 sec)
INFO:tensorflow:global_step/sec: 158.15
INFO:tensorflow:loss = 1.136996, step = 701 (0.629 sec)
INFO:tensorflow:global_step/sec: 145.949
INFO:tensorflow:loss = 0.43331262, step = 801 (0.691 sec)
INFO:tensorflow:global_step/sec: 151.637
INFO:tensorflow:loss = 0.15766326, step = 901 (0.650 sec)
INFO:tensorflow:global_step/sec: 184.995
INFO:tensorflow:loss = 0.13001205, step = 1001 (0.543 sec)
INFO:tensorflow:global_step/sec: 155.695
INFO:tensorflow:loss = 0.06971967, step = 1101 (0.649 sec)
INFO:tensorflow:global_step/sec: 155.241
INFO:tensorflow:loss = 0.06948236, step = 1201 (0.637 sec)
INFO:tensorflow:global_step/sec: 187.938
```

```
INFO:tensorflow:loss = 0.00019657437, step = 7401 (0.535 sec)
INFO:tensorflow:global_step/sec: 197.353
INFO:tensorflow:loss = 0.0001716603, step = 7501 (0.511 sec)
INFO:tensorflow:global_step/sec: 206.236
INFO:tensorflow:loss = 0.00021552885, step = 7601 (0.484 sec)
INFO:tensorflow:global_step/sec: 171.397
INFO:tensorflow:loss = 0.00025045624, step = 7701 (0.583 sec)
INFO:tensorflow:global_step/sec: 165.036
INFO:tensorflow:loss = 0.0001879914, step = 7801 (0.607 sec)
INFO:tensorflow:global_step/sec: 196.422
INFO:tensorflow:loss = 0.00017440198, step = 7901 (0.509 sec)
INFO:tensorflow:global_step/sec: 186.05
INFO:tensorflow:loss = 0.00019347534, step = 8001 (0.538 sec)
INFO:tensorflow:global_step/sec: 203.113
INFO:tensorflow:loss = 0.0001602166, step = 8101 (0.489 sec)
INFO:tensorflow:global_step/sec: 183.977
INFO:tensorflow:loss = 0.0001504411, step = 8201 (0.544 sec)
INFO:tensorflow:global_step/sec: 215.552
INFO:tensorflow:loss = 0.0001243344, step = 8301 (0.466 sec)
INFO:tensorflow:global_step/sec: 197.765
INFO:tensorflow:loss = 8.952583e-05, step = 8401 (0.506 sec)
INFO:tensorflow:global_step/sec: 163.836
INFO:tensorflow:loss = 0.00012516923, step = 8501 (0.607 sec)
INFO:tensorflow:global_step/sec: 135.586
INFO:tensorflow:loss = 8.201572e-05, step = 8601 (0.747 sec)
INFO:tensorflow:global_step/sec: 151.461
INFO:tensorflow:loss = 0.00011289062, step = 8701 (0.656 sec)
INFO:tensorflow:global_step/sec: 175.899
INFO:tensorflow:loss = 0.000105380605, step = 8801 (0.569 sec)
INFO:tensorflow:Saving checkpoints for 8816 into C:\Users\alexa\AppData\
Local\Temp\tmpzqaa51pj\model.ckpt.
INFO:tensorflow:Loss for final step: 9.226757e-05.
```

Out[241]: <tensorflow.python.estimator.canned.dnn.DNNClassifier at 0x2ad90c1e7b8>

Creating predictions

In [242]:
```
pred_fn = tf.estimator.inputs.pandas_input_fn(x=X_test,batch_size=len(X_
test),shuffle=False)
```

In [243]:
```
predictions = list(model.predict(input_fn=pred_fn))
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\alexa\AppData\Local\T
emp\tmpzqaa51pj\model.ckpt-8816
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

Confusion matrix

In [244]:
```
final_preds = []

for pred in predictions:
    final_preds.append(pred['class_ids'][0])
```

In [245]:
```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test,final_preds))
```

```
             precision    recall  f1-score   support

          0       0.22      0.22      0.22        69
          1       0.23      0.16      0.19        75
          2       0.28      0.34      0.31       107
          3       0.27      0.28      0.28       117

avg / total       0.26      0.26      0.26       368
```

Summary

With 1479 values it is only possible to classify 25-30% of employee satisfaction level. I suppose more entries are required to build a stable model

132 - 138

2.5. Convolutional Neural Network that is trained to distinguish emotions (Keras)

Convolutional Neural Network that is trained to distinguish emotions (Keras)

- Number of pictures: **36 076 (size 48 by 48)**
- Number of emotions to classify: **7**
- Dataset: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

Convert csv to images:
This is **not** a required step, but I had curiosity to see how images looked like before they were converted to csv. The initial file provided in kaggle - csv file.

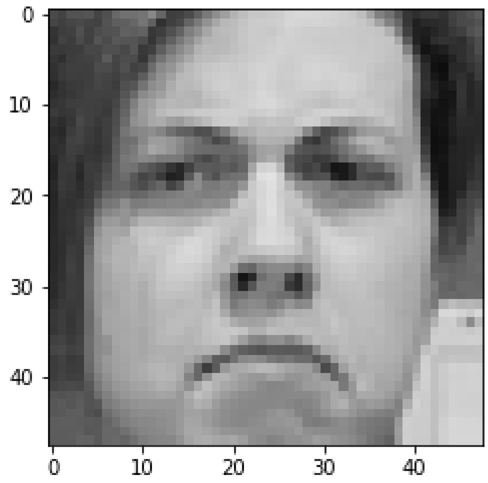
This code transforms csv back to images, saving the category of the image in the name of the images. This allows to distinguish categories easier in the future.

In [1]:
```
import pandas as pd
import numpy as np
from PIL import Image

df = pd.read_csv('fer2013.csv', sep=',')
h,w = 48,48 #setting width and height

for i in range(len(df)):
    input = df.iloc[i, 1]
    my_list = input.replace(" ", ",").split(",") #replace spaces and con
vert to list
    narray = np.asarray(my_list)
    img = Image.fromarray(np.uint8(narray.reshape(h,w)) , 'L')
    label = df['emotion'][i]
    img.save(str(label)+"_"+str(i)+'image.png')
```

Example of a image

In [175]:
```
from matplotlib.pyplot import imshow
import numpy as np
from PIL import Image

%matplotlib inline
pil_im = Image.open('data/train/0/0_40image.png', 'r').convert('RGB')
imshow(np.asarray(pil_im))
```

Out[175]:
```
<matplotlib.image.AxesImage at 0x24b8b0c8eb8>
```



Import Kaggle

In [8]:
```
import warnings
warnings.filterwarnings('ignore')

from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

Build CNN Model

Initialising the CNN

In [9]:
```
classifier = Sequential()
```

First convolutional layer
64 - number of feature detectors of size 5 by 5
Relu as an activation function
Even though pictures initialy were white-and-black, the number of channels is 3. Converting csv back to pictures applied channel 3

In [10]:
```
classifier.add(Conv2D(64, (5, 5), input_shape = (48, 48, 3), activation
= 'relu', data_format="channels_last")) #some mistake, needs 3, but pict
ures are not colorful
```

Pooling Step
Size of a stride: 2 by 2

In [11]:
```
classifier.add(MaxPooling2D(pool_size = (2, 2)))
```

Second convolutional and pooling layers
64 - number of feature detectors of size 5 by 5
Relu as an activation function
Size of a stride: 3 by 3

In [12]:
```
classifier.add(Conv2D(64, (5, 5), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (3, 3)))
```

Flattening

In [13]:
```
classifier.add(Flatten())
```

Dense layers
units = 7 - number of categories
softmax is better as an activation function for models with >2 categories

In [14]:
```
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 7, activation = 'softmax'))
```

Compile the CNN

In [15]:
```
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy'
, metrics = ['accuracy'])
```

Fitting images into a model

Rotating, rescalling, zooming images - creating new pictures from old ones;enriching and preventing overfitting

In [17]:
```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255, #all pixels between
 0 and 1
                                   shear_range = 0.2,
                                   zoom_range = 0.2, #zooming
                                   horizontal_flip = True) #flipping hor
izontally
                                    #there are more transformations in d
ocs
```

Rescale test data

In [19]:
```
test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [20]:
```
training_set = train_datagen.flow_from_directory('data/train',
                                                 target_size = (48, 48),
 #size of images that are expected
                                                 batch_size = 10, #after
 what number of phoos weights will be updated
                                                 class_mode = 'categoric
al')

test_set = test_datagen.flow_from_directory('data/test',
                                            target_size = (48, 48),
                                            batch_size = 10,
                                            class_mode = 'categorical')
```

```
Found 29300 images belonging to 7 classes.
Found 6770 images belonging to 7 classes.
```

In [21]:

```
classifier.fit_generator(training_set,
                         steps_per_epoch = (29300/10), #size of a batch
                         epochs = 25,
                         validation_data = test_set,
                         validation_steps = (6776/10))
```

```
Epoch 1/25
2930/2930 [==============================] - 1016s 347ms/step - loss: 1.
6488 - acc: 0.3429 - val_loss: 1.4820 - val_acc: 0.4173
Epoch 2/25
2930/2930 [==============================] - 331s 113ms/step - loss: 1.4
503 - acc: 0.4447 - val_loss: 1.3899 - val_acc: 0.4616
Epoch 3/25
2930/2930 [==============================] - 330s 113ms/step - loss: 1.3
822 - acc: 0.4646 - val_loss: 1.3504 - val_acc: 0.4716
Epoch 4/25
2930/2930 [==============================] - 295s 101ms/step - loss: 1.3
344 - acc: 0.4907 - val_loss: 1.3146 - val_acc: 0.4870
Epoch 5/25
2930/2930 [==============================] - 294s 100ms/step - loss: 1.2
962 - acc: 0.5079 - val_loss: 1.2785 - val_acc: 0.5034
Epoch 6/25
2930/2930 [==============================] - 312s 107ms/step - loss: 1.2
620 - acc: 0.5178 - val_loss: 1.2941 - val_acc: 0.5078
Epoch 7/25
2930/2930 [==============================] - 307s 105ms/step - loss: 1.2
406 - acc: 0.5257 - val_loss: 1.2525 - val_acc: 0.5164
Epoch 8/25
2930/2930 [==============================] - 292s 100ms/step - loss: 1.2
210 - acc: 0.5352 - val_loss: 1.2758 - val_acc: 0.5194
Epoch 9/25
2930/2930 [==============================] - 291s 99ms/step - loss: 1.19
60 - acc: 0.5473 - val_loss: 1.2684 - val_acc: 0.5102
Epoch 10/25
2930/2930 [==============================] - 285s 97ms/step - loss: 1.18
37 - acc: 0.5509 - val_loss: 1.2590 - val_acc: 0.5130
Epoch 11/25
2930/2930 [==============================] - 289s 99ms/step - loss: 1.16
67 - acc: 0.5574 - val_loss: 1.2588 - val_acc: 0.5309
Epoch 12/25
2930/2930 [==============================] - 286s 98ms/step - loss: 1.14
70 - acc: 0.5666 - val_loss: 1.2179 - val_acc: 0.5365
Epoch 13/25
2930/2930 [==============================] - 286s 98ms/step - loss: 1.13
86 - acc: 0.5681 - val_loss: 1.2311 - val_acc: 0.5285
Epoch 14/25
2930/2930 [==============================] - 532s 182ms/step - loss: 1.1
201 - acc: 0.5746 - val_loss: 1.2576 - val_acc: 0.5233
Epoch 15/25
2930/2930 [==============================] - 325s 111ms/step - loss: 1.1
071 - acc: 0.5810 - val_loss: 1.2390 - val_acc: 0.5300
Epoch 16/25
2930/2930 [==============================] - 301s 103ms/step - loss: 1.0
951 - acc: 0.5848 - val_loss: 1.2535 - val_acc: 0.5211
Epoch 17/25
2930/2930 [==============================] - 289s 99ms/step - loss: 1.08
69 - acc: 0.5857 - val_loss: 1.2265 - val_acc: 0.5390
Epoch 18/25
2930/2930 [==============================] - 289s 99ms/step - loss: 1.07
15 - acc: 0.5945 - val_loss: 1.2855 - val_acc: 0.5349
```

```
Epoch 19/25
2930/2930 [==============================] - 290s 99ms/step - loss: 1.07
00 - acc: 0.5978 - val_loss: 1.2452 - val_acc: 0.5470
Epoch 20/25
2930/2930 [==============================] - 293s 100ms/step - loss: 1.0
572 - acc: 0.6032 - val_loss: 1.2430 - val_acc: 0.5482
Epoch 21/25
2930/2930 [==============================] - 291s 99ms/step - loss: 1.04
38 - acc: 0.6046 - val_loss: 1.2584 - val_acc: 0.5433
Epoch 22/25
2930/2930 [==============================] - 993s 339ms/step - loss: 1.0
367 - acc: 0.6091 - val_loss: 1.2542 - val_acc: 0.5335
Epoch 23/25
2930/2930 [==============================] - 343s 117ms/step - loss: 1.0
275 - acc: 0.6129 - val_loss: 1.2680 - val_acc: 0.5439
Epoch 24/25
2930/2930 [==============================] - 327s 112ms/step - loss: 1.0
200 - acc: 0.6115 - val_loss: 1.2630 - val_acc: 0.5493
Epoch 25/25
2930/2930 [==============================] - 310s 106ms/step - loss: 1.0
121 - acc: 0.6192 - val_loss: 1.2842 - val_acc: 0.5340
```

Out[21]:
```
<keras.callbacks.History at 0x24b86e914a8>
```

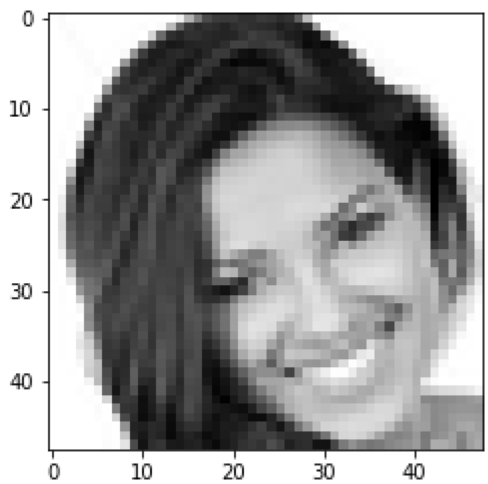
The accuracy of a model is aroung 54%. I suppose more epochs, more complicated model can improve the quality. Moreover, some emotions (Anger, Disgust) are similar and photos of these categories look alike, so it is hard to distinguish some of categories from one another.

Predict category for a single image

Now, after the model was trained, a prediction on new images can be made. A single image from test data will be categorized to find what emotion is presented on this image.

In [176]:
```
image_example = Image.open('data/test/3/3_13991image.png', 'r').convert(
'RGB')
imshow(np.asarray(image_example))
```

Out[176]:
```
<matplotlib.image.AxesImage at 0x24b8b119320>
```



This is an image of a label "Happy"

Reshape the image

In [166]:
```
from scipy.misc import imread,imresize

x=imread('3_13991image.png',mode='RGB')
x=imresize(x,(48,48))
x=np.invert(x)
x=x.reshape(-1,48,48,3)
```

In [167]:
```
x = x/255.
```

Predict emotion on the image

In [168]:
```
prediction = classifier.predict(x)
```

Chunk of code that tranforms index in the emotion name

In [169]:
```
#(0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)
labels = {
    0: "Angry",
    1: "Disgust",
    2: "Fear",
    3: "Happy",
    4: "Sad",
    5: "Surprise",
    6: "Neutral"
}
```

Index of the prediction

In [171]:
```
np.argmax(prediction)
```

Out[171]: 3

Transform index to label/emotion

In [172]:
```
labels.get(np.argmax(prediction))
```

Out[172]: 'Happy'

139 -144

2.6. Apply RNN LSTM to create a model for sentiment analysis of Amazon reviews (Keras)

Apply RNN LSTM to create a model for sentiment analysis of Amazon reviews (Keras)

- Number of reviews: >568 000 (the model was built on 30 000, laptop can not handle all reviews)
- Source: https://www.kaggle.com/snap/amazon-fine-food-reviews/data

Import required libraries

In [1]:
```
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re
```

```
Using TensorFlow backend.
```

Import the dataset and keep only required columns

In [2]:
```
data = pd.read_csv('Reviews.csv')
# Keeping only the neccessary columns
data = data[['Score','Text']]
```

Set a limit (laptop is not able to process all of reviews)
To balance the input, the same number of positive and negative reviews are taken

In [3]:
```
#LIMIT - 15 000 positive and 15 000 negative
data_pos = data.loc[data['Score'] >= 4][:15000]
data_neg = data.loc[data['Score'] <= 2][:15000]
```

In [4]:
```
df = pd.concat([data_pos, data_neg])
```

Shufle rows

In [5]:
```
df = df.sample(frac=1).reset_index(drop=True)
```

In [6]:
```
df.head()
```

Out[6]:

	Score	Text
0	5	Are you sick of the regular flavors? Plain, ap...
1	4	These 'ramen-like' noodles will be a tasty add...

2	1	These Emerils Gourmet Coffee, Emeril's Big Eas...
3	4	These bars have the texture of a soft cookie, ...
4	5	My cats both LOVE this food. I've had varying...

Create types of reviews by converting numbers to (1,2 - negative, 4,5 - positive. Neutral are removed)

In [7]:
```
df.loc[df['Score'] > 3, 'Type'] = "Positive"
df.loc[df['Score'] < 3, 'Type'] = "Negative"
```

In [8]:
```
df.head()
```

Out[8]:

	Score	Text	Type
0	5	Are you sick of the regular flavors? Plain, ap...	Positive
1	4	These 'ramen-like' noodles will be a tasty add...	Positive
2	1	These Emerils Gourmet Coffee, Emeril's Big Eas...	Negative
3	4	These bars have the texture of a soft cookie, ...	Positive
4	5	My cats both LOVE this food. I've had varying...	Positive

Convert every word to lower cases and remove any non-digit or non-letter symbols

In [9]:
```
df['Text'] = df['Text'].apply(lambda x: x.lower()) #lower cases
df['Text'] = df['Text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','',x)))
```

In [10]:
```
print("Number of positive reviews: ", len(df[ df['Type'] == 'Positive'])
)
print("Number of negative reviews: ", len(df[ df['Type'] == 'Negative'])
)
```

```
Number of positive reviews:  15000
Number of negative reviews:  15000
```

Tokenizer is used to vectorize the text and convert it into sequence of integers after restricting the tokenizer to use only top most common 2000 words. Pad_sequences is used to convert the sequences into 2-D numpy array.

In [11]:
```
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(df['Text'].values)
X = tokenizer.texts_to_sequences(df['Text'].values)
X = pad_sequences(X)
```

Build RNN with Keras

In [12]:
```
embed_dim = 128
lstm_out = 196
```

```
In [13]: model = Sequential()
model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2,activation='sigmoid'))
#model.compile(loss = 'categorical_crossentropy', optimizer='adam',metri
cs = ['accuracy'])
model.compile(loss = 'binary_crossentropy', optimizer='adam',metrics = [
'accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1258, 128)         256000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 1258, 128)         0
_________________________________________________________________
lstm_1 (LSTM)                (None, 196)               254800
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 394
=================================================================
Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [14]: Y = pd.get_dummies(df['Type']).values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2
5, random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
(22500, 1258) (22500, 2)
(7500, 1258) (7500, 2)
```

```
In [15]: batch_size = 64
model.fit(X_train, Y_train, epochs = 3, batch_size=batch_size, verbose =
 2)
```

```
Epoch 1/3
 - 4534s - loss: 0.4303 - acc: 0.7998
Epoch 2/3
 - 4206s - loss: 0.3372 - acc: 0.8658
Epoch 3/3
 - 4131s - loss: 0.2879 - acc: 0.8844
```

```
Out[15]: <keras.callbacks.History at 0x2236492c358>
```

```
In [16]: validation_size = 1500

X_validate = X_test[-validation_size:]
Y_validate = Y_test[-validation_size:]
X_test = X_test[:-validation_size]
Y_test = Y_test[:-validation_size]
score,acc = model.evaluate(X_test, Y_test, verbose = 2, batch_size = bat
ch_size)
print("score: %.2f" % (score))
print("acc: %.2f" % (acc))
```

```
score: 0.28
```

```
acc: 0.88
```

In [17]:
```
pos_cnt, neg_cnt, pos_correct, neg_correct = 0, 0, 0, 0
for x in range(len(X_validate)):

    result = model.predict(X_validate[x].reshape(1,X_test.shape[1]),batc
h_size=1,verbose = 2)[0]

    if np.argmax(result) == np.argmax(Y_validate[x]):
        if np.argmax(Y_validate[x]) == 0:
            neg_correct += 1
        else:
            pos_correct += 1

    if np.argmax(Y_validate[x]) == 0:
        neg_cnt += 1
    else:
        pos_cnt += 1



print("pos_acc", pos_correct/pos_cnt*100, "%")
print("neg_acc", neg_correct/neg_cnt*100, "%")
```

```
pos_acc 85.92297476759629 %
neg_acc 88.08567603748327 %
```

Model predicts positive review with an accuracy of 86% and negative reviews with 88%

Example

Some review for the model proving

In [27]:
```
rev = [df["Text"][50]]
#vectorizing the text by the pre-fitted tokenizer instance
print(rev)
```

```
['i love these crackers and decided its best just to buy in bulk  the cr
ackers came fast and they are in tact  very fresh and delicious']
```

In [28]:
```
rev = tokenizer.texts_to_sequences(rev)
#padding the tweet to have exactly the same shape as `embedding_2` input
rev = pad_sequences(rev, maxlen=1258, dtype='int32', value=0)
sentiment = model.predict(rev,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("negative")
elif (np.argmax(sentiment) == 1):
    print("positive")
```

```
positive
```

Model says that this is a positive review

In [30]:
```
rev = [df["Text"][99]]
#vectorizing the text by the pre-fitted tokenizer instance
print(rev)
```

```
['i just received these today for my wife  they were the most horrible t
```

```
asting coffee ever  this wasnt just a case of not caring for this  the c
offee was down right nasty  both my wife and i thought so  i dont know i
f we just got a bad batch or what  but we wont be ordering this again']
```

In [31]:

```
rev = tokenizer.texts_to_sequences(rev)
#padding the tweet to have exactly the same shape as `embedding_2` input
rev = pad_sequences(rev, maxlen=1258, dtype='int32', value=0)
sentiment = model.predict(rev,batch_size=1,verbose = 2)[0]
if(np.argmax(sentiment) == 0):
    print("negative")
elif (np.argmax(sentiment) == 1):
    print("positive")
```

```
negative
```

Model indicates that this is a negative review and it is right

145 -151

2.7. Recurrent Neural Network (LSTM) that generates haiku (Japanese poems) in Keras/Tensorflow

Recurrent Neural Network (LSTM) that generates haiku (Japanese poems) in Keras/Tensorflow

Information about the dataset

- 10 000 haikus of Issa were used to train RNN
- Poems were taken from this website: http://haikuguy.com/issa/searchenglish2.php

Keras implementation

Importing main libraries

In [1]:
```
import sys
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
import warnings
warnings.filterwarnings('ignore')
```

```
Using TensorFlow backend.
```

Text loading, opening and coverting it to lowercase

In [2]:
```
filename = "issa.txt"
raw_text = open(filename).read()
raw_text = raw_text.lower()
```

Creating unique id for every character

In [3]:
```
chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
```

In [4]:
```
n_chars = len(raw_text)
print("Total number of characters in the text: ", n_chars)
```

```
Total number of characters in the text:  541081
```

In [5]:
```
n_vocab = len(chars)
print("Total number of unique characters: ", n_vocab)
```

```
Total number of unique characters:  36
```

Preparing the input by encoding characters, dividing text by 54 characters (creating inputs, every input is 54 characters)

54 - average number of characters in one haiku (54 000/10 000)

In [6]:
```
seq_length = 54 #average
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
    seq_in = raw_text[i:i + seq_length]
    seq_out = raw_text[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print("Total Patterns: ", n_patterns)
```

```
Total Patterns:  541027
```

In [7]:
```
len(dataX[1])
```

Out[7]: 54

Input is transformed into the form [samples, time steps, features] expected by an LSTM network.

Then imput is scaled from 0 to 1.

Lastly the output pattern is OneHotEncoded

Reshape X to be [samples, time steps, features]

In [8]:
```
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
```

Normalization

In [9]:
```
X = X / float(n_vocab)
```

One hot encode the output variable

In [10]:
```
y = np_utils.to_categorical(dataY)
```

The LSTM model:
1 layer, 256 neurons
Dropout - 0.2
"Softmax" activation function

In [11]:
```
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Define checkpoint

In [12]:
```
filepath="weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_b
est_only=True, mode='min')
callbacks_list = [checkpoint]
```

Fit data

In [ ]:
```
model.fit(X, y, epochs=20, batch_size=128, callbacks=callbacks_list)
```

The pre-trained model is loaded:

In [13]:
```
filename = "weights-improvement-14-1.7628.hdf5"
model.load_weights(filename)
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Code to convert encoded characters back

In [14]:
```
int_to_char = dict((i, c) for i, c in enumerate(chars))
```

In [15]:
```
start = numpy.random.randint(0, len(dataX)-1)
pattern = dataX[start]
print("Seed:")
print("\"", ''.join([int_to_char[value] for value in pattern]), "\"")
# generate characters
for i in range(50):
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(n_vocab)
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    sys.stdout.write(result)
    pattern.append(index)
    pattern = pattern[1:len(pattern)]
```

```
Seed:
" ce field
the greatest sight of all!
summer's early daw "
n


the siae field soow
fer she sorw falls...
poum
```

Tensorflow implementation

Importing main libraries

In [16]:
```
import tensorflow as tf
import numpy as np
```

Set parameters

In [17]:
```
#set hyperparameters
max_len = 50
step = 2
num_units = 256
```

```
learning_rate = 0.001
batch_size = 128
epoch = 20
temperature = 0.5
```

Text loading, opening and coverting it to lowercase

In [18]:
```
filename = "issa.txt"
text = open(filename, 'r').read()
text = text.lower()
```

Creating unique id for every character

In [19]:
```
unique_chars = list(set(text))
len_unique_chars = len(unique_chars)

input_chars = []
output_char = []

for i in range(0, len(text) - max_len, step):
    input_chars.append(text[i:i+max_len])
    output_char.append(text[i+max_len])

train_data = np.zeros((len(input_chars), max_len, len_unique_chars))
target_data = np.zeros((len(input_chars), len_unique_chars))

for i , each in enumerate(input_chars):
    for j, char in enumerate(each):
        train_data[i, j, unique_chars.index(char)] = 1
    target_data[i, unique_chars.index(output_char[i])] = 1
```

Define RNN

In [20]:
```
def rnn(x, weight, bias, len_unique_chars):

    x = tf.transpose(x, [1, 0, 2])
    x = tf.reshape(x, [-1, len_unique_chars])
    x = tf.split(x, max_len, 0)

    cell = tf.contrib.rnn.BasicLSTMCell(num_units, forget_bias=1.0)
    outputs, states = tf.contrib.rnn.static_rnn(cell, x, dtype=tf.float3
2)
    prediction = tf.matmul(outputs[-1], weight) + bias
    return prediction
```

Helper function to sample an index from a probability array

In [21]:
```
def sample(predicted):
    '''

    '''
    exp_predicted = np.exp(predicted/temperature)
    predicted = exp_predicted / np.sum(exp_predicted)
    probabilities = np.random.multinomial(1, predicted, 1)
    return probabilities
```

In [22]:
```
x = tf.placeholder("float", [None, max_len, len_unique_chars])
y = tf.placeholder("float", [None, len_unique_chars])
weight = tf.Variable(tf.random_normal([num_units, len_unique_chars]))
bias = tf.Variable(tf.random_normal([len_unique_chars]))

prediction = rnn(x, weight, bias, len_unique_chars)
softmax = tf.nn.softmax_cross_entropy_with_logits(logits=prediction, lab
els=y)
cost = tf.reduce_mean(softmax)
optimizer = tf.train.RMSPropOptimizer(learning_rate=learning_rate).minim
ize(cost)

init_op = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init_op)

num_batches = int(len(train_data)/batch_size)
```

```
WARNING:tensorflow:From <ipython-input-22-e162b3960fc5>:7: softmax_cross
_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated a
nd will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See @{tf.nn.softmax_cross_entropy_with_logits_v2}.
```

In [ ]:
```
for i in range(epoch):
    print("Epoch {0}/{1}".format(i+1, epoch))
    count = 0
    for _ in range(num_batches):
        train_batch, target_batch = train_data[count:count+batch_size],
target_data[count:count+batch_size]
        count += batch_size
        sess.run([optimizer] ,feed_dict={x:train_batch, y:target_batch})

    #get on of training set as seed
    seed = train_batch[:1:]

    #to print the seed 40 characters
    seed_chars = ''
    for each in seed[0]:
        seed_chars += unique_chars[np.where(each == max(each))[0][0]]
    print("Seed:", seed_chars)

    #predict next 100 characters
    for i in range(100):
        if i > 0:
            remove_fist_char = seed[:,1:,:]
            seed = np.append(remove_fist_char, np.reshape(probabilities,
 [1, 1, len_unique_chars]), axis=1)
        predicted = sess.run([prediction], feed_dict = {x:seed})
        predicted = np.asarray(predicted[0]).astype('float64')[0]
        probabilities = sample(predicted)
        predicted_chars = unique_chars[np.argmax(probabilities)]
        seed_chars += predicted_chars
    print('Result:', seed_chars)
sess.close()
```

Some generated haikus

three men
use it for a pillow...
green rice field

willow tree
catch the blossom-scented wind
of the cherry

honeybees--
but right next door
hornets

following
the setting sun...
a frog

Social Data Mining and Sentiment analysis

153 - 167

3.1. Twitter API: extract tweets of Trump and Trudeau to compare their activity on Twitter (NLTK/Regex)

Twitter API: extract tweets of Trump and Trudeau to compare their activity on Twitter (NLTK/Regex)

Data extraction with API

- Firstly, Twitter does not allow to extract more than 200 tweets per one time. This issue can be solved by adding **“max_id”** option. Each iteration will extract different tweets because max_id is changing in the end of each iteration
- Secondly, by default twitter extracts a truncated version of a tweet. To force twitter to output the full version of tweet, an option of the call should be set to **“tweet_mode=extended”**;
  To compare different attributes of Trumps’ and Trudeaus’ tweets, python language, some libraries(pandas, numpy) and regular expressions were used.
- Thirdly, **"include_rts"** is set to "False" to prevent retweets from being saved. Only tweets of autors are analysed

Import required libraries

In [30]:
```
import requests
import pandas as pd
import re
from nltk import FreqDist
import operator
import warnings
warnings.filterwarnings('ignore')
```

Extract data with the help of API(Trudeau)

First loop is needed to initialize max_id, second loop updates it constantly

In [4]:
```
headers = {
    'Authorization': 'Bearer AAAAAAAAAAAAAAAAAAAAADmO4QAAAAAA1l35b3JfTyY
e8rDAX0q7nhR%2BBis%3D90KK4CMhxUBYfLFslUyJmusiEVnhBRLVmIG4Nnb2b6R2SlVxmU'
,
}

params = (
    ('screen_name', 'JustinTrudeau'),
    ('include_rts', 'false'),
    ('count', '200'),
    ('tweet_mode', 'extended'),
)

response = requests.get('https://api.twitter.com:443/1.1/statuses/user_t
imeline.json', headers=headers, params=params)
response_json=response.json()
max_id = response_json[-1]['id']

import requests
#1500 - number of tweets
while len(response_json)<1500:
    headers = {
```

```
        'Authorization': 'Bearer AAAAAAAAAAAAAAAAAAAAADmO4QAAAAAA1l35b3J
fTyYe8rDAX0q7nhR%2BBis%3D90KK4CMhxUBYfLFslUyJmusiEVnhBRLVmIG4Nnb2b6R2SlV
xmU',
    }

    params = (
        ('screen_name', 'JustinTrudeau'),
        ('include_rts', 'false'),
        ('count', '200'),
        ('tweet_mode', 'extended'),
        ('max_id', max_id),
    )

    response = requests.get('https://api.twitter.com:443/1.1/statuses/us
er_timeline.json', headers=headers, params=params)
    response_json_new=response.json()
    response_json  += response_json_new
    max_id = response_json[-1]['id']
```

Create a dataframe of the date of a tweet and its' content (text)

In [5]:
```
created_at_list_trud = []
for time in range(len(response_json)):
    created_at_list_trud.append(response_json[time]["created_at"])

texts_trud = []
for el in range(len(response_json)):
    texts_trud.append(response_json[el]["full_text"])

df_trud = pd.DataFrame(
    {'Text': texts_trud,
     'created_at': created_at_list_trud
    })
```

Required to see more text of tweet in Jupyter

In [6]:
```
pd.options.display.max_colwidth = 140
pd.options.display.max_colwidth
```

Out[6]: 140

In [7]:
```
df_trud.head()
```

Out[7]:

	Text	created_at
0	Innovators, researchers, and entrepreneurs move Canada forward - creating good jobs and growing our economy. Here’s how we’re making sur...	Fri Jul 27 16:10:24 +0000 2018
1	Les innovateurs, les chercheurs et les entrepreneurs font avancer le Canada : ils créent de bons emplois et font croître notre économie....	Fri Jul 27 16:09:50 +0000 2018
2	Today, we honour the brave Canadians who fought for freedom and democracy during the Korean War: https://t.co/4cVOhJ4PFz	Fri Jul 27 12:07:57 +0000 2018
		Fri Jul 27

3	Aujourd’hui, nous rendons hommage aux braves Canadiens qui se sont battus au nom de la liberté et de la démocratie pendant la guerre de ...	12:07:45 +0000 2018
4	Hardworking Canadians shouldn’t have to worry about having enough money to retire. That’s why we’ve improved and strengthened the Canada...	Thu Jul 26 23:01:13 +0000 2018

Extract data with the help of API(Trump)

In [8]:

```
headers = {
    'Authorization': 'Bearer AAAAAAAAAAAAAAAAAAAAADmO4QAAAAAA1l35b3JfTyY
e8rDAX0q7nhR%2BBis%3D90KK4CMhxUBYfLFslUyJmusiEVnhBRLVmIG4Nnb2b6R2SlVxmU'
,
}

params = (
    ('screen_name', 'realDonaldTrump'),
    ('include_rts', 'false'),
    ('count', '200'),
    ('tweet_mode', 'extended'),
)

response = requests.get('https://api.twitter.com:443/1.1/statuses/user_t
imeline.json', headers=headers, params=params)
response_json_trump=response.json()
#max id to insert into next loop
max_id = response_json_trump[-1]['id']

import requests
#include_rts is set to False, but count includes retweets. so the easies
t way to get some number of tweets - while loop
while len(response_json_trump)<1500:
    headers = {
        'Authorization': 'Bearer AAAAAAAAAAAAAAAAAAAAADmO4QAAAAAA1l35b3J
fTyYe8rDAX0q7nhR%2BBis%3D90KK4CMhxUBYfLFslUyJmusiEVnhBRLVmIG4Nnb2b6R2SlV
xmU',
    }

    params = (
        ('screen_name', 'realDonaldTrump'),
        ('include_rts', 'false'),
        ('count', '200'),
        ('tweet_mode', 'extended'),
        ('max_id', max_id),
    )

    response = requests.get('https://api.twitter.com:443/1.1/statuses/us
er_timeline.json', headers=headers, params=params)
    response_json_new=response.json()
    response_json_trump  += response_json_new
    max_id = response_json_trump[-1]['id']
```

Create a dataframe of the date of a tweet and its' content (text)

In [9]:

```
created_at_list_trump = []
for time in range(len(response_json_trump)):
    created_at_list_trump.append(response_json_trump[time]["created_at"]
```

```
)

texts_trump = []
for el in range(len(response_json_trump)):
    texts_trump.append(response_json_trump[el]["full_text"])


df_trump = pd.DataFrame(
    {'Text': texts_trump,
     'created_at': created_at_list_trump
    })
```

In [10]:
```
df_trump.head()
```

Out[10]:

	Text	created_at
0	We must have Border Security, get rid of Chain, Lottery, Catch & Release Sanctuary Cities - go to Merit based Immigration. Protect I...	Mon Jul 30 11:57:34 +0000 2018
1	....Also, why is Mueller only appointing Angry Dems, some of whom have worked for Crooked Hillary, others, including himself, have worke...	Sun Jul 29 20:20:39 +0000 2018
2	Is Robert Mueller ever going to release his conflicts of interest with respect to President Trump, including the fact that we had a very...	Sun Jul 29 20:12:15 +0000 2018
3	There is No Collusion! The Robert Mueller Rigged Witch Hunt, headed now by 17 (increased from 13, including an Obama White House lawyer)...	Sun Jul 29 19:35:14 +0000 2018
4	...and the Amazon Washington Post do nothing but write bad stories even on very positive achievements - and they will never change!	Sun Jul 29 19:09:19 +0000 2018

Limit number of tweets to 1500

In [11]:
```
df_trud = df_trud[:1500]
df_trump = df_trump[:1500]
```

Questions to answer

- What is the average number of times they tweet per day?
- What day of the week do they tweet most frequently?
- Ratio of the word "fake" vs "real"?
- Ratio of the word "good" vs "bad"?
- How many times a week does Trump use his surname?
- Peak times they tweet?
- Number of times they use their countries name in their tweets?
- What is the average number of words per tweet?
- What are the most popular words used by Trump and Trudeau?

What is the average number of times they tweet per day?

In order to answers this question, datasets should be grouped on a daily basis to count number of tweets per day. It was done by extracting part of a date, converting this part to datetime and counting number of tweets per day on average.

In [12]:
```
#extract date from text
for i, row in df_trump.iterrows():
    value = df_trump['created_at'][i][4:10] +", "+ df_trump['created_at'
][i][-4:]
    df_trump.set_value(i,'date', value)

#convert to datetime
df_trump['date_conv'] = pd.to_datetime(df_trump['date'])

#group on a daily basis, count tweets
df_trump_1 = df_trump.resample('D', on='date_conv').count()

#Trudeau
#extract date from text
for i, row in df_trud.iterrows():
    value = df_trud['created_at'][i][4:10] +", "+ df_trud['created_at'][
i][-4:]
    df_trud.set_value(i,'date', value)

#convert to datetime
df_trud['date_conv'] = pd.to_datetime(df_trud['date'])

#group on a daily basis, count tweets
df_trud_1 = df_trud.resample('D', on='date_conv').count()
```

In [13]:
```
print("Average number of tweets per day of Trump: ", "%.2f" % df_trump_1
["Text"].mean())
```

```
Average number of tweets per day of Trump:  7.21
```

In [14]:
```
print("Average number of tweets per day of Trudeau: ", "%.2f" % df_trud_
1["Text"].mean())
```

```
Average number of tweets per day of Trudeau:  6.82
```

Consequently, on average, Trump posts more tweets by 1 per day. (Even though Trudeau posts same tweets in English and in French)

What day of the week do they tweet most frequently?

To answer this question the same method will be used, the only difference that now the day of the week will be extracted and grouped by the variable that represents this day of the week

In [15]:
```
#extract day of week from text
for i, row in df_trump.iterrows():
    df_trump.set_value(i,'dayofweek', df_trump['created_at'][i][:3])

#group by day of a week to find the most popular day to tweet
df_trump_2 = df_trump.groupby('dayofweek').count()

#Trud
#extract day of week from text
```

```
for i, row in df_trud.iterrows():
    df_trud.set_value(i,'dayofweek', df_trud['created_at'][i][:3])

#group by day of a week to find the most popular day to tweet
df_trud_2 = df_trud.groupby('dayofweek').count()
```

Trudeau

In [16]:
```
df_trud_2.sort_values(by=['date_conv'], ascending=False)['Text']
```

Out[16]:
```
dayofweek
Wed    292
Thu    279
Fri    248
Tue    244
Mon    162
Sun    138
Sat    137
Name: Text, dtype: int64
```

Trump

In [17]:
```
df_trump_2.sort_values(by=['date_conv'], ascending=False)['Text']
```

Out[17]:
```
dayofweek
Wed    269
Thu    247
Tue    220
Fri    218
Sat    197
Mon    192
Sun    157
Name: Text, dtype: int64
```

For both Trump and Trudeau, Wednesday and Thursday are the most popular days to tweet.

Ratio of the word "fake" vs "real"?

Regex was used to find out how many times a word “fake” and a word “real” appeared in tweets. The regex “findall” code was looped across every tweet to count instances of these words

Trump

In [18]:
```
count_real = 0
count_fake = 0

#count number of occurences for both words
for element in range(0, df_trump['Text'].count()):
    text = df_trump['Text'][element]
    extr_real = re.findall(r'(real)', text)
    extr_fake = re.findall(r'(fake)', text)
    for el in extr_real:
        count_real +=1
    for el in extr_fake:
        count_fake +=1
```

```
print("Trump: Counts of 'real'", count_real,
      "\nCounts of 'fake'", count_fake,
      "\nProportion of fake to real",
      count_fake/count_real)
```

```
Trump: Counts of 'real' 67
Counts of 'fake' 1
Proportion of fake to real 0.014925373134328358
```

Trudeau

In [19]:
```
count_real = 0
count_fake = 0

#count number of occurences for both words
for element in range(0, df_trud['Text'].count()):
    text = df_trud['Text'][element]
    extr_real = re.findall(r'(real)', text)
    extr_fake = re.findall(r'(fake)', text)
    for el in extr_real:
        count_real +=1
    for el in extr_fake:
        count_fake +=1

print("Trudeau: Counts of 'real'", count_real,
      "\nCounts of 'fake'", count_fake,
      "\nProportion of fake to real",
      count_fake/count_real)
```

```
Trudeau: Counts of 'real' 20
Counts of 'fake' 0
Proportion of fake to real 0.0
```

In summary, the proportion of using these words for Trump in 1 to 68, while Trudeau did not use a word ‘fake’ in last 1500 tweets at all.

Ratio of the word "good" vs "bad"?

In [20]:
```
count_good = 0
count_bad = 0

for element in range(0, df_trump['Text']
                     .count()):
    text = df_trump['Text'][element]
    extr_good = re.findall(r'(good)', text)
    extr_bad = re.findall(r'(bad)', text)
    for el in extr_good:
        count_good +=1
    for el in extr_bad:
        count_bad +=1

print("Trump: Counts of 'good'", count_good,
      "\nCounts of 'bad'", count_bad,
      "\nProportion of bad to good",
      count_bad/count_good)
```

```
Trump: Counts of 'good' 94
Counts of 'bad' 63
Proportion of bad to good 0.6702127659574468
```

In [21]:
```python
count_good = 0
count_bad = 0

for element in range(0, df_trud['Text']
                     .count()):
    text = df_trud['Text'][element]
    extr_good = re.findall(r'(good)', text)
    extr_bad = re.findall(r'(bad)', text)
    for el in extr_good:
        count_good +=1
    for el in extr_bad:
        count_bad +=1

print("Trudeau: Counts of 'good'", count_good,
      "\nCounts of 'bad'", count_bad,
      "\nProportion of bad to good",
      count_bad/count_good)
```

```
Trudeau: Counts of 'good' 46
Counts of 'bad' 7
Proportion of bad to good 0.15217391304347827
```

Trump uses a word “good” by 67% more often than a word “bad”, while Trudeau uses "good" 15% more often.

How many times a week does Trump use his surname?

1. A new column was created that indicates the week number of a tweet
2. A regex was used to create another column with a Boolean variables that show if a surname was used in this tweet or not
3. And finally, data was grouped by a week number to count number of times a surname was used per week

In [22]:
```python
#extract a week number
for i, row in df_trump.iterrows():
    df_trump.set_value(i,'weekn', df_trump['date_conv'][i].week)

#create a boolean variable that shows surname usage
for i, row in df_trump.iterrows():
    text = df_trump['Text'][i]
    lst = re.findall(r'(Trump)', text)
    if len(lst)>0:
        df_trump.set_value(i,'last_name', "True")
    else:
        df_trump.set_value(i,'last_name', "False")

#mean tweets with last name per weeek
df_trump_4 = df_trump.groupby(['weekn', 'last_name']).count()
df_trump_4 = df_trump_4.reset_index()
df_trump_4 = df_trump_4.loc[df_trump_4['last_name'] == 'True']
print("Average number of times per week when Trump uses his surname in T
witter: ",df_trump_4["created_at"].mean())
```

```
Average number of times per week when Trump uses his surname in Twitter:
  4.172413793103448
```

Peak times they tweet?

This time an hour of the tweet was extracted and used for grouping to count number of tweets per hour

Trump

In [23]:
```
#extract time
for i, row in df_trump.iterrows():
    value = df_trump['created_at'][i][11:13]
    df_trump.set_value(i,'timev', value)

#the most popular time of day
df_trump_6=df_trump.groupby('timev').count()
df_trump_6.sort_values(by=['created_at'], ascending=False)[:10]
```

Out[23]:

	Text	created_at	date	date_conv	dayofweek	weekn	last_name
timev							
13	177	177	177	177	177	177	177
11	166	166	166	166	166	166	166
12	159	159	159	159	159	159	159
20	104	104	104	104	104	104	104
14	99	99	99	99	99	99	99
10	88	88	88	88	88	88	88
22	75	75	75	75	75	75	75
19	67	67	67	67	67	67	67
17	65	65	65	65	65	65	65
00	60	60	60	60	60	60	60

Trudeau

In [24]:
```
#extract time
for i, row in df_trud.iterrows():
    value = df_trud['created_at'][i][11:13]
    df_trud.set_value(i,'timev', value)

#the most popular time of day
df_trud_6=df_trud.groupby('timev').count()
df_trud_6.sort_values(by=['created_at'], ascending=False)[:10]
```

Out[24]:

	Text	created_at	date	date_conv	dayofweek
timev					
22	145	145	145	145	145
18	132	132	132	132	132

14	118	118	118	118	118
21	112	112	112	112	112
20	111	111	111	111	111
15	104	104	104	104	104
19	101	101	101	101	101
00	100	100	100	100	100
17	95	95	95	95	95
23	89	89	89	89	89

Trump prefers to tweet from 10AM till 2PM, in the afternoon, while Trudeau prefers to leave tweets from 8PM to 10PM, in the evening.

Number of times they use their countries name in their tweets?

A Boolean variable was created to indicate if country name was used or not. After that the total number of usages and non-usages were found by grouping:

Trump

In [25]:
```
#create a boolean variable that shows country name usage
for i, row in df_trump.iterrows():
    text = df_trump['Text'][i]
    lst = re.findall(r'(US|USA|United States)', text)
    if len(lst)>0:
        df_trump.set_value(i,'country', "True")
    else:
        df_trump.set_value(i,'country', "False")

#counting
df_trump_4 = df_trump.groupby(['country']).count()
#df_trump_4 = df_trump_4.reset_index()
print("%.2f" % (df_trump_4['Text']['True']/df_trump_4['Text']['False']),
 "% of times name of contry is used")
```

```
0.10 % of times name of contry is used
```

Trudeau

In [26]:
```
#create a boolean variable that shows country name usage
for i, row in df_trud.iterrows():
    text = df_trud['Text'][i]
    lst = re.findall(r'(Canada)', text)
    if len(lst)>0:
        df_trud.set_value(i,'country', "True")
    else:
        df_trud.set_value(i,'country', "False")

#counting
df_trud_4 = df_trud.groupby(['country']).count()
#df_trud_4 = df_trud.reset_index()
```

```
print("%.2f" % (df_trud_4['Text']['True']/df_trud_4['Text']['False']), "
% of times name of contry is used")
```

```
0.43 % of times name of contry is used
```

Trudeau tends to use the name of Canada much more often than Trump uses the name of US. More than 40% of tweets of Trudeau contain name of his country, while only every 10th tweet of Trump has a name of US in some form.

What is the average number of words per tweet?

To calculate average number of words in a tweet, first of all, a total number of words of every tweet was put in list and then the mean of numbers of this list was found

Trudeau

In [27]:

```
#average number of words in a tweet
lst=[]

for i, row in df_trump.iterrows():
    lst.append(len(df_trump['Text'][i].split(' ')))

import numpy as np
print("Trudeau.\nAverage number of words: ","%.2f" % np.mean(lst))
```

```
Trudeau.
Average number of words:  33.82
```

Trump

In [28]:

```
#average number of words in a tweet
lst=[]

for i, row in df_trump.iterrows():
    lst.append(len(df_trud['Text'][i].split(' ')))

import numpy as np
print("Trump.\nAverage number of words: ","%.2f" % np.mean(lst))
```

```
Trump.
Average number of words:  31.06
```

Trump tends to use on average 3 words less than Trudeau.

What are the most popular words used by Trump and Trudeau?

In [31]:

```
import re
#the words that appear he most in positive reviews
import nltk
porter = nltk.PorterStemmer()
list_pos=[]
for i in range(len(df_trump)):
    list_pos.append(df_trump["Text"].iloc[i])
lst_words_pos = []
for line in list_pos:
```

```
    text_pos = re.split('\n| |\?|\!|\:|\"|\(|\)|\...|\;',line)
    for word in text_pos:
        if (len(word)>3 and not word.startswith('@') and not word.starts
with('#') and word != 'RT'):
            lst_words_pos.append(porter.stem(word.lower()))


dist_pos = FreqDist(lst_words_pos)
sorted_dist_pos = sorted(dist_pos.items(), key=operator.itemgetter(1), r
everse=True)
sorted_dist_pos[:50]
```

```
Out[31]: [('that', 434),
 ('with', 425),
 ('will', 382),
 ('great', 380),
 ('http', 365),
 ('have', 341),
 ('they', 267),
 ('&amp', 210),
 ('thi', 191),
 ('peopl', 189),
 ('countri', 181),
 ('veri', 172),
 ('democrat', 162),
 ('from', 148),
 ('their', 144),
 ('more', 143),
 ('want', 142),
 ('mani', 140),
 ('news', 130),
 ('just', 127),
 ('border', 123),
 ('been', 123),
 ('trade', 120),
 ('state', 119),
 ('about', 116),
 ('presid', 115),
 ('there', 114),
 ('fake', 113),
 ('make', 112),
 ('work', 107),
 ('must', 106),
 ('would', 105),
 ('time', 104),
 ('than', 102),
 ('year', 102),
 ('american', 100),
 ('good', 100),
 ('much', 99),
 ('thank', 97),
 ('trump', 91),
 ('back', 91),
 ('come', 88),
 ('what', 86),
 ('need', 84),
 ('should', 84),
 ('into', 83),
 ('america', 83),
 ('meet', 81),
 ('be', 81),
```

```
 ('never', 79)]
```

Some of the most frequent words of Trump: *people, country, democracy, fake, border, news, america*

In [33]:
```
import re
#the words that appear he most in positive reviews
import nltk
porter = nltk.PorterStemmer()
list_pos=[]
for i in range(len(df_trud)):
    list_pos.append(df_trud["Text"].iloc[i])
lst_words_pos = []
for line in list_pos:
    text_pos = re.split('\n| |\?|\!|\:|\"|\(|\)|\...|\;',line)
    for word in text_pos:
        if (len(word)>3 and not word.startswith('@') and not word.starts
with('#') and word != 'RT'):
            lst_words_pos.append(porter.stem(word.lower()))


dist_pos = FreqDist(lst_words_pos)
sorted_dist_pos = sorted(dist_pos.items(), key=operator.itemgetter(1), r
everse=True)
sorted_dist_pos[:50]
```

Out[33]:
```
[('http', 821),
 ('ttp', 594),
 ('pour', 502),
 ('nou', 382),
 ('canada', 341),
 ('&amp', 328),
 ('with', 221),
 ('work', 177),
 ('dan', 166),
 ('plu', 164),
 ('avec', 159),
 ('thi', 150),
 ('notr', 123),
 ('more', 121),
 ('vou', 120),
 ('canadian', 119),
 ('today', 111),
 ('leur', 108),
 ('canadien', 101),
 ('peopl', 101),
 ('tou', 101),
 ('travail', 96),
 ('thank', 93),
 ('will', 86),
 ('their', 85),
 ('we’r', 84),
 ('make', 81),
 ('tout', 81),
 ('about', 80),
 ('creat', 79),
 ('job', 79),
 ('cett', 79),
 ('great', 77),
 ('avon', 76),
```

```
('have', 74),
('from', 74),
('meet', 74),
('help', 72),
('congratul', 71),
('pay', 71),
('erci', 67),
('your', 67),
('togeth', 66),
('that', 64),
('félicit', 63),
('votr', 63),
('countri', 62),
('discuss', 62),
('protect', 59),
('aujourd’hui', 58)]
```

Some of the most frequent words of Trudeau: *Canada, people, today, thank, congats, protect*

168 - 174

3.2. YouTube API: Extraction and sentiment analysis of comments about Asus Zenbook Pro (Regex/NLTK)

Youtube API: Extraction and sentiment analysis of comments about Asus Zenbook Pro (Regex/NLTK)

The topic of analysis is "Asus Zenbook Pro", a laptop from Asus. The idea is to find out what people think about the product by analysing comments, extracted from videos on this topic.

Import libraries

In [1]:
```
import requests
import pandas as pd
import numpy as np
```

Extract videos that contain specific search words

Key-words are: *asus zenbook pro*

Number of videos: *50*

Relevance: *English language*

In [2]:
```
params = (
    ('key', 'AIzaSyDyPycUEc7szd7NWABwbAULVdAxBo36W3w'),
    ('part', 'snippet'),
    ('type', 'video'),
    ('maxResults', 50),
    ('q', 'asus zenbook pro'),
    ('relevanceLanguage', 'en'), #is not guaranteed to work
)

response = requests.get('https://www.googleapis.com/youtube/v3/search',
params=params)

response_json=response.json()


channel_ids = []
videoid_name = {}
for i in range(len(response_json['items'])):
    channel_ids.append(response_json['items'][i]['snippet']['channelId']
)
    videoid_name[response_json['items'][i]['snippet']['title']] = respon
se_json['items'][i]['id']['videoId']
```

Even though **'relevanceLanguage'** is set to English, API outputs videos of non-English channels. Consequently, only comments from English-speaking videos will be selected for the Analysis

To find out what videos are in English language, a library called **"langid"** is used

A language will be determined from a title of a video

In [3]:
```
import langid

#create a list of videos with english names
videos_required=[]
for name in videoid_name.keys():
    lang = langid.classify(name)
```

```
    #print("Lang: ", lang, "Name: ", name)
    if lang[0] == 'en':
        videos_required.append(videoid_name.get(name))
```

In [4]:
```
print("Number of English videos: ", len(videos_required))
```

```
Number of English videos:  24
```

Now when video Id's are stored, API can be used once more to extract comments of videos in a list "videos_required"

In [5]:
```
import time
comments=[]
video_id = []
for video in videos_required:

    params_v = (
        ('key', 'AIzaSyDyPycUEc7szd7NWABwbAULVdAxBo36W3w'),
        ('part', 'snippet'),
        ('videoId', video),
        ('maxResults', '100'),
    )

    response_v = requests.get('https://www.googleapis.com/youtube/v3/com
mentThreads', params=params_v)
    response_json_v=response_v.json()


    for i in range(len(response_json_v['items'])):
        comments.append(response_json_v['items'][i]['snippet']['topLevel
Comment']['snippet']['textOriginal'])
        video_id.append(response_json_v['items'][i]['snippet']['topLevel
Comment']['snippet']['videoId'])
    time.sleep(3)
```

Now Comments are put into a dataframe
Moreover, API from text-processing.com is used to detect **positive** and **negative** comments

In [6]:
```
df = pd.DataFrame(columns=['textDisplay', 'video_id','label','pos','neg'
,'neutral']) #creates empty dataframe

for i in range(len(comments)):
    lst=[]
    comment = comments[i]
    vid_id = video_id[i]
    data = [('text', comment),]
    response = requests.post('http://text-processing.com/api/sentiment/'
, data=data)
    json_sent = response.json()
    lst.append(comment)
    lst.append(vid_id)
    lst.append(json_sent['label'])
    lst.append(json_sent["probability"]["pos"])
    lst.append(json_sent["probability"]["neg"])
    lst.append(json_sent["probability"]["neutral"])
    df.loc[i] = lst
```

Summary: number of positive, negative, neutral comments

In [7]:
```
df.groupby(['label'])['textDisplay'].count()
```

Out[7]:
```
label
neg        654
neutral    330
pos        468
Name: textDisplay, dtype: int64
```

Subset of negative comments

In [14]:
```
pd.options.display.max_colwidth = 140
df.loc[df['label'] == 'neg'].sort_values(by=['neg'], ascending =False)[:
5]
```

Out[14]:

	textDisplay	video_id	label	pos	neg	neutral
1378	Ther is NOTHING WORST than scrolling a touch screen and that it lags so terribly.\n\nThe touch pad completely turned me off from this la...	jR1V_7RxrIk	neg	0.028725	0.971275	0.001585
679	Stupid idea and boring naming	otLtSbzWgrA	neg	0.058390	0.941610	0.061381
1190	I bought an UX430UA from Asus and I'm really mad at them for not having Asus health charging app. the website says all 2017 zenbook have...	A0cLS0ZHWNc	neg	0.067910	0.932090	0.211965
63	I'm all for an extra screen on a laptop, but why on earth did they put it in the worse possible place to put a screen?\nDoes anyone seri...	b5wGGp88nBs	neg	0.106105	0.893895	0.146019
332	Who the hell measures battery life with the screen off? That's so stupid!	ycsCNY-wSHg	neg	0.106472	0.893528	0.012235

Subset of positive comments

In [13]:
```
df.loc[df['label'] == 'pos'].sort_values(by=['pos'], ascending =False)[:
5]
```

Out[13]:

	textDisplay	video_id	label	pos	neg	neutral
1429	VERY NICE. GOOD BRAND. I use this brand for many years and I feel very comfortable. this is the top of the PC and of the various brands....	EcaDhN_OD_Q	pos	0.898260	0.101740	0.093861

872	Nice one Saf! This is probably the best coverage of Computex haha	phGShu0LzwQ	pos	0.871374	0.128626	0.161591
1078	Asus always deliver a great, durable, and beautiful product.	CEWrNY0u-Gc	pos	0.869709	0.130291	0.111447
1430	Fiero utilizzatore della Asus da più di 15 anni. Una marca davvero ottima. Eccelle in ogni sua funzionalità e prestazioni. Design e graf...	EcaDhN_OD_Q	pos	0.864242	0.135758	0.157376
877	Now that is awesome innovation. especially the extension display option. that is nice.	phGShu0LzwQ	pos	0.859301	0.140699	0.111939

Use PorterStemmer to normalize words and find the most frequent words used in positive and negative comments

In [11]:
```
from nltk import FreqDist
import operator

import re
#the words that appear he most in positive reviews
import nltk
porter = nltk.PorterStemmer()
list_pos=[]
for i in range(len(df.loc[df['label'] == 'pos'])):
    list_pos.append(df.loc[df['label'] == 'pos']["textDisplay"].iloc[i])
lst_words_pos = []
for line in list_pos:
    text_pos = re.split('\n| |\?|\!|\:|\"|\(|\)|\...|\;',line)
    for word in text_pos:
        if (len(word)>3 and not word.startswith('@') and not word.starts
with('#') and word != 'RT'):
            lst_words_pos.append(porter.stem(word.lower()))


dist_pos = FreqDist(lst_words_pos)
sorted_dist_pos = sorted(dist_pos.items(), key=operator.itemgetter(1), r
everse=True)
sorted_dist_pos[:50]
```

Out[11]:
```
[('thi', 112),
 ('laptop', 81),
 ('with', 79),
 ('asu', 66),
 ('that', 58),
 ('video', 58),
 ('great', 57),
 ('review', 48),
 ('good', 45),
 ('have', 42),
 ('would', 42),
 ('your', 41),
 ('thank', 41),
```

```
 ('nice', 37),
 ('more', 36),
 ('love', 35),
 ('look', 35),
 ('will', 33),
 ('zenbook', 31),
 ('what', 31),
 ('like', 31),
 ('than', 30),
 ('awesom', 28),
 ('better', 27),
 ('macbook', 26),
 ('realli', 26),
 ("it'", 25),
 ('just', 25),
 ('screen', 24),
 ('game', 23),
 ('vivobook', 23),
 ('cool', 22),
 ('appl', 22),
 ('use', 20),
 ('veri', 20),
 ('know', 19),
 ('could', 18),
 ('from', 18),
 ('price', 18),
 ('work', 17),
 ('about', 17),
 ('think', 16),
 ('amaz', 16),
 ('make', 16),
 ('best', 15),
 ('display', 15),
 ('some', 15),
 ('want', 15),
 ('edit', 14),
 ('pleas', 13)]
```

Some useful words that help understand what users in Zenbook laptops **like**: *look, video, screen, game, price, display*

In [12]:
```
list_neg=[]
for i in range(len(df.loc[df['label'] == 'neg'])):
    list_neg.append(df.loc[df['label'] == 'neg']["textDisplay"].iloc[i])
lst_words_neg = []
for line in list_neg:
    text_neg = re.split('\n| |\?|\!|\:|\"|\(|\)|\...|\;',line)
    for word in text_neg:
        if (len(word)>3 and not word.startswith('@') and not word.starts
with('#') and word != 'RT'):
            lst_words_neg.append(porter.stem(word.lower()))
dist_neg = FreqDist(lst_words_neg)
sorted_dist_neg = sorted(dist_neg.items(), key=operator.itemgetter(1), r
everse=True)
sorted_dist_neg[:50]
```

Out[12]:
```
[('thi', 253),
 ('laptop', 191),
 ('that', 158),
 ('have', 116),
```

```
 ('with', 112),
 ('asu', 106),
 ('screen', 99),
 ('zenbook', 85),
 ('like', 79),
 ('look', 71),
 ('just', 67),
 ('about', 67),
 ('what', 58),
 ('want', 56),
 ('when', 55),
 ('would', 54),
 ('they', 53),
 ('macbook', 49),
 ('than', 47),
 ('think', 46),
 ('more', 45),
 ('better', 45),
 ('game', 45),
 ('onli', 45),
 ('realli', 45),
 ('will', 45),
 ("don't", 44),
 ('your', 44),
 ('touch', 44),
 ("it'", 42),
 ('need', 42),
 ('much', 40),
 ('releas', 38),
 ('review', 36),
 ('video', 36),
 ('use', 35),
 ('price', 35),
 ('appl', 33),
 ('could', 33),
 ('from', 32),
 ('there', 32),
 ('make', 31),
 ('pleas', 31),
 ('time', 28),
 ('doe', 28),
 ('where', 27),
 ('thing', 27),
 ('some', 27),
 ('come', 26),
 ('trackpad', 26)]
```

Some useful words that help understand what users in Zenbook laptops **dislike**: *time, game, price, touch(pad), screen, trackpad*

Hadoop (MapReduce, Pig, Hive)

176 - 182

4.1. Calculate average temperature across months. Small example of how to extract data from json into MapReduce

Calculate average temperature across months with mapreduce

Small example of how to extract data from json into mapreduce Data stored in **weather**
The **mapreduce output**can be found in mapreduce_output.txt
The **log** file of the mapreduce - mapreduce_log.txt
The **jar** file used to run a mapreduce job - weather.jar
Driver, Mapper and Reducer are saved as separate files for a reference
The code to **extract data from API** is in weather_extraction.py

Driver

```
package com.mop.weather;
import org.apache.hadoop.conf.Configured;
import org.json.simple.parser.JSONParser;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
//import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
//import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;


import com.mop.weather.weatherDriver;
import com.mop.weather.weatherMapper;
import com.mop.weather.weatherReducer;


public class weatherDriver extends Configured implements Tool {



    @Override
    public int run(String[] args) throws Exception {
```

```java
        if (args.length != 2) {
            System.err.println("Usage: fberature <input path> <output path>");
            System.exit(-1);
        }

        //Job Setup
        Job fb = Job.getInstance(getConf(), "facebook-friends");

        fb.setJarByClass(weatherDriver.class);


        //File Input and Output format
        FileInputFormat.addInputPath(fb, new Path(args[0]));
        FileOutputFormat.setOutputPath(fb, new Path(args[1]));

        fb.setInputFormatClass(TextInputFormat.class);
        fb.setOutputFormatClass(SequenceFileOutputFormat.class);

        //Output types


        fb.setMapperClass(weatherMapper.class);
        fb.setReducerClass(weatherReducer.class);

        fb.setOutputKeyClass(IntWritable.class); //type of a key (stock code)
        fb.setOutputValueClass(DoubleWritable.class); //type of a value (price);


        //Submit job
        return fb.waitForCompletion(true) ? 0 : 1;

    }

    public static void main(String[] args) throws Exception {

        int exitCode = ToolRunner.run(new weatherDriver(), args);
        System.exit(exitCode);
    }}
```

Mapper

```java
package com.mop.weather;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
```

```java
//import java.time.LocalDate;
//import java.time.format.DateTimeFormatter;
//import java.util.Locale;

import org.apache.hadoop.io.DoubleWritable;
//import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
//import org.apache.hadoop.mapreduce.Mapper.Context;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

//import com.hirw.maxcloseprice.MaxClosePriceMapper.Volume;


public class weatherMapper extends Mapper<LongWritable, Text, IntWritable, Doubl
eWritable> {

    double init = 184.65; //price of a stock in the end of 2014;
    @Override //we need to overwrite "map" method;
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {



        String line = value.toString(); //convert the value (record) to a string;

        try {
            JSONParser parser = new JSONParser();
            Object obj = parser.parse(line);

            JSONObject jsonObject = (JSONObject) obj;

            long time = (long) jsonObject.get("time");
            //double l = jsonObject.get("temperatureHigh");
            String tempstring = String.valueOf(jsonObject.get("temperatureHigh"));
            double temperature = Double.valueOf(tempstring);

            Date date = new java.util.Date((long)time*1000L);
            SimpleDateFormat sdf = new java.text.SimpleDateFormat("dd-MM-yyyy");
            sdf.setTimeZone(java.util.TimeZone.getTimeZone("GMT+2"));
            String formattedDate = sdf.format(date);
            String subs = formattedDate.substring(3, 5);
            int month = Integer.parseInt(subs);
            context.write(new IntWritable(month), new DoubleWritable(temperature))
; //used to submit a mapper output;



        } catch (IOException e) {
```

```
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        }


    }

}
```

Reducer

```
package com.mop.weather;

import java.io.IOException;
import org.json.simple.parser.JSONParser;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

import org.apache.hadoop.io.DoubleWritable;
//import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
//import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;


public class weatherReducer extends Reducer<IntWritable, DoubleWritable, IntWrit
able, DoubleWritable> {
    //first two define the input to a reducer - Text, FloatWritable
    //two others - the output from the reducer (Text, FloatWritable)

     @Override
     public void reduce(IntWritable key, Iterable<DoubleWritable> values, Context
 context)
             throws IOException, InterruptedException {

         //key - month; values - iterable list of percentage difference stocks;

         double sum_temp = Double.MIN_VALUE; //puts the smallest value possible?;
         int elementNumb = 0;

         //Iterate all closing prices and calculate maximum
         for (DoubleWritable value : values) {
             sum_temp = sum_temp + value.get(); //get allows using DoubleWritable
as double
             elementNumb += 1;
         }

         double aver_temp = sum_temp/elementNumb;
```

```
        //Write output
        context.write(key, new DoubleWritable(aver_temp));
    }

}
```

Log

hirwuser864@ip-172-31-45-217:~$ hadoop jar /home/hirwuser864/weather/weather.jar com.mop.weather.weatherDriver -libjars /hirw-workshop/mapreduce/facebook/json-simple-1.1.jar /user/hirwuser864/weather_input/input/ /user/hirwuser864/weather_output 18/06/13 19:54:25 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-45-216.ec2.internal/172.31.45.216:8032 18/06/13 19:54:26 INFO input.FileInputFormat: Total input paths to process : 1 18/06/13 19:54:26 INFO mapreduce.JobSubmitter: number of splits:1 18/06/13 19:54:26 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1525967314796_1006 18/06/13 19:54:26 INFO impl.YarnClientImpl: Submitted application application_1525967314796_1006 18/06/13 19:54:26 INFO mapreduce.Job: The url to track the job: http://ec2-54-92-244-237.compute-1.amazonaws.com:8088/proxy/application_1525967314796_1006/ 18/06/13 19:54:26 INFO mapreduce.Job: Running job: job_1525967314796_1006 18/06/13 19:54:32 INFO mapreduce.Job: Job job_1525967314796_1006 running in uber mode : false 18/06/13 19:54:32 INFO mapreduce.Job: map 0% reduce 0% 18/06/13 19:54:37 INFO mapreduce.Job: map 100% reduce 0% 18/06/13 19:54:43 INFO mapreduce.Job: map 100% reduce 100% 18/06/13 19:54:43 INFO mapreduce.Job: Job job_1525967314796_1006 completed successfully 18/06/13 19:54:43 INFO mapreduce.Job: Counters: 53 File System Counters FILE: Number of bytes read=10226 FILE: Number of bytes written=271177 FILE: Number of read operations=0 FILE: Number of large read operations=0 FILE: Number of write operations=0 HDFS: Number of bytes read=762281 HDFS: Number of bytes written=335 HDFS: Number of read operations=6 HDFS: Number of large read operations=0 HDFS: Number of write operations=2 Job Counters Launched map tasks=1 Launched reduce tasks=1 Data-local map tasks=1 Total time spent by all maps in occupied slots (ms)=14404 Total time spent by all reduces in occupied slots (ms)=12536 Total time spent by all map tasks (ms)=3601 Total time spent by all reduce tasks (ms)=3134 Total vcore-milliseconds taken by all map tasks=3601 Total vcore-milliseconds taken by all reduce tasks=3134 Total megabyte-milliseconds taken by all map tasks=3687424 Total megabyte-milliseconds taken by all reduce tasks=3209216 Map-Reduce Framework Map input records=730 Map output records=730 Map output bytes=8760 Map output materialized bytes=10226 Input split bytes=151 Combine input records=0 Combine output records=0 Reduce input groups=12 Reduce shuffle bytes=10226 Reduce input records=730 Reduce output records=12 Spilled Records=1460 Shuffled Maps =1 Failed Shuffles=0 Merged Map outputs=1 GC time elapsed (ms)=169 CPU time spent (ms)=1660 Physical memory (bytes) snapshot=700690432 Virtual memory (bytes) snapshot=2772324352 Total committed heap usage (bytes)=579338240 Peak Map Physical memory (bytes)=498581504 Peak Map Virtual memory (bytes)=1383337984 Peak Reduce Physical memory (bytes)=202108928 Peak Reduce Virtual memory (bytes)=1388986368 Shuffle Errors BAD_ID=0 CONNECTION=0 IO_ERROR=0 WRONG_LENGTH=0 WRONG_MAP=0 WRONG_REDUCE=0 File Input Format Counters Bytes Read=762130 File Output Format Counters Bytes Written=335

Output

1 2.0919354838709685
2 7.350701754385967
3 10.817741935483872
4 13.178666666666668
5 19.28870967741935
6 23.587166666666672
7 24.801774193548386
8 24.490645161290313
9 19.6145
10 14.002096774193552
11 7.229833333333333
12 3.977213114754097

183 - 192

4.2. Hadoop MapReduce script to find tweets that contain some word or words

Hadoop Mapreduce script to find tweets that contain some word or words

This mapreduce task finds files where some code is stored. In this example every file is a tweet from Trumps twitter. I tried to find tweets where Trump used the name of US in some form (United States, US, USA etc.) or words Good and Bad.

Every tweet is stored in json file (for instance, **js_0.json**)
The **mapreduce output**can be found in result.txt
The **log** file of the mapreduce - log.txt
The **jar** file used to run a mapreduce job - find_word_twitter_wordsinside.jar
Driver, Mapper and Reducer are saved as separate files for a reference
The code to **extract tweets from Twitter API** is in extract_tweet.py

The code in hadoop to run a script
Because data is stored in json, some additional libjar is needed to run the script - json-simple-1.1.jar

```
hadoop jar /home/hirwuser864/findwordtwitter/find_word_twitter_wordsinside.jar
com.mop.findword.findwordDriver -libjars /hirw-workshop/mapreduce/facebook/json-
simple-1.1.jar /user/hirwuser864/findwordtwitter_input/input/
/user/hirwuser864/findwordtwitterwordsinside_output
```

Driver

```
package com.mop.findword;

import org.apache.hadoop.conf.Configured;
import org.json.simple.parser.JSONParser;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
//import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```java
import com.mop.findword.findwordDriver;
import com.mop.findword.findwordMapper;
import com.mop.findword.findwordReducer;


public class findwordDriver extends Configured implements Tool {



    @Override
    public int run(String[] args) throws Exception {

        if (args.length != 2) {
            System.err.println("Usage: libjar <input path> <output path>");
            System.exit(-1);
        }

        //Job Setup
        Job fb = Job.getInstance(getConf(), "findword");

        fb.setJarByClass(findwordDriver.class);


        //File Input and Output format
        FileInputFormat.addInputPath(fb, new Path(args[0]));
        FileOutputFormat.setOutputPath(fb, new Path(args[1]));

        fb.setInputFormatClass(TextInputFormat.class);
        fb.setOutputFormatClass(SequenceFileOutputFormat.class);

        //Output types


        fb.setMapperClass(findwordMapper.class);
        fb.setReducerClass(findwordReducer.class);


        fb.setOutputKeyClass(Text.class); //type of a key (stock code)
        fb.setOutputValueClass(Text.class); //type of a value (price);

        //Submit job
        return fb.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {

        int exitCode = ToolRunner.run(new findwordDriver(), args);
        System.exit(exitCode);
    }}
```

Mapper

```java
package com.mop.findword;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import org.apache.hadoop.mapreduce.Mapper;

public class findwordMapper extends Mapper<LongWritable, Text, Text, Text> {
    @Override //we need to overwrite "map" method;
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String fileName = ((FileSplit) context.getInputSplit()).getPath().getName()
;

        String line = value.toString();
        Path pt=new Path("hdfs://ip-172-31-45-216.ec2.internal:8020/user/hirwuser8
64/
                findwordtwitter_words/words");
        FileSystem fs = FileSystem.get(context.getConfiguration());
        BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(pt)));
        String words=br.readLine();
        String[] items = words.split(" ");
        //String[] items = {"US","USA","[Gg]ood","[Bb]ad"};
        for (int i = 0; i<items.length; i+=1 ) {

            String pattern = items[i];

            try {
                JSONParser parser = new JSONParser();
                Object obj = parser.parse(line);

                JSONObject jsonObject = (JSONObject) obj;

                String full_text = (String) jsonObject.get("full_text");
                Pattern r = Pattern.compile(pattern);
```

```
                CharSequence cs = full_text;
                Matcher m = r.matcher(cs);
                if (m.find( )) {
                    context.write(new Text(items[i]), new Text(fileName));

                    }


            } catch (IOException e) {
                e.printStackTrace();
            } catch (ParseException e) {
                e.printStackTrace();
            }

              // Create a Pattern object


        }
        }
}
```

Reducer

```
package com.mop.findword;

import java.io.IOException;

//import org.apache.hadoop.io.DoubleWritable;
//import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class findwordReducer extends Reducer<Text, Text, Text, Text> {

     @Override
     public void reduce(Text key, Iterable<Text> values, Context context)
             throws IOException, InterruptedException {
         String result = "[";
         for (Text value : values) {
             result = result + ", " + value;
         }
         result = result + "]";
         context.write(key, new Text(result));




     }
}
```

Log

```
hirwuser864@ip-172-31-45-217:~$ hadoop jar
    /home/hirwuser864/findwordtwitter/find_word_twitter.jar
    com.mop.findword.findwordDriver
    -libjars /hirw-workshop/mapreduce/facebook/json-simple-1.1.jar
    /user/hirwuser864/findwordtwitter_input/input/ /user/hirwuser864/findwordtwitt
er_output
18/06/17 16:02:01 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-
45-216.ec2.internal/172.31.45.216:8032
18/06/17 16:02:02 INFO input.FileInputFormat: Total input paths to process : 1000
18/06/17 16:02:02 INFO mapreduce.JobSubmitter: number of splits:1000
18/06/17 16:02:03 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_152
5967314796_1105
18/06/17 16:02:03 INFO impl.YarnClientImpl: Submitted application application_1525
967314796_1105
18/06/17 16:02:03 INFO mapreduce.Job: The url to track the job: http://ec2-54-92-2
44-237.compute-1.amazonaws.com:8088/proxy/application_1525967314796_1105/
18/06/17 16:02:03 INFO mapreduce.Job: Running job: job_1525967314796_1105
18/06/17 16:02:09 INFO mapreduce.Job: Job job_1525967314796_1105 running in uber m
ode : false
18/06/17 16:02:09 INFO mapreduce.Job:  map 0% reduce 0%
18/06/17 16:02:22 INFO mapreduce.Job:  map 1% reduce 0%
18/06/17 16:02:38 INFO mapreduce.Job:  map 2% reduce 0%
18/06/17 16:02:53 INFO mapreduce.Job:  map 3% reduce 0%
18/06/17 16:03:08 INFO mapreduce.Job:  map 4% reduce 0%
18/06/17 16:03:25 INFO mapreduce.Job:  map 5% reduce 0%
18/06/17 16:03:42 INFO mapreduce.Job:  map 6% reduce 0%
18/06/17 16:03:58 INFO mapreduce.Job:  map 7% reduce 0%
18/06/17 16:04:13 INFO mapreduce.Job:  map 8% reduce 0%
18/06/17 16:04:28 INFO mapreduce.Job:  map 9% reduce 0%
18/06/17 16:04:46 INFO mapreduce.Job:  map 10% reduce 0%
18/06/17 16:05:02 INFO mapreduce.Job:  map 11% reduce 0%
18/06/17 16:05:15 INFO mapreduce.Job:  map 12% reduce 0%
18/06/17 16:05:31 INFO mapreduce.Job:  map 13% reduce 0%
18/06/17 16:05:48 INFO mapreduce.Job:  map 14% reduce 0%
18/06/17 16:06:06 INFO mapreduce.Job:  map 15% reduce 0%
18/06/17 16:06:22 INFO mapreduce.Job:  map 16% reduce 0%
18/06/17 16:06:38 INFO mapreduce.Job:  map 17% reduce 0%
18/06/17 16:06:55 INFO mapreduce.Job:  map 18% reduce 0%
18/06/17 16:07:10 INFO mapreduce.Job:  map 19% reduce 0%
18/06/17 16:07:27 INFO mapreduce.Job:  map 20% reduce 0%
18/06/17 16:07:43 INFO mapreduce.Job:  map 21% reduce 0%
18/06/17 16:07:56 INFO mapreduce.Job:  map 22% reduce 0%
18/06/17 16:08:14 INFO mapreduce.Job:  map 23% reduce 0%
18/06/17 16:08:30 INFO mapreduce.Job:  map 24% reduce 0%
18/06/17 16:08:45 INFO mapreduce.Job:  map 25% reduce 0%
18/06/17 16:09:02 INFO mapreduce.Job:  map 26% reduce 0%
18/06/17 16:09:14 INFO mapreduce.Job:  map 26% reduce 9%
```

```
18/06/17 16:09:18 INFO mapreduce.Job:  map 27% reduce 9%
18/06/17 16:09:34 INFO mapreduce.Job:  map 28% reduce 9%
18/06/17 16:09:50 INFO mapreduce.Job:  map 29% reduce 9%
18/06/17 16:09:56 INFO mapreduce.Job:  map 29% reduce 10%
18/06/17 16:10:07 INFO mapreduce.Job:  map 30% reduce 10%
18/06/17 16:10:22 INFO mapreduce.Job:  map 31% reduce 10%
18/06/17 16:10:39 INFO mapreduce.Job:  map 32% reduce 10%
18/06/17 16:10:44 INFO mapreduce.Job:  map 32% reduce 11%
18/06/17 16:10:55 INFO mapreduce.Job:  map 33% reduce 11%
18/06/17 16:11:11 INFO mapreduce.Job:  map 34% reduce 11%
18/06/17 16:11:29 INFO mapreduce.Job:  map 35% reduce 11%
18/06/17 16:11:32 INFO mapreduce.Job:  map 35% reduce 12%
18/06/17 16:11:46 INFO mapreduce.Job:  map 36% reduce 12%
18/06/17 16:11:59 INFO mapreduce.Job:  map 37% reduce 12%
18/06/17 16:12:15 INFO mapreduce.Job:  map 38% reduce 12%
18/06/17 16:12:21 INFO mapreduce.Job:  map 38% reduce 13%
18/06/17 16:12:34 INFO mapreduce.Job:  map 39% reduce 13%
18/06/17 16:12:48 INFO mapreduce.Job:  map 40% reduce 13%
18/06/17 16:13:03 INFO mapreduce.Job:  map 41% reduce 13%
18/06/17 16:13:09 INFO mapreduce.Job:  map 41% reduce 14%
18/06/17 16:13:23 INFO mapreduce.Job:  map 42% reduce 14%
18/06/17 16:13:36 INFO mapreduce.Job:  map 43% reduce 14%
18/06/17 16:13:52 INFO mapreduce.Job:  map 44% reduce 14%
18/06/17 16:13:57 INFO mapreduce.Job:  map 44% reduce 15%
18/06/17 16:14:11 INFO mapreduce.Job:  map 45% reduce 15%
18/06/17 16:14:27 INFO mapreduce.Job:  map 46% reduce 15%
18/06/17 16:14:43 INFO mapreduce.Job:  map 47% reduce 15%
18/06/17 16:14:45 INFO mapreduce.Job:  map 47% reduce 16%
18/06/17 16:14:59 INFO mapreduce.Job:  map 48% reduce 16%
18/06/17 16:15:14 INFO mapreduce.Job:  map 49% reduce 16%
18/06/17 16:15:32 INFO mapreduce.Job:  map 50% reduce 16%
18/06/17 16:15:34 INFO mapreduce.Job:  map 50% reduce 17%
18/06/17 16:15:48 INFO mapreduce.Job:  map 51% reduce 17%
18/06/17 16:16:04 INFO mapreduce.Job:  map 52% reduce 17%
18/06/17 16:16:20 INFO mapreduce.Job:  map 53% reduce 17%
18/06/17 16:16:22 INFO mapreduce.Job:  map 53% reduce 18%
18/06/17 16:16:36 INFO mapreduce.Job:  map 54% reduce 18%
18/06/17 16:16:52 INFO mapreduce.Job:  map 55% reduce 18%
18/06/17 16:17:08 INFO mapreduce.Job:  map 56% reduce 18%
18/06/17 16:17:10 INFO mapreduce.Job:  map 56% reduce 19%
18/06/17 16:17:28 INFO mapreduce.Job:  map 57% reduce 19%
18/06/17 16:17:45 INFO mapreduce.Job:  map 58% reduce 19%
18/06/17 16:18:01 INFO mapreduce.Job:  map 59% reduce 19%
18/06/17 16:18:04 INFO mapreduce.Job:  map 59% reduce 20%
18/06/17 16:18:17 INFO mapreduce.Job:  map 60% reduce 20%
18/06/17 16:18:35 INFO mapreduce.Job:  map 61% reduce 20%
18/06/17 16:18:54 INFO mapreduce.Job:  map 62% reduce 20%
18/06/17 16:18:58 INFO mapreduce.Job:  map 62% reduce 21%
18/06/17 16:19:10 INFO mapreduce.Job:  map 63% reduce 21%
18/06/17 16:19:30 INFO mapreduce.Job:  map 64% reduce 21%
18/06/17 16:19:46 INFO mapreduce.Job:  map 65% reduce 22%
18/06/17 16:20:02 INFO mapreduce.Job:  map 66% reduce 22%
```

```
18/06/17 16:20:18 INFO mapreduce.Job:  map 67% reduce 22%
18/06/17 16:20:36 INFO mapreduce.Job:  map 68% reduce 22%
18/06/17 16:20:40 INFO mapreduce.Job:  map 68% reduce 23%
18/06/17 16:20:55 INFO mapreduce.Job:  map 69% reduce 23%
18/06/17 16:21:13 INFO mapreduce.Job:  map 70% reduce 23%
18/06/17 16:21:30 INFO mapreduce.Job:  map 71% reduce 23%
18/06/17 16:21:34 INFO mapreduce.Job:  map 71% reduce 24%
18/06/17 16:21:46 INFO mapreduce.Job:  map 72% reduce 24%
18/06/17 16:22:01 INFO mapreduce.Job:  map 73% reduce 24%
18/06/17 16:22:19 INFO mapreduce.Job:  map 74% reduce 24%
18/06/17 16:22:23 INFO mapreduce.Job:  map 74% reduce 25%
18/06/17 16:22:37 INFO mapreduce.Job:  map 75% reduce 25%
18/06/17 16:22:55 INFO mapreduce.Job:  map 76% reduce 25%
18/06/17 16:23:13 INFO mapreduce.Job:  map 77% reduce 25%
18/06/17 16:23:17 INFO mapreduce.Job:  map 77% reduce 26%
18/06/17 16:23:31 INFO mapreduce.Job:  map 78% reduce 26%
18/06/17 16:23:43 INFO mapreduce.Job:  map 79% reduce 26%
18/06/17 16:24:01 INFO mapreduce.Job:  map 80% reduce 26%
18/06/17 16:24:05 INFO mapreduce.Job:  map 80% reduce 27%
18/06/17 16:24:19 INFO mapreduce.Job:  map 81% reduce 27%
18/06/17 16:24:37 INFO mapreduce.Job:  map 82% reduce 27%
18/06/17 16:24:55 INFO mapreduce.Job:  map 83% reduce 27%
18/06/17 16:24:59 INFO mapreduce.Job:  map 83% reduce 28%
18/06/17 16:25:13 INFO mapreduce.Job:  map 84% reduce 28%
18/06/17 16:25:27 INFO mapreduce.Job:  map 85% reduce 28%
18/06/17 16:25:43 INFO mapreduce.Job:  map 86% reduce 28%
18/06/17 16:25:47 INFO mapreduce.Job:  map 86% reduce 29%
18/06/17 16:26:02 INFO mapreduce.Job:  map 87% reduce 29%
18/06/17 16:26:20 INFO mapreduce.Job:  map 88% reduce 29%
18/06/17 16:26:38 INFO mapreduce.Job:  map 89% reduce 29%
18/06/17 16:26:42 INFO mapreduce.Job:  map 89% reduce 30%
18/06/17 16:26:56 INFO mapreduce.Job:  map 90% reduce 30%
18/06/17 16:27:14 INFO mapreduce.Job:  map 91% reduce 30%
18/06/17 16:27:29 INFO mapreduce.Job:  map 92% reduce 30%
18/06/17 16:27:35 INFO mapreduce.Job:  map 92% reduce 31%
18/06/17 16:27:45 INFO mapreduce.Job:  map 93% reduce 31%
18/06/17 16:28:01 INFO mapreduce.Job:  map 94% reduce 31%
18/06/17 16:28:20 INFO mapreduce.Job:  map 95% reduce 31%
18/06/17 16:28:23 INFO mapreduce.Job:  map 95% reduce 32%
18/06/17 16:28:37 INFO mapreduce.Job:  map 96% reduce 32%
18/06/17 16:28:52 INFO mapreduce.Job:  map 97% reduce 32%
18/06/17 16:29:09 INFO mapreduce.Job:  map 98% reduce 32%
18/06/17 16:29:11 INFO mapreduce.Job:  map 98% reduce 33%
18/06/17 16:29:25 INFO mapreduce.Job:  map 99% reduce 33%
18/06/17 16:29:41 INFO mapreduce.Job:  map 100% reduce 33%
18/06/17 16:29:51 INFO mapreduce.Job:  map 100% reduce 100%
18/06/17 16:29:51 INFO mapreduce.Job: Job job_1525967314796_1105 completed success
fully
18/06/17 16:29:51 INFO mapreduce.Job: Counters: 53
        File System Counters
                FILE: Number of bytes read=3586
                FILE: Number of bytes written=125531392
```

```
		FILE: Number of read operations=0
		FILE: Number of large read operations=0
		FILE: Number of write operations=0
		HDFS: Number of bytes read=444519
		HDFS: Number of bytes written=2124
		HDFS: Number of read operations=3003
		HDFS: Number of large read operations=0
		HDFS: Number of write operations=2
	Job Counters
		Launched map tasks=1000
		Launched reduce tasks=1
		Data-local map tasks=1000
		Total time spent by all maps in occupied slots (ms)=17399300
		Total time spent by all reduces in occupied slots (ms)=5017684
		Total time spent by all map tasks (ms)=4349825
		Total time spent by all reduce tasks (ms)=1254421
		Total vcore-milliseconds taken by all map tasks=4349825
		Total vcore-milliseconds taken by all reduce tasks=1254421
		Total megabyte-milliseconds taken by all map tasks=4454220800
		Total megabyte-milliseconds taken by all reduce tasks=1284527104
	Map-Reduce Framework
		Map input records=1032
		Map output records=246
		Map output bytes=3088
		Map output materialized bytes=9580
		Input split bytes=157890
		Combine input records=0
		Combine output records=0
		Reduce input groups=6
		Reduce shuffle bytes=9580
		Reduce input records=246
		Reduce output records=6
		Spilled Records=492
		Shuffled Maps =1000
		Failed Shuffles=0
		Merged Map outputs=1000
		GC time elapsed (ms)=30408
		CPU time spent (ms)=405960
		Physical memory (bytes) snapshot=264764542976
		Virtual memory (bytes) snapshot=1378467610624
		Total committed heap usage (bytes)=188008628224
		Peak Map Physical memory (bytes)=304279552
		Peak Map Virtual memory (bytes)=1390415872
		Peak Reduce Physical memory (bytes)=404221952
		Peak Reduce Virtual memory (bytes)=1383501824
	Shuffle Errors
		BAD_ID=0
		CONNECTION=0
		IO_ERROR=0
		WRONG_LENGTH=0
		WRONG_MAP=0
		WRONG_REDUCE=0
```

```
    File Input Format Counters
            Bytes Read=286629
    File Output Format Counters
            Bytes Written=2124
```

Output

Shows files where specific word was contained

Good [, js_30, js_469, js_534, js_396, js_499, js_602, js_839, js_111, js_944, js_242]

US [, js_77, js_770, js_195, js_459, js_186, js_886, js_194, js_463, js_260, js_196, js_609, js_477, js_257, js_856, js_536, js_487, js_636, js_18, js_660, js_261, js_180, js_838, js_381, js_625, js_481, js_758, js_552, js_986, js_429, js_351, js_33, js_735, js_634, js_437, js_865, js_410, js_955, js_593, js_654, js_895, js_259, js_271, js_303, js_775, js_725, js_643, js_664, js_355, js_741, js_710, js_947, js_697, js_241, js_540, js_522, js_430, js_876, js_904, js_435]

USA [, js_895, js_381, js_838, js_593, js_955, js_634, js_33, js_437]

[Bb]ad [, js_51, js_359, js_490, js_440, js_684, js_645, js_217, js_393, js_212, js_848, js_731, js_517, js_910, js_893, js_335, js_106, js_3, js_412, js_353, js_322, js_956, js_139, js_377, js_580, js_553, js_546, js_835, js_244, js_482, js_154, js_15, js_811, js_789, js_118, js_585, js_650, js_373, js_826, js_810, js_901, js_612, js_549, js_170]

[Gg]ood [, js_379, js_166, js_491, js_40, js_867, js_922, js_602, js_756, js_30, js_51, js_242, js_528, js_45, js_389, js_43, js_706, js_11, js_731, js_99, js_516, js_611, js_3, js_572, js_464, js_849, js_499, js_809, js_641, js_604, js_797, js_839, js_475, js_699, js_352, js_515, js_381, js_838, js_792, js_396, js_219, js_527, js_251, js_944, js_783, js_518, js_422, js_750, js_326, js_318, js_169, js_992, js_466, js_208, js_469, js_632, js_636, js_217, js_72, js_218, js_111, js_766, js_455, js_534, js_541, js_291, js_188, js_517, js_178]

good [, js_178, js_72, js_636, js_632, js_169, js_318, js_750, js_422, js_251, js_527, js_792, js_838, js_475, js_379, js_604, js_641, js_849, js_464, js_3, js_611, js_11, js_706, js_389, js_45, js_51, js_867, js_922, js_491, js_541, js_218, js_766, js_217, js_992, js_326, js_783, js_219, js_699, js_797, js_166, js_572, js_731, js_43, js_40, js_756, js_291, js_455, js_466, js_518, js_352, js_809, js_99, js_528, js_188, js_208, js_515, js_516, js_517, js_381]

193 - 194

4.3. Merge and aggregate json datasets in pig to calculate number of bikes available in Toronto

Merge and aggregate json datasets in pig to calculate number of bikes available in Toronto

Merging two datasets that are stored in json files. Calculating % of bikes available on bike stations in Toronto. Pig code for a reference

Datasets available here (https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#84045f23-7465-0892-8889-7b6f91049b29)

Load json files with JsonLoader

```
station_information = LOAD '/user/hirwuser864/bikes_input/bikes/station_information.json'  USING JsonLoader('station_id:int, name:chararray, lat:float, lon:float, address:chararray, capacity:int, rental_methods:{(items:chararray)}');

station_status = LOAD '/user/hirwuser864/bikes_input/bikes/station_status.json' USING JsonLoader('station_id:int, num_bikes_available:int,  num_bikes_disabled:int, num_docks_available:int, num_docks_disabled:int, is_installed:int, is_renting:int, is_returning:int, last_reported:long');
```

Merging datasets by station_id

```
join_inner = JOIN station_information BY (station_id) , station_status BY (station_id);


join_project  = FOREACH join_inner GENERATE station_information::station_id,
station_information::address, station_information::capacity,
station_status::num_bikes_available, station_status::num_docks_available,
station_status::num_docks_disabled, station_information::lat, station_information::lon;
```

Calculating number of bikes available for every station

```
join_project_f  = FOREACH join_project GENERATE
   station_information::station_id as station_id,
   station_information::address as address,
   station_information::capacity as capacity,
   station_status::num_bikes_available as num_bikes_available,
   station_status::num_docks_available as num_docks_available,
   station_status::num_docks_disabled as num_docks_disabled,
   1-((float)station_information::capacity-(float)station_status::num_bikes_available)/(float)station_information::capacity  as percent_bikes,
   station_information::lat as lat,
   station_information::lon as lon;
```

Saving result as json file with JsonStorage

```
STORE join_project_f INTO '/user/hirwuser864/bikes_output/output.json' USING JsonStorage();
```

Subset of output

{"station_id":7000,"address":"Fort York Blvd / Capreol Crt","capacity":31,"num_bikes_available":31,"num_docks_available":0,"num_docks_disabled":0,"percent_bikes":1.0,"lat":43.63983,"lon":-79.39595}
{"station_id":7078,"address":"College St / Major St","capacity":11,"num_bikes_available":8,"num_docks_available":2,"num_docks_disabled":0,"percent_bikes":0.72727275,"lat":43.6576,"lon":-79.4032}

195 - 202

4.4. Find regions of the world with the highest usage of smartphones and social media in 2016 and 2017 with Hive

Find regions of the world with the highest usage of smartphones and social media in 2016 and 2017 with Hive

Aggregating, merging datasets, creating partitions in Hive to analyze the state of smartphones and Internet across countries of the world
Countries available: 39

Plan:

1. Load data in two tables, one holds population data, second holds social media and smartphone data
2. Merge two tables
3. Use CASE to divide data by regions
4. Group by regions
5. Create partitions

Population data was taken from World Bank (https://data.worldbank.org/indicator/SP.POP.TOTL) Data about smartphone and social media usage was taken from Pew Research Center (http://www.pewglobal.org/2018/06/19/social-media-use-continues-to-rise-in-developing-countries-but-plateaus-across-developed-ones/)

Subset of population data

country	year	population
Aruba	2016	104822
Aruba	2017	105264
Afghanistan	2016	34656032
Afghanistan	2017	35530081

Subset of internet/smartphone data

country	internet_use	smartphone_own	social_media_usage	year
United States	0.89	0.77	0.69	2017
Canada	0.91	0.71	0.68	2017

country	internet_use	smartphone_own	social_media_usage	year
France	87	0.62	0.53	2017
Germany	0.87	0.72	0.4	2017

Create a database

```
CREATE DATABASE project;
USE project;
```

Create and load data insde a table that holds smartphone/internet data

```
CREATE EXTERNAL TABLE IF NOT EXISTS internet_data (
country STRING,
internet_use FLOAT,
smartphone_own FLOAT,
social_media_usage FLOAT,
year INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/hirwuser864/internet_data'
TBLPROPERTIES ('creator'='me', 'created_on' = '2018-08-10',
  'description'='This table holds internet data', "skip.header.line.count"="1");

SELECT * FROM internet_data
LIMIT 10;
```

Result of a select statement

United States 0.89 0.77 0.69 2017
Canada 0.91 0.71 0.68 2017
France 87.0 0.62 0.53 2017
Germany 0.87 0.72 0.4 2017
Greece 0.66 0.53 0.45 2017
Hungary 0.74 0.61 0.56 2017
Italy 0.71 0.67 0.48 2017
Netherlands 0.93 0.8 0.61 2017
Poland 0.75 0.57 0.46 2017
Spain 0.87 0.79 0.59 2017

Create and load data insde a table that holds population data

```
CREATE EXTERNAL TABLE IF NOT EXISTS population_data (
country STRING,
year INT,
Population INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/hirwuser864/population_data'
TBLPROPERTIES ('creator'='me', 'created_on' = '2018-08-10',
  'description'='This table holds population data', "skip.header.line.count"="1");
```

```
SELECT * FROM population_data
LIMIT 10;
```

Result of a select statement

Aruba 2016 104822
Aruba 2017 105264
Afghanistan 2016 34656032
Afghanistan 2017 35530081
Angola 2016 28813463
Angola 2017 29784193
Albania 2016 2876101
Albania 2017 2873457
Andorra 2016 77281
Andorra 2017 76965

Merge these two tables on country and year

```
CREATE TABLE merged_i_p as
SELECT i.country, i.year, i.internet_use, i.smartphone_own, i.social_media_usage, p
.population
FROM internet_data i INNER JOIN population_data p
ON i.country = p.country AND i.year = p.year;

SELECT * FROM merged_i_p
LIMIT 10;
```

Argentina 2016 0.71 0.48 0.59 43847430
Argentina 2017 0.78 0.65 0.65 44271041
Australia 2016 0.93 0.79 0.7 24210809
Australia 2017 0.93 0.82 0.69 24598933
Brazil 2016 0.6 0.41 0.48 207652865
Brazil 2017 0.7 0.54 0.53 209288278
Canada 2016 0.91 0.72 0.65 36264604
Canada 2017 0.91 0.71 0.68 36708083
Chile 2016 0.78 0.65 0.66 17909754
Chile 2017 0.78 0.72 0.63 18054726

Add region to a table

6 regions is total: Europe, Asia, Africa, North America, Latin America and Middle East

```
CREATE TABLE merge_continent as
SELECT *,
CASE
WHEN country in ( 'United States', 'Canada')  THEN 'North America'
WHEN country in ( 'France', 'Germany', 'Greece', 'Hungary',
  'Italy', 'Netherlands', 'Poland', 'Spain', 'Sweden',
  'United Kingdom', 'Russia')  THEN 'Europe'
WHEN country in ('Australia', 'China', 'India', 'Indonesia',
  'Japan', 'Philippines', 'South Korea', 'Vietnam')  THEN 'Asia'
```

```
WHEN country in ('Israel', 'Jordan', 'Lebanon', 'Tunisia', 'Turkey')  THEN 'Middle
 East'
WHEN country in ('Ghana', 'Kenya', 'Nigeria', 'Senegal',
  'South Africa', 'Tanzania')  THEN 'Africa'
WHEN country in ('Argentina', 'Brazil', 'Chile', 'Colombia',
  'Mexico', 'Peru', 'Venezuela')  THEN 'Latin America'
ELSE null
END AS continent
FROM merged_i_p;

SELECT * FROM merge_continent
LIMIT 10;
```

Argentina 2016 0.71 0.48 0.59 43847430 Latin America
Argentina 2017 0.78 0.65 0.65 44271041 Latin America
Australia 2016 0.93 0.79 0.7 24210809 Asia
Australia 2017 0.93 0.82 0.69 24598933 Asia
Brazil 2016 0.6 0.41 0.48 207652865 Latin America
Brazil 2017 0.7 0.54 0.53 209288278 Latin America
Canada 2016 0.91 0.72 0.65 36264604 North America
Canada 2017 0.91 0.71 0.68 36708083 North America
Chile 2016 0.78 0.65 0.66 17909754 Latin America
Chile 2017 0.78 0.72 0.63 18054726 Latin America

Top-3 continents with highest average rating of smartphone ownership in 2016 and 2017 and region with highest difference (the one that develops the fastest)

```
CREATE TABLE smart_2017 as
SELECT continent, round(smartphone_own_avg, 3) as rounded_smart FROM continent_grou
p
WHERE year = '2017'
SORT BY rounded_smart DESC;

CREATE TABLE smart_2016 as
SELECT continent, round(smartphone_own_avg, 3) as rounded_smart FROM continent_grou
p
WHERE year = '2016'
SORT BY rounded_smart DESC;

SELECT continent, rounded_smart FROM smart_2016
SORT BY rounded_smart DESC
LIMIT 3;

SELECT continent, rounded_smart FROM smart_2017
SORT BY rounded_smart DESC
LIMIT 3;
```

In 2016

0.72 = 72%

Continent	% of people who owns smartphones on average

North America |0.72| Europe |0.626| |Middle East |0.496|

In 2017

Continent	% of people who owns smartphones on average
North America	0.74
Europe	0.675
Middle East	0.67

Differece between years

```
CREATE TABLE merged_smart as
SELECT s.continent, s.rounded_smart as data_2016, d.rounded_smart as data_2017
FROM smart_2016 s INNER JOIN smart_2017 d
ON s.continent = d.continent;

SELECT continent, round(data_2017-data_2016, 3) as diff FROM merged_smart
SORT BY diff DESC;
```

Continent	% of change from 16 to 17
Middle East	0.174
Latin America	0.117
Africa	0.088

In summary, the same 3 continents were leaders in number of people who owns smartphones in both 2016 and 2017 - North America, Europa, Middle East. As of continents that develops the fastest, Middle East, Latin America and Africa have the highest development rate.

Top-3 continents with highest average rating of social media in 2016 and 2017 and region with highest difference (the one that develops the fastest)

```
CREATE TABLE sm_2017 as
SELECT continent, round(social_media_usage_avg, 3) as rounded_sm FROM continent_gro
```

```
up
WHERE year = '2017'
SORT BY rounded_sm DESC;

CREATE TABLE sm_2016 as
SELECT continent, round(social_media_usage_avg, 3) as rounded_sm FROM continent_gro
up
WHERE year = '2016'
SORT BY rounded_sm DESC;

SELECT continent, rounded_sm FROM sm_2017
SORT BY rounded_sm DESC
LIMIT 3;

SELECT continent, rounded_sm FROM sm_2016
SORT BY rounded_sm DESC
LIMIT 3;
```

In 2016

Continent	% of people who uses social medias on average
North America	0.67
Europe	0.557
Middle East	0.546

In 2017

Continent	% of people who uses social medias on average
North America	0.685
Middle East	0.632
Latin America	0.581

Differece between years

```
CREATE TABLE merged_sm as
SELECT s.continent, s.rounded_sm as data_2016, d.rounded_sm as data_2017
FROM sm_2016 s INNER JOIN sm_2017 d
ON s.continent = d.continent;

SELECT continent, round(data_2017-data_2016, 3) as diff FROM merged_sm
SORT BY diff DESC;
```

Continent	% of change from 16 to 17

Continent	% of change from 16 to 17
Middle East	0.086
Africa	0.065
Latin America	0.061

In summary, North America and Middle East were leaders in number of people who uses social media in both 2016 and 2017. In contrast, Latin America managed to get 3rd place in 2017, taking a position of Europe which lost its' second place in 2017. As of continents that develops the fastest, Middle East, Latin America and Africa have the fastest spread of social media.

Create partitions to store data by continents
Number of continents is not huge, so "dynamic partitioning" can be applied

```
SET hive.exec.dynamic.partition.mode=nonstrict;
INSERT OVERWRITE TABLE merged_partition_dynamic
PARTITION (cont)
SELECT m.*, m.continent
FROM merge_continent m;

SHOW PARTITIONS merged_partition_dynamic;
```

cont=Africa
cont=Asia
cont=Europe
cont=Latin America
cont=Middle East
cont=North America

```
SELECT * FROM merged_partition_dynamic
WHERE cont='Europe'
LIMIT 5;
```

Germany 2016 0.85 0.66 0.37 82348669 Europe Europe
Germany 2017 0.87 0.72 0.4 82695000 Europe Europe
Spain 2016 0.9 0.79 0.63 46484062 Europe Europe
Spain 2017 0.87 0.79 0.59 46572028 Europe Europe
France 2016 0.81 0.58 0.48 66859768 Europe Europe

Spark and Scala

204 - 211

5.1. Spark/Scala: predict price of a diamond with decision tree and random forest

Spark/Scala: predict price of a diamond with decision tree and random forest

Usage of Spark machine learning (Linear Regression, Decision tree, Random forest) to create a model that predicts a price of diamonds on a basis of different features of them. GridSearch is applied to find the best combination of parameters of a model

Information about the dataset

- Number of inputs: 53 941
- Number of features: 11
- Source of data: https://www.kaggle.com/shivam2503/diamonds (https://www.kaggle.com/shivam2503/diamonds)

Import libraries and start of spark session

```
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.ml.tuning.{ParamGridBuilder, TrainValidationSplit}

// To see less warnings
import org.apache.log4j._
Logger.getLogger("org").setLevel(Level.ERROR) //less warnings pop up


// Start a simple Spark Session
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().getOrCreate()
```

Import dataset and print schema

```
val data = spark.read.option("header","true").option("inferSchema","true").format("
csv").load("diamonds.csv")
data.printSchema()
```

|-- _co: integer (nullable = true)
|-- carat: double (nullable = true)
|-- cut: string (nullable = true)
|-- color: string (nullable = true)
|-- clarity: string (nullable = true)
|-- depth: double (nullable = true)
|-- table: double (nullable = true)
|-- price: integer (nullable = true)

|-- x: double (nullable = true)
|-- y: double (nullable = true)
|-- z: double (nullable = true)

Subset of data

```
data.show
```

+---+-----+---------+-----+-------+-----+-----+-----+----+----
+----+
|_c0|carat| cut|color|clarity|depth|table|price| x| y| z|
+---+-----+---------+-----+-------+-----+-----+-----+----+----
+----+
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.2	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Some data preprocessing

```
//drop column with ids
val df_noid = data.drop(data.col("_c0"))

val df_no_na = df_noid.na.drop()

val df_label = df_no_na.select(data("price").as("label"), $"carat", $"cut", $"color
", $"clarity",
  $"depth", $"table", $"x", $"y", $"z")
```

Encode categorical variables: convert strings to integers and encode with OneHotEncoderEstimator

```
// Import VectorAssembler and Vectors
import org.apache.spark.ml.feature.{VectorAssembler,StringIndexer,VectorIndexer,On
eHotEncoder}
import org.apache.spark.ml.linalg.Vectors

val cutIndexer = new StringIndexer().setInputCol("cut").setOutputCol("cutIndex")
val colorIndexer = new StringIndexer().setInputCol("color").setOutputCol("colorInd
ex")
val clarityIndexer = new StringIndexer().setInputCol("clarity").setOutputCol("clar
ityIndex")

import org.apache.spark.ml.feature.OneHotEncoderEstimator
val encoder = new OneHotEncoderEstimator().setInputCols(Array("cutIndex", "colorIn
dex", "clarityIndex"))
  .setOutputCols(Array("cutIndexEnc", "colorIndexEnc", "clarityIndexEnc"))
```

Vector assembler

```
val assembler = (new VectorAssembler()
                      .setInputCols(Array("carat", "cutIndexEnc", "colorIndexEnc", "c
larityIndexEnc",
                      "depth", "table", "x", "y", "z"))
                      .setOutputCol("features_assem") )
```

Scalling of features with MinMaxScaler

```
import org.apache.spark.ml.feature.MinMaxScaler
val scaler = new MinMaxScaler().setInputCol("features_assem").setOutputCol("featur
es")
```

Train/Test split

```
val Array(training, test) = df_label.randomSplit(Array(0.75, 0.25))
```

Decision Tree

Building a decision tree, contructing a pipeline and creating a ParamGrid

Parameters: Max depth(5, 10, 15, 20, 30) and Max Bins(10, 20, 30, 50)

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.regression.DecisionTreeRegressionModel
import org.apache.spark.ml.regression.DecisionTreeRegressor

import org.apache.spark.ml.tuning.{CrossValidator, ParamGridBuilder}

val dt = new DecisionTreeRegressor().setLabelCol("label").setFeaturesCol("features
")

val pipeline = new Pipeline().setStages(Array(cutIndexer,colorIndexer,
  clarityIndexer,encoder, assembler,scaler, dt))

val paramGrid = new ParamGridBuilder().addGrid(dt.maxDepth, Array(5, 10, 15, 20, 3
0))
  .addGrid(dt.maxBins, Array(10, 20, 30, 50)).build()
```

Cross-validation (3 splits); Predict test data

```
val cv = new CrossValidator().setEstimator(pipeline).setEvaluator(new RegressionEv
aluator)
  .setEstimatorParamMaps(paramGrid).setNumFolds(3)
val cvModel = cv.fit(training)
val predictions = cvModel.transform(test)
```

Evaluate a model

```
// Select (prediction, true label) and compute test error.
```

```
val evaluator = new RegressionEvaluator().setLabelCol("label").setPredictionCol("p
rediction")
  .setMetricName("rmse")
val rmse = evaluator.evaluate(predictions)
println("Root Mean Squared Error (RMSE) on test data = " + rmse)

// Select (prediction, true label) and compute test error.
val evaluator_r2 = new RegressionEvaluator().setLabelCol("label").setPredictionCol
("prediction")
  .setMetricName("r2")
val r2 = evaluator_r2.evaluate(predictions)
println("R-squared (r^2) on test data = " + r2)

// Select (prediction, true label) and compute test error.
val evaluator_mae = new RegressionEvaluator().setLabelCol("label").setPredictionCo
l("prediction")
  .setMetricName("mae")
val mae = evaluator_mae.evaluate(predictions)
println("Mean Absolute Error (MAE) on test data = " + mae)

// Select (prediction, true label) and compute test error.
val evaluator_mse = new RegressionEvaluator().setLabelCol("label").setPredictionCo
l("prediction")
  .setMetricName("mse")
val mse = evaluator_mse.evaluate(predictions)
println("Mean Squared Error (MSE) on test data = " + mse)
predictions.select("features", "label", "prediction").show()
```

Root Mean Squared Error (RMSE) on test data = 839.790709763866
R-squared (r^2) on test data = 0.9556915131409848
Mean Absolute Error (MAE) on test data = 381.5670094175047
Mean Squared Error (MSE) on test data = 705248.4362056978

Predictions of decision tree model

```
+-----+------------------+
|label| prediction|
+-----+------------------+
| 326| 724.0|
| 334| 445.74|
| 337| 362.0|
| 337| 360.0|
| 340| 445|
| 344|448|
| 357| 445.74|
| 357| 445.74|
| 357|388.27|
| 360| 371.5|
| 361| 564.26|
| 362| 445.74|
| 363| 385.0|
```

```
|363|435.57|
|365| 533.22|
|367| 625.0|
|367| 445.74|
|367| 445.74|
|367| 445.74|
|367| 445.74|
+-----+------------------+
```

Random Forest

Building a random forest, contructing a pipeline and creating a ParamGrid

Parameters to tune: Max Depth (5, 10, 15, 20, 30, 50), Max Bins (10, 20, 30, 50), Number of trees (10, 20).

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.regression.{RandomForestRegressionModel, RandomForestRe
gressor}
import org.apache.spark.ml.tuning.{CrossValidator, ParamGridBuilder}

val rf = new RandomForestRegressor().setLabelCol("label").setFeaturesCol("features
")

val pipeline = new Pipeline().setStages(Array(cutIndexer,colorIndexer, clarityInde
xer,
  encoder, assembler,scaler, rf))

val paramGrid = new ParamGridBuilder().addGrid(rf.maxDepth, Array(5, 10, 15, 20, 3
0, 50))
  .addGrid(rf.maxBins, Array(10, 20, 30, 50)).addGrid(rf.numTrees, Array(10, 20)).b
uild()
```

Cross-validation (3 splits); Predict test data

```
val cv = new CrossValidator().setEstimator(pipeline).setEvaluator(new RegressionEv
aluator)
  .setEstimatorParamMaps(paramGrid).setNumFolds(3)
val cvModel = cv.fit(training)
val predictions = cvModel.transform(test)
```

Evaluate a model

```
// Select (prediction, true label) and compute test error.
val evaluator = new RegressionEvaluator().setLabelCol("label").setPredictionCol("p
rediction")
  .setMetricName("rmse")
val rmse = evaluator.evaluate(predictions)
println("Root Mean Squared Error (RMSE) on test data = " + rmse)
```

```
// Select (prediction, true label) and compute test error.
val evaluator_r2 = new RegressionEvaluator().setLabelCol("label").setPredictionCol
("prediction")
  .setMetricName("r2")
val r2 = evaluator_r2.evaluate(predictions)
println("R-squared (r^2) on test data = " + r2)

// Select (prediction, true label) and compute test error.
val evaluator_mae = new RegressionEvaluator().setLabelCol("label").setPredictionCo
l("prediction")
  .setMetricName("mae")
val mae = evaluator_mae.evaluate(predictions)
println("Mean Absolute Error (MAE) on test data = " + mae)

// Select (prediction, true label) and compute test error.
val evaluator_mse = new RegressionEvaluator().setLabelCol("label").setPredictionCo
l("prediction")
  .setMetricName("mse")
val mse = evaluator_mse.evaluate(predictions)
println("Mean Squared Error (MSE) on test data = " + mse)
predictions.select("features", "label", "prediction").show()
```

Root Mean Squared Error (RMSE) on test data = 570.8512705968417
Root Mean Squared Error (r^2) on test data = 0.9792948676072686
Root Mean Squared Error (MAE) on test data = 257.77994671313667
Root Mean Squared Error (MSE) on test data = 325871.1731420286

Predictions of random forest model

```
+------------------+-----+
| prediction|label|
+------------------+-----+
| 445.74| 334|
| 445.66| 340|
| 351.0| 351|
| 352.0| 352|
| 410.0| 353|
| 355.0| 355|
| 445.74| 357|
| 574.17| 357|
| 362.0| 362|
| 385.0| 363|
|435.57| 363|
| 385.0| 364|
| 445.74| 367|
| 445.74| 367|
| 377.5| 367|
| 410.0| 367|
|384.853| 368|
```

416.0	*368*
371.5	*371*
373.0	*373*

212 - 217

5.2. Spark/Scala Classification task: predict student performance

Spark/Scala Classification task: predict student performance

Predict if student passes a course or not on a basis of his or her personal life, activities in school and outside

Information about the dataset

- Number of inputs: 650
- Number of features: 29
- Source of data: https://archive.ics.uci.edu/ml/datasets/student+performance (https://archive.ics.uci.edu/ml/datasets/student+performance)

Import libraries and start of spark session

```
// To see less warnings
import org.apache.log4j._
Logger.getLogger("org").setLevel(Level.ERROR) //less warnings pop up


// Start a simple Spark Session
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().getOrCreate()
```

Import dataset and print schema

```
val data = spark.read.option("header","true").option("inferSchema","true").format("
csv").load("student-por.csv")
data.printSchema()
```

|-- *sex: string (nullable = true)*
|-- *age: integer (nullable = true)*
|-- *address: string (nullable = true)*
|-- *famsize: string (nullable = true)*
|-- *Pstatus: string (nullable = true)*
|-- *Medu: integer (nullable = true)*
|-- *Fedu: integer (nullable = true)*
|-- *Mjob: string (nullable = true)*
|-- *Fjob: string (nullable = true)*
|-- *reason: string (nullable = true)*
|-- *guardian: string (nullable = true)*
|-- *traveltime: integer (nullable = true)*
|-- *studytime: integer (nullable = true)*

|-- *failures: integer (nullable = true)*
|-- *schoolsup: string (nullable = true)*
|-- *famsup: string (nullable = true)*
|-- *paid: string (nullable = true)*
|-- *activities: string (nullable = true)*
|-- *nursery: string (nullable = true)*
|-- *higher: string (nullable = true)*
|-- *internet: string (nullable = true)*
|-- *romantic: string (nullable = true)*
|-- *famrel: integer (nullable = true)*
|-- *freetime: integer (nullable = true)*
|-- *goout: integer (nullable = true)*
|-- *Dalc: integer (nullable = true)*
|-- *Walc: integer (nullable = true)*
|-- *health: integer (nullable = true)*
|-- *absences: integer (nullable = true)*
|-- *G1: integer (nullable = true)*
|-- *G2: integer (nullable = true)*
|-- *G3: integer (nullable = true)*

Some data preprocessing

```
val df_pass = data.withColumn("pass", when($"G3" >= 10 , 1).otherwise(0)).drop("G1
").drop("G2").drop("G3")

val df_label = df_pass.select($"school", $"sex", $"age", $"address", $"famsize", $"
Pstatus", $"Medu",
  $"Fedu", $"Mjob", $"Fjob", $"reason", $"guardian", $"traveltime", $"studytime", $
"failures",
  $"schoolsup", $"famsup", $"paid", $"activities", $"nursery",$"higher", $"internet
", $"romantic",
   $"famrel", $"freetime", $"goout", $"Dalc", $"Walc", $"health", $"absences",
   df_pass("pass").as("label"))
```

Encode categorical variables: convert strings to integers and encode with OneHotEncoderEstimator

```
// Import VectorAssembler and Vectors
import org.apache.spark.ml.feature.{VectorAssembler,StringIndexer,VectorIndexer,On
eHotEncoder}
import org.apache.spark.ml.linalg.Vectors

//categorical_var = [0,1,3,4,5,8,9,10,11,15,16,17,18,19,20,21,22]

val schoolIndexer = new StringIndexer().setInputCol("school").setOutputCol("school
Index")
val sexIndexer = new StringIndexer().setInputCol("sex").setOutputCol("sexIndex")
val addressIndexer = new StringIndexer().setInputCol("address").setOutputCol("addr
essIndex")
val famsizeIndexer = new StringIndexer().setInputCol("famsize").setOutputCol("fams
izeIndex")
```

```
val PstatusIndexer = new StringIndexer().setInputCol("Pstatus").setOutputCol("PstatusIndex")
val MjobIndexer = new StringIndexer().setInputCol("Mjob").setOutputCol("MjobIndex")
val FjobIndexer = new StringIndexer().setInputCol("Fjob").setOutputCol("FjobIndex")
val reasonIndexer = new StringIndexer().setInputCol("reason").setOutputCol("reasonIndex")
val guardianIndexer = new StringIndexer().setInputCol("guardian").setOutputCol("guardianIndex")
val schoolsupIndexer = new StringIndexer().setInputCol("schoolsup").setOutputCol("schoolsupIndex")
val famsupIndexer = new StringIndexer().setInputCol("famsup").setOutputCol("famsupIndex")
val paidIndexer = new StringIndexer().setInputCol("paid").setOutputCol("paidIndex")
val activitiesIndexer = new StringIndexer().setInputCol("activities").setOutputCol("activitiesIndex")
val nurseryIndexer = new StringIndexer().setInputCol("nursery").setOutputCol("nurseryIndex")
val higherIndexer = new StringIndexer().setInputCol("higher").setOutputCol("higherIndex")
val internetIndexer = new StringIndexer().setInputCol("internet").setOutputCol("internetIndex")
val romanticIndexer = new StringIndexer().setInputCol("romantic").setOutputCol("romanticIndex")

import org.apache.spark.ml.feature.OneHotEncoderEstimator

val encoder = new OneHotEncoderEstimator().setInputCols(Array("schoolIndex", "sexIndex",
"addressIndex", "famsizeIndex", "PstatusIndex", "Medu", "Fedu", "MjobIndex", "FjobIndex",
  "reasonIndex", "guardianIndex", "traveltime", "studytime", "failures", "schoolsupIndex",
  "famsupIndex", "paidIndex", "activitiesIndex", "nurseryIndex",
  "higherIndex", "internetIndex", "romanticIndex", "famrel", "freetime",
  "goout", "Dalc", "Walc", "health"))
  .setOutputCols(Array("schoolEnc", "sexEnc", "addressEnc", "famsizeEnc", "PstatusEnc",
  "MeduEnc", "FeduEnc", "MjobEnc", "FjobEnc",
  "reasonEnc", "guardianEnc", "traveltimeEnc", "studytimeEnc", "failuresEnc", "schoolsupEnc",
  "famsupEnc", "paidIndexEnc", "activitiesEnc", "nurseryEnc",
  "higherEnc", "internetEnc", "romanticEnc", "famrelEnc", "freetimeEnc", "gooutEnc", "DalcEnc",
  "WalcEnc", "healthEnc"))
```

Vector assembler

```
val assembler = (new VectorAssembler()
                  .setInputCols(Array("schoolEnc", "sexEnc", "addressEnc", "famsize
```

```
Enc",
                    "PstatusEnc", "MeduEnc", "FeduEnc", "MjobEnc", "FjobEnc",
                    "reasonEnc", "guardianEnc", "traveltimeEnc", "studytimeEnc", "fai
luresEnc",
                    "schoolsupEnc", "famsupEnc", "paidIndexEnc", "activitiesEnc", "nu
rseryEnc",
                    "higherEnc", "internetEnc", "romanticEnc", "famrelEnc", "freetime
Enc", "gooutEnc",
                    "DalcEnc", "WalcEnc", "healthEnc", "age", "absences"))
                    .setOutputCol("features_assem") )
```

Scalling of features with MinMaxScaler

```
import org.apache.spark.ml.feature.MinMaxScaler
val scaler = new MinMaxScaler().setInputCol("features_assem").setOutputCol("featur
es")
```

Train/Test split

```
val Array(training, test) = df_label.randomSplit(Array(0.75, 0.25))
```

Decision Tree

Building a decision tree, contructing a pipeline and creating a ParamGrid

Parameters: Max depth(5, 10, 15, 20, 30) and Max Bins(10, 20, 30, 50)

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.DecisionTreeClassificationModel
import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator
import org.apache.spark.ml.tuning.{CrossValidator, ParamGridBuilder}

val dt = new DecisionTreeClassifier().setLabelCol("label").setFeaturesCol("feature
s")

val pipeline = new Pipeline().setStages(Array(schoolIndexer,sexIndexer,addressInde
xer,famsizeIndexer,
  PstatusIndexer,MjobIndexer,FjobIndexer,reasonIndexer,
  guardianIndexer,schoolsupIndexer,famsupIndexer,paidIndexer,activitiesIndexer,nurs
eryIndexer,
  higherIndexer,internetIndexer,romanticIndexer,encoder, assembler,scaler, dt))

val paramGrid = new ParamGridBuilder().addGrid(dt.maxDepth, Array(5, 10, 15, 20, 3
0)).addGrid(dt.maxBins,
  Array(10, 20, 30, 50)).build()
```

Cross-validation (3 splits); Predict test data

```scala
val cv = new CrossValidator().setEstimator(pipeline).setEvaluator(new BinaryClassificationEvaluator)
  .setEstimatorParamMaps(paramGrid).setNumFolds(3)
val cvModel = cv.fit(training)
val predictions = cvModel.transform(test)
```

Evaluate a model

```scala
import org.apache.spark.mllib.evaluation.MulticlassMetrics

// Convert the test results to an RDD using .as and .rdd
val predictionAndLabels = predictions.select($"prediction",$"label").as[(Double, Double)].rdd

// Instantiate a new MulticlassMetrics object
val metrics = new MulticlassMetrics(predictionAndLabels)

// Print out the Confusion matrix
println("Confusion matrix:")
println(metrics.confusionMatrix)
```

Confusion matrix:
14.0 11.0
12.0 134.0

Predictions of decision tree model

```
+-----+----------+
|label|prediction|
+-----+----------+
| 1| 1.0|
| 0| 0.0|
| 1| 1.0|
| 1| 1.0|
| 1| 1.0|
| 1| 1.0|
| 0| 1.0|
| 1| 0.0|
| 1| 1.0|
| 1| 1.0|
| 1| 1.0|
| 1| 1.0|
| 1| 1.0|
| 1| 0.0|
| 1| 1.0|
| 1| 1.0|
| 1| 1.0|
| 1| 1.0|
| 1| 1.0|
```

Other (Web scraping, graphs, MongoDB)

219 - 224

6.1. Plot temperature records in Munich for 5 years (2012-2016) and days in 2017 that broke these records with matplotlib

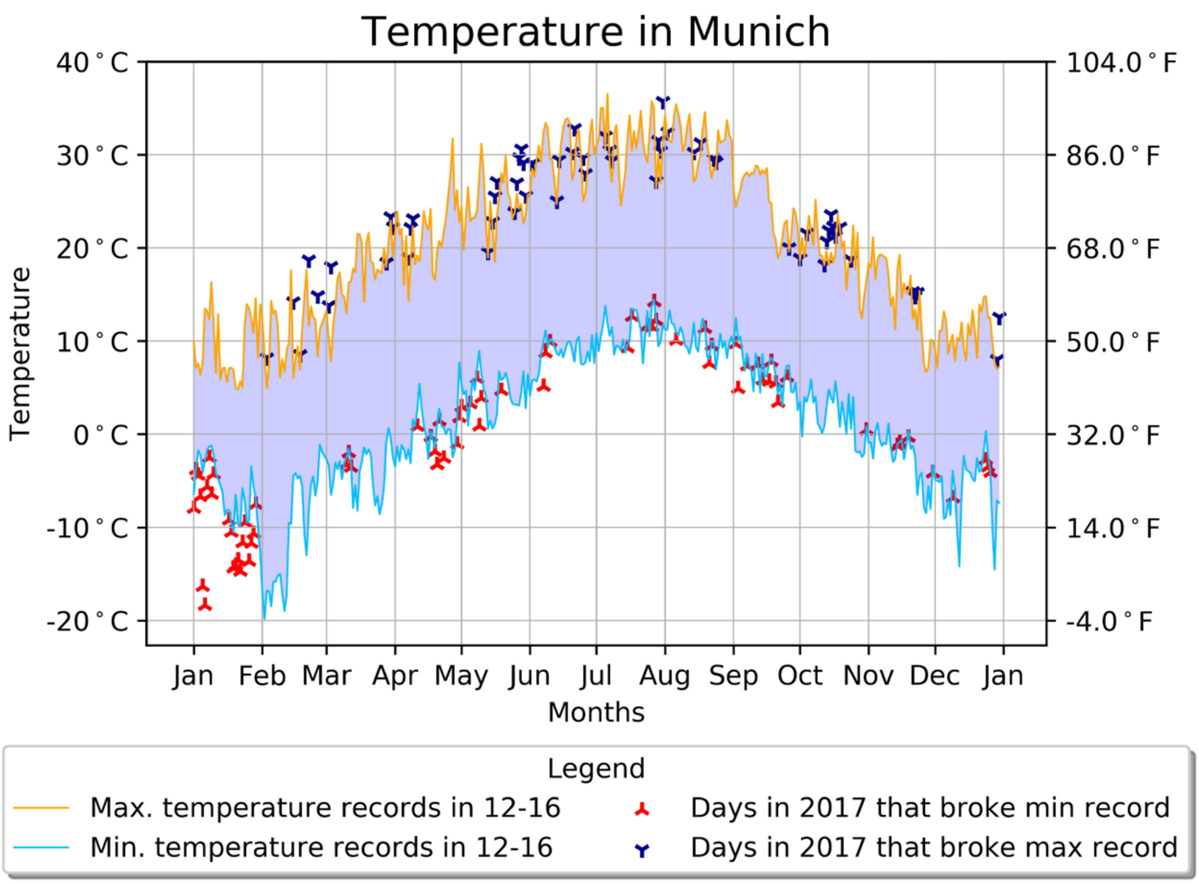
Plot temperature records in Munich for 5 years (2012-2016) and days in 2017 that broke these records with matplotlib

Temperature data was extracted with **Dark Sky API**: https://darksky.net/dev
Total number of rows: 2193
City to analyze: Munich
Date range: 2012-2017

The plot looks like this:

In [8]:
```
from IPython.display import Image
Image(filename='munich.png')
```

Out[8]:



The code to create this plot

Data extraction

Extracting historical weather data from darksky api; Data is stored in JSON format for further processing

In [ ]:
```
import requests
```

```python
import time
import os
import json

#create file to put info in
with open(os.path.join("D:/weather_matplotlib/", "data_.json"), 'w'):
    pass

#API parameters
params = (
    ('exclude', 'currently,flags,minutely,hourly,alerts'),
    ('units', 'si'),
)

API_key = '[Insert API key]'
timestamp = 1388271600 #31.12.12 - unix

lat = '48.135125'
lon = '11.581980'
day_num =  365*6 #(6 years)

for i in range(day_num):
    print("Iteration: "+str(i))
    request = 'https://api.darksky.net/forecast/'+API_key+'/'+lat+','+lo
n+','+ str(timestamp)
    response = requests.get(request, params=params)
    data = response.json()
    with open("D:/weather_matplotlib/data_.json", 'a') as f:
        f.write(json.dumps(data['daily']['data'][0]))
        f.write("\n")
    timestamp = timestamp + 86400  #adds one day to a timestamp
    #if i % 5 == 0: #making a pause for 3 seconds every 5th day
        #time.sleep(2)
f.close()
```

Import data

In [9]:
```python
import pandas as pd
import numpy as np

df = pd.read_json("data_.json", lines=True)
df = df[["time", "apparentTemperatureMax",
         "apparentTemperatureMin"]]
```

Convert unix to datetime

In [10]:
```python
from datetime import datetime
df["datetime"] = pd.to_datetime(df['time'],unit='s').dt.date #leave only
 date, remove time
```

Extract year, month and day from date

In [11]:
```python
df["year"] = pd.DatetimeIndex(df['datetime']).year
df["month"] = pd.DatetimeIndex(df['datetime']).month
df["day"] = pd.DatetimeIndex(df['datetime']).day
```

Divide the datadrame in two, one contains data from 2012 to 2016, the second contains

data from 2017

```
In [12]: df_12_16 =  df.drop(df[df["year"] == 2017].index)
df_17 =  df.drop(df[df["year"] != 2017].index)
```

Apply groupby to calculate max and min temperature of years 2012-2016

```
In [13]: df_groupped_min = df_12_16.groupby(['month', 'day'])[["apparentTemperatu
reMin"]].min().reset_index()
df_groupped_max = df_12_16.groupby(['month', 'day'])[["apparentTemperatu
reMax"]].max().reset_index()
```

Merge month and day in one column

```
In [14]: df_groupped_min["date"] = df_groupped_min["month"].map(str) + "/" + df_g
roupped_min["day"].map(str)
df_groupped_max["date"] = df_groupped_max["month"].map(str) + "/" + df_g
roupped_max["day"].map(str)
```

Drop useless columns

```
In [15]: df_groupped_min = df_groupped_min[["date","apparentTemperatureMin"]]
df_groupped_max = df_groupped_max[["date","apparentTemperatureMax"]]
```

Merge columns with min and max values of years 2012-2016

```
In [16]: merged_df=pd.merge(df_groupped_min, df_groupped_max, how='inner', left_o
n='date', right_on='date')
merged_df = merged_df.rename(index=str, columns={"apparentTemperatureMin
": "min_12_16", "apparentTemperatureMax": "max_12_16"})
```

Dataframe with values of the year 2017 is processed

```
In [17]: df_17["date"] = df_17["month"].map(str) + "/" + df_17["day"].map(str)
df_17 = df_17[["date","apparentTemperatureMin","apparentTemperatureMax"]
]
df_17 = df_17.rename(index=str, columns={"apparentTemperatureMin": "min_
17", "apparentTemperatureMax": "max_17"})
```

Merge dataframes of years 2012-16 with data from 2017

```
In [18]: df_fin=pd.merge(df_17, merged_df, how='inner', left_on='date', right_on=
'date')
```

Add some random year (this step is required to plot data)

```
In [19]: df_fin["date"] = df_fin["date"].map(str) + "/" + "2000"
df_fin['date'] = pd.to_datetime(df_fin['date'])
```

Create boolen variable that indicates either temperature from year 2017 broke record or not

In [20]:
```
df_fin['max_overcame'] = np.where(df_fin['max_17']>df_fin["max_12_16"],
'true', 'false')
df_fin['min_overcame'] = np.where(df_fin['min_17']<df_fin["min_12_16"],
'true', 'false')
```

Variables to create scatterplots later

In [22]:
```
df_min_o = df_fin.loc[df_fin['min_overcame'] == 'true'][["date","min_17"]
]]
df_max_o = df_fin.loc[df_fin['max_overcame'] == 'true'][["date","max_17"]
]]
```

Plot the data

In [23]:
```
import matplotlib.pyplot as plt

#scatterplot needs lists
plt.plot(df_fin["date"], df_fin["max_12_16"], linestyle='-', color='oran
ge', linewidth=0.7, label='Max. temperature records in 12-16')
plt.plot(df_fin["date"], df_fin["min_12_16"], linestyle='-', color='deep
skyblue', linewidth=0.7, label='Min. temperature records in 12-16')
plt.scatter(df_min_o["date"].tolist(), df_min_o['min_17'], marker ='2',
c='red', label='Days in 2017 that broke min record')
plt.scatter(df_max_o["date"].tolist(), df_max_o['max_17'], marker ='1',
c='darkblue', label='Days in 2017 that broke max record')
plt.title('Temperature in Munich', fontsize=15)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), fancybox=Tru
e, shadow=True, ncol=2, title='Legend')


import matplotlib.dates as mdates
maxv=df_fin['max_12_16']
minv=df_fin['min_12_16']

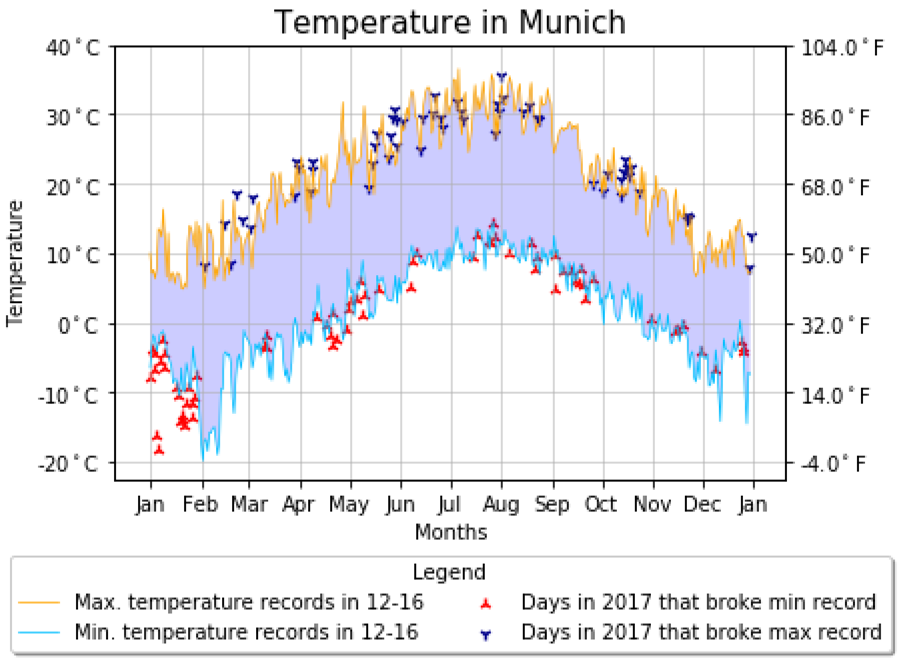
#show only months
ax = plt.gca()
ax.xaxis.set_major_locator(mdates.MonthLocator())
monthFmt = mdates.DateFormatter('%b')
ax.xaxis.set_major_formatter(monthFmt)
plt.xlabel('Months')

#add grid and filling betwees min and max values
d = df_fin['date'].values
plt.gca().fill_between(d, minv, maxv, facecolor='blue', alpha=0.2)
ax.grid(True, linewidth=0.5)

##add celsius
ax.set_yticks(np.arange(-20,50,10)) #ads numbers to y axis
ax.set_yticklabels(str(i)+'$^\circ$C' for i in np.arange(-20,50,10)) #ad
s Celsium symbol
plt.ylabel('Temperature')

##add fahrenheit
ax1=ax.twinx()
ax1.set_yticks(ax.get_yticks())
ax1.set_ylim(ax.get_ylim())
ax1.set_yticklabels(map(lambda x : '{:}'.format((x*1.8)+32)+'$^\circ$F',
 ax1.get_yticks()))
```

```
Out[23]: [Text(1,0,'-4.0$^\\circ$F'),
          Text(1,0,'14.0$^\\circ$F'),
          Text(1,0,'32.0$^\\circ$F'),
          Text(1,0,'50.0$^\\circ$F'),
          Text(1,0,'68.0$^\\circ$F'),
          Text(1,0,'86.0$^\\circ$F'),
          Text(1,0,'104.0$^\\circ$F')]
```



Save plot

```
In [ ]: plt.savefig('munich.png', bbox_inches='tight',dpi = 600)
```

225 - 235

6.2. Web scraping from TripAdvisor’s' pages to extract info about restaurants (Beautifulsoup)

Web scraping from Tripadvisors' pages to extract info about restaurants (Beautifulsoup)

Different attributes of restaurants are extracted from the page of the most popular restaurants in Toronto anf from individual web page of these restaurants. These attributes are then put into a pandas dataframe. Beautifulsoup is used.

Import libraries

In [1]:
```
import requests
import numpy as np
from bs4 import BeautifulSoup
import time
import pandas as pd
```

Extract info from a website

In [2]:
```
url = "https://www.tripadvisor.com/Restaurants-g155019-Toronto_Ontario.h
tml"
headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6
) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537
.36'}

r = requests.get(url, headers=headers)
html_doc = r.content

soup = BeautifulSoup(html_doc, 'html.parser')
elements = soup.findAll('a', attrs={'class': "property_title"})
```

Extract names of restaurants (30 in total)

Tag: 'a'
Attribute: 'class': 'property_title'

In [3]:
```
elements = soup.findAll('a', attrs={'class': 'property_title'})
restaurants=[]
for el in elements:
    restaurants.append(el.text.strip("\n"))
```

In [4]:
```
restaurants[:3]
```

Out[4]:
```
['STK Toronto', 'ALO RESTAURANT', 'Scaramouche Restaurant']
```

Extract number of reviews

Tag: 'div'
Attribute: 'class': 'rating rebrand'

In [5]:
```
reviews=[]
reviews_number = soup.findAll('div', attrs={'class': "rating rebrand"})
for rew in reviews_number:
    reviews.append(rew.contents[3].text.strip("\n"))
```

In [6]:
```
reviews[:3]
```

Out[6]:
```
['350 reviews ', '453 reviews ', '1,388 reviews ']
```

Extract ratings

Tag: 'div'

Attribute: 'class': 'rating rebrand'

In [7]:
```
ratings=[]
ratings_ = soup.findAll('div', attrs={'class': "rating rebrand"})

for ratg in ratings_:
    ratings.append(ratg.contents[1]['alt'].strip("\n"))
```

In [8]:
```
ratings[:3]
```

Out[8]:
```
['4.5 of 5 bubbles', '4.5 of 5 bubbles', '4.5 of 5 bubbles']
```

Extract prices

Tag: 'span'

Attribute: 'class': 'item price'

In [9]:
```
prices=[]
prices_ = soup.findAll('span', attrs={'class': "item price"})

for p in prices_:
    prices.append(p.text)
```

Replace symbols with values

In [10]:
```
for n, i in enumerate(prices):
    if i == "$":
        prices[n] = "Cheap"
    elif i == "$$ - $$$":
        prices[n] = "Medium range"
    elif i == "$$$$":
        prices[n] = "Expensive"
```

In [11]:
```
prices[:3]
```

Out[11]:
```
['Medium range', 'Expensive', 'Expensive']
```

Extract cuisines of restaurants

Tag: 'div'

Attribute: 'class': 'cuisines'

Info is extracted through "children"

In [12]:
```
cuisines_ = soup.findAll('div', attrs={'class': "cuisines"})
cuisines=[]
for c in cuisines_:
    children = c.findChildren("a" , recursive=False)
    lst_temp=[]
    for child in children:
        lst_temp.append(child.text)
    cuisines.append(lst_temp)
```

In [13]:
```
cuisines[1] #single restaurant
```

Out[13]:
```
['French',
 'European',
 'Vegetarian Friendly',
 'Vegan Options',
 'Gluten Free Options']
```

Web links of restaurants to extract info from individual web pages of restaurants

In [14]:
```
links_ = soup.findAll('a', attrs={'class': "property_title"})
links = []
for l in links_:
    links.append(l['href'])
```

All restaraunts are processed via loop

Different attributes are extracted and put into lists

Listst are:

In [15]:
```
addresses = []
locations = []
countries = []
phone_numbs = []
ratings_all = []
details_list = []
reviews_full=[]
```

In [16]:
```
for link in links:
    time.sleep(5)
    url_r = "https://www.tripadvisor.com" + str(link)
    #print("Processing: ", link)
    r_r = requests.get(url_r, headers=headers)
    html_doc_r = r_r.content
    soup_r = BeautifulSoup(html_doc_r, 'html.parser')

##/##/##/##/##/##/##/##/##/##/##/##/##/
#address is extracted
    address_ = soup_r.find('span', attrs={'class': "street-address"})
    addresses.append(address_.text)


##/##/##/##/##/##/##/##/##/##/##/##/##/
#locations
    locations_ = soup_r.find('span', attrs={'class': "locality"})
    locations.append(locations_.text[:-2]) #exclude last comma



##/##/##/##/##/##/##/##/##/##/##/##/##/
#country is extracted
    country_ = soup_r.find('span', attrs={'class': "country-name"})
    countries.append(country_.text.strip("\n"))



##/##/##/##/##/##/##/##/##/##/##/##/##/
#phone number is extracted
```

```
    phone = soup_r.find('div', attrs={'class': "blEntry phone"})
    phone_numbs.append(phone.text.strip("\n"))

##/##/##/##/##/##/##/##/##/##/##/##/##/
#ratings are extracted
    ratings_names = []
    ratings_numbs = []
    ratings_name = soup_r.findAll('div', attrs={'class': "wrap row part
"})
    for r in ratings_name:
        ratings_names.append(r.span['alt'])

    ratings_numb = soup_r.findAll('div', attrs={'class': "label part "})
    for r_n in ratings_numb:
        ratings_numbs.append(r_n.text.strip())

    ratings_dict = {}
    for i in range(len(ratings_names)):
        ratings_dict[ratings_numbs[i]] = ratings_names[i]

    ratings_all.append(ratings_dict)

##/##/##/##/##/##/##/##/##/##/##/##/##/
#details (different features of a resturant) are extracted

    details = soup_r.findAll('div', attrs={'id': "RESTAURANT_DETAILS"})
    for d in details:
        rest_det = str(d.contents[3])

    soup_det = BeautifulSoup(rest_det, 'html.parser')

    ttls = []
    cont = []

    details_ = soup_det.findAll('div', attrs={'class': "title"})
    for det in details_:
        ttls.append(det.text.strip())

    contents_ = soup_det.findAll('div', attrs={'class': "content"})
    for cn in contents_:
        cont.append(cn.text.strip())


    detail_dict = {}
    for i in range(len(ttls)):
        detail_dict[ttls[i]] = cont[i]

    details_list.append(detail_dict)

##/##/##/##/##/##/##/##/##/##/##/##/##/
#long reviews
    reviews_temp=[]
    reviewsf_ = soup_r.findAll('p', attrs={'class': "partial_entry"})
    for rev in reviewsf_:
        reviews_temp.append(rev.text)
    reviews_full.append(reviews_temp)
```

Create a table(dataframe) from attributes extracted before the loop

In [17]:
```
df = pd.DataFrame(
    {'restaurant_name': restaurants,
     'adress': addresses,
     'country': countries,
     'phone_number': phone_numbs,
     'review': reviews,
     'overall_rating': ratings,
     'price': prices,
    })
```

In [18]:
```
df.head()
```

Out[18]:

	restaurant_name	adress	country	phone_number	review	overall_rating	price
0	STK Toronto	153 Yorkville Ave	Canada	+1 416-613-9660	350 reviews	4.5 of 5 bubbles	Medium range
1	ALO RESTAURANT	163 Spadina Ave	Canada	+1 416-260-2222	453 reviews	4.5 of 5 bubbles	Expensive
2	Scaramouche Restaurant	1 Benvenuto Pl	Canada	+1 416-961-8011	1,388 reviews	4.5 of 5 bubbles	Expensive
3	New Orleans Seafood & Steakhouse	267 Scarlett Rd	Canada	+1 416-766-7001	209 reviews	4.5 of 5 bubbles	Medium range
4	Richmond Station	1 Richmond St. West	Canada	+1 647-748-1444	1,825 reviews	4.5 of 5 bubbles	Medium range

Convert cuisines python list to a list with commas, put it inside a table

In [19]:
```
cuisines_un=[]
for i in range(len(cuisines)):
    cuisines_un.append(",".join(cuisines[i]))

df['cuisines'] = cuisines_un
```

Extract ratings of individual pieces of ratings

Some restaurants do not have "atmosphere", so info is put into a table via "try...except"

In [20]:
```
df['rating_food'] = np.nan
df['rating_service'] = np.nan
df['rating_atmosphere'] = np.nan
df['rating_value'] = np.nan

for i in range(30):
    try:
        df.loc[i,'rating_food'] = ratings_all[i]['Food']
    except KeyError:
            continue
```

```
for i in range(30):
    try:
        df.loc[i,'rating_service'] = ratings_all[i]['Service']
    except KeyError:
         continue
for i in range(30):
    try:
        df.loc[i,'rating_atmosphere'] = ratings_all[i]['Atmosphere']
    except KeyError:
         continue
for i in range(30):
    try:
        df.loc[i,'rating_value'] = ratings_all[i]['Value']
    except KeyError:
         continue
```

Remove "bubbles" from ratings

In [21]:
```
for i in range(30):
    df.loc[i,'rating_food'] = df.loc[i,'rating_food'].replace('bubbles',
 '')
    df.loc[i,'rating_service'] = df.loc[i,'rating_food'].replace('bubble
s', '')
    df.loc[i,'rating_atmosphere'] = df.loc[i,'rating_food'].replace('bub
bles', '')
    df.loc[i,'rating_value'] = df.loc[i,'rating_food'].replace('bubbles'
, '')
    df.loc[i,'overall_rating'] = df.loc[i,'overall_rating'].replace('bub
bles', '')
```

Add details to a table as empty columns

In [22]:
```
df['Average prices'] = np.nan
df['Cuisine'] = np.nan
df['Meals'] = np.nan
df['Restaurant features'] = np.nan
df['Dining Style'] = np.nan
df['Good for'] = np.nan
df['Open Hours'] = np.nan
df['Location and Contact Information'] = np.nan
df['Description'] = np.nan
```

Most of the restaurants do not have all features that are stored in "details", so "try...except" is needed to store info about restaurants

In [23]:
```
for i in range(30):
    try:
        df.loc[i,'Average prices'] = details_list[i]['Average prices']
    except KeyError:
         continue
for i in range(30):
    try:
        df.loc[i,'Cuisine'] = details_list[i]['Cuisine']
    except KeyError:
         continue
for i in range(30):
    try:
```

```
            df.loc[i,'Meals'] = details_list[i]['Meals']
        except KeyError:
             continue
for i in range(30):
    try:
        df.loc[i,'Restaurant features'] = details_list[i]['Restaurant fe
atures']
    except KeyError:
         continue

for i in range(30):
    try:
        df.loc[i,'Dining Style'] = details_list[i]['Dining Style']
    except KeyError:
         continue
for i in range(30):
    try:
        df.loc[i,'Good for'] = details_list[i]['Good for']
    except KeyError:
         continue
for i in range(30):
    try:
        df.loc[i,'Open Hours'] = details_list[i]['Open Hours']
    except KeyError:
         continue
for i in range(30):
    try:
        df.loc[i,'Location and Contact Information'] = details_list[i]['
Location and Contact Information']
    except KeyError:
         continue

for i in range(30):
    try:
        df.loc[i,'Description'] = details_list[i]['Description']
    except KeyError:
         continue
```

Store reviews that are separated by ";"

In [24]:
```
new_full_reviews = []
for res in reviews_full:
    new_full_reviews.append(";".join(res))

df['reviews'] = new_full_reviews
```

Final Table

First 5 columns

In [31]:
```
df.iloc[:,0:5] .head()
```

Out[31]:

	restaurant_name	adress	country	phone_number	review
0	STK Toronto	153 Yorkville Ave	Canada	+1 416-613-9660	350 reviews

1	ALO RESTAURANT	163 Spadina Ave	Canada	+1 416-260-2222	453 reviews
2	Scaramouche Restaurant	1 Benvenuto Pl	Canada	+1 416-961-8011	1,388 reviews
3	New Orleans Seafood & Steakhouse	267 Scarlett Rd	Canada	+1 416-766-7001	209 reviews
4	Richmond Station	1 Richmond St. West	Canada	+1 647-748-1444	1,825 reviews

Next 5 columns

In [26]:
```
df.iloc[:,5:10].head()
```

Out[26]:

	overall_rating	price	cuisines	rating_food	rating_service
0	4.5 of 5	Medium range	Steakhouse,Vegetarian Friendly,Gluten Free Opt...	4.5 of 5	4.5 of 5
1	4.5 of 5	Expensive	French,European,Vegetarian Friendly,Vegan Opti...	5.0 of 5	5.0 of 5
2	4.5 of 5	Expensive	French,International,Vegetarian Friendly,Vegan...	4.5 of 5	4.5 of 5
3	4.5 of 5	Medium range	Steakhouse,Cajun & Creole,Seafood,Gluten Free ...	4.5 of 5	4.5 of 5
4	4.5 of 5	Medium range	American,International,Vegetarian Friendly,Glu...	4.5 of 5	4.5 of 5

Next 5 columns

In [28]:
```
df.iloc[:,10:15].head()
```

Out[28]:

	rating_atmosphere	rating_value	Average prices	Cuisine	Meals
0	4.5 of 5	4.5 of 5	UAH 620 - \nUAH 3,098	Steakhouse, Contemporary, Vegetarian Friendly,...	Dinner, Drinks
1	5.0 of 5	5.0 of 5	UAH 1,832 - \nUAH 2,667	French, European, Vegetarian Friendly, Vegan O...	Dinner, Drinks
2	4.5 of 5	4.5 of 5	UAH 781 - \nUAH 1,158	French, International, Vegetarian Friendly, Ve...	Dinner
3	4.5 of 5	4.5 of 5	UAH 350 - \nUAH 781	Steakhouse, Cajun & Creole, Seafood, Gluten Fr...	Dinner
				American, International,	Lunch,

4	4.5 of 5	4.5 of 5	NaN	Canadian, Vegetarian ...	Dinner, Brunch

Next 5 columns

In [29]:
```
df.iloc[:,15:20].head()
```

Out[29]:

	Restaurant features	Dining Style	Good for	Open Hours	Location and Contact Information
0	Reservations, Private Dining, Seating, Waitsta...	NaN	Romantic, Large groups, Bar scene, Special occ...	Sunday\n5:00 PM - 12:00 AM\n\n\nMonday\n3:30 P...	Address:\n 153 Yorkville Ave, Toronto, Ontario...
1	Reservations, Seating, Waitstaff, Serves Alcoh...	NaN	Special occasions, Local cuisine, Bar scene, R...	Tuesday\n5:00 PM - 1:00 AM\n\n\nWednesday\n5:0...	Address:\n 163 Spadina Ave | 3rd Floor, Toront...
2	Seating, Waitstaff, Wheelchair Accessible, Ser...	Fine Dining	Scenic view, Business meetings, Large groups, ...	Monday\n5:30 PM - 9:30 PM\n\n\nTuesday\n5:30 P...	Address:\n 1 Benvenuto Pl, Toronto, Ontario M4...
3	Takeout, Reservations, Seating, Waitstaff, Par...	NaN	Business meetings, Special occasions, Families...	Tuesday\n5:00 PM - 10:00 PM\n\n\nWednesday\n5:...	Address:\n 267 Scarlett Rd | York, Toronto, On...
4	Waitstaff, Highchairs Available, Serves Alcoho...	NaN	Large groups, Romantic, Local cuisine, Special...	Monday\n11:00 AM - 10:30 PM\n\n\nTuesday\n11:0...	Address:\n 1 Richmond St. West, Toronto, Ontar...

Last 2 columns

In [30]:
```
df.iloc[:,20:].head()
```

Out[30]:

	Description	reviews
0	STK is a unique concept that artfully blends t...	Food ambiance and service is amazing. Matthew ...
1	Hospitality [hos-pi-tal-i-tee] Origin: French;...	Walk-ins with no reservations are no problem f...

2	Scaramouche has long been celebrated by custom...	I had carbonara and it was fantastic! My husba...
3	NaN	I had the steak and it was cooked to perfectio...
4	Richmond Station is a stopping place, a bustli...	My wife and I had a great meal, watching the c...

236 - 239

6.3. Analyse film industry with MongoDB and Python

Analyze film industry with MongoDB and Python

Small code that explains how to merge 3 datasets into one with MongoDb Compass and save it as a separate table in a database. Datasets are taken from Unesco, World Bank and Numbeo. Files are available at the repository.

The idea is to analyze the film industry a little bit, compare the state of cinema industry across countries all over the world and analyze the correlation between variables. Hypotheses are:

- Hypothesis 1
- H1: there is a relationship between population grows and ticket sold
- H0: there is no relationship between population grows and ticket sold.

- Hypothesis 2
- H1: there is a positive relationship between GDP and number of cinemas per country
- H0: there is no relationship between GDP and number of cinemas per country

- Hypothesis 3
- H1: People who live in developed country buy more tickets in cinema
- H0: People who live in developed country do not buy more tickets in cinema

MongoDB

Start a MongoDB shell

Reference on StackOverflow (https://stackoverflow.com/questions/42739166/could-not-connect-to-mongodb-on-the-provided-host-and-port?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa)

1. Create folder "C:\data\db"
2. access "mongod" in bin folder of MongoDB to set up the MongoDB on a machine and establish connection
3. access "mongo" in bin folder of MongoDB to start shell

Create a database

`use project`

> *switched to db project*

Add collections (tables) to a database

```
db.createCollection("unesco")
db.createCollection("numbeo")
db.createCollection("worldbank")
```

> *{ "ok" : 1 }*

`show collections`

> *unesco*
> *numbeo*
> *worldbank*

Fill collections with data. Data is stored in csv (through command line)

```
mongoimport --db project --collection unesco --type csv --headerline --file “D:/mongodb/DatasetsFilm/unesco.csv”
mongoimport --db project --collection numbeo --type csv --headerline --file “D:/mongodb/DatasetsFilm/tickets.csv”
mongoimport --db project --collection worldbank --type csv --headerline --file “D:/mongodb/DatasetsFilm/worldbank.csv”
```

First merge (unesco with worldbank)

Mutual field is "country"

db.worldbank.aggregate%28%5B%7B%24lookup%3A%20%7Bfrom%3A%20%22unesco%22%2ClocalField%3A%20%22country%22%2C%0A%20%20%20%20foreignF

Second merge (first merge with worldbank)

Mutual field is "country"

db.merged_one.aggregate%28%5B%7B%24lookup%3A%20%7Bfrom%3A%20%22numbeo%22%2C%0A%20%20%20%20localField%3A%20%22country%22%2Cforei

Export collection as json

```
mongoexport --db project --collection merged_two
    --out “C:/Users/alexa/Desktop/GC/GC2/Data collection and curation/mongodb/merged_t.json”
```

Python code and data analysis

Create dataframe by extracing data from json

```
import pandas as pd
import json
countries=[]
admissions=[]
ticket_prices=[]
gdp_s=[]
gdp_per_capita=[]
population=[]
cinemas=[]
for line in open("C:/Users/alexa/Desktop/GC/GC2/Data collection and curation/mongodb/merged_t.json", 'r'):
    #lines.append(line)
    js = json.loads(line)
    countries.append(js['country'])
    gdp_s.append(js['gdp'])
    gdp_per_capita.append(js['gdp_capita'])
    population.append(js['population'])
    admissions.append(js['cinemas'][0]['admiss'])
    cinemas.append(js['cinemas'][0]['cinemas'])
    ticket_prices.append(js['ticket_price'][0]['ticket_price'])

dict={}

dict["countries"] = countries
dict["admissions"] = admissions
dict["ticket_prices"] = ticket_prices
dict["gdp_s"] = gdp_s
dict["gdp_per_capita"] = gdp_per_capita
dict["population"] = population
dict["cinemas"] = cinemas

df = pd.DataFrame(data=dict)
```

Question 1

Hypothesis: There is a relationship between population grows and ticket sold

Create new column (number of tickets sold) required for hypothesis testing

```
df['tickets_sold '] = df.apply(lambda row: row['admissions'] / row['ticket_prices'], axis=1)
```

Remove outliers - 68-95-99 rule is applied on "admission" column

Countries to remove - Brazil, China, Mexico, Republic of Korea, Russia, US

```
df = df[(df['countries'] != 'Brazil') & (df['countries'] != 'China') &
        (df['countries'] != 'Mexico') & (df['countries'] != 'Republic of Korea') &
        (df['countries'] != 'Russian Federation') & (df['countries'] != 'United States of America')]
```

Calculate correlation and p-value

%0A%3E0.73

from scipy.stats import ttest_ind ttest_ind(df['tickets_sold'], df['population'])

```
>Ttest_indResult(statistic=-5.396667468504574, pvalue=3.0475777179812855e-07)

In summary, p-value is < 0.05, cinsequently there is a relationship between number of tickets sold and population;

Conclusion:
Film companies, when entering a market of a company, do not need to take the consideration the population because the number of tickets sold
 per person is the almost the same across countries of the world.


## Question 2

###### Hypothesis: There is a positive relationship between GDP and number of cinemas per country

```df['cinemas'].corr(df['gdp_s'])
```

0.76

```
ttest_ind(df['cinemas'], df['gdp_s'])
```
```

Ttest_indResult(statistic=-4.313915342388997, pvalue=3.117987914035751e-05)

In summary, p-value is < 0.05, cinsequently there is a relationship between number of cinemas and GDP;

Question 3

Hypothesis: People who live in developed country buy more tickets in cinema

Create a column with ration "tickets sold" to population

```
df['tickets_population_ratio'] = df.apply(lambda row: row['tickets_sold'] / row['population'], axis=1)
```

Top-10 countries with the highest tickets sold to population ration:

```
df.sort_values('tickets_population_ratio', ascending=False)['countries'].head(10)
```

43 Malaysia
30 Cuba
61 Singapore
3 Belarus
48 New Zealand
29 Iceland
12 Colombia
13 Costa Rica
6 Australia
21 Estonia

In summary, top 10 countries are not those that are considered as first-world countries by Nationsonline http://www.nationsonline.org/oneworld/first_world.htm (http://www.nationsonline.org/oneworld/first_world.htm)

Summary

Hypotheses 1 and 2 were not rejected, while the 3rd one was rejected