Data Science Task

By: Shehabeldin Elzoheiry

Outline

- **Introduction**
- **I. Data exploration and wrangling**
  - Importing data
  - Inspecting missing values
  - Choosing features
- **II. Model Determination**
  - K-nearest neighbors
  - Logistic regression
  - Decision tree
  - Support-vector machine
- **III. Model Presentation**
  - Confusion matrix
  - Identifying top customers

Introduction

- **Data has nine features and one target (label)**

- **The target is a categorical value**
  - A customer either responded or didn’t respond

- **Therefore, we search for a classification model**

I. Data exploration and wrangling

- **Observations after importing data:**
  1. The column 'sports' include missing data for 1500 customers
     - All sports are almost equally distributed
     - Ratio of response to no response is also similar

```
]: print(df.sports.value_counts())

athletics    2853
badminton    2828
soccer       2819
Name: sports, dtype: int64
```

```
]: dummies_label = pd.get_dummies(df.label)
test_quick = pd.concat([df.sports, df.label, dummies_label ], axis = 1)
tested = test_quick.pivot_table(index='sports', columns='label',aggfunc='sum')
tested
```

]:

	no response		response	
label	**no response**	**response**	**no response**	**response**
sports				
athletics	1906.0	0.0	0.0	947.0
badminton	1891.0	0.0	0.0	937.0
soccer	1897.0	0.0	0.0	922.0

I. Data exploration and wrangling

- **Observations after importing data:**

  1. The column 'sports' include missing data for 1500 customers
     - All sports are almost equally distributed
     - Ratio of response to no response is also similar

     ➔ No predominant sport that can be used to fill the empty observations

     ➔ Therefore, they will be left as 0 (after get_dummies)

I. Data exploration and wrangling

- **Observations after importing data:**
  1. The column 'sports' include missing data for 1500 customers
  2. The column 'name' has 10,000 different names
     - No repeated names

     ➔ Names will be dropped

```
df.describe(include = 'all')
```

	name	age	lifestyle	zip code
count	10000	10000.000000	10000	10000.000000
unique	10000	NaN	3	NaN
top	VnSEFOuL	NaN	active	NaN
freq	1	NaN	3375	NaN
mean	NaN	42.090700	NaN	55227.270600
std	NaN	15.874416	NaN	26139.756227
min	NaN	15.000000	NaN	10003.000000
25%	NaN	28.000000	NaN	32708.250000
50%	NaN	42.000000	NaN	55290.000000
75%	NaN	56.000000	NaN	77967.750000
max	NaN	69.000000	NaN	99982.000000

I. Data exploration and wrangling

- **Observations after importing data:**
  1. The column 'sports' include missing data for 1500 customers
  2. The column 'name' has 10,000 different names
  3. The column 'zip code' has 9451 different codes
     - The most frequent code is repeated only 3 times!

```
zip_["zip code"] = df["zip code"]
zip_.describe(include='all')
```

```
count     10001.0
unique     9451.0
top       68953.0
freq          3.0
Name: zip code, dtype: float64
```

➔ Zip codes will also be dropped

I. Data exploration and wrangling

- **Observations after importing data:**

  1. The column 'sports' include missing data for 1500 customers
  2. The column 'name' has 10,000 different names
  3. The column 'zip code' has 9451 different codes
  4. The rest of other columns (except age and earnings) are categorical

  ➔ get_dummies will be used

I. Data exploration and wrangling

- **Summary**
  - ➔ name / zip code are dropped
  - ➔ get_dummies will be used
  - ➔ label will be transferred to binary

```
test_df = []
test_df = df[['age','earnings']]
life_sty = pd.get_dummies(df.lifestyle)
fam_stat = pd.get_dummies(df['family status'])
car_     = pd.get_dummies(df.car)
sports_  = pd.get_dummies(df.sports)
liv_area = pd.get_dummies(df['living area'])

test_df = pd.concat([test_df, life_sty, fam_stat, car_, sports_, liv_area], axis = 1)
test_df['label'] = df['label'].apply(lambda x: 1 if (x == 'response')  else 0)
test_df.head()
```

	age	earnings	active	cozily	healthy	married	single	expensive	practical	athletics	badminton	soccer	rural	urban	lab
0	62.0	102526.0	0	1	0	1	0	0	1	1	0	0	0	1	
1	34.0	33006.0	1	0	0	1	0	1	0	0	0	1	0	1	
2	69.0	118760.0	0	0	1	0	1	1	0	0	1	0	0	1	
3	57.0	131429.0	0	1	0	1	0	0	1	0	0	1	0	1	
4	66.0	96003.0	0	1	0	0	1	0	1	0	1	0	0	1	

II. Model Determination

- For simple comparison between different models, metrics.accuracy_score() will be used

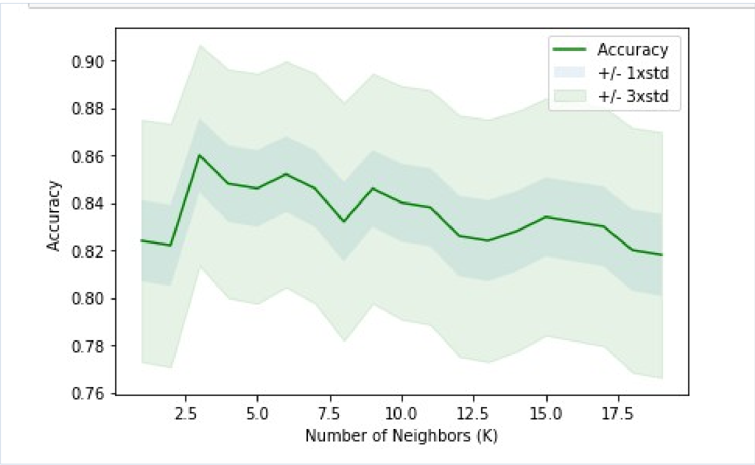
- Parameters of each model will be adjusted to output most accurate performance before comparison

II. Model Determination

1. K-nearest neighbors

- Different K values were tested
- Different test_size ratios were tested

➔ K of 3 and test_size of 0.05 showed the highest accuracy

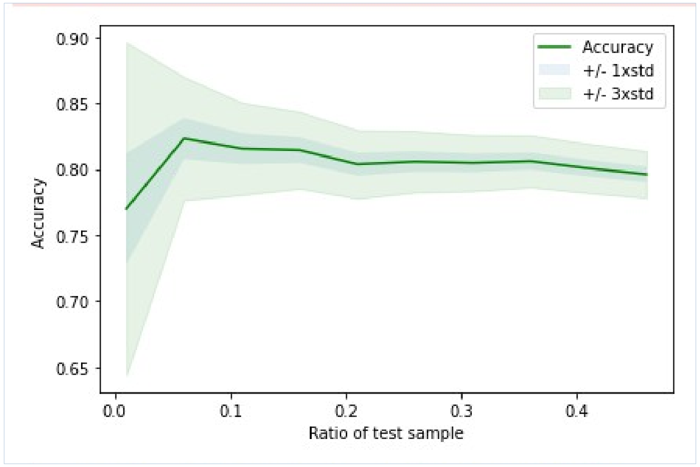


II. Model Determination

2. Logistic regression

- Different C and solver values were tested
- Different test_size ratios were tested

➔ C of 0.01, ‘lbfgs’ as solver and test_size of 0.05 showed the highest accuracy

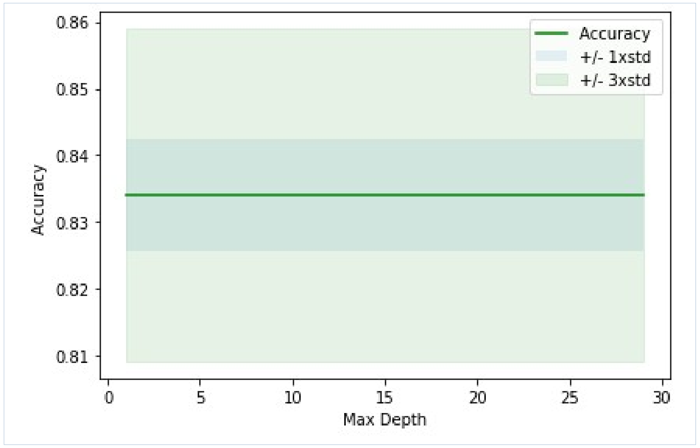


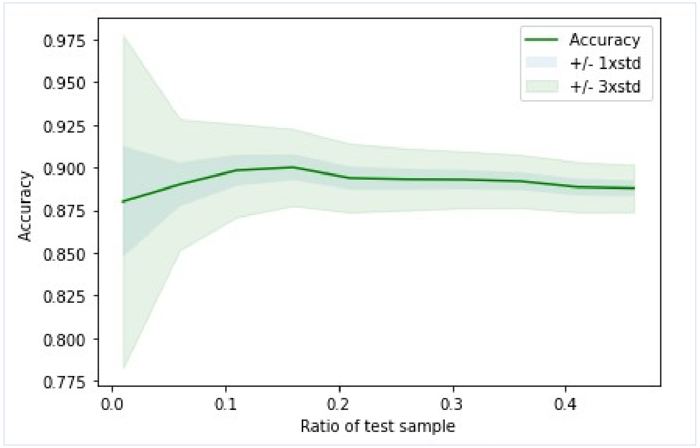
II. Model Determination

3. Decision tree

- Different max_depth values &
- Different test_size ratios

  were tested





➔ Depth didn’t affect accuracy as long as random_state was set

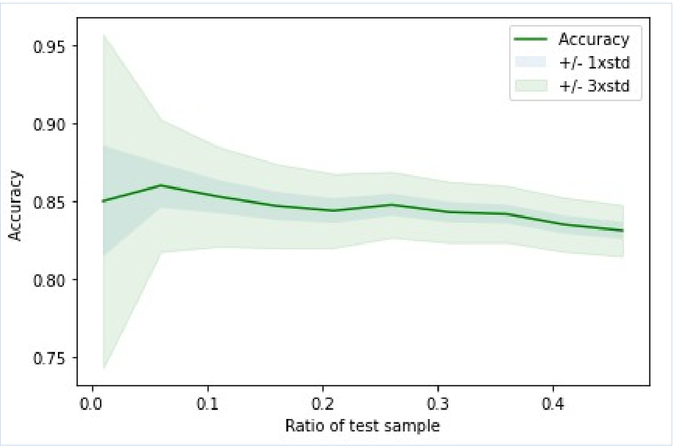
➔ test_size of 1.5 showed the highest accuracy

II. Model Determination

4. Support-vector machine

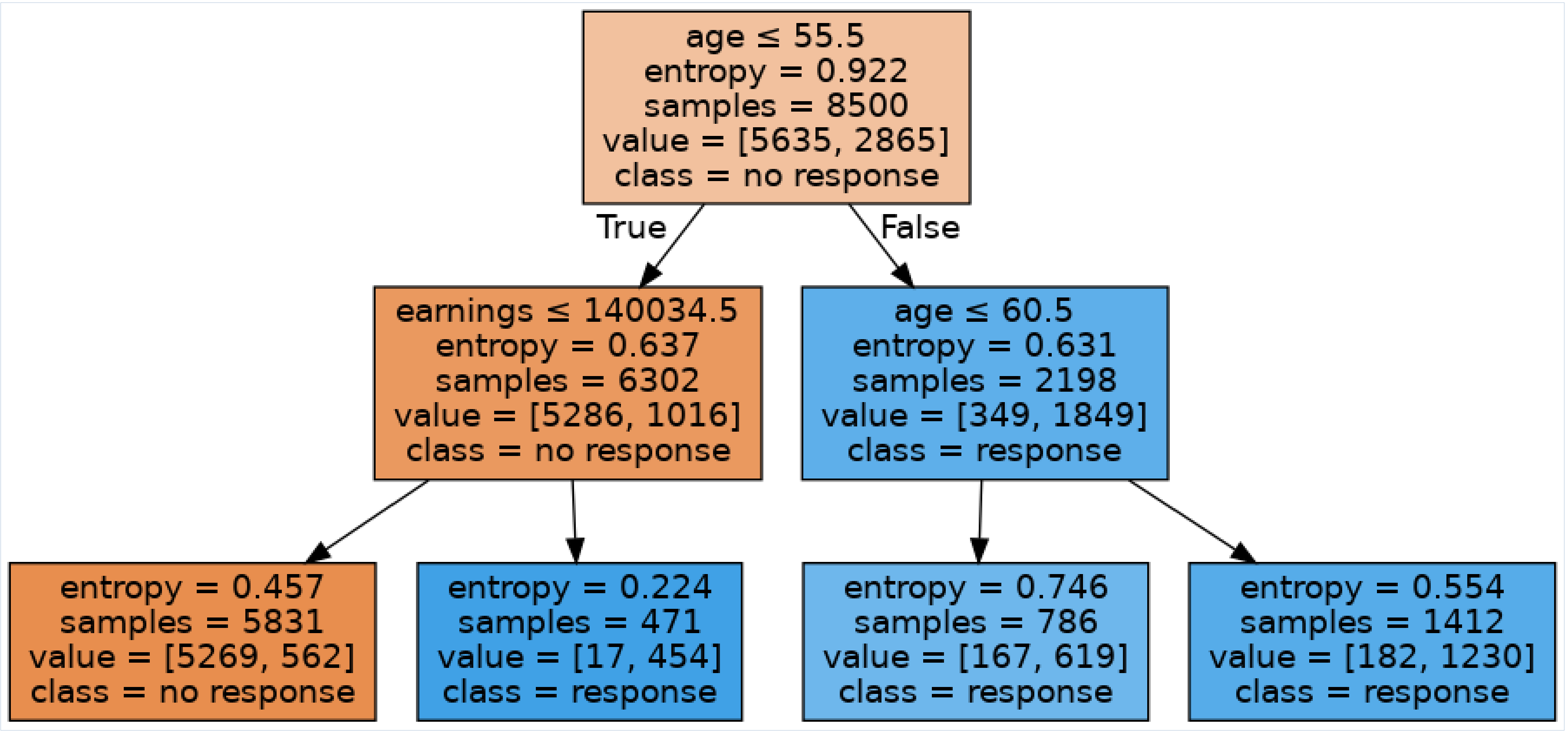
- Different Kernels were tested
- Different test_size ratios were tested

➔ Kernel as ‘rbf’ and test_size of 0.05 showed the highest accuracy



III. Model Presentation

- **Tree Plot**



The code for plotting didn't run in Jupyter, but ran on PyCharm, which will be attached separately, called 'Capgemini_plot_Tree.py'

II. Model Determination

- **Summary**

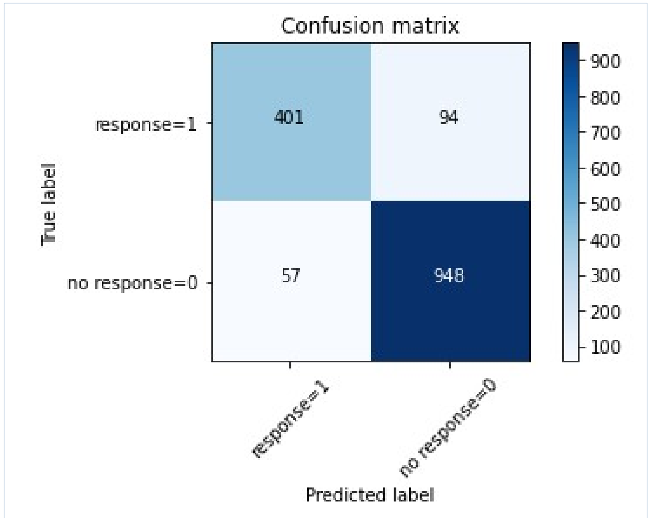
Scores	
– K-nearest neighbors	0.86
– Logistic regression	0.82
– **Decision tree**	**0.9**
– Support-vector machine	0.86

III. Model Presentation

- **Decision tree model is trained based on optimal parameters (determined in section II)**

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.15, random_state=4)
#decision tree object (model)
decTree   = DecisionTreeClassifier(criterion="entropy", max_depth = 2,random_state=4).fit(X_train,y_train)
predTree  = decTree.predict(X_test)
```

- **Confusion matrix is plotted**
  - True/false positives & negatives



III. Model Presentation

- **Determining best customer features**
  - Decision tree (previous slide) gave impression that age plays a crucial role.
  - To further determine the ideal features, logistic regression model will be used since it returns probabilities
    - Customer with highest probability to respond

```python
# trial_df is equivalent to original dataframe but with probabilities of responding
trial_df=df[['name', 'age', 'lifestyle', 'zip code', 'family status', 'car','sports', 'earnings', 'living area']]
trial_df['proba']= LR.predict_proba(X)[:,1]
ideal = trial_df[trial_df["proba"] == max(trial_df["proba"])]
ideal.head()
```

	name	age	lifestyle	zip code	family status	car	sports	earnings	living area	proba
5930	NLRP5nIR	69.0	cozily	33619.0	married	practical	NaN	149239.0	urban	0.872824

III. Model Presentation

- **Extra: overview of top 10 customers likely to respond, based on probability to respond.**
  - Further points towards age

```
trial_df.sort_values(by=['proba'], ascending=False).head(10)
```

	name	age	lifestyle	zip code	family status	car	sports	earnings	living area	proba
5930	NLRP5nIR	69.0	cozily	33619.0	married	practical	NaN	149239.0	urban	0.872824
8870	8IZX4IbI	69.0	active	33276.0	single	practical	NaN	145998.0	rural	0.869986
4594	34UDY7tV	69.0	healthy	41310.0	single	expensive	soccer	149630.0	urban	0.868554
5710	IE4Gudzt	69.0	active	44027.0	married	practical	NaN	149522.0	urban	0.865897
4977	cn3UdDWz	69.0	healthy	60469.0	married	expensive	NaN	139824.0	urban	0.864299
5591	Hw5CYSS4	69.0	cozily	28133.0	single	expensive	athletics	146890.0	urban	0.864078
9999	eBED7EpQ	69.0	healthy	90816.0	single	practical	athletics	149683.0	rural	0.864054
1437	kC6415B9	69.0	cozily	54912.0	single	practical	athletics	149575.0	urban	0.862858
4722	rpBV49cc	69.0	healthy	25576.0	married	practical	NaN	139759.0	rural	0.861594
5728	RbTIN9wy	69.0	active	63838.0	single	expensive	NaN	135039.0	rural	0.860355

Thank you for your time !