Projektarbeitsbericht

Teilnehmer

Muhammed Aydin
Mansour Mazaheri
Jamal Abdel Naser
(“Alle haben in jedem Bereich teilgenommen.”)

Ziel der Untersuchung

Wir wollen herausfinden, ob jemand anhand der sozialen, politischen, wirtschaftlichen, religiösen und kulturellen Faktoren in seinem/ihrem Leben zufrieden ist.
(Unsere Arbeitsschwerpunkte waren verschiedene Daten-aufbereitungstechnologien, ML-Algorithmen, Webentwicklung und Server-Technologien.)

Arbeitshypothese

Die Bevölkerung der Welt ist heutzutage unglaublich unzufrieden mit ihrem täglichen Leben. Die Unzufriedenheit hängt von verschiedenen Faktoren ab. Wir wollen untersuchen, ob jemand wirklich ein glückliches oder sorgenvolles Leben führt. Wir stellen ein paar Fragen an unsere Webbesucher. Daraus berechnen wir, ob er/sie überhaupt glücklich ist und wenn, dann wie zufrieden er/sie ist. Wir haben einen Datensatz von ‘ World Values Survey’ genutzt, (https://www.worldvaluessurvey.org/WVSContents.jsp) ,in dem Datensätze ordinal skaliert sind.

Technologien

1. Python (Pandas, Numpy, Matplotlib, Seaborn)
2. ML- Algorithmen ('BernoulliNB', 'GaussianNB', 'KNeighborsClassifier','LogisticRegression', 'SGDClassifier', 'SVC', ’DecisionTreeClassifier', 'RandomForestClassifier', 'AdaBoostClassifier', 'MLPClassifier')
3. Server Technologie (Ngrok)
4. Web Technologie (Streamlit)

Qualitätskriterien :

1. Wir vermuten, dass >=75% score gebende Algorithmen gute Algorithmen sind.
2. Unsere Classifier Algorithmen bezeichnen jemanden als glücklichen Menschen ,der zu 50% glücklich ist.

Die Daten herunterladen und importieren

Der World Values Survey (WVS) ist ein internationales Forschungsprogramm, das sich der wissenschaftlichen und akademischen Untersuchung der sozialen, politischen, wirtschaftlichen, religiösen und kulturellen Werte der Menschen in der Welt widmet.

Ziel des Projekts ist es zu untersuchen, welche Auswirkungen die Stabilität oder Veränderung der Werte auf die soziale, politische und wirtschaftliche Entwicklung von Ländern haben.

Wir haben einen Datensatz von ‘ World Values Survey’ genutzt.

Wir laden zuerst die Daten mit pandas.

```
import pandas as pd
```

```
df = pd.read_csv("C:/Users/Alfa/Downloads/F00011356-WVS_Cross-National_Wave_7_csv_v1_6_2/WVS_Cross-National_Wave_7_csv_v1_6_2.
```

Diese Funktion liefert ein pandas-DataFrame-Objekt mit sämtlichen Daten.

Einen kurzen Blick auf die Datenstruktur werfen

Schauen wir uns die ersten 5 Zeilen des Data-Frames mit der Methode head() an.

```
df.head()
```

	version	doi	A_WAVE	A_STUDY	B_COUNTRY	B_COUNTRY_ALPHA	C_COW_NUM	C_COW_ALPHA	A_YEAR	D_INTERVIEW	...	WVS_Polmistr
0	1-6-2 (2021-04-25)	doi.org/10.14281/18241.1	7	2	20	AND	232	AND	2018	20070001	...	
1	1-6-2 (2021-04-25)	doi.org/10.14281/18241.1	7	2	20	AND	232	AND	2018	20070002	...	
2	1-6-2 (2021-04-25)	doi.org/10.14281/18241.1	7	2	20	AND	232	AND	2018	20070003	...	
3	1-6-2 (2021-04-25)	doi.org/10.14281/18241.1	7	2	20	AND	232	AND	2018	20070004	...	
4	1-6-2 (2021-04-25)	doi.org/10.14281/18241.1	7	2	20	AND	232	AND	2018	20070005	...	

5 rows × 548 columns

Die Methode info() hilft, schnell eine Beschreibung der Daten zu erhalten. Dies sind insbesondere die Anzahl der Zeilen, der Typ jedes Attributs und die Anzahl der Werte ungleich null.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70867 entries, 0 to 70866
Columns: 548 entries, version to v2xps_party
dtypes: float64(523), int64(16), object(9)
memory usage: 296.3+ MB
```

Wie wir sehen, haben wir ursprünglich 70867 Zeilen Einträge, 548 Spalten Einträge, unterschiedliche Datentypen und wir haben 296+ Speicherplatz genutzt.

Betrachten wir auch die anderen Spalten. Die Methode describe() fasst die numerischen Merkmale zusammen.

```
df.describe()
```

	A_WAVE	A_STUDY	B_COUNTRY	C_COW_NUM	A_YEAR	D_INTERVIEW	J_INTDATE	FW_END	FW_START	K_TIME_START	...	WVS_F
count	70867.0	70867.0	70867.000000	70867.000000	70867.000000	7.086700e+04	6.798700e+04	70867.000000	70867.000000	59231.000000	...	
mean	7.0	2.0	427.417134	514.574978	2018.356922	4.277022e+08	2.018323e+07	201839.059365	201823.025865	14.031349	...	
std	0.0	0.0	248.622770	291.107448	0.955816	2.486329e+08	8.813118e+03	90.562074	89.629041	3.754853	...	
min	7.0	2.0	20.000000	2.000000	2017.000000	2.007000e+07	2.017010e+07	201703.000000	201701.000000	0.000000	...	
25%	7.0	2.0	196.000000	160.000000	2018.000000	1.967009e+08	2.018020e+07	201805.000000	201801.000000	11.300000	...	
50%	7.0	2.0	400.000000	640.000000	2018.000000	4.000706e+08	2.018072e+07	201808.000000	201806.000000	14.060000	...	
75%	7.0	2.0	642.000000	740.000000	2019.000000	6.420702e+08	2.019071e+07	201906.000000	201905.000000	16.590000	...	
max	7.0	2.0	840.000000	993.000000	2020.000000	8.400736e+08	2.020081e+07	202008.000000	202007.000000	23.580000	...	

8 rows × 539 columns

Da die wichtigsten Spalten kategorisch skaliert sind, betrachten wir den statistischen Durchschnitt (max, mean, min, mode, ...) zu errechnen, für unnötig.

Aufbereiten der Daten

Wir haben die Spalten je nach Ihrem Bereich sortiert und mathematisch untereinander hinzugefügt und ganz neue Spalten gestellt. Dadurch konnten wir unseren Datensatz von ungefähr 300 Spalten zu 30 Spalten reduzieren.

```
economic_values = pd.DataFrame(df['Q106'].rename('economic_values'))

corruption_preceptions = pd.DataFrame(df['Q112'].rename('corruption_preceptions'))

corruption_of_institutions = (df['Q113']+df['Q114']+df['Q115']+df['Q116']+df['Q117'])/5
corruption_of_institutions = pd.DataFrame(corruption_of_institutions, columns=['corruption_of_institutions'])

effect_of_immigration = (df['Q122']+df['Q123']+df['Q124']+df['Q125']+df['Q126']+df['Q127']+df['Q128']+df['Q129'])/8
effect_of_immigration = pd.DataFrame(effect_of_immigration, columns=['effect_of_immigration'])

secure_in_neighborhood = (df['Q132']+df['Q133']+df['Q134']+df['Q135']+df['Q136']+df['Q137']+df['Q138'])/7
secure_in_neighborhood = pd.DataFrame(secure_in_neighborhood, columns=['secure_in_neighborhood'])

worried_about_risks_and_war = (df['Q142']+df['Q143']+df['Q146']+df['Q147']+df['Q148'])/5
worried_about_risks_and_war = pd.DataFrame(worried_about_risks_and_war, columns=['worried_about_risks_and_war'])


science_and_technologie = (df['Q158']+df['Q159']+df['Q160']+df['Q161']+df['Q162']+df['Q163']) / 6
science_and_technologie = pd.DataFrame(science_and_technologie, columns=['science_and_technologie'])

importance_of_god = pd.DataFrame(df['Q164'].rename('importance_of_god'))

religion_and_science = (df['Q169']+df['Q170'])/2
religion_and_science = pd.DataFrame(religion_and_science, columns=['religion_and_science'])

moral_rules = pd.DataFrame(df['Q176'].rename('moral_rules'))

Ethical_justification = (df['Q179']+df['Q180']+df['Q181']+df['Q182']+df['Q183']+df['Q184']+df['Q185']+df['Q186']+df['Q187']+df['Q188']
Ethical_justification = pd.DataFrame(Ethical_justification, columns=['Ethical_justification'])

Interest_for_politics = pd.DataFrame(df['Q199'].rename('Interest_for_politics'))

Political_action = (df['Q209']+df['Q210']+df['Q211']+df['Q212']+df['Q213']+df['Q214']+df['Q215']+df['Q216']+
df['Q217']+df['Q218']+df['Q219']+df['Q220']) /12
Political_action = pd.DataFrame(Political_action, columns=['Political_action'])

Electoral_integrity_battery = pd.DataFrame(df['Q232'].rename('Electoral_integrity_battery'))

Importance_of_democracy = pd.DataFrame(df['Q250'].rename('Importance_of_democracy'))

Perceived_democraticness = pd.DataFrame(df['Q251'].rename('Perceived_democraticness'))

Satisfaction_with_the_political_system = pd.DataFrame(df['Q252'].rename('Satisfaction_with_system'))

Respect_for_human_rights = pd.DataFrame(df['Q253'].rename('Respect_human_rights'))

Identity_levels = (df['Q255']+df['Q256']+df['Q257']+df['Q258']+df['Q259']) /5
Identity_levels = pd.DataFrame(Identity_levels, columns=['Identity_levels'])

Social_class = pd.DataFrame(df['Q258'].rename('Social_class'))



df_m = pd.concat([economic_values,corruption_preceptions, corruption_of_institutions, effect_of_immigration,secure_in_neighborhood,\
           worried_about_risks_and_war,science_and_technologie,importance_of_god,religion_and_science,moral_rules,\
          Ethical_justification,Interest_for_politics,Political_action,Electoral_integrity_battery,Importance_of_democracy,\
          Perceived_democraticness,Satisfaction_with_the_political_system, Respect_for_human_rights,Identity_levels,Social_class, df_
```

```
df_m.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70867 entries, 0 to 70866
Data columns (total 30 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   economic_values               70006 non-null  float64
 1   corruption_preceptions        69750 non-null  float64
 2   corruption_of_institutions    61207 non-null  float64
 3   effect_of_immigration         61075 non-null  float64
 4   secure_in_neighborhood        60280 non-null  float64
 5   worried_about_risks_and_war   59374 non-null  float64
 6   science_and_technologie       66460 non-null  float64
 7   importance_of_god             68965 non-null  float64
 8   religion_and_science          64407 non-null  float64
 9   moral_rules                   68599 non-null  float64
 10  Ethical_justification         53861 non-null  float64
 11  Interest_for_politics         70376 non-null  float64
 12  Political_action              52069 non-null  float64
 13  Electoral_integrity_battery   63364 non-null  float64
 14  Importance_of_democracy       69629 non-null  float64
 15  Perceived_democraticness      69002 non-null  float64
 16  Satisfaction_with_system      68179 non-null  float64
 17  Respect_human_rights          69475 non-null  float64
 18  Identity_levels               65728 non-null  float64
 19  Social_class                  68709 non-null  float64
 20  neighbours                    65006 non-null  float64
 21  Interpersonal_trust           66497 non-null  float64
 22  Organizational_membership     61820 non-null  float64
 23  financial                     70508 non-null  float64
 24  main_goal                     69724 non-null  float64
 25  court                         68032 non-null  float64
 26  comparing_parents             69700 non-null  float64
 27  Importance_of_family          70757 non-null  float64
 28  Importance_of_work            70184 non-null  float64
 29  Q46                           70437 non-null  float64
dtypes: float64(30)
memory usage: 16.2 MB
```

Unsere Zielspalte haben wir von ordinal zu kategorisch umgewandelt, in dem sie lediglich booliche Werte nämlich “happy” und “not happy” enthält.

```
def happy(x):
    if x==1 or x ==2:
        return "happy"
    elif x ==3 or x == 4:
        return "not_happy"
    else:
        return np.NaN
```

```
df_m['happy']=df_m.Q46.apply(happy)
df_m.drop('Q46',1, inplace=True)
```

Mit der Methode isnull() haben wir berechnet, wie viele fehlende Werte jede Spalte enthält.

```
t = ((df_m.isnull().sum())/df_m.shape[0]).to_frame()
t = t.rename(columns={0: 'NULL values'})
t
```

	NULL values
economic_values	0.012150
corruption_preceptions	0.015762
corruption_of_institutions	0.136312
effect_of_immigration	0.138174
secure_in_neighborhood	0.149393
worried_about_risks_and_war	0.162177
science_and_technologie	0.062187
importance_of_god	0.026839
religion_and_science	0.091157
moral_rules	0.032004
Ethical_justification	0.239971
Interest_for_politics	0.006928
Political_action	0.265257
Electoral_integrity_battery	0.105874
Importance_of_democracy	0.017469
Perceived_democraticness	0.026317
Satisfaction_with_system	0.037930
Respect_human_rights	0.019642
Identity_levels	0.072516
Social_class	0.030451
neighbours	0.082704
Interpersonal_trust	0.061665
Organizational_membership	0.127662
financial	0.005066
main_goal	0.016129
court	0.040005
comparing_parents	0.016467
Importance_of_family	0.001552
Importance_of_work	0.009638
happy	0.006068

Wir mussten auch mit den fehlenden Werten umgehen. Dass wir Survey Datensatz nutzten, um fehlende Werte zu eliminieren, war nach unserer Ansicht die beste Lösung.
Imputing haben wir in unserem Datensatz deswegen auch nicht genutzt.

```
df_m.dropna().info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21674 entries, 2 to 70866
Data columns (total 30 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   economic_values              21674 non-null  float64
 1   corruption_preceptions       21674 non-null  float64
 2   corruption_of_institutions   21674 non-null  float64
 3   effect_of_immigration        21674 non-null  float64
 4   secure_in_neighborhood       21674 non-null  float64
 5   worried_about_risks_and_war  21674 non-null  float64
 6   science_and_technologie      21674 non-null  float64
 7   importance_of_god            21674 non-null  float64
 8   religion_and_science         21674 non-null  float64
 9   moral_rules                  21674 non-null  float64
 10  Ethical_justification        21674 non-null  float64
 11  Interest_for_politics        21674 non-null  float64
 12  Political_action             21674 non-null  float64
 13  Electoral_integrity_battery  21674 non-null  float64
 14  Importance_of_democracy      21674 non-null  float64
 15  Perceived_democraticness     21674 non-null  float64
 16  Satisfaction_with_system     21674 non-null  float64
 17  Respect_human_rights         21674 non-null  float64
 18  Identity_levels              21674 non-null  float64
 19  Social_class                 21674 non-null  float64
 20  neighbours                   21674 non-null  float64
 21  Interpersonal_trust          21674 non-null  float64
 22  Organizational_membership    21674 non-null  float64
 23  financial                    21674 non-null  float64
 24  main_goal                    21674 non-null  float64
 25  court                        21674 non-null  float64
 26  comparing_parents            21674 non-null  float64
 27  Importance_of_family         21674 non-null  float64
 28  Importance_of_work           21674 non-null  float64
 29  happy                        21674 non-null  object
dtypes: float64(29), object(1)
memory usage: 5.1+ MB
```

Erstellen ein Train- und Testdatensatz

Aus Sklearn, Unterbibliothek model_selection , haben wir train_test_split importiert, damit wir es in Trainings- und Testsätze aufteilen können.
Erstens gibt es den Parameter random_state, der den Seed-Wert des Zufallszahlengenerators festlegt. Und zweitens können wir mehrere Datensätze mit einer identischen Anzahl Zeilen übergeben, die anhand der gleichen Indizes aufgeteilt werden.

```
def data_spliting (df,target, size = 0.2, random_state=42):
    """
    it takes dataframe and return train and test datasets
    """
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split( df.drop(target,1),df[target], test_size=size ,
                                                         random_state=random_state)
    return  X_train, X_test, y_train, y_test
```

Mithilfe obiger Funktion haben wir X_train, X_test, y_train und y_test erstellt.

```
X_train, X_test, y_train,y_test = data_spliting(df_m.dropna(), target='happy')
```

Skalieren von Merkmalen

Skalieren von Merkmalen wird verwendet, um die Werte unabhängiger Variablen zu standardisieren. Wir haben mit hilfe der folgenden Funktion “StandardScaler” zum Skalieren des Datensatzes benutzt:

```
def data_scaling (df, target, method = 'StandardScaler', balance = False, **kwargs):
    """
    it takes train and test dataframe and return scaled train and test dataframes
    method {'StandardScaler': StandardScaler, 'MinMaxScaler':MinMaxScaler}
    """
    from sklearn.preprocessing import StandardScaler, MinMaxScaler
    if method == 'StandardScaler':
        scaler = StandardScaler()
    elif method == 'MinMaxScaler':
        scaler = MinMaxScaler()
    else:
          raise ValueError("please chose a vaild method")


    if balance:
        X_train, X_test, y_train, y_test = data_balancing (df, target)
    else:
        X_train, X_test, y_train, y_test = data_spliting (df, target)

    X_train_s = scaler.fit_transform(X_train)
    X_test_s = scaler.transform(X_test)

    return X_train_s, X_test_s, y_train, y_test
```

Datensatz balancieren

Randomresampling stellt eine naive Technik dar, um die Klassenverteilung für unbalancierte Datensätze wieder zu balancieren. Random oversampling wird für die Verdopplung der Beispiele der Minderheit Klasse in den Trainingdatensätzen genutzt und führt zu overfitting mancher Modelle . Random undersampling löscht Beispiele der Mehrheit Klasse und verursacht Beseitigung der unnötigen Daten.

Zum Balancieren haben wir die folgende Funktion benutzt:

```
def data_balancing (df, target, over_sampeling = True, under_sampling = False):
    """
    it takes train dataframe and return either over sampled dataset or under sampled dataset or both them
    """
    from imblearn.under_sampling import RandomUnderSampler
    from imblearn.over_sampling import RandomOverSampler
    over = RandomOverSampler(random_state=0)
    under = RandomUnderSampler(random_state=0)

    X_train, X_test, y_train, y_test = data_spliting (df, target)

    if over_sampeling!=under_sampling:
        if over_sampeling:
            X_resample_over, y_resample_over = over.fit_resample(X_train, y_train)
            return X_resample_over, X_test, y_resample_over, y_test
        else:
            X_resample_under, y_resample_under = under.fit_resample(X_train, y_train)
            return X_resample_under,X_test, y_resample_under, y_test
    else:
        raise ValueError('Please choose either over_sampeling or under_sampeling')
```

Trainieren und Auswerten auf der Trainings- und Testdatensätze

Danach wurden unterschiedliche ML Algorithmen ('BernoulliNB', 'GaussianNB', 'KNeighborsClassifier', 'LogisticRegression', 'SGDClassifier', 'SVC', 'DecisionTreeClassifier', 'RandomForestClassifier', 'AdaBoostClassifier', 'MLPClassifier') genutzt, um Trainings Datensätze zu trainieren und Testdatensätze zu überprüfen , um schließlich über künftige Datensätze prognostizieren zu können. Mit scrore Funktionen wurde überprüft, wie gut die Modelle über künftige Datensätze vorhersagen können.

Wie wir oben sehen konnten, haben wir lediglich Klassifikation Algorithmen genutzt. Aufgrund der Anwesenheit der Labels haben wir keinen unüberwachten LernAlgorithmus genutzt und da wir in Labels nur kategorische Daten behalten haben , finden wir stets Klassifikationsalgorithmen besser geeignet und nicht Regression - Algorithmen.

Ergebnisse

ohne Skalieren und **ohne** Balancieren

	model	accuracy	f1 score
0	BernoulliNB	0.862976	0.463224
1	GaussianNB	0.814302	0.617365
2	KNeighborsClassifier	0.865283	0.535293
3	LogisticRegression	0.860900	0.542748
4	SGDClassifier	0.848212	0.618097
5	SVC	0.862976	0.463224
6	DecisionTreeClassifier	0.788927	0.572058
7	RandomForestClassifier	0.869666	0.551014
8	AdaBoostClassifier	0.862976	0.564396
9	MLPClassifier	0.854902	0.600225

mit Skalieren und **ohne** Balancieren

	model	accuracy	f1 score
0	BernoulliNB	0.832065	0.610667
1	GaussianNB	0.814302	0.617365
2	KNeighborsClassifier	0.869896	0.551207
3	LogisticRegression	0.861361	0.545414
4	SGDClassifier	0.862745	0.480885
5	SVC	0.866436	0.520551
6	DecisionTreeClassifier	0.788005	0.578019
7	RandomForestClassifier	0.869204	0.549424
8	AdaBoostClassifier	0.862976	0.564396
9	MLPClassifier	0.849135	0.614853

ohne Skalieren und **mit** Balancieren

	model	accuracy	f1 score
0	BernoulliNB	0.372318	0.352197
1	GaussianNB	0.694348	0.579433
2	KNeighborsClassifier	0.814764	0.588652
3	LogisticRegression	0.717647	0.608488
4	SGDClassifier	0.820531	0.643455
5	SVC	0.733795	0.620169
6	DecisionTreeClassifier	0.792849	0.572869
7	RandomForestClassifier	0.868973	0.584471
8	AdaBoostClassifier	0.732411	0.619017
9	MLPClassifier	0.721569	0.593071

mit Skalieren und **mit** Balancieren

	model	accuracy	f1 score
0	BernoulliNB	0.832065	0.610667
1	GaussianNB	0.814302	0.617365
2	KNeighborsClassifier	0.869896	0.551207
3	LogisticRegression	0.861361	0.545414
4	SGDClassifier	0.863899	0.471778
5	SVC	0.866436	0.520551
6	DecisionTreeClassifier	0.786621	0.569101
7	RandomForestClassifier	0.869204	0.549424
8	AdaBoostClassifier	0.862976	0.564396
9	MLPClassifier	0.847290	0.605052

Die Ergebnisse zeigen, dass wir mit skalierte und balancierte Datensatz bessere Ergebnisse als ohne skalierte und balancierte Datensatz liefern können. (für jede Version, obwohl einige Modelle besser funktionieren als andere Versionen, wird möglicherweise keine verallgemeinerte Tabelle bereitgestellt. Aus Gründen der Konsistenz wird es jedoch mit einem skalierten und balancierten Datensatz fortgesetzt)

Optimiere das Modell

Gridsearch

In diesem bereich haben wir gridsearch API benutzt um die beste kombination der hyper Parameter für die Algorithmen nämlich RandomForestClassifier, BernoulliNB, SVM, MLPClassifier zu suchen und damit die zu optimieren.

```
from sklearn.model_selection import GridSearchCV
```

```
X_train, X_test, y_train,y_test = data_scaling(df_m.dropna(), target='happy', balanced = True)
```

```
from sklearn.ensemble import RandomForestClassifier
rf_params = {"n_estimators":[100, 300,600],
             "max_depth":[17,20,22],
             "max_features": [10,12,16],
            }
rf = RandomForestClassifier()
rf_cv_model = GridSearchCV(rf, rf_params, cv = 5, n_jobs = -1, verbose = 2).fit(X_train, y_train)
rf_cv_model.best_params_
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

```
{'max_depth': 20, 'max_features': 12, 'n_estimators': 600}
```

```
y_pred = rf_cv_model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[3684   57]
 [ 511   83]]
              precision    recall  f1-score   support

       happy       0.88      0.98      0.93      3741
   not_happy       0.59      0.14      0.23       594

    accuracy                           0.87      4335
   macro avg       0.74      0.56      0.58      4335
weighted avg       0.84      0.87      0.83      4335
```

Die Ergebnisse von Random Forest Classifier nach Gridsearch sind besser geworden aber hat keine größe Unterschied mit unserem Modell.

Nach der Untersuchung der Algorithmen nämlich RandomForestClassifier, BernoulliNB, SVM, MLPClassifier kommen wir zur Entscheidung dass Gridsearch keine riesiege verbesserung zum Ergebnis bringt. Nichtsdestotrotz haben wir Gridserach API beibehalten, weil erstens eine gute gewohnheit ist, Gridsearch API zu nutzen, Darüber Hinaus hat sie das Ergebnis bisschen verbessert.

Voting classifier

Diese API nimmt unterschiedliche Algorithmen in sich, vergleicht am ende die Ergebnisse der algorithmen und nimmt die mehrheit als eigene Ergebnis. Z.B. wenn drei unterschiedliche Algorithmen die in Voting classifier beinhaltet sind,liefern ‘1’ als ergebnis und zwei ‘0’ als Ergebnis liefern, Die API wird ‘1’ als Ergebnis liefern. Obwohl soft version der API rechnet das ergebnis bisschen anderes, es rechnet sein ergebnis im bezug auf wahrscheinlichkeit.

```
from sklearn.ensemble import VotingClassifier
```

```
rf_model = RandomForestClassifier(max_depth=20, max_features = 12,n_estimators = 600 )
bel_model = BernoulliNB(alpha = 1)
svm = SVC(C=10, gamma = 0.01, kernel ='rbf')
ml = MLPClassifier(activation = 'relu',alpha =0.0001, learning_rate =  'constant')
```

```
eclf1 = VotingClassifier(estimators=[ ('random', rf_model),('bel',bel_model), ('svm', svm),('ml',ml)], voting='h
eclf1 = eclf1.fit(X_train, y_train)
```

```
/Users/aydin/opt/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/_multilayer_perceptron.py:617: C
onvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converge
d yet.
  % self.max_iter, ConvergenceWarning)
```

```
y_pred = eclf1.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[3703   38]
 [ 543   51]]
              precision    recall  f1-score   support

       happy       0.87      0.99      0.93      3741
   not_happy       0.57      0.09      0.15       594

    accuracy                           0.87      4335
   macro avg       0.72      0.54      0.54      4335
weighted avg       0.83      0.87      0.82      4335
```

Wie weist der obiger Datensatz auf, Voting classifier macht auch keine unterschied zur Ergebnis wenn wir es mit der einzelnen Algorithmen vergleichen. Manche haben sogar bisschen besseres Ergebnis geliefert, z.b. Bernouli classifier.

Cross validation

Beim Training eines Modells ist es wichtig, es nicht mit komplexen Algorithmen zu sehr (Overfitting) oder mit einfachen Algorithmen zu wenig (Underfitting) anzupassen. Die Wahl des Trainings- und Testsatzes ist entscheidend für die Reduzierung dieses Risikos. Es ist jedoch schwierig, den Datensatz so aufzuteilen, dass das Lernen und die Gültigkeit der Testergebnisse maximiert werden. Hier kommt die Kreuzvalidierung zum Einsatz. Kreuzvalidierung sucht der besten train und test kombination durch unterschiedlichen iterationen der zeilen für die lösung.

```
from sklearn.model_selection import cross_val_score
```

```
df_ = df_m.dropna()
X, y = df_.drop('happy',1), df_['happy']
```

```
y.value_counts()
```

```
happy        18813
not_happy     2861
Name: happy, dtype: int64
```

```
cross_val_score(rf_model,X,y,cv=4).mean()
```

```
0.8649530203225856
```

```
cross_val_score(bel_model,X,y,cv=4,).mean()
```

```
0.8679985267101229
```

```
cross_val_score(ml,X,y,cv=4,).mean()
```

```
0.848389076251743
```

```
cross_val_score(eclf2,X,y,cv=4,).mean()
```

```
0.8661528101601144
```

Nach der kreuzvalidierung kommen wir zum Ergebnis dass mit diesem Datensatz können wir höchsten <> 85% score erreichen. Trotzdem sind wir zufrieden weil wir unserem eigen Ziel (75%) längst erreicht haben. Alle vier Algorithmen haben uns mit diesem Datensatz gute Score geliefert.

```
rf_model = RandomForestClassifier(max_depth=20, max_features = 12,n_estimators = 600 ).fit(X_train, y_train)
```

```
t = pd.DataFrame(rf_model.feature_importances_, index=list(X.columns)).sort_values(0, ascending=False)
t = t.rename(columns= {0:'values'})
t
```

	values
financial	0.118503
Ethical_justification	0.058156
science_and_technologie	0.057459
secure_in_neighborhood	0.047526
trust	0.046855
Political_action	0.046323
effect_of_immigration	0.042934
corruption_of_institutions	0.040600
Identity_levels	0.039628
worried_about_risks_and_war	0.036836
active	0.036191
Perceived_democraticness	0.034162
nachbarn	0.033658
moral_rules	0.032992
Satisfaction_with_system	0.031860
economic_values	0.030368
comparing_parents	0.028206
religion_and_science	0.027603
corruption_preceptions	0.025991
Importance_of_democracy	0.025468
importance_of_god	0.021919
Respect_human_rights	0.019528
Electoral_integrity_battery	0.019346
Interest_for_politics	0.019238
court	0.018763
Social_class	0.016460
main_goal	0.016179
work	0.014580
family	0.012669

Von Randomforestclassifier haben wir feature_importance Attribut genutzt, in dem wir die alle wichtigste features nach Ihrer wichtigkeit sortiert kriegen konnten. Wie wir oben sehen können, ökonomische, Ethische und wissenschaftliche vielfalt haben eine besondere starke einfluss auf menschlicher zufriedenheit als vielfalt von alle andere aspekten.

```
t = pd.DataFrame(-bel_model.coef_[0], index=list(X.columns)).sort_values(0, ascending=False)
t = t.rename(columns= {0:'values'})
t
```

	values
family	1.946351
financial	1.613412
active	1.097731
Satisfaction_with_system	1.067801
work	1.066519
Perceived_democraticness	1.019010
Electoral_integrity_battery	1.017790
worried_about_risks_and_war	1.014138
moral_rules	0.892986
religion_and_science	0.889762
Ethical_justification	0.874852
secure_in_neighborhood	0.823347
science_and_technologie	0.755426
Identity_levels	0.736838
Social_class	0.730413
main_goal	0.725849
economic_values	0.667477
Political_action	0.621308
Respect_human_rights	0.614767
trust	0.578749
effect_of_immigration	0.572479
corruption_of_institutions	0.559285
Interest_for_politics	0.516276
Importance_of_democracy	0.499432
court	0.465869
nachbarn	0.465167
comparing_parents	0.432717
importance_of_god	0.414541
corruption_preceptions	0.356864

Bernouli liefert aber bisschen andere Ergebnis Als Randomforest. Hier sehen wir Familie an der ersten position, so die andere feature auch andere Ergebnisse aufweist haben.

Web software entwicklung

Werkzeug: Streamlit

Mit dieser software kann man seine/ihre Glück messen. Mit sidebar muss man ein paar einstellungen seines/ihres Leben eingeben und mit ‘Predict’ schaltfläche kann man ergenis erhalten.