1. Predspracovanie

1.1. Spracovanie jedného oleja

súbor:

Predprocessing/convert_functions.py

volanie:

convert_oil(file_name, output, head)

vstupy:

file_name:

cesta k vstupnému súboru.

output:

otvorený súbor pre výpis použitím funkcie open.

head:

hlavička obsahujúca typ oleja a id oleja (v našom prípade názov súboru).

výstupy:

žiaden (zmeny sa zapisujú priamo na output).

opis:

Funkcia načíta údaje z textového súboru a prerobí ich na formát csv. Tieto údaje následne uloží do výstupu. Funkcia môže byť spustená opakovane, aby spracovala viacero súborov, v tom prípade sa údaje z nich zapíšu na samostatné riadky.

príklad:

```
convert_oil("data/Grupo
C-LOO/200428_071902/Spectra.POSITIVE.txt",
           "converted.csv",
           "LOO,200428_071902")
```

1.2. Spracovanie jedného oleja do DataFramu

súbor:

Predprocessing/convert_functions.py

volanie:

convert_oil_to_df(df, file_name, head)

vstupy:

df:

DataFrame, do ktorého sa budú ukladať dáta.

file_name:

cesta k vstupnému súboru.

head:

hlavička obsahujúca typ oleja a id oleja (v našom prípade názov súboru).

výstupy:

df:

DataFrame, obsahujúci spracované dáta.

opis:

Funkcia načíta údaje z textového súboru a vloží ich do DataFramu. Funkcia môže byť spustená opakovane, aby spracovala viacero súborov. Funkcia vráti DataFrame so spacovaným súborom.

príklad:

```
df = convert_oil_to-df(df,"data/GrupoC-LOO/200428_071902
/Spectra.POSITIVE.txt", "LOO,200428_071902")
```

1.3. Spracovanie viacerých olejov

súbor:

Predprocessing/convert.py

volanie:

convert_multiple_oils(path, limit_sample, oil_type, output)

vstupy:

path:

cesta k priečinku s priečinkami vzoriek.

limit_sample:

číslo, koľko vzoriek chceme (-1 ak všetky).

oil_type:

názov triedy.

output:

otvorený súbor s výstupom.

výstupy:

žiaden (výstup je zapísaný priamo na output).

opis:

Funkcia vykoná spracovanie meraní s poskytnutých. Spracuje niekoľko text súborov a zapíše ich do poskytnutého cvs súboru. Tento súbor na zápis ostáva otvorený, aby sa do neho mohli zapísať vzorky z iných tried.

príklad:

```
out = open(SAVE_PATH, "w")
print_header(out)

convert_multiple_oils(LOAD_PATH_EVOO, 1, "EVOO", out)
convert_multiple_oils(LOAD_PATH_VOO, 1, "VOO", out)
convert_multiple_oils(LOAD_PATH_LOO, 1, "LOO", out)
```

1.4. Odstránenie nepotrebných stĺpcov

1.4.1. v_1 až v_150

súbor:

Predprocessing/filter_values.py

volanie:

filter_values_1_150(data)

vstupy:

data:

DataFrame, pre ktorý chceme odstrániť stĺpce.

výstupy:

data:

DataFrame s odstránenými stĺpcami.

opis:

Funkcia odstráni stĺpce v_1 až v_150. Tieto stĺpce boli pri dátovej analýze vyhodnotené ako nezaujímavé, nakoľko majú stále rovnaké hodnoty. Funkcia vráti DataFrame bez týchto stĺpcov.

1.4.2. Posledný stĺpec

súbor:

Predprocessing/filter_values.py

volanie:

filter_values_last(data)

vstupy:

data:

DataFrame, pre ktorý chceme odstrániť stĺpec.

výstupy:

data:

DataFrame s odstráneným stĺpcom.

opis:

Funkcia odstráni posledný stĺpec. Používa sa na odstránenie v_448, ktorý obsahuje takmer iba nan hodnoty a preto ho nie je možné použiť. Funkcia vráti DataFrame bez tohto stĺpca.

1.5. Overenie chýbajúcich spektier

súbor:

Predprocessing/check_missing_spectra.py

volanie:

add_missing_spectra(df)

vstupy:

df:

DataFrame súboru, v ktorom chceme overiť, či má všetky spektrá.

output:

súbor, ktorý obsahuje všetky spektrá pre každý olej.

výstupy:

df:

DataFrame súboru s chýbajúcimi riadkami.

opis:

Funkcia overí, či súbor obsahuje záznam o všetkých spektrách olejov, ktoré sa v súbore nachádzajú. V prípade, že chýba nejaké spektrum pre konkrétny olej, na chýbajúce miesto sa doplní riadok, ktorý obsahuje hodnoty oil_id, spectrum a analysis_time, ostatné hodnoty sú NaN.

1.6. Vytvorenie hlavičky súboru

súbor:

Predprocessing/convert_functions.py

volanie:

print_header(output)

vstupy:

output:

otvorený súbor csv pre výpis.

výstupy:

žiaden (hlavička sa zapíše priamo na output).

opis:

Vypíše hlavičku spracovaných dát do poskytnutého súboru. Hlavička obsahuje názvy stĺpcov:"class, oil_id, spectrum, drift_field_intensity, pressure, temperature, drift_tube_length, analysis_time" a následne t_0-t_448 v_0-v_448.

1.7. Vytvorenie hlavičky súboru ako pole

súbor:

Predprocessing/convert_functions.py

volanie:

print_header_df()

vstupy:

-

výstupy:

header:

hlavička pre DataFrame vo forme pola.

opis:

Vypíše hlavičku spracovaných dát do poľa. Hlavička obsahuje názvy stĺpcov:

"class, oil_id, spectrum, drift_field_intensity, pressure, temperature, drift_tube_length, analysis_time" a následne t_0-t_448 v_0-v_448.

1.8. Celkové predspracovanie

súbor:

Predprocessing/predprocess_data.py

volanie:

predprocess(oil_type, files, train, id)

vstupy:

oil_type:

trieda oleja (“EVOO”, “VOO”, “LOO”).

files:

pole txt súborov, ktoré treba predspracovať.

train:

True/False podľa toho, či sú dáta trénovacie alebo validačné.

id:

identifikátor prvej vzorky oleja.

výstupy:

žiadne (výstup sa uloží na na predefinovanú adresu).

opis:

Funkcia predspracuje súbory vo formáte txt a uloží ich ako csv.

2. Normalizácia

Normalizovanie stĺpcov v_xy na hodnoty v rozmedzí [0,1].

2.1. Spracovanie normalizácie (fit a transform)

súbor:

Predprocessing/normalization.py

volanie:

normalize(df)

vstupy:

df:

DataFrame s dátami.

výstupy:

df:

DataFrame s normalizovanými dátami.

opis:

Funkcia z priloženého DataFramu odseparuje stĺpce v_xy, nad ktorými vykoná fit a transform normalizácie. Upravené stĺpce nahradí v pôvodnom DataFrame a odstráni stĺpec v_448. Funkcia vracia takto upravený DataFrame.

2.2. Spracovanie iba transformácie

súbor:

Predprocessing/normalization.py

volanie:

transform_only(df)

vstupy:

df:

DataFrame s dátami.

výstupy:

df:

DataFrame s normalizovanými dátami.

opis:

Funkcia z priloženého DataFramu odseparuje stĺpce v_xy, nad ktorými vykoná transformáciu. Upravené stĺpce nahradí v pôvodnom DataFrame a odstráni stĺpec v_448. Funkcia vracia takto upravený DataFrame v prípade, že bolo možné transformáciu vykonať. V opačnom prípade (nebol nájdený súbor s fit-om) funkcia vráti None.

2.3. Fit normalizácie

súbor:

Predprocessing/normalization.py

volanie:

normalize_fit(df1)

vstupy:

df:

DataFrame zložený len z dátových stĺpcov v_xy.

výstupy:

žiadne (vykonaný fit sa uloží do súboru).

opis:

Funkcia z priloženého DataFramu identifikuje minimálnu a maximálnu hodnotu. Na základe týchto hodnôt sa vykoná fit pre škálovanie stĺpcov na hodnoty v rozmedzí [0,1]. Nafitovaný model sa uloží.

2.4. Transformácia

súbor:

Predprocessing/normalization.py

volanie:

normalize_transform(df1)

vstupy:

df:

DataFrame zložený len z dátových stĺpcov v_xy.

výstupy:

df1:

DataFrame s normalizovanými dátami.

opis:

Funkcia nad stĺpcami DataFramu vykoná transformáciu na hodnoty v rozmedzí [0,1]. ďalej zavolá funkciu clip_values, ktorá zabezpečí, aby boli všetky hodnoty v rozmedzí [0,1]. Návratom funkcie je takto upravený DataFrame. V prípade, že ne nebol nájdený súbor s fit-om funkcia vráti None.

2.5. Odstránenie stĺpca v_448

súbor:

Predprocessing/normalization.py

volanie:

df = delete_v_448_column(df)

vstupy:

df:

DataFrame s dátami.

výstupy:

df:

DataFrame bez daného stĺpca.

opis:

Funkcia odstráni stĺpec v_448 ak sa v df taký stĺpec nachádza. Návratom funkcie je upravený alebo pôvodný DataFrame na vstupe v závislosti od toho či df obsahoval stĺpec v_448.

2.6. Clip hodnôt

súbor:

Predprocessing/normalization.py

volanie:

clip_values(df1)

vstupy:

df1:

DataFrame zložený len z dátových stĺpcov v_xy.

výstupy:

df1:

DataFrame s orezanými hodnotami.

opis:

Funkcia zabezpečí orezanie hodnôt na hodnoty v rozmedzí [0,1]. Získanie v_xy stĺpcov. Návratom hodnoty je upravený DataFrame.

2.7. Normalizácia

súbor:

Predprocessing/normalization.py

volanie:

cut_v_columns(df)

vstupy:

df1:

DataFrame s dátami.

výstupy:

df1:

DataFrame obsahujúci iba v stĺpce.

opis:

Funkcia vytvorí nový DataFrame obsahujúci iba v_xy stĺpce vstupného df. Tento nový df je návratovou hodnotou funkcie.

2.8. Získanie minimálnej a maximálnej hodnoty

súbor:

Predprocessing/normalization.py

volanie:

get_min_max_values(df1)

vstupy:

df1:

DataFrame zložený len z dátových stĺpcov v_xy.

výstupy:

min_value:

minimálna hodnota.

max_value:

maximálna hodnota.

opis:

Funkcia z v_xy získa celkovú najmenšiu a najväčšiu hodnotu, ktoré vráti ako návratovú hodnotu funkcie.

3. Práca s črtami

3.1. Črty

3.1.1. Priemer

súbor:

FeatureSelection/features.py

volanie:

def mean(df_values, column)

vstupy:

df_values:

DataFrame obsahujúci surové predspracované dáta pre práve jeden olej.

column:

názov stĺpca pre ktorý sa vytvorí priemer.

výstupy:

mean_value:

vypočítaný priemer.

opis:

Funkcia na vstupe získa DataFrame dát pre práve jeden olej a názov stĺpca z tohto DataFramu (alebo slova ,,ALL"). V prípade názvu stĺpca sa pre daný stĺpec vypočíta priemer. V prípade slova ,,ALL" v argumente sa vypočíta priemerná hodnota pre všetky bunky v DataFrame. Vypočítaná hodnota je výstupom funkcie.

V prípade využitia pre jeden konkrétny stĺpec sa využíva zápis v tvare "v_n" čiže napríklad mean(df,"v_0") vypočíta priemer zo stĺpca v_0.

príklad:

```
df = pd.DataFrame(data = {'v_0':[1, 4], 'v_1':[4, 7]})
value = mean(df, 'ALL')
assert value == 4
```

3.1.2. Maximum

súbor:

FeatureSelection/features.py

volanie:

max(df_values, column)

vstupy:

df_values:

DataFrame, tabuľka s hodnotami.

column:

názov stĺpca (v_0-v_448).

výstupy:

max_value:

vypočítané maximum.

opis:

Funkcia na vypočítanie maxima v danom stĺpci.

3.1.3. Minimum

súbor:

FeatureSelection/features.py

volanie:

min(df_values, column)

vstupy:

df_values:

DataFrame, tabuľka s hodnotami.

column:

názov stĺpca (v_0-v_448).

výstupy:

mean_value:

vypočítané minimum.

opis:

Funkcia na vypočítanie minima v danom stĺpci.

3.1.4. Medián

súbor:

FeatureSelection/features.py

volanie:

median(df_values, column)

vstupy:

df_values:

DataFrame, tabuľka s hodnotami.

column:

názov stĺpca (v_0-v_448).

výstupy:

median_value:

vypočítaný medián.

opis:

Funkcia na vypočítanie mediánu v danom stĺpci.

3.1.5. Smerodajná odchýlka

súbor:

FeatureSelection/features.py

volanie:

std(df_values, column)

vstupy:

df_values:

DataFrame, tabuľka s hodnotami.

column:

názov stĺpca (v_0-v_448).

výstupy:

std_value:

vypočítaná smerodajná odchýlka.

opis:

Funkcia na vypočítanie smerodajnej odchýlky (Standard deviation) v danom stĺpci.

3.1.6. Rozdiel medzi maximom a minimom

súbor:

FeatureSelection/features.py

volanie:

delta(df_values, column)

vstupy:

df_values:

DataFrame, tabuľka s hodnotami.

column:

názov stĺpca (v_0-v_448).

výstupy:

delta_value:

vypočítaný rozdiel maxima a minima.

opis:

Funkcia na vypočítanie rozdielu medzi maximom a minimom v danom stĺpci.

3.1.7. Súčet

súbor:

FeatureSelection/features.py

volanie:

sum(df_values, column)

vstupy:

df_values:

DataFrame, tabuľka s hodnotami.

column:

názov stĺpca (v_0-v_448).

výstupy:

sum_value:

vypočítaný súčet.

opis:

Funkcia na vypočítanie súčtu hodnôt v danom stĺpci.

3.1.8. Point

súbor:

FeatureSelection/features.py

volanie:

point(df, attribute)

vstupy:

df:

DataFrame, vstup je tabuľka, musí obsahovať stĺpec v_0.

attribute:

[function]_[value]-[value_len]-[spectrum]-[spectrum_len]:

function:

žiadaná funkcia max alebo mean.

value:

číslo, ktoré určuje začiatočný stĺpec.

value_len:

počet hodnôt (stĺpcov).

spectrum:

číslo spektra od ktorého začať.

spectrum_len:

počet spektier.

výstupy:

point:

vypočítaná hodnota črty point.

opis:

Výpočet črty point, čo je maximum/priemer z vyrezanej oblasti. Používa sa najmä na odhalené anomálie medzi triedami oleja. Získanie hodnôt pre attribute je popísané v Zistenie hodnôt pre črtu point (hľadanie anomálií).

príklad:

```
point(df, "mean_0-3-1-2")

      v_0  v_1   v_2   v_3
0     1    1     2     -1
1     9    5     4     -1
2     14   224   22    0

23.333333333333332
```

```
(zobrazený index nieje spektrum, nakoľko spektrum začína od 1)
```

3.1.9. Diff - odchýlka od šablóny

súbor:

FeatureSelection/features.py

volanie:

diff(df, oil_type)

vstupy:

df:

DataFrame, tabuľka s hodnotami.

oil_type:

typ oleja, na základe ktorého sa vyberie šablóna. v prípade, ak je k typu pripojené “-mask” použije sa aj maska.

výstupy:

diff:

vypočítaná hodnota črty diff.

opis:

Funkcia vypočíta celkovú odchýlku od jednotlivých tried.

postup:

- Načíta sa šablóna pre olej, prípadne maska.
  - šablóna(DataFrame) bola vytvorená pomocou priemeru hodnôt oleju danej triedy (prvých 10 olejov).
  - maska (DataFrame) je veľkosti šablóny a obsahuje na všetkých miestach hodnotu 1, okrem miest, ktoré boli vyhodnotené ako variabilné v rámci triedy na základe analýzy rovnakých 10-tich olejov.
- Vytvorí sa Diff mapa odčítaním šablóny a aktuálneho oleja.
  - predstavuje rozdiel/odchylku.
- Odstránia sa riadky obsahujúce nan hodnoty (spôsobené rôznym počtom spektier v olejoch).
- Z hodnôt sa spraví absolútna hodnota, čiže vzdialenosť hodnôt.
- Rozdiely menšie ako určená hranica (threshold=2) sa zamenia za hodnotu 0.
- Ak je nastavené použitie masky, aplikuje sa vynásobením aktuálnej mapy maskou.
- Funkcia vráti súčet všetkých hodnôt.

príklad:

```
features.diff(df,”diff_EVOO-mask”)
```

3.2. Vytvorenie tabuľky s črtami

súbor:

create_features_df.py

volanie:

def create_features_df(df)

vstupy:

df:

DataFrame obsahujúci predspracované dáta.

výstupy:

df_features:

výstup funkcie. DataFrame obsahujúci údaje o triede k jedinečným id olejov.

opis:

Funkcia z DataFramu surových predspracovaných dát vytvorí DataFrame obsahujúci len informácie o jedinečných id olejov s informáciou a kategórií oleja.

príklad:

```
df = pd.DataFrame(data = {'class':['EVOO', 'EVOO', 'EVOO'],
'oil_id':['1', '1', '2'], 'data' :['1234', '2345', '3456']})

df_control = pd.DataFrame(data = {'class':['EVOO', 'EVOO'],
'oil_id':['1', '2']})

df_features = create_features_df(df)

assert df_features.equals(df_control) == True
```

3.3. Vytvorenie tabuľky so všetkými črtami

súbor:

create_features_df.py

volanie:

def create_features(df)

vstupy:

df:

DataFrame obsahujúci predspracované dáta.

výstupy:

df_features:

výstup funkcie. DataFrame obsahujúci vytvorené črty pre všetky unikátne oleje.

opis:

Funkcia z DataFramu surových predspracovaných dát vytvorí DataFrame obsahujúci vypočítané všetky črty pre každý olej z daného DataFramu.

3.4. Zistenie chýbajúcich čŕt

súbor:

FeatureSelection/add_features.py - súbor obsahuje funkciu na zistenie chýbajúcich hodnôt (features) vo výslednom DataFrame.

volanie:

check_missing(df, features_to_chceck, df_features)

vstupy:

df:

DataFrame obsahujúci predspracované dáta.

features_to_chceck:

pole features, ktoré majú svoje funkcie pre ich výpočet.

df_features:

DataFrame obsahujúci údaje o triede k jedinečným id olejov a môže už obsahovať aj niektoré features.

výstupy:

žiadny (črty sa pridajú priamo do DataFramu).

opis:

Funkcia zistí chýbajúce črty a pre ne zavolá príslušné funkcie na výpočet. Následne sú do DataFramu pridané chýbajúce črty.

3.5. Vytvorenie obrázka z dát

3.5.1. Vytvorenie obrázka

súbor:

FeatureSelection/create_pictures.py

volanie:

create_image(oil_df, directory, oil_id)

vstupy:

oil_df:

DataFrame obsahujúci dáta (v_x stĺpce) jedného oleja. Index DataFrame(u) je stĺpec Spectrum.

directory:

trieda do ktorej je olej zaradený (EVOO, VOO, LOO).

oil_id:

ID oleja.

výstupy:

žiadne (obrázok sa uloží do súboru)

opis:

Funkcia z vytvorí obrázok z DataFramu dát pre jeden olej. Atribúty obrázka (výška a šírka) sú nastavované ako globálne premenné. Vytvorený obrázok sa uloží do priečinka, ktorý označuje triedu oleja, pod menom zadefinovaným ako ID oleja.

3.5.2. Lineárna interpolácia farby

súbor:

FeatureSelection/create_pictures.py

volanie:

linear_interpolation_color(value)

vstupy:

value:

hodnota konkrétnej bunky z DataFrame.

výstupy:

red:

hodnota červenej farby, ktorá bude zapísaná do obrázka. Hodnota je v rozmedzí 0 - 255.

blue:

hodnota modrej farby, ktorá bude zapísaná do obrázka. Hodnota je v rozmedzí 0 - 255.

opis:

Funkcia robí lineárnu interpoláciu medzi modrou a červenou farbou. Vstupom funkcie je hodnota, ktorá je interpolovaná. Maximálna a minimálna hodnota, medzi ktorými sa interpoluje sú nastavené ako globálne premenné.

3.6. Vytvorenie diff mapy

3.6.1. Funkcia na vytvorenie diff mapy

súbor:

FeatureSelection/features.py

volanie:

diff_map(df1, df2, start_column, end_column)

vstupy:

df1,df2:

DataFrame, jednotlivé tabuľky medzi ktorými chceme vytvoriť diffmapu.

start_column, end_column:

názov začiatočného a koncového stĺpca, odkiaľ a pokiaľ chceme zobrať hodnoty pre diff mapy.

výstupy:

result:

DataFrame, ktorý predstavuje diff mapu.

opis:

Funkcia na vytvorenie takzvanej diff mapy, čiže odčítanie hodnôt v dvoch tabuľkách údajov. Používa sa na zistenie rozdielov medzi hodnotami rôznych tried alebo v rámci jednej triedy.

príklad:

```
diff_map(df1, df2, "v_0", "v_447")
```

3.6.2. Vygenerovanie diff máp pre jednu triedu

súbor:

FeatureSelection/create_diff_maps.py

volanie:

same_class_diff(source_file, output_path, size)

vstupy:

source_file:

súbor, ktorý obsahuje oleje rovnakej triedy, po predspracovaní (napr.dataEVOO.csv v Data/Raw).

output_path:

cesta, kde chceme aby sa vytvorili súbory s diff mapami. Každá mapa je samostatný súbor. cesta končí znakom “/”.

size:

veľkosť vzorky, počet súborov, ktoré chceme vytvoriť, “-1” ak chceme všetky možné.

výstupy:

žiadne (výstup sa ukladá priamo do súboru).

opis:

Táto funkcia vytvorí diff mapu pre za sebou idúce dvojice olejov. Pre súbor s 10-timi olejmi tak vytvorí 5 diff máp.

príklad:

```
same_class_diff('../Data/Raw/dataEVOO-10.csv',
'../Data/Feature/DiffMaps/EVOO/', 5)
```

3.6.3. Vygenerovanie diff máp pre rôzne triedy

súbor:

FeatureSelection/create_diff_maps.py

volanie:

different_class_diff(source_file1, source_file2, output_path, size)

vstupy:

source_file1,source_file1:

súbory, kde každý obsahuje oleje rovnakej triedy, po predspracovaní (napr.dataEVOO.csv v Data/Raw).

output_path:

cesta, kde chceme aby sa vytvorili súbory s diff mapami. Každá mapa je samostatný súbor. cesta končí znakom "/".

size:

veľkosť vzorky, počet súborov, ktoré chceme vytvoriť, "-1" ak chceme všetky možné.

výstupy:

žiadne (výstup sa ukladá priamo do súboru).

opis:

Táto funkcia vytvorí diff mapu pre dvojice olejov pričom sa zoberie vždy i-ty olej z každej triedy. Pre súbory s 10-timi olejmi tak vytvorí 10 diff máp.

príklad:

```
different_class_diff('../Data/Raw/dataEVOO-10.csv',
'../Data/Raw/dataLOO-10.csv','../Data/Feature/DiffMaps/EVOO-LOO/'
, 5)
```

3.6.4. Vygenerovanie diff máp medzi olejom a jeho šablónou

súbor:

FeatureSelection/create_diff_maps.py

volanie:

template_diff(source_file, template, output_path, size)

vstupy:

source_file:

súbor obsahuje oleje rovnakej triedy, po predspracovaní (napr.dataEVOO.csv v Data/Raw)

template:

cesta k šablóne, ktorá bude použitá.

output_path:

cesta, kde chceme aby sa ukladali vzniknuté diff mapy.

size:

veľkosť vzorky, počet súborov, ktoré chceme vytvoriť, "-1" ak chceme všetky možné.

výstupy:

žiadne (výstup sa ukladá priamo do súboru).

opis:

Táto funkcia vytvorí diff mapu pre oleje zo vstupu vzhľadom na zadanú šablónu. Tieto mapy ďalej slúžia pre vytvorenie masky oleja.

3.6.5. Vygenerovanie šablóny

súbor:

FeatureSelection/create_diff_maps.py

volanie:

create_template(source_file, output_file)

vstupy:

source_file:

súbor obsahuje oleje rovnakej triedy, po predspracovaní (napr.dataEVOO.csv v Data/Raw).

output_file:

cesta aj s názvom súboru, kam sa uloží šablóna.

výstupy:

žiadne (výstup sa ukladá priamo do súboru).

opis:

Táto funkcia vytvorí šablónu danej triedy oleja s poskytnutých olejov, a to priemerom hodnôt v olejoch.

príklad:

```
create_template('../Data/Raw/dataEVOO-10.csv',
'../Data/Feature/DiffMaps/templates/EVOO.csv')
```

3.6.6. Vygenerovanie masky

súbor:

FeatureSelection/create_diff_maps.py

volanie:

create_mask(source_file, template, output_path)

vstupy:

source_file:

cesta k súborom ktoré vznikli funkciou template_diff. Sú to diff mapy medzi olejmi a šablónou ich triedy.

template:

cesta k použitej šablóne, pre ktorú chceme vytvoriť masku, je potrebná aby maska bola rovnako veľká.

output_file:

cesta aj s názvom súboru, kam sa uloží maska.

výstupy:

žiadne (výstup sa ukladá priamo do súboru).

opis:

Táto funkcia vytvára masku danej triedy oleja s poskytnutých olejov, a to nasledovným postupom:

- Zistia sa oblasti v šablóne, kde sa nachádzajú variabilné hodnoty v rámci olejov jednej triedy.
- Vytvorí sa základ masky, DataFrame rovnako veľký ako šablóna, obsahujúci len hodnoty = 1.
- Na miesta zistené v kroku 1 sa doplnia hodnoty 0.

Maska sa používa tak, že ňou vynásobíme hodnoty, čiže miesta kde je maska rovná 1 sa zachovajú a miesta, kde je hodnota 0 sa vynulujú.

3.7. Zistenie hodnôt pre črtu point (hľadanie anomálií)

3.7.1. Aplikovanie hraničnej hodnoty (threshold)

súbor:

FeatureSelection/select_point_feature.py

volanie:

apply_threshold(data, threshold)

vstupy:

data:

DataFrame, na ktorý chceme aplikovať threshold.

threshold:

hraničná hodnota (odporúčaná hodnota 2).

výstupy:

data:

DataFrame po aplikovaní threshold.

opis:

Funkcia zmení hodnoty menšie ako hraničná hodnota na 0.

3.7.2. Nájdenie oblastí, kde sú hodnoty (threshold)

súbor:

FeatureSelection/select_point_feature.py

volanie:

get_points(data)

vstupy:

data:

DataFrame, pre ktorý chceme nájsť oblasti.

výstupy:

points:

Dataframe obsahujúci nájdené oblasti.

opis:

Funkcia je používaná na identifikovanie oblastí, kde sú hodnoty. Je používaná pre črtu point na identifikovanie hraníc anomálií.

Ohraničenie je vykonané oddelením častí DataFrame pomocou odstránenia nulových riadkov/stĺpcov. Kvôli tomu nemusí oddeliť anomálie blízko seba alebo v niektorých špecifických pozíciách.

príklad:

	start index	end index	start v	end v	max	mean
0	1699.0	1700.0	v_205	v_205	2.0086	2.005700
6	5270.0	5280.0	v_204	v_206	2.1764	1.841061
3	4579.0	4595.0	v_204	v_208	2.7110	1.787266
2	3724.0	3741.0	v_202	v_208	2.6580	1.758643
5	5085.0	5105.0	v_204	v_208	2.4000	1.675175

3.7.3. Spriemerovanie diff máp

súbor:

FeatureSelection/select_point_feature.py

volanie:

average_diff_maps(maps_path)

vstupy:

maps_path:

cesta k mapám, ktoré chceme spriemerovať.

výstupy:

map_sum:

DataFrame tvorený priemernými hodnotami.

opis:

Vytvorí mapu pomocou priemeru hodnôt v poskytnutých mapách. Funkcia sa používa na zovšeobecnenie odchýlok pre dva oleje na odhalenie všeobecných odchýlok. Čiže ak vytvoríme 10 máp, ktoré reprezentujú rozdiely medzi 20-timi olejmi, tak spriemerovaním týchto máp dostaneme priemerné rozdiely v danej triede. Rozdiely boli používané v absolútnej hodnote.

3.7.4. Získanie hodnôt pre črtu point

súbor:

FeatureSelection/select_point_feature.py

volanie:

get_diff_points(path, show, threshold)

vstupy:

path:

cesta k mapám, na ktorých chceme vykonať analýzu, (napr. "../data/Feature/DiffMaps/templates/EVOO").

show:

hodnota True/False podľa toho či chceme ukázať anomálie pomocou heatmapy.

threshold:

hraničná hodnota, určuje ako veľké/malé anomálie hľadáme, odporúčaná hodnota je 2 pre normalizované dáta.

výstupy:

DataFrame obsahujúci hodnoty len na zaujímavých oblastiach

opis:

Zistí hodnoty pre črty point, čiže hranice anomálií pre zadané diff mapy olejov. Táto funkcia zahŕňa funkcie vyššie:

- average_diff_maps(path).
- apply_threshold(averaged, threshold).
- get_points(data_threshold).

3.8. Feature selection

3.8.1. Filtrácia

súbor:

FeatureSelection/select_features.py

volanie:

filter_method(df)

vstupy:

df:

celý DataFrame spracovaných dát s úpravou stĺpca class z typu string na int.

výstupy:

žiadne (výstup sa vypíše priamo do konzoly).

opis:

Metóda hľadá korelácie medzi črtami. Následne určí relevantné črty(x>0,5). Všetky črty s koreláciou väčšou ako 0,5 vypíše.

3.8.2. RFE - Hľadanie vhodného počtu čŕt

súbor:

FeatureSelection/select_features.py

volanie:

recursive_feature_elimination(df,labels)

vstupy:

df:

spracované dáta načítané do DataFrame (bez stĺpca class).

label:

DataFrame stĺpca class.

výstupy:

žiadne (výstup sa vypíše priamo do konzoly).

opis:

Funkcia greedy prístupom vyhľadáva optimálny počet čŕt. Metóda vypíše optimálny počet features a skóre zvolených features s decision tree klasifikátorom

3.8.3. Výpis najvhodnejších čŕt s daným množstvom

súbor:

FeatureSelection/select_features.py

volanie:

print_rfe_features(df, labels, head)

vstupy:

df:

spracované dáta načítané do DataFrame (bez stĺpca class).

label:

DataFrame stĺpca class.

nof:

počet features, pre ktoré má nájsť optimálne features.

výstupy:

žiadne (výstup sa vypíše priamo do konzoly).

opis:

Funkcia zistí optimálne features pri danom množstve.

4. Tradičné klasifikátory

4.1. Vytvorenie modelu random forest

súbor:

StandardClasification/random_forest.py

volanie:

classify_random_forest(df)

vstupy:

df:

DataFrame s vypočítanými features.

výstupy:

evaulate_clf :

funkcia vracia skóre pre metriky accuracy, F1 micro a F1 macro

opis:

Vo funkcii sa delia vzorky na testovacie a trénovacie. Klasifikátor sa trénuje, následne sa uloží model a vyhodnotí klasifikátor. Ak už model existuje použije sa pri trénovaní.

Pri prvom spustení je výstupom natrénovaný model klasifikátoru random forest a zobrazená úspešnosť klasifikátora pomocou metík accuracy, F1 micro a F1 macro skóre. Taktiež sa vypíše matica zámen. Pri ďalších spusteniach je výstupom iba úspešnosť.

4.1.1. Train random forest

súbor:

StandardClasification/random_forest.py

volanie:

train_random_forest(train_features, train_labels, tuning=False)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

tuning=False:

volanie funkcie na vylepšenie hyperparametrov je primárne vypnuté.

výstupy:

clf:

nastavený klasifikátor na random forest spolu s hyperparametrami

opis:

Funkcia, ktorá vytvorí a vráti klasifikátor Random Forest.

4.1.2. Tuning random forest

súbor:

StandardClasification/random_forest.py

volanie:

rf_tuning(train_features, train_labels)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

výstupy:

clf.best_params_:

najlepšie parametre zvoleného klasifikátora.

opis:

Funkcia na nájdenie najlepších hyperparametrov zo zvolenej množiny.

4.2. Vytvorenie modelu decision tree

4.2.1. Classify decision tree

súbor:

StandardClasification/decision_tree.py

volanie:

classify_decision_tree(df)

vstupy:

df:

DataFrame s vypočítanými features.

výstupy:

evaulate_clf :

funkcia vracia skóre pre metriky accuracy, F1 micro a F1 macro

opis:

Vo funkcii sa delia vzorky na testovacie a trénovacie. Klasifikátor sa trénuje, následne sa uloží model a vyhodnotí klasifikátor. Ak už model existuje použije sa pri trénovaní.

Pri prvom spustení je výstupom natrénovaný model klasifikátoru decision tree a zobrazená úspešnosť klasifikátora pomocou metík accuracy, F1 micro a F1 macro skóre. Taktiež sa vypíše matica zámen. Pri ďalších spusteniach je výstupom iba úspešnosť.

4.2.2. Train decision tree

súbor:

StandardClasification/decision_tree.py

volanie:

train_decision_tree(train_features, train_labels, tuning=False)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

tuning=False:

volanie funkcie na vylepšenie hyperparametrov je primárne vypnuté.

výstupy:

clf:

nastavený klasifikátor na decision tree spolu s hyperparametrami

opis:

Funkcia, ktorá vytvorí a vráti klasifikátor Decision Tree.

4.2.3. Tuning decision tree

súbor:

StandardClasification/decision_tree.py

volanie:

decision_tree_tuning(train_features, train_labels)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

výstupy:

clf.best_params_:

najlepšie parametre zvoleného klasifikátora.

opis:

Funkcia na nájdenie najlepších hyperparametrov zo zvolenej množiny.

4.3. Vytvorenie modelu SVM

4.3.1. Classify SVM

súbor:

StandardClasification/svm.py

volanie:

classify_svm(df)

vstupy:

df:

DataFrame s vypočítanými features.

výstupy:

evaulate_clf :

funkcia vracia skóre pre metriky accuracy, F1 micro a F1 macro

opis:

Vo funkcii sa delia vzorky na testovacie a trénovacie. Klasifikátor sa trénuje, následne sa uloží model a vyhodnotí klasifikátor. Ak už model existuje použije sa pri trénovaní.

Pri prvom spustení je výstupom natrénovaný model klasifikátoru SVM a zobrazená úspešnosť klasifikátora pomocou metík accuracy, F1 micro a F1 macro skóre. Taktiež sa vypíše matica zámen. Pri ďalších spusteniach je výstupom iba úspešnosť.

4.3.2. Train SVM

súbor:

StandardClasification/svm.py

volanie:

train_svm(train_features, train_labels, tuning=False)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

tuning=False:

volanie funkcie na vylepšenie hyperparametrov je primárne vypnuté.

výstupy:

clf:

nastavený klasifikátor na SVC spolu s hyperparametrami

opis:

Funkcia, ktorá vytvorí a vráti klasifikátor SVM.

4.3.3. Tuning SVM

súbor:

StandardClasification/svm.py

volanie:

svm_tuning(train_features, train_labels)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

výstupy:

clf.best_params_:

najlepšie parametre zvoleného klasifikátora.

opis:

Funkcia na nájdenie najlepších hyperparametrov zo zvolenej množiny.

4.4. Vytvorenie modelu KNN

4.4.1. Classify KNN

súbor:

StandardClasification/knn.py

volanie:

classify_knn(df)

vstupy:

df:

DataFrame s vypočítanými features.

výstupy:

evaulate_clf :

funkcia vracia skóre pre metriky accuracy, F1 micro a F1 macro

opis:

Vo funkcii sa delia vzorky na testovacie a trénovacie. Klasifikátor sa trénuje, následne sa uloží model a vyhodnotí klasifikátor. Ak už model existuje použije sa pri trénovaní.

Pri prvom spustení je výstupom natrénovaný model klasifikátoru KNN a zobrazená úspešnosť klasifikátora pomocou metík accuracy, F1 micro a F1 macro skóre. Taktiež sa vypíše matica zámen. Pri ďalších spusteniach je výstupom iba úspešnosť.

4.4.2. Train KNN

súbor:

StandardClasification/knn.py

volanie:

train_knn(train_features, train_labels, tuning=False)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

tuning=False:

volanie funkcie na vylepšenie hyperparametrov je primárne vypnuté.

výstupy:

clf:

nastavený klasifikátor na knn spolu s hyperparametrami

opis:

Funkcia, ktorá vytvorí a vráti klasifikátor knn.

4.4.3. Tuning KNN

súbor:

StandardClasification/knn.py

volanie:

knn_tuning(train_features, train_labels)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

výstupy:

clf.best_params_:

najlepšie parametre zvoleného klasifikátora.

opis:

Funkcia na nájdenie najlepších hyperparametrov zo zvolenej množiny

4.5. Vytvorenie modelu naive bayes (gaussian)

4.5.1. Classify naive bayes

súbor:

StandardClasification/naive_bayes.py

volanie:

classify_naive_bayes(df)

vstupy:

df:

DataFrame s vypočítanými features.

výstupy:

evaulate_clf :

funkcia vracia skóre pre metriky accuracy, F1 micro a F1 macro

opis:

Vo funkcii sa delia vzorky na testovacie a trénovacie. Klasifikátor sa trénuje, následne sa uloží model a vyhodnotí klasifikátor. Ak už model existuje použije sa pri trénovaní.

Pri prvom spustení je výstupom natrénovaný model klasifikátoru naivný bayes a zobrazená úspešnosť klasifikátora pomocou metík accuracy, F1 micro a F1 macro skóre. Taktiež sa vypíše matica zámen. Pri ďalších spusteniach je výstupom iba úspešnosť.

4.5.2. Train naive bayse

súbor:

StandardClasification/naive_bayes.py

volanie:

train_naive_bayes(train_features, train_labels, tuning=False)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

tuning=False:

volanie funkcie na vylepšenie hyperparametrov je primárne vypnuté.

výstupy:

clf:

nastavený klasifikátor na naive bayes spolu s hyperparametrami

opis:

Funkcia, ktorá vytvorí a vráti klasifikátor Naive Bayes.

4.5.3. Tuning naive bayes

súbor:

StandardClasification/naive_bayes.py

volanie:

naive_bayes_tuning(train_features, train_labels)

vstupy:

train_features:

trénovacie features.

train_labels:

trénovacie labels.

výstupy:

clf.best_params_:

najlepšie parametre zvoleného klasifikátora.

opis:

Funkcia na nájdenie najlepších hyperparametrov zo zvolenej množiny.

5. Neurónové siete

5.1. Multilayer Perceptron

5.1.1. Classify MLP

súbor:

Neurons/train_mlp.py

volanie:

classify_mlp(df)

vstupy:

df:

DataFrame obsahujúci nájdené črty.

výstupy:

žiadne (výstup sa vypíše do konzoly).

opis:

Súbor obsahuje funkciu na trénovanie klasifikátora Multilayer Perceptron (MLP). Funkcia na vstupe získa DataFrame čŕt pre všetky oleje. Z DataFramu si uloží pole tried olejov, aby sme na konci mohli vyhodnotiť úspešnosť klasifikátora. Načítané dáta sa náhodne rozdelia na trénovacie a testovacie. Na trénovacích dátach as natrénuje model MLP a následne sa vyhodnotí jeho úspešnosť na testovacích dátach.

5.1.2. Train MLP

súbor:

StandardClasification/mlp.py

volanie:

train_mlp(train_features, train_labels, tuning=False)

vstupy:

train_features:

trénovacie fetures.

train_labels:

trénovacie labels.

tuning=False:

volanie funkcie na vylepšenie hyperparametrov je primárne vypnuté.

výstupy:

clf

nastavený klasifikátor MLP spolu s hyperparametrami

opis:

Funkcia, ktorá vytvorí a vráti klasifikátor Multilayer Perceptron.

5.1.3. Tuning MLP

súbor:

Neuronsn/train_mlp.py

volanie:

mlp_tuning(train_features, train_labels)

vstupy:

train_features:

trénovacie fetures.

train_labels:

trénovacie labels.

výstupy:

clf.best_params_

najlepšie hyperparametre MLP klasifikátora.

opis:

Funkcia na nájdenie najlepších hyperparametrov zo zvolenej množiny.

5.2. VGG

5.2.1. Init VGG

súbor:

Neurons/Vgg16Model.py

volanie:

def __init__(self, num_classes)

vstupy:

num_classes:

počet tried, ktoré klasifikujeme.

výstupy:

out:

posledná vrstva siete, ktorá obsahuje predikcie.

opis:

Trieda Vgg16 reprezentuje architektúru konvolučnej neurónovej siete.

5.2.2. Classify VGG

súbor:

Neurons/train_vgg_net.py

volanie:

classify_vgg16()

vstupy:

žiadne.

výstupy:

žiadne (výstup sa vypíše do konzoly).

opis:

Súbor obsahuje funkciu na trénovanie neurónovej siete Vgg16. Funkcia si načíta obrázky zo súboru pomocou datasetu. Dáta sú náhodne rozdelené na trénovacie a testovacie. Na trénovacích dátach as natrénuje model Vgg16 a následne sa vyhodnotí jeho úspešnosť na testovacích dátach.

6. Trénovanie

6.1. Spúšťanie trénovania 1 modelu

súbor:

StandardClasification/train.py

volanie:

train_model(classifier, train_features, train_labels)

vstupy:

classifier:

klasifikátor, ktorý chceme natrénovať.

train_features:

načítané trénovacie dáta.

train_labels:

triedy, ktoré patria k jednotlivým dátam.

výstupy:

clf:

natrénovaný klasifikátor.

opis:

Vo funkcii sa načíta klasifikátor, ktorý je následne natrénovaný na zvolených dátach. Natrénovaný klasifikátor je uložený. Funkcia vracia natrénovaný klasifikátor.

6.2. Dotrénovanie modelu

súbor:

StandardClasification/train.py

volanie:

retrain_model()

vstupy:

-

výstupy:

-

opis:

Prázdna funkcia, ktorá bude obsahovať kód na dotrénovanie vybraného modelu na nových dátach.

6.3. Spúšťanie trénovania

súbor:

StandardClasification/train.py

volanie:

train(classifiers, data)

vstupy:

classifiers:

pole klasifikátorov, ktoré chceme natrénovať.

data:

načítané trénovacie dáta vo forme DataFramu.

výstupy:

žiadne (natrénované klasifikátory sa priamo uložia)

opis:

Funkcia rozdelí trénovacie dáta na dáta a triedy. Následne natrénuje na dátach zvolené klasifikátory.

7. Klasifikovanie

7.1. Klasifikácia pomocou 1 modelu

súbor:

StandardClasification/predict_classes.py

volanie:

predict_class(classifier, test_features)

vstupy:

classifier:

klasifikátor, ktorý chceme vyhodnotiť.

test_features:

načítané testovacie dáta.

výstupy:

pred_labels:

pole predikovaných tried.

opis:

Funkcia načíta natrénovaný klasifikátor a predikuje triedu načítaných dát. Funkcia vráti pole predikovaných tried.

7.2. Klasifikácia modelov

súbor:

StandardClasification/predict_classes.py

volanie:

predict(classifiers, data)

vstupy:

classifiers:

pole klasifikátorov, ktoré chceme vyhodnotiť.

data:

načítané testovacie dáta vo forme DataFramu.

výstupy:

predictions:

pole predikovaných tried.

predictions_arr:

pole predikcií jednotlivých klasifikátorov pre každú predikciu.

opis:

Funkcia rozdelí testovacie dáta na dáta a triedy. Klasifikátory natrénuje a vyhodnotí úspešnosť predikcie jednotlivých klasifikátorov..

7.3. Klasifikácia pomocou využitia kombinácie modelov

súbor:

StandardClasification/predict_classes.py

volanie:

combine_predictions(predictions_arr)

vstupy:

predictions_arr:

pole predikcií jednotlivých klasifikátorov.

výstupy:

final_arr:

pole predikcií.

opis:

Funkcia načíta pole predikcí samostatných klasifikátorov. Pre jednotlivé vzorky identifikuje triedu, ktorá sa vyskytuje najčastejšie ako odpoveď.

Funkcia vracia pole predikcií, ktoré boli vytvorené na základe jednotlivých predikcií klasifikátorov.

8. Evaluácia a grafy

8.1. Evaluácia klasifikátora

súbor:

Statistic/evaluation.py

volanie:

evaluate_clf(test_labels, pred_labels, name)

vstupy:

test_labels:

skutočné lable z klasifikátora.

pred_labels:

predikované lable z klasifikátora.

name:

meno vstupujúceho klasifikátora.

výstupy:

evaluation:

úspešnosť klasifikátora, pole accuracy, F1 micro a F1 macro.

opis:

Funkcia vypočíta metriky accuracy, F1 micro a F1 macro pre klasifikátor tie vracia v percentách. Taktiež sa vytvára matica zámen, ktorá sa vypíše do konzoly a uloží ako obrázok do StandardClasification/charts pod názvom daného klasifikátora vo formáte názov-klasifikátora_matrix.png.

8.2. Vytvorenie grafu výsledkov pre všetky tradičné klasifikátory

súbor:

Statistic/create_chart.py, Súbor obsahuje funkciu na vytvorenie grafu výsledkov pre všetky tradičné klasifikátory.

volanie:

scores_charts(score, file_name=”score”)

vstupy:

scores:

výsledky klasifikátora z funkcie def evaluate_clf(test_labels, pred_labels, name).

file_name:

názov grafu pod ktorým bude uložený obrázok.

výstupy:

žiadne (výstup sa priamo uloží do súboru).

opis:

Funkcia vytvorí graf výsledkov všetkých tradičných klasifikátor pre metriky accuracy, F1 micro a F1 macro.