TEHNIČKA I KORISNIČKA DOKUMENTACIJA - PREDVIĐANJE TIPA OSOBNOSTI

Ivan Krcivoj i Antonela Bogdanić

Projekt iz kolegija Umjetna inteligencija
Prirodoslovno-matematički fakultet, Matematički odsjek
veljača 2021., Zagreb

SADRŽAJ

---

1. UVOD

Ova je dokumentacija popratni materijal uz Seminarski rad i program *prediction.py*. Prvi dio dokumentacije predstavljaju detalji funkcija i modela korištenih u kodu. Drugi dio dokumentacije odnosi se na korisničke upute o pokretanju i korištenju.

Program smo pisali jezikom Python, uz korištenje pomoćnih *libraryja* i *modula*. Također, spomenuli bismo i *Jupyter notebook* u kojem smo određene dijelove koda pisali i pokretali. Ova dokumentacija, kao i Seminarski rad, napisani su korištenjem *Jupyter notebooka*. Time se omogućuje interaktivno pokretanje našeg koda. Zbog toga su funkcije pisane tako da se mogu pokrenuti nezavisno.

---

2. TEHNIČKA DOKUMENTACIJA

U ovom odjeljku fokusirat ćemo se na objašnjenje koda koji se nalazi u datoteci *prediction.py*. Opise funkcija napisanih u kodu podijelit ćemo u više odjeljaka radi lakšeg snalaženja. Najprije ćemo objasniti funkcije vezane uz pripremu podataka. Zatim će slijediti opisi funkcija koje su korištene pri analiziranju podataka. Na samom kraju, podijelit ćemo u posebna poglavlja svaki od korištenih modela radi lakšeg snalaženja.

Na početku samog programa nalaze se naredbe s kojima *importamo* sve potrebne *module* za daljnji rad. Nećemo ih redom obrazlagati.

2.1 PRIPREMA PODATAKA

Podaci s kojima radimo nalaze se u dvije .csv datoteke naziva *train.csv* i *train.csv* u mapi *Data*. Ta se mapa nalazi u istoj mapi kao i ova dokumentacija. U seminarskom radu opisali smo zašto te dvije datoteke spajamo u jednu, a ovdje ćemo prikazati kako.

In [ ]:
```
#ucitavanje podataka
data_train = pd.read_csv('Data/train.csv')
data_test = pd.read_csv('Data/test.csv')
#mjenjamo ime zadnjeg stupca
data_train.rename(columns={'Personality (Class label)':'Personality'}, inplace=True)
data_test.rename(columns={'Personality (class label)':'Personality'}, inplace=True)
data = pd.concat([data_train, data_test])

flag_valid = 0
```

U ovom kratkom isječku koda vidimo čitanje datoteka u kojima se nalaze podaci, promjena imena zadnjeg stupca radi jednostavnosti i na kraju spajanje u *data*. *Data*, *data_train*, *data_test* bit će globalne varijable tijekom cijelog programa. Također, jedna od globalnih varijabli bit će i *flag_valid*, koja je na početku 0, a kasnije ćemo reći zašto smo je koristili. Za detaljan opis podataka u tablici i samog izgleda tablice upućujemo Vas na Seminarski rad.

In [ ]:
```
#funkcija sređuje podatke
def adjust_data():
  array = data.values
  global flag_valid
  if flag_valid !=0:
      return

  flag_valid +=1
  for i in range(len(array)):
    if array[i][0]!='Male' and array[i][0]!='Female':
        data['Gender'].replace(array[i][0],'Female', inplace=True)

  for i in range(len(array)):
    if array[i][1]<15 or array[i][1] > 30:
      data['Age'].replace(array[i][1],round(data['Age'].mean()), inplace=True)

  columns = [ 'Gender','Age','openness','neuroticism','conscientiousness','agreeableness','e
xtraversion', 'Personality']
  for j in range(2,7):
    for i in range(len(array)):
      if array[i][j]<1:
         data[columns[j]].replace(array[i][j],1, inplace=True)
      elif array[i][j] > 8:
        data[columns[j]].replace(array[i][j],8, inplace=True)
```

Funkcija *adjust_data*() izgleda kao gore, a ima za zadatak provjeriti ispravnost podataka i promijeniti neispravne podatke. Prije provjere ispravnosti htjeli smo smanjiti složenost i provjeriti je li ovaj posao već odrađen. Ukoliko je vrijednost *flag_valid* veća od 0, već smo promijenili i nije potrebno nastavljati s funkcijom, a ukoliko nije nastavljamo s funkcijom i povećavamo vrijednost *flag_valid*.

Moramo provjeriti 3 stvari: jesu li podaci za spol korektno zapisani (imaju li vrijednosti '*Female*' i '*Male*'), jesu li godine u normalnom rasponu i jesu li sve ocjene zapisane brojevima od 1 do 8. Obrazloženja zašto na taj način mijenjamo vrijednosti opisan je u Seminarskom radu.

Nadalje, uvidom u tip podataka u tablici, htjeli smo da neki podaci promjene svoj tip. U nekim će nam funkcijama biti pogodno gledati spol numeričkim oznaka, isto vrijedi i za klase osobnosti. Zato smo pretvorbu u numeričke tipove podijelili na dva dijela, odnosno dvije funkcije.

In [ ]:
```
#funkcija koja mjenja spolove u brojeve
def gender_to_num():
    adjust_data()
    data['Gender'].replace(['Male'], 0, inplace=True)
    data['Gender'].replace(['Female'], 1, inplace=True)

#funckija promjene osobnosti u brojeve
def personality_to_num():
    adjust_data()
    data['Personality'].replace(['extraverted'], 1, inplace = True)
    data['Personality'].replace(['serious'], 2, inplace = True)
    data['Personality'].replace(['dependable'], 3, inplace = True)
    data['Personality'].replace(['lively'], 4, inplace = True)
    data['Personality'].replace(['responsible'], 5, inplace = True)
```

Funkcije su jednostavne, dodana je samo još jedna provjera jesu li svi podaci korektni (u slučaju kada se funkcije pozivaju izvan konteksta cijele "priče").

Zadnji dio pripreme podataka je dijeljenje podataka kako bi ih mogli koristiti na modelima. Možemo reći da u kodu imamo dvije vrste dijeljenja. Prvo dijeljenje označava dijeljenje na podatke za trening i podatke za testiranje. To činimo pomoću funkcije *split_data*(). Cjelokupan skup podataka dijelimo u (standardnom) omjeru 70% za treniranje i 30% podataka za testiranje, a spremamo ih u pripadne globalne varijable *train* i *test*.

In [ ]:
```
#dijenje podataka
def split_data():
    global train
    global test
    train_length = round(0.7 * len(data))
    test = data[train_length:]
    train = data[:train_length]
```

U drugom tipu dijeljenja podatke dijelimo na "zavisan" i "nezavisan" dio. Konkretnije, pomoću prvih 7 stupaca želimo predvidjeti vrijednosti u osmom stupcu. Osim toga, htjeli smo da ta funkcija zaokružuje cjelokupnu pripremu podataka za modeliranje. Upravo se zato zove *prepare_date*() i njezin je zadatak proći kroz sve spomenute korake. Nju pozivamo na početku svakog modeliranja.

In [ ]:
```
#funkcija koja priprema podatke za model
def prepare_data():
    adjust_data()
    gender_to_num()
    personality_to_num()
    split_data()

    features = [ 'Gender','Age','openness','neuroticism','conscientiousness','agreeableness'
,'extraversion']
    global X
    global y
    global X_test
    global y_test
    X, y = train[features].values, train['Personality'].values
    X_test, y_test = test[features].values, test['Personality'].values
```

Također, dodatne globalne varijable u cijelom kodu su i *X*, *X_test* koje predstavljaju "nezavisan" dio trening i test podataka, dok varijable *y* i *y_test* predstavljaju "zavisan" dio.

2.2 ANALIZA PODATAKA

U ovome ćemo dijelu opisati kratke funkcije koje pozivamo prilikom analize podataka. Za svaku funkciju u Seminarskom radu detaljno je opisan razlog korištenja, način pozivanja i zaključak izveden iz prikaza, stoga ćemo u ovoj dokumentaciji objasniti sami kod vezan uz njih.

Prva funkcija je vezana uz grafički prikaz PCA analize.

In [ ]:
```
#graf PCA po dvije komponente
def graph_PCA_2():
    gender_to_num()
    features = [ 'Gender','Age','openness','neuroticism','conscientiousness','agreeableness'
,'extraversion']
    n_components = 7

    pca = PCA(n_components=n_components)
    components = pca.fit_transform(data[features])

    total_var = pca.explained_variance_ratio_.sum() * 100

    labels = {str(i): f"PC {i+1}" for i in range(n_components)}
    labels['color'] = 'Personality'

    fig = px.scatter_matrix(components, color=data['Personality'], dimensions=range(n_compon
ents), labels=labels, title=f'Total Explained Variance: {total_var:.2f}%')
    fig.update_traces(diagonal_visible=False)
    fig.show()
```

Htjeli smo pojednostaviti korištenje pa smo uz pomoć funkcije *gender_to_num*() pretvorili sve stringove spola u brojeve 0 ili 1. Napomenimo, da se pri nezavisnom korištenju funkcije crtanja grafa (kao i ostalih) ne treba brinuti o ispravnosti podataka jer je ta provjera uključena u funkciji *gender_to_num*(). Koristit ćemo funkcije i metode iz biblioteke *sklearn.decomposition.PCA* , poput *PCA*(), *fit_transform*(), *pca.explained_varianceratio.sum*(). Dok iz biblioteke *plotly.express* (koja je importana kao px) koristimo funkcije za grafički prikaz. (Za detalje pogladati u Seminarskom radu pod literatura [4])

In [ ]:
```
def scree_plot():
    gender_to_num()
    features = [ 'Gender','Age','openness','neuroticism','conscientiousness','agreeableness'
,'extraversion']
    n_components = 7

    pca = PCA(n_components=n_components)
    components = pca.fit_transform(data[features])

    per_var = np.round(pca.explained_variance_ratio_ * 100, decimals=2)
    print(per_var)

    labels = ['PC' + str(x) for x in range(1, len(per_var)+1)]
    plt.bar(x=range(1,len(per_var)+1), height=per_var, tick_label= labels )
    plt.xlabel('Principal Component')
    plt.ylabel('Persentage of Explaind Variance')
    plt.title('Scree Plot')
    plt.show()
```

Sljedeća funkcija nam "crta" *scree plot*. Opet ćemo radi jednostavnosti pretvoriti spol u numeričke vrijednosti. Nadalje, slijedi sličan dio koda kao kod prethodne funkcije. Ovdje dodatno koristimo funkcije iz *mathplotlib.pyplot*. Jedan dio koda tiče se detalja uređenja samog grafa.

In [ ]:
```
def graph_PCA():
    gender_to_num()
    features = [ 'Gender','Age','openness','neuroticism','conscientiousness','agreeableness'
,'extraversion']
    X = data[features]

    pca = PCA(n_components=2)
    components = pca.fit_transform(X)

    loadings = pca.components_.T * np.sqrt(pca.explained_variance_)
    fig = px.scatter(components, x=0, y=1, color=data['Personality'])
    for i, feature in enumerate(features):
        fig.add_shape(
            type='line',
            x0=0, y0=0,
            x1=loadings[i, 0],
            y1=loadings[i, 1]
        )
        fig.add_annotation(
            x=loadings[i, 0],
            y=loadings[i, 1],
            ax=0, ay=0,
            xanchor="center",
            yanchor="bottom",
            text=feature,
        )
        fig.update_layout(title_text= 'PCA', xaxis_title='PC1', yaxis_title='PC2', )
    fig.show()
```

Još jedan graf vezan uz PCA je *graph_PCA*(). Njime prikazujemo sve podatke u dvije dimenzije na grafu i nadamo se dobiti "lijepo" grupiranje. Također, veliki dio koda otpada na uređenje samog prikaza, dok je u srži ponovo korištenje funkcija iz navedene biblioteke *sklearn.decomposition.PCA* .

Sljedeća funkcija odnosno grafički prikaz je često korišteni *heatmap* graf. Njega smo implementirali na sljedeći način.

In [ ]:
```
#heatmap
def graph_heatmap():
    gender_to_num()
    sns.heatmap(data.corr(), vmin=-0.1, vmax=0.1, annot=True, cmap='viridis')
    plt.title('Heatmap', fontsize =20)
    plt.show()
```

Biblioteka od koristi nam je ovog puta bila *seaborn* koja u sebi sadrži funkciju za crtanje *heatmape*. Detalji u izgledu grafa dodatno nam pomažu da istaknemo koreliranost podataka.

Zanimljiva inačica *box-plota* je *violin plot*. Stoga smo njega htjeli prikazati u našem projektu. Prilikom poziva funkcije u y varijablu spremamo naziv jednog od 7 "pitanja" iz testa. Na taj način, vidimo kako su odgovori na to pitanje distribuirani ovisno o rezultantnoj klasi osobnosti. Primjer jednog poziva funkcije je: violin_plot('openness'). I ovaj graf nalazimo u biblioteci *seaborn*.

In [ ]:
```
#violinplot
def violin_plot(Y):
  fig = sns.violinplot(data=data, y=Y, x="Personality", hue="Personality", box=True).set_tit
le('Violin plot')
  plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.).set_title('Personality:')
  plt.show()
```

Zadnja funkcija iz analize podataka služi nam za prikaz distribucija 5 klasa osobnosti u *train* i *test* podacima.

In [ ]:
```
#ispis distribucija
def distribution():
    gender_to_num()
    split_data()
    print('Train distribution:\n', train['Personality'].value_counts() / len(train))
    print('\n\nTest distribution:\n', test['Personality'].value_counts() / len(test))
```

Ovdje smo ponovo spol pretvorili u numeričke vrijednosti i podijelili podatke u one za treniranje i testiranje, s ciljem da prikažemo distribuciju 5 rezultantnih klasa za svaki od dva podskupova podataka.

2.3 MODELI PREDVIĐANJA

Za početak, objasnit ćemo 3 pomoćne funkcije koje nam olakšavaju rad s modelima. Prva od pomoćnih funkcija je *accuracy*() koja prima model i podatke i ispisuje točnost modela u postocima na tim podacima. Tu nam pomaže funkcija *score*() koja je funkcija članica svih modela koje ćemo mi koristiti.

In [ ]:
```
#funckija za tocnost
def accuracy(model, v_X, v_y):
  print("Accuracy:",round(model.score(v_X,v_y)*100,2,), "%")
```

Sljedeća funkcija je veoma bitna pri treniranju modela jer upravo pomoću nje poboljšavamo svojstva modela. Ona prima model, listu parametara koja ovisi o modelu te podatke na kojima želimo tražiti najbolje pripadne parametre.

In [ ]:
```
#funkcija za odabir parametara
def best_hyperparameters(model, param_grid, X_train, y_train):
  gs = GridSearchCV(model, param_grid)
  gs.fit(X_train, y_train)
  print(gs.best_estimator_)
  return gs.best_estimator_
```

Pomoću funkcija iz biblioteke *sklearn.model_selection* koristeći unakrsnu validaciju i isprobavanje svih mogućih kombinacija parametara iz liste koju smo proslijedili pronalazimo odgovarajući model za naše podatke.

In [ ]:
```
#funkcija za k-validaciju
def k_validation(model, df, k):
  features = [ 'Gender','Age','openness','neuroticism','conscientiousness','agreeableness',
'extraversion']
  acc = []
  sub_df = split(df,range(int64(ceil(len(df)/k)), len(df), int64(ceil(len(df)/k))))

  for i in range(k):
    df_train = pd.DataFrame()
    df_test = pd.DataFrame()

    for j in range(k):
      if i!=j:
        df_train = pd.concat([df_train, sub_df[j]])
      else: df_test = sub_df[j]

    X, y = df_train[features].values, df_train['Personality'].values
    X_test, y_test = df_test[features].values, df_test['Personality'].values

    model.fit(X,y)

    acc.append(model.score(X_test,y_test)*100)

  title = str(type(model)).split('.')[-1].split("'")[0]
  label = 'Average accuracy: {:.2f}%'.format(sum(acc) / len(acc))
  fig = sns.boxplot(y = acc)
  fig.set(ylabel='Accuracy [%]', title= title, xlabel=label)
  plt.show()
```

Zadnja pomoćna funkcija je važna za testiranje naših modela. Ona prima model, podatke i varijablu k. Podatke podijelimo u k dijelova i napravimo k-struku unakrsnu validaciju. Na kraju, sve točnosti spremimo u listu i pomoću nje nacrtamo, odnosno prikažemo *box-plot* tih vrijednosti.

2.4 STABLO ODLUKE

In [ ]:
```
#funkcija vezana uz model STABLA ODLUKE
def decision_tree():
    prepare_data()
    tree = DecisionTreeClassifier(criterion="entropy")
    tree_param ={
        'splitter': ['best', 'random'],
        'max_features': [None, 'sqrt', 'log2'],
        'class_weight': [None, 'balanced']
        #'min_samples_split' : np.arange(2,10,1),
        #'min_samples_leaf' : np.arange(1,50,1)
        }
    tree = best_hyperparameters(tree, tree_param, X, y)
    tree = tree.fit(X,y)
    accuracy(tree,X,y)

    accuracy(tree,X_test, y_test)

    k_validation(tree,data,30)
```

Kako smo dosta pomoćnih funkcija prije implementirali, stvaranje modela za predikcije bit će nam veoma jednostavno. Prije svega pripremimo podatke na već opisani način. Zatim pomoću biblioteke *sklearn.DecisionTreeClassifier* stvorimo model s unaprijed zadanim kriterijem *entropy*. Ostale parametre spremimo u listu koju zatim prosljeđujemo funkciji *best_hyperparameters*() da nam vrati one najbolje u odnosu na podatke X i y. Nakon toga moramo *fitati* model na tim skupovima i radi predostrožnosti ispišemo točnost na podacima za treniranje. Isto to napravimo i za test podatke. To nije jedina mjera točnosti modela koju ćemo promatrati, već pozovemo i pomoćnu funkciju *k_validation*() koja nam u konačnici i grafički prikaže točnost.

2.5 NEURONSKE MREŽE

In [ ]:
```
#funkcija za neuronsku mrezu uz vec najbolje parametre
def neural_network():
    prepare_data()
    clf = MLPClassifier(activation='logistic', solver='lbfgs', max_iter=6000, hidden_layer_s
izes=(600,600,600), tol=1e-6 )
    clf.fit(X, y)
    accuracy(clf,X,y)
    accuracy(clf,X_test, y_test)
    k_validation(clf, data,30)
```

Objašnjenje koda kod ovog modela veoma je pojednostavljeno s obzirom na to da smo više manje sve već objasnili na prethodnom modelu. Jedina je razlika što u ovome slučaju koristimo biblioteku *sklearn.neural_network.MLPClassifier* i što u kodu ne pozivamo funkciju za traženje najboljih parametara jer bi samo traženje bilo vremenski predugo za demonstraciju koda. Zato dobivene parametre upisujemo ručno u model. Također, i u ovome modelu na isti način testiramo točnost.

2.6 SLUČAJNA ŠUMA STABALA ODLUKA

In [ ]:
```
#funkcija za random forest uz vec najbolje parametre
def random_forest():
     prepare_data()
     rfc=RandomForestClassifier(criterion='entropy', n_estimators=150, max_depth=350, min_sa
mples_leaf=70, min_samples_split=40)
     rfc.fit(X,y)
     accuracy(rfc,X_test, y_test)
```

Ovdje se priča oko vremenski predugog izvršavanja traženja najboljih parametara ponavlja, pa samo ručno ponovo upisujemo najbolje parametre u funkciju. Funkcija koju koristimo nalazi se u biblioteci *sklearn.ensemble.RandomForestClassifier*. Ovdje pozivanje k-validacije nije potrebno, o čemu smo više rekli u Seminarskom radu.

2.7 LOGISTIČKA REGRESIJA

Za kraj, prikazujemo funkciju *log_reg*().

In [ ]:
```
#funkcija za logisticku regresija
def log_reg():
    prepare_data()
    mul_lr = linear_model.LogisticRegression(multi_class='multinomial', max_iter =1000)
    lr_param ={
        #'C':np.arange(0.01, 1.01, 0.01),
        'solver' : ['newton-cg', 'lbfgs'],
        'tol' : np.arange(1e-6,1e-4,0.000001)
        }
    mul_lr.fit(X, y)
    mul_lr = best_hyperparameters(mul_lr, lr_param, X, y)
    accuracy(mul_lr, X, y)
    accuracy(mul_lr, X_test, y_test)
```

Korištenjem biblioteke *sklearn.linear_model* stvaramo model logističke regresije i opet demonstriramo korištenje pomoćne funkcije *best_hyperparameters*(). Na kraju ispisujemo točnosti modela.

3. KORISNIČKA DOKUMENTACIJA

U paketu koji ste dobili nalaze se više datoteka. Ako želite testirati kod to možete učiniti na više načina. Način koji je najprirodniji je korištenjem datoteke prediction.py. Možete pokrenuti datoteku kao i bilo koju drugu ekstenzije .py i zatim pozivajući funkcije isprobavati kod. Obraćamo pažnju da prilikom toga treba paziti da se podaci na kojima radimo uvijek nalaze u mapi imena Data koja je na istoj razini kao i prediction.py.

Još jedan od načina je koristeći Seminarski rad koji je pisan u *Jupyter bilježnici* (ekstenzije .ipynb) i interaktivno pokretati već napisane dijelove koda. Također, možete na sličan način u novoj *Jupyter bilježnici* kreirati dijelove koda i tako isprobavati kod.