Classificatore Problemi Cardiaci

Membri del gruppo:

-Niccolò Caldarulo 710070 n.caldarulo2@studenti.uniba.it

Repository:

https://github.com/firefive555/ClassificatoreProblemiCardiaci-

Introduzione

Il progetto in questione consiste nello studio di un dataset fornito dal sito kaggle.com di pazienti affetti da problemi cardiaci attraverso metodi di apprendimento supervisionato e l'implementazione di query SPARQL.

Pre-Requisiti

Le librerie che richiede questo programma per funzionare sono:

-scikit-learn

-pandas

-bnlearn

Query SPARQL

La query viene utilizzata per ottenere valori che verranno utilizzati come punto di split per attributi continui, cosi da poter utilizzare il dataset per creare una rete bayesiana.

```
endpoint_url = "https://query.wikidata.org/sparql"

query = """SELECT ?numLabel ?num1Label ?ageAdultLabel ?peakLabel
WHERE
{
   wd:Q41861 wdt:P5135 ?num.
   wd:Q762713 wdt:P5135 ?num1.
   wd:Q1202615 wdt:P5136 ?ageAdult
   wd:Q188151 wdt:P5135 ?peak

  SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
}"""
```

Per la query viene utilizzato l'endpoint di wikidata, vengono chiesti i valori per i quali i livelli di colesterolo e pressione arteriosa superano i livelli normali e il valore di età dopo il quale si viene considerati anziani.

```
def get_results(endpoint_url, query):
    user_agent = "WDQS-example Python/%s.%s" % (sys.version_info[0], sys.version_info[1])
    # TODO adjust user agent; see https://w.wiki/CX6
    sparql = SPARQLWrapper(endpoint_url, agent=user_agent)
    sparql.setQuery(query)
    sparql.setReturnFormat(JSON)
    return sparql.query().convert()

results = get_results(endpoint_url, query)

for result in results["results"]["bindings"]:
    hypertension = int(result["numLabel"]["value"])
    hypercholesterol = int(result["num1Label"]["value"])
    adult = int(result["ageAdultLabel"]["value"])
    peak = int(result["peakLabel"]["value"])
```

Elaborazione Dataset

Dopo aver immagazzinato questi valori si utilizza pandas per leggere il csv contenente il dataset che intendiamo studiare.

```
data = pd.read_csv("pyRDF2Vec/samples/countries-cities/heart.csv")
data1=data
```

Una volta letto ed immagazzinato in due variabili (data e data1) si procede con il trasformare tutti gli attributi continui di data in attributi categorici grazie all'utilizzo dei dati precedentemente raccolti tramite query cosi da poter creare la nostra rete bayesiana.

```
data["Hypertension"] = data['RestingBP']
data['Hypertension'].values[data['RestingBP'].values > hypertension] = 1
data['Hypertension'].values[data['RestingBP'].values <= hypertension] = 0

data["Hypercholesterol"] = data["Cholesterol"]
data['Hypercholesterol'].values[data["Cholesterol"].values > hypercholesterol] = 1
data['Hypercholesterol'].values[data["Cholesterol"].values <= hypercholesterol] = 0

data["LifePeriod"] = data["Age"]
data["LifePeriod"].values[data["Age"].values >= adult] = 1
data["LifePeriod"].values[data["Age"].values < adult] = 0

data["Ischemy"] = data["Oldpeak"]
data["Ischemy"].values[data["Oldpeak"].values > peak] = 2
data["Ischemy"].values[data["Oldpeak"].values < peak] = 1

data = data.drop(columns=["Age" , "Oldpeak", "MaxHR", "RestingBP", "Cholesterol" ])
```

Si modifica poi il valore data1 in modo tale da avere solo valori numerici su tutti i domini degli attributi, si procede con un hot encoding delle colonne contenenti valori non numerici e senza una relazione d'ordine tra loro creando quindi una colonna per ogni possibile valore del dominio dell'attributo preso in esame, queste nuove colonne create avranno dominio [0,1].

```
data1["Sex"].values[data["Sex"].values == "M"] = 0
data1["Sex"].values[data["Sex"].values == "F"] = 1

data1["ExerciseAngina"].values[data["ExerciseAngina"].values == "N"] = 0
data1["ExerciseAngina"].values[data["ExerciseAngina"].values == "Y"] = 1


data1["ST_SlopeUp"] = 0
data1["ST_SlopeFlat"] = 0
data1["ST_SlopeDown"] = 0

data1["ST_SlopeUp"].values[data["ST_Slope"].values == "Up"] = 1
data1["ST_SlopeFlat"].values[data["ST_Slope"].values == "Flat"] = 1
data1["ST_SlopeDown"].values[data["ST_Slope"].values == "Down"] = 1

data1["ChestPainTypeATA"] = 0
data1["ChestPainTypeNAP"] = 0
data1["ChestPainTypeASY"] = 0
data1["ChestPainTypeTA"] = 0

data1["ChestPainTypeTA"].values[data["ChestPainType"].values == "TA"] = 1
data1["ChestPainTypeASY"].values[data["ChestPainType"].values == "ASY"] = 1
data1["ChestPainTypeNAP"].values[data["ChestPainType"].values == "NAP"] = 1
data1["ChestPainTypeATA"].values[data["ChestPainType"].values == "ATA"] = 1

data1["ResrtingECGNormal"] = 0
data1["RestingECGST"] = 0
data1["ResrtingECGLHV"] = 0

data1["ResrtingECGNormal"].values[data["RestingECG"].values == "Normal"] = 1
data1["RestingECGST"].values[data["RestingECG"].values == "ST"] = 1
data1["ResrtingECGLHV"].values[data["RestingECG"].values == "LHV"] = 1

data1 = data1.drop(columns=["RestingECG" , "ChestPainType", "ST_Slope"])
```

Una volta create questi nuovi attributi si tolgono dal dataset data1 gli attributi originali cosi da non avere problemi quando si passerà alla fase di apprendimento sueprvisionato.

Si splitta quindi il dataset sia data che data1 in test e set utilizzando la funzione tran_test_split con test_size = 0.2.

Apprendimento Rete Bayesiana

Si crea un loop che andarà si occuperà di dividere il dataset in training e test set 5 volte con un random seed che va da 0 a 4, per ogni iterazione di questo ciclo esterno si creerà un StratifiedKFold con 5 split anch'esso,

```
scores = []
nLoop = 5
g = 0
while(g < nLoop):
    train, test = train_test_split(data , test_size=0.2, random_state=g)
    outher_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
    Xdata = train.drop(columns ="HeartDisease")
    ydata = train["HeartDisease"]
    b = outher_cv.split(Xdata , ydata)
```

e si apprende la struttura della rete tramite la libreria bnlearn tramite il metodo structure_learning.

Una volta creata la rete per ogni esempio del validation set si calcola l'inferenza delle probabilità avendo come osservazioni i valori degli attributi di quella riga e come target l'attributo "HeartDisease", per calcolarne l'accuratezza si trasforma la predizione in 0 o 1 a seconda se essa supera o meno lo 0.5, si calcola poi il numero di predizioni errate e lo si divide per il numero di predizioni totali, a questo punto si sottrae ad uno e si ottiene l'accuratezza della rete bayesiana per il validation set in questione.

```
    for(train_idx, valid_idx) in b:
        train.iloc[train_idx].columns
        DAG = bn.structure_learning.fit(train.iloc[train_idx], methodtype='chow-liu', root_node='HeartDisease')
        model_mle = bn.parameter_learning.fit(DAG, train.iloc[train_idx], methodtype='maximumlikelihood')
        sum = 0
        for example in train.iloc[valid_idx].index:
            q1 = bn.inference.fit(model_mle, variables=['HeartDisease'], evidence={
                "Hypercholesterol":train["Hypercholesterol"][example],
                "Hypertension":train["Hypertension"][example],
                "ST_Slope":train["ST_Slope"][example],
                "ExerciseAngina":train["ExerciseAngina"][example],
                "RestingECG":train["RestingECG"][example],
                "FastingBS":train["FastingBS"][example],
                "ChestPainType":train["ChestPainType"][example],
                "Sex":train["Sex"][example],
                "LifePeriod":train["LifePeriod"][example]
                })
            if(train["HeartDisease"][example] == 0):
                if (q1.df["p"][1] > 0.5):
                    sum = sum + 1
            else:
                if(q1.df["p"][1] <= 0.5):
                    sum = sum+1

        probability = 1-(sum / len(test.index))
        scores.append(probability)
    g = g+1
print('\n   Outher Loop:')
print('Bayesian Network ACC %.2f%% +/- %.2f' % (np.mean(scores) *100, np.std(scores)*100))
```

Ad ogni iterata dopo aver calcolato l'accuratezza si inserisce in una lista di risultati chiamata score, alla fine di entrambi i cicli otterremo una lista di lunghezza 25 e su di essa calcoleremo la media e la varianza.

Apprendimento Supervisionato

Si mettono ora in comparazione i risultati ottenuti dalla rete bayesiana con quelli ottenuti da altri metodi di apprendimento supervisionato per classificatori (LogisticRegression, KNN, DecisionTree, SVC, RandomForest).

```
clf1 = LogisticRegression(multi_class='multinomial' , solver = 'newton-cg', random_state=1)

clf2 = KNeighborsClassifier( algorithm='ball_tree' , leaf_size=50)

clf3 = DecisionTreeClassifier(random_state=1)

clf4 = SVC(random_state=1)

clf5 = RandomForestClassifier(random_state=1)
```

Per ogni classificatore si crea una rispettiva griglia di parametri cosi da poterlo testare con opzioni differenti.

```
param_grid1 = [{'clf1__penalty': ['l2'], 'clf1__C': np.power(10., np.arange(-4, 4))}]

param_grid2 = [{'clf2__n_neighbors': list(range(1,10)), 'clf2__p': [1, 2]}]

param_grid3 = [{'max_depth': list(range(1 , 10)) + [None], 'criterion': ['gin1', 'entropy']}]

param_grid4 = [{'clf4__kernel': ['rbf'], 'clf4__C': np.power(10. , np.arange(-4, 4)),
                'clf4__gamma': np.power(10., np.arange(-5, 0))},
               {'clf4__kernel': ['linear'], 'clf4__C': np.power(10. , np.arange(-4, 4))}]

param_grid5 = [{'n_estimators': [10, 100, 500, 1000, 10000]}]
```

Si crea ora un dizionario nel quale si utilizza come chiave il nome del metodo di apprendimento e che contiene i valori delle param_grid

```
gridcvs = {}
innder_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

for pgrid, est, name in zip((param_grid1, param_grid2, param_grid3 , param_grid4, param_grid5),
                            (pipe1, pipe2, clf3, pipe4 , clf5),
                            ('Softmax', 'KNN', 'DTree', 'SVM', 'RForest')):
    print(pgrid , est , name)
    print('ok')
    gcv = GridSearchCV(estimator= est, param_grid=pgrid, scoring='accuracy', n_jobs=-1,
                       cv=innder_cv, verbose=0, refit=True)
    gridcvs[name] = gcv
```

Una volta fatto cio per ogni metodo che abbiamo scelto di utilizzare si istanza uno StratifiedKFold con 5 split come gia visto per l'apprendimento della rete bayesiana.

```
for name, gs_est in sorted(gridcvs.items()):
    print(name)
    nLoop = 5
    g = 0
    print(50 * '-', '\n')
    print('Alghorithm:', name)
    outer_scores = []
    while(g < nLoop):
        X_train , X_test , y_train , y_test = train_test_split(X, y, test_size=0.2, random_state=g, stratify=y)

        outher_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
        b = outher_cv.split(X_train , y_train)
        print('      Inner loop:')
        for (train_idx, valid_idx) in b:
            gridcvs[name].fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
            print('\n       Best ACC %.2f%%' % (gridcvs[name].best_score_ * 100))
            print('    Best Parameters:', gridcvs[name].best_params_)

            outer_scores.append(gridcvs[name].best_estimator_.score(X_train.iloc[valid_idx] , y_train.iloc[valid_idx]))
            print('         ACC (on outher test fold %.2f%%)' % (outer_scores[-1]*100))
        g = g+1
    print('\n   Outher Loop:')
    print('      ACC %.2f%% +/- %.2f' % (np.mean(outer_scores) * 100, np.std(outer_scores) * 100))
```

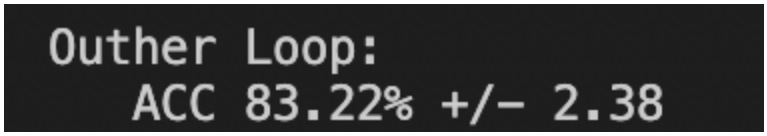
Si ha anche in questo caso un ciclo esterno che andrà a creare 5 differenti training e test set per ogni tipologia di algoritmo scelto, anche qui abbiamo una lista di risultati chiamata score e per ogni algoritmo viene calcolata la media e la varianza

Valutazione

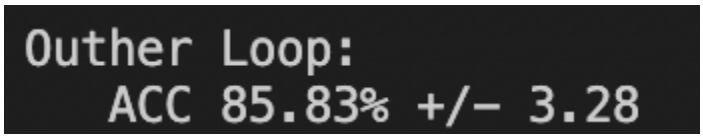
Notiamo come in tutti i classificatori il punteggio ottenuto è piu o meno similare, nella rete bayesiana otteniamo un punteggio di 88.26% con una varianza di +/- 1.99

```
   Outher Loop:
>>> print('Bayesian Network ACC %.2f%% +/- %.2f' % (np.mean(scores) *100, np.std(scores)*100))
Bayesian Network ACC 88.26% +/- 1.99
```

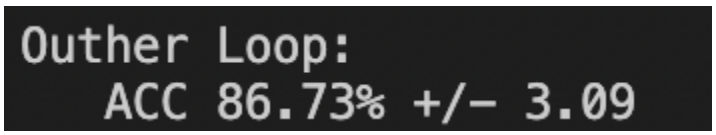
Decision Tree:



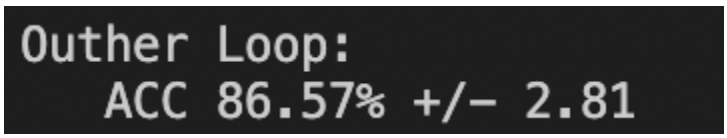
KNN:



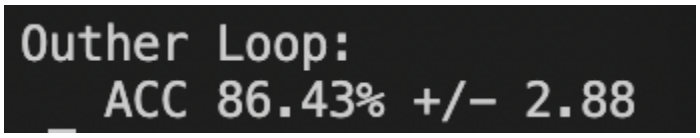
Random Forest:



SVM:



Softmax:



Notiamo dunque che tutti gli algoritmi sono in un range di accuratezza 83-88% con la rete bayesiana con la performance migliore invece notiamo che l'albero decisione è quello che performa peggio.