comp3009_assignment

September 24, 2019

[1]:
```
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.utils import resample

import cufflinks as cf
import ipywidgets as widgets
from ipywidgets import interact, interact_manual
from ipywidgets.embed import embed_minimal_html

cf.go_offline(connected = False)
```

[2]:
```
df = pd.read_csv('data2019.student.csv')
```

[3]:
```
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_missing})
missing_value_df[missing_value_df['percent_missing'] > 0]
```

[3]:

	column_name	percent_missing
Class	Class	9.090909
att3	att3	0.363636
att9	att9	0.454545
att13	att13	93.454545
att19	att19	94.000000
att25	att25	0.272727
att28	att28	0.545455

[4]:
```
to_drop = ['ID', 'att13', 'att19']
df.drop(labels = to_drop, axis = 1, inplace = True)

df['att3'].fillna(value = df['att3'].mode()[0], inplace = True)
df['att9'].fillna(value = df['att9'].mode()[0], inplace = True)
```

[5]:
```
# percent_missing = df.isnull().sum() * 100 / len(df)
# missing_value_df = pd.DataFrame({'column_name': df.columns,
#                                  'percent_missing': percent_missing})
# missing_value_df
```

[6]:
```
df['att25'].iplot(kind = 'box',
                  title = 'Boxplot of Attribute 25',
                  xTitle = '',
                  yTitle = 'value')
```

[7]:
```
df['att28'].iplot(kind = 'box',
                  title = 'Boxplot of Attribute 28',
                  xTitle = '',
                  yTitle = 'value')
```

[8]:
```
# median
df['att25'].fillna(value = int(df['att25'].median()), inplace = True)
# mean
df['att28'].fillna(value = int(np.floor(df['att28'].mean())), inplace = True)
```

[9]:
```
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_missing})
missing_value_df[missing_value_df['percent_missing'] > 0]
```

[9]:
```
      column_name  percent_missing
Class       Class         9.090909
```

[10]:
```
# for i in df.columns:
#     if df[i].nunique() <= 1:
#         print(i)
```

[11]:
```
to_drop = ['att14', 'att17']
df.drop(labels = to_drop, axis = 1, inplace = True)
```

[12]:
```
# df.shape
```

[13]:
```
df = df.T.drop_duplicates().T
```

[14]:
```
# df.shape
```

[15]:
```
df.drop_duplicates(inplace = True)
```

[16]:
```
# df.shape
```

[17]:
```
# @interact
# def box_plots(attribute=list(df.columns)):
#     df[attribute].iplot(kind = 'box')
```

[18]:
```
df['att20'].iplot(kind = 'box', title = 'att20 Boxplot (numeric)')
```

[19]:
```
df['att12'].iplot(kind = 'box', title = 'att12 Boxplot (categorical)')
```

[20]:
```
df = df.apply(pd.to_numeric, errors='ignore')
```

[21]:
```
#df.dtypes
```

[22]:
```
to_numeric = ['att18', 'att20', 'att21', 'att22', 'att25', 'att28']

# for i in to_numeric:
```

```
#     display(df[i].describe())
```

[23]:
```
# @interact_manual
# def hist_plots(attribute=list(to_numeric)):
#     df[attribute].iplot(kind = 'hist',
#                          title = str(attribute) + ' Before Scaling',
#                         xTitle = 'value')
```

[24]:
```
df['att21'].iplot(kind = 'hist', title = 'att21 Before Scaling', xTitle =␣
 ↪'value')
```

[25]:
```
to_gauss = ['att18', 'att20', 'att21', 'att22']
to_ln = ['att25', 'att28']
```

[26]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaled = pd.DataFrame(scaler.fit_transform(df.loc[:,to_gauss].values), columns␣
 ↪= to_gauss)
df.loc[:,to_gauss] = scaled.values
```

[27]:
```
scaler = preprocessing.RobustScaler()
scaled = pd.DataFrame(scaler.fit_transform(df.loc[:,to_ln].values), columns =␣
 ↪to_ln)
df.loc[:,to_ln] = scaled.values
```

[28]:
```
scaler = preprocessing.MinMaxScaler()
scaled = pd.DataFrame(scaler.fit_transform(df.loc[:,to_numeric].values),␣
 ↪columns = to_numeric)
df.loc[:,to_numeric] = scaled.values
```

[29]:
```
# @interact_manual
# def hist_plots(attribute=list(to_numeric)):
#     df[attribute].iplot(kind = 'hist',
#                         title = str(attribute) + ' After Scaling',
#                         xTitle = 'value')
```

[30]:
```
df['att21'].iplot(kind = 'hist', title = 'att21 After Scaling', xTitle =␣
 ↪'value')
```

[31]:
```
to_categorical = ['att30', 'att29', 'att27', 'att26', 'att23', 'att16',␣
 ↪'att15', 'att12', 'att11',
                  'att10', 'att9', 'att7', 'att6', 'att5', 'att4', 'att3',␣
 ↪'att2', 'att1']
df = pd.get_dummies(df, columns = to_categorical)
```

[32]:
```
#df.shape
```

[33]:
```
df['Class'].value_counts()
```

[33]:
```
1.0    650
0.0    250
```

```
Name: Class, dtype: int64
```

[34]:
```
minority_class = df[df.Class == 0]
minority_upsampled = resample(minority_class, replace = True, n_samples = (650␣
 ↪- 250))
```

[35]:
```
df_train = df[:-100]
df_train = pd.concat([df_train, minority_upsampled])
df_train = df_train.reset_index(drop = True)
y_train = df_train.loc[:,'Class'].values
X_train = df_train.iloc[:,1:].values

df_test = df[-100:]
X_test_final = df_test.iloc[:,1:].values
```

[36]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

Xx = df_train.iloc[:,1:]
yy = df_train.loc[:,'Class']

bestfeatures = SelectKBest(score_func = chi2, k = 10)
fit = bestfeatures.fit(Xx, yy)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(Xx.columns)

featureScores = pd.concat([dfcolumns, dfscores], axis = 1)
featureScores.columns = ['Specs', 'Score']
featureScores = featureScores.set_index('Specs').sort_values(by = ['Score'],␣
 ↪axis = 0, ascending = True)
#print(featureScores.nlargest(35,'Score'))%xdel

featureScores.iplot(kind='barh', title = 'Univariate Feature Importance (K-Best␣
 ↪from Chi^2)')
```

[37]:
```
feat_list_1 = featureScores[-15:].index
```

[38]:
```
from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier(n_estimators = 100)
model.fit(Xx,yy)
#print(model.feature_importances_)
feat_importances = pd.Series(model.feature_importances_, index=Xx.columns)
feat_importances = feat_importances.sort_values()

feat_importances.iplot(kind='barh', title = 'Tree-Based Feature Importance')
```

[39]:
```
feat_list_2 = feat_importances[-11:].index
```

[40]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.feature_selection import SelectFromModel


# lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(Xx, yy)
# model = SelectFromModel(lsvc, prefit=True)
# feat_list_3 = Xx.iloc[:,list(model.get_support(indices=True))].columns

lr = LogisticRegression(C = 0.000000001, penalty = 'l2', dual = False, solver =␣
↪'lbfgs').fit(Xx, yy)
model = SelectFromModel(lr, prefit=True)
feat_list_3 = Xx.iloc[:,list(model.get_support(indices=True))].columns
#feat_list_3
```

[41]:
```
# common within all three methods - probably can be considered rather important␣
↪as a result
# list(set(feat_list_1) & set(feat_list_2) & set(feat_list_3))
```

[42]:
```
# every single attribute seen across all three methods
important_cols = list(set(list(feat_list_1) + list(feat_list_2) +␣
↪list(feat_list_3)))

X_train = Xx.loc[:,important_cols].values
X_test = df_test.loc[:,important_cols].values
```

[43]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score

X_test_final = df_test.loc[:,important_cols].values
df_test.loc[:,important_cols].to_csv('final_100_set.csv', index = False)
X = df_train.loc[:,important_cols]
y = df_train.loc[:,'Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=6346)
```

[44]:
```
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=6346)
# X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,␣
↪test_size=0.2, random_state=4653)
```

[45]:
```
# pd.concat([y_val, X_val], axis=1).to_csv('val.csv', index = True)
# pd.concat([y_test, X_test], axis=1).to_csv('test.csv', index = True)
# pd.concat([y_train, X_train], axis=1).to_csv('train.csv', index = True)
```

[46]:
```
# import numpy as np
```

```
# def pandas2arff(df,filename,wekaname =␣
↪"pandasdata",cleanstringdata=True,cleannan=True):
#     """
#     converts the pandas dataframe to a weka compatible file
#     df: dataframe in pandas format
#     filename: the filename you want the weka compatible file to be in
#     wekaname: the name you want to give to the weka dataset (this will be␣
↪visible to you when you open it in Weka)
#     cleanstringdata: clean up data which may have spaces and replace with␣
↪"_", special characters etc which seem to annoy Weka.
#                      To suppress this, set this to False
#     cleannan: replaces all nan values with "?" which is Weka's standard for␣
↪missing values.
#               To suppress this, set this to False
#     """
#     import re

#     def cleanstring(s):
#         if s!="?":
#             return re.sub('[^A-Za-z0-9]+', "_", str(s))
#         else:
#             return "?"

#     dfcopy = df #all cleaning operations get done on this copy


#     if cleannan!=False:
#         dfcopy = dfcopy.fillna(-999999999) #this is so that we can swap this␣
↪out for "?"
#         #this makes sure that certain numerical columns with missing values␣
↪don't get stuck with "object" type

#     f = open(filename,"w")
#     arffList = []
#     arffList.append("@relation " + wekaname + "\n")
#     #look at each column's dtype. If it's an "object", make it "nominal"␣
↪under Weka for now (can be changed in source for dates.. etc)
#     for i in range(df.shape[1]):
#         if dfcopy.dtypes[i]=='O' or (df.columns[i] in␣
↪["Class","CLASS","class"]):
#             if cleannan!=False:
#                 dfcopy.iloc[:,i] = dfcopy.iloc[:,i].
↪replace(to_replace=-999999999, value="?")
#             if cleanstringdata!=False:
#                 dfcopy.iloc[:,i] = dfcopy.iloc[:,i].apply(cleanstring)
```

```
#             _uniqueNominalVals = [str(_i) for _i in np.unique(dfcopy.iloc[:
↪,i])]
#             _uniqueNominalVals = ",".join(_uniqueNominalVals)
#             _uniqueNominalVals = _uniqueNominalVals.replace("[","")
#             _uniqueNominalVals = _uniqueNominalVals.replace("]","")
#             _uniqueValuesString = "{" + _uniqueNominalVals +"}"
#             arffList.append("@attribute " + df.columns[i] +␣
↪_uniqueValuesString + "\n")
#         else:
#             arffList.append("@attribute " + df.columns[i] + " real\n")
#             #even if it is an integer, let's just deal with it as a real␣
↪number for now
#     arffList.append("@data\n")
#     for i in range(dfcopy.shape[0]):#instances
#         _instanceString = ""
#         for j in range(df.shape[1]):#features
#                 if dfcopy.dtypes[j]=='O':
#                     _instanceString+="\"" + str(dfcopy.iloc[i,j]) + "\""
#                 else:
#                     _instanceString+=str(dfcopy.iloc[i,j])
#                 if j!=dfcopy.shape[1]-1:#if it's not the last feature, add a␣
↪comma
#                     _instanceString+=","
#         _instanceString+="\n"
#         if cleannan!=False:
#             _instanceString = _instanceString.replace("-999999999.0","?")␣
↪#for numeric missing values
#             _instanceString = _instanceString.replace("\"?\"","?") #for␣
↪categorical missing values
#         arffList.append(_instanceString)
#     f.writelines(arffList)
#     f.close()
#     del dfcopy
#     return True
```

[47]:
```
# pandas2arff(pd.concat([y_val, X_val], axis=1), 'val.arff', wekaname="val",␣
↪cleanstringdata=False, cleannan=False)
# pandas2arff(pd.concat([y_test, X_test], axis=1), 'test.arff',␣
↪wekaname="test", cleanstringdata=False, cleannan=False)
# pandas2arff(pd.concat([y_train, X_train], axis=1), 'train.arff',␣
↪wekaname="train", cleanstringdata=False, cleannan=False)
```

[48]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict

from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn import svm
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier


results = pd.DataFrame()
final_proportions = pd.DataFrame()

all_models = dict([
    ('kmeans', KMeans(n_clusters = 2)),
    ('rForest', RandomForestClassifier(n_estimators=100, max_depth=2)),
    ('logReg', LogisticRegression(solver = 'lbfgs', max_iter = 2500)),
    ('svm', svm.SVC(gamma='scale', decision_function_shape='ovo')),
    ('nBayes', MultinomialNB(alpha=2.5)),
    ('adaBoost', AdaBoostClassifier(n_estimators=100)),
    ('knearest', KNeighborsClassifier(n_neighbors=73, p=1, weights="distance"))
])

for i in all_models.keys():
    results[i + '_acc'] = cross_val_score(all_models[i], X_train, y_train,␣
 ↪cv=10, scoring = 'accuracy')
    results[i + '_f1'] = cross_val_score(all_models[i], X_train, y_train,␣
 ↪cv=10, scoring = 'f1')

    all_models[i].fit(X_train, y_train)
    pred_final = all_models[i].predict(X_test_final)
    zeros = np.bincount(pred_final.astype(int))[0]
    ones = np.bincount(pred_final.astype(int))[1]

    final_proportions[i] = [zeros, ones, (abs(ones - zeros) / 100), (1 -␣
 ↪(abs(ones - zeros) / 100))]
```

[49]:
```
# from sklearn.pipeline import Pipeline
# from sklearn.model_selection import GridSearchCV

# pipe = Pipeline([('classifier' , svm.SVC())])

# param_grid = [
# #     {'classifier' : [LogisticRegression()],
# #      'classifier__penalty' : ['l1', 'l2'],
# #     'classifier__C' : np.logspace(-4, 4, 20),
# #     'classifier__solver' : ['liblinear']},
# #     {'classifier' : [RandomForestClassifier()],
# #     'classifier__n_estimators' : list(range(10,101,10)),
# #     'classifier__max_features' : list(range(6,32,5))},
```

```
# #     {'classifier' : [MultinomialNB()],
# #     'classifier__alpha' : [0.2,0.5,0.6,1,1.5,2.
↪5,5,10,20,50,75,90,130,250,500]},
#     {'classifier' : [svm.SVC()],
#     'classifier__kernel' : ['linear', 'poly', 'rbf', 'sigmoid'],
#     'classifier__C' : [1,2,5,10,100,50,90],
#     'classifier__gamma' : ['scale', 1, 2, 3, 5, 0.5, 100, 25000],
#     'classifier__decision_function_shape' : ['ovr', 'ovo']}

# ]

# clf = GridSearchCV(pipe, param_grid = param_grid, cv = 5, verbose=True,␣
↪n_jobs=-1)



# best_clf = clf.fit(X_train, y_train)

# best_clf.best_estimator_
# best_clf.best_params_
```

[50]:
```
results.iplot(kind = 'box',
             colors = ['#172144', '#172144', '#306E46', '#306E46', '#5D9E39',␣
↪'#5D9E39',
                       '#6E0D09', '#6E0D09', '#1D3168', '#1D3168', '#286263',␣
↪'#286263',
                       '#DB7C26', '#DB7C26'],
              yrange=[0,1],
              title = 'Classification Comparison',
             layout_update=dict([('yaxis', {'nticks':11}),
                                 ('margin', {'b':100})]))
#                , asPlot = True) ## uncomment (and remove bracket) to display␣
↪in browser
```

[51]:
```
final_proportions.index = ['n_zeros', 'n_ones', 'difference', 'correct']
final_proportions
```

[51]:

	kmeans	rForest	logReg	svm	nBayes	adaBoost	knearest
n_zeros	54.00	52.00	49.00	45.0	47.00	42.00	51.00
n_ones	46.00	48.00	51.00	55.0	53.00	58.00	49.00
difference	0.08	0.04	0.02	0.1	0.06	0.16	0.02
correct	0.92	0.96	0.98	0.9	0.94	0.84	0.98

[52]:
```
df_predictions = pd.DataFrame()
df_predictions['ID'] = [i for i in range(1001, 1101)]

all_models['knearest'].fit(X_train, y_train)
df_predictions['Predict 1'] = all_models['knearest'].predict(X_test_final)
```

```python
all_models['logReg'].fit(X_train, y_train)
df_predictions['Predict 2'] = all_models['logReg'].predict(X_test_final)

df_predictions = df_predictions.astype(int)
```

[53]:
```python
#df_predictions.to_csv('predict_actual.csv', index = False)
```

[54]:
```python
df_predictions[0:10]
```

[54]:

	ID	Predict 1	Predict 2
0	1001	0	0
1	1002	0	0
2	1003	0	0
3	1004	0	0
4	1005	0	0
5	1006	1	1
6	1007	1	1
7	1008	0	0
8	1009	1	1
9	1010	0	0