Final_Source

June 3, 2021

[171]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import matplotlib.cm as cm
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,␣
 ↪QuadraticDiscriminantAnalysis
from scipy.stats import zscore

from sklearn.metrics import confusion_matrix, accuracy_score #Must remove later

#NOTE: Many of the initial pre-processing is gotten from: https://www.kaggle.
 ↪com/julenn/titanic
```

1 Preprocessing

[172]:
```
# Load training dataset

train = pd.read_csv("./train.csv", index_col=0)
train.describe() #shows some data is missing based on count (Although only␣
 ↪shows numerical data)
# train.head()
```

[172]:

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

[173]:
```
#View Cabin since it contains some missing values and is related to other␣
 ↪columns
train_cabin_null = train[['Cabin']].isnull().any(axis=1)
```

```
print("No. of unique Cabin values: " + str(len(train['Cabin'].unique())))
print("No. of null Cabin values: " + str(train[train_cabin_null].shape[0]))

#drop cabin since it's unique to person/family and its high null value. Drip␣
↪ticket too since it's useless.
#We also drop the name which basically corresponds to passengerID
train.drop('Cabin', axis=1, inplace=True)
train.drop('Ticket', axis=1, inplace=True)
train.drop('Name', axis=1, inplace=True)
train.head()
```

```
No. of unique Cabin values: 148
No. of null Cabin values: 687
```

[173]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
PassengerId								
1	0	3	male	22.0	1	0	7.2500	S
2	1	1	female	38.0	1	0	71.2833	C
3	1	3	female	26.0	0	0	7.9250	S
4	1	1	female	35.0	1	0	53.1000	S
5	0	3	male	35.0	0	0	8.0500	S

[174]:

```
#We need to change some of the class types
CATEGORICAL_COLUMNS = ('Pclass', 'Sex', 'Embarked')

for column in CATEGORICAL_COLUMNS:
    train[column] = train[column].astype('category')


#Show missing values
train_null = train.isnull().any(axis=1)
print("There are",len(train[train_null]),"missing entries of data") #We have␣
↪entries with at least some missing values
train[train_null]
```

```
There are 179 missing entries of data
```

[174]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
PassengerId								
6	0	3	male	NaN	0	0	8.4583	Q
18	1	2	male	NaN	0	0	13.0000	S
20	1	3	female	NaN	0	0	7.2250	C
27	0	3	male	NaN	0	0	7.2250	C
29	1	3	female	NaN	0	0	7.8792	Q
…	…	…	…	…	…	…	…	…
860	0	3	male	NaN	0	0	7.2292	C
864	0	3	female	NaN	8	2	69.5500	S

```
869                0     3    male  NaN      0      0    9.5000        S
879                0     3    male  NaN      0      0    7.8958        S
889                0     3  female  NaN      1      2   23.4500        S

[179 rows x 8 columns]
```

[175]:
```
#Fill missing age values
train['Age'] = train.groupby('Pclass')['Age'].apply(lambda x: x.fillna(x.
 ↪mean()))
train_sibsp_5_mean = train[train['SibSp'] == 5]['Age'].mean()
train[['Age']] = train[['Age']].fillna(value=train_sibsp_5_mean)

print("There are",len(train[train.isnull().any(axis=1)]),"missing entries of␣
 ↪data left to fill")
# train[train.isnull().any(axis=1)]
```

```
There are 2 missing entries of data left to fill
```

[176]:
```
#Fill missing embarked value and also scaling other features
#creating a binary feature to idenfity passengers that have or don't have␣
 ↪siblings/spouse.
train['Embarked'].fillna(train['Embarked'].mode()[0], inplace=True)
train['Age_z'] = zscore(train['Age'])
train['Age_Fare'] = train['Age'] * train['Fare']
train['SibSp_true'] = 0
train.loc[train['SibSp'] > 0, 'SibSp_true'] = 1
```

[177]:
```
#Create dummy values for categorical predictors
DUMMIES_TO_DROP = ['Embarked_C', 'Embarked_Q', 'Sex_female', 'Pclass_2']

def get_dummies(df):
    df = pd.get_dummies(df)

    return df.drop(DUMMIES_TO_DROP, axis=1)


train_df = get_dummies(train)
train_df
```

[177]:
```
             Survived       Age  SibSp  Parch     Fare     Age_z     Age_Fare  \
PassengerId
1                   0  22.00000      1      0   7.2500 -0.552360   159.500000
2                   1  38.00000      1      0  71.2833  0.659475  2708.765400
3                   1  26.00000      0      0   7.9250 -0.249401   206.050000
4                   1  35.00000      1      0  53.1000  0.432256  1858.500000
5                   0  35.00000      0      0   8.0500  0.432256   281.750000
…                 …       …     …      …     …        …          …
```

```
887                    0  27.00000       0       0  13.0000 -0.173662   351.000000
888                    1  19.00000       0       0  30.0000 -0.779579   570.000000
889                    0  25.14062       1       2  23.4500 -0.314491   589.547532
890                    1  26.00000       0       0  30.0000 -0.249401   780.000000
891                    0  32.00000       0       0   7.7500  0.205037   248.000000

             SibSp_true  Pclass_1  Pclass_3  Sex_male  Embarked_S
PassengerId
1                     1         0         1         1           1
2                     1         1         0         0           0
3                     0         0         1         0           1
4                     1         1         0         0           1
5                     0         0         1         1           1
…                   …       …       …       …         …
887                   0         0         0         1           1
888                   0         1         0         0           1
889                   1         0         1         0           1
890                   0         1         0         1           0
891                   0         0         1         1           0

[891 rows x 12 columns]
```

2 Building our models

3 LDA & QDA

[178]:
```python
#Split the data

TRAIN_DATA_RATIO = 0.7
TARGET = 'Survived'

train_subset_df, test_subset_df = train_test_split(train_df, train_size=TRAIN_DATA_RATIO, random_state=42)
X_train_df, y_train_df = train_subset_df.loc[:, train_subset_df.columns != TARGET], train_subset_df[[TARGET]]
X_test_df, y_test_df = test_subset_df.loc[:, test_subset_df.columns != TARGET], test_subset_df[[TARGET]]

X_train_array, y_train_array = X_train_df.to_numpy(), y_train_df.to_numpy().ravel()
X_test_array, y_test_array = X_test_df.to_numpy(), y_test_df.to_numpy().ravel()
```

[282]:
```python
# let's standardize our combined train and test data
comb_train = np.concatenate((X_train_array, X_test_array), axis=0)
mean = comb_train.mean(axis=0)
std = np.std(comb_train, axis=0, ddof=1)
```

```
X_train_array = (X_train_array - mean)/std
X_test_array = (X_test_array - mean)/std

# Using the skLearn LDA & QDA
lda = LinearDiscriminantAnalysis()
lda.fit(X_train_array, y_train_array)
y_predicted = lda.predict(X_test_array)

print(f"Accuracy for LDA skLearn: {accuracy_score(y_test_array, y_predicted)}")

predictors = set(X_train_df.columns) - {'Age_z'}
X_train_array_qda = X_train_df.loc[:, predictors].to_numpy()
X_test_array_qda = X_test_df.loc[:, predictors].to_numpy()

#Note that the predictors were amended because the Age_z variable was collinear␣
↪with the Age variable
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train_array_qda, y_train_array)
y_predicted_2 = qda.predict(X_test_array_qda)

print(f"Accuracy for QDA skLearn: {accuracy_score(y_test_array,␣
↪y_predicted_2)}")
```

```
Accuracy for LDA skLearn: 0.7985074626865671
Accuracy for QDA skLearn: 0.8171641791044776
```

[283]:
```
def qdaOrlda(QDA = True):

    y_train_reshape = y_train_array.reshape(1,y_train_array.shape[0])

    if QDA:
        all_ = np.concatenate((X_train_array_qda, y_train_reshape.T), axis=1)
    else:
        all_ = np.concatenate((X_train_array, y_train_reshape.T), axis=1)

    train_survive = all_[all_[:,-1] == 1]
    train_dead = all_[all_[:,-1] == 0]

    y_train_survive = train_survive[:,-1]
    y_train_dead = train_dead[:,-1]

    X_train_survive = np.delete(train_survive, -1, axis=1)
    X_train_dead = np.delete(train_dead, -1, axis=1)


    mean_survive = X_train_survive.mean(axis=0)
    mean_dead = X_train_dead.mean(axis=0)
```

```
    prob_survive = len(y_train_survive)/(len(y_train_survive) + len(y_train_dead))
    prob_dead = len(y_train_dead)/(len(y_train_survive) + len(y_train_dead))

    cov_matrix_survive = (X_train_survive.T @ X_train_survive)/(len(X_train_survive)-1)
    cov_matrix_dead = (X_train_dead.T @ X_train_dead)/(len(X_train_dead)-1)




    #Note that I use the diagonal as the Mahalanobis Distance between each row of x and the mean
    small_random_number = np.finfo(float).eps
    if QDA:
        left_survive = (-1/2 * (np.log(np.linalg.det(cov_matrix_survive) + small_random_number)))
        left_term = (X_test_array_qda - mean_survive) @ np.linalg.inv(cov_matrix_survive)
        mahal = left_term @ (X_test_array_qda - mean_survive).T
        final_mahal = mahal.diagonal()
        right_survive = (-1/2 * (final_mahal)) + np.log(prob_survive)

        left_dead = (-(1/2) * (np.log(np.linalg.det(cov_matrix_dead) + small_random_number)))
        left_term2 = (X_test_array_qda - mean_dead) @ np.linalg.inv(cov_matrix_dead)
        mahal2 = left_term2 @ (X_test_array_qda - mean_dead).T
        final_mahal2 = mahal2.diagonal()
        right_dead = (-1/2 * (final_mahal2)) + np.log(prob_dead)

    else:
        avg_cov_matrix = (cov_matrix_survive + cov_matrix_dead)/2

        right_survive = (-1/2 * (mean_survive.T @ np.linalg.inv(avg_cov_matrix) @ mean_survive)) + np.log(prob_survive)
        left_survive = X_test_array @ np.linalg.inv(avg_cov_matrix) @ mean_survive

        right_dead = (-1/2 * (mean_dead.T @ np.linalg.inv(avg_cov_matrix) @ mean_dead)) + np.log(prob_dead)
        left_dead = X_test_array @ np.linalg.inv(avg_cov_matrix) @ mean_dead
```

```
    survive = left_survive + right_survive
    dead = left_dead + right_dead


    TP=0
    FP=0
    TN=0
    FN=0

    y_pred = []
    for i in range(len(dead)):
        survived = survive[i] ##
        died = dead[i]  ##
        if survived >= died:
            y_pred.append(1)
            if y_test_array[i] == 1:
                TP = TP + 1
            else:
                FP = FP + 1
        else:
            y_pred.append(0)
            if y_test_array[i] == 0:
                TN = TN + 1
            else:
                FN = FN + 1

    Precision = TP/(TP + FP)
    Recall = TP/(TP + FN)
    F_measure = (2 * Precision * Recall)/(Precision + Recall)
    Accuracy = (TP + TN)/(TP + TN + FP + FN)


    print("Precision = %.4f " %(Precision))
    print("Recall = %.4f " %(Recall))
    print("F_measure = %.4f " %(F_measure))
#     print("Accuracy = %.4f " %(Accuracy))
    if QDA:
        print(f"Accuracy for QDA Mine: {accuracy_score(y_test_array, y_pred)}")
    else:
        print(f"Accuracy for LDA Mine: {accuracy_score(y_test_array, y_pred)}")

qdaOrlda(True)
```

```
Precision = 0.7358
Recall = 0.7027
F_measure = 0.7189
Accuracy for QDA Mine: 0.7723880597014925
```