GE 461 INTRODUCTION TO DATA SCIENCE
Project W13: Telehealth - Fall Detection

Yagiz Yaman

Contents

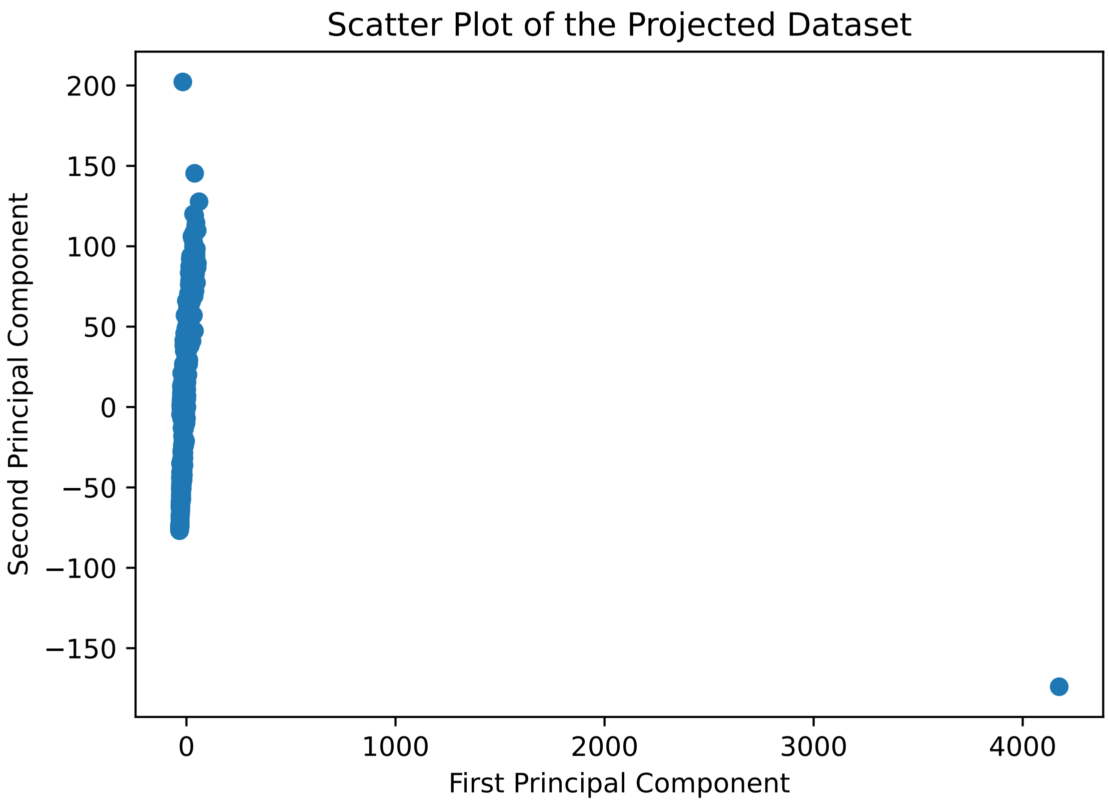
PART A [40 points]

PCA

0.84% of the variance is captured by the first two principals.

The dataset is projected onto the first two principal components.

The scatter plot of the projected dataset is as follows,



K-means clustering

Now, k-means clustering is going to be applied to separate data into clusters.

```
##    nclusters  accuracy
## 0          2      0.55
## 1          3      0.20
## 2          4      0.08
## 3          5      0.08
## 4          6      0.13
## 5          7      0.03
## 6          8      0.02
## 7          9      0.10
## 8         10      0.10
```

When we set F to 0 and NF to 1, the best accuracy is achieved when the number of clusters is set to 2 with 55% accuracy. When we examine the clusters it generates,

```
## Number of observations labeled as 0 is 565 &
##  number of observations labeled as 1 is 1.
```

So, the result of k-means clustering does not seem logical. It assigns all of the observations to one cluster and only one observation to other cluster. However, when the outlier observation is removed from the dataset, it is expected to get a better result.

```
## The accuracy of the kmeans clustering with 2 clusters become 0.78 .
```

So, when the outlier is removed, the accuracy of kmeans clustering is improved. However, without any changes, the kmeans clustering is not able to cluster the dataset in a meaningful way.

Part B [60 points]

SVM

Now, an SVM classifier is going to be constructed with different parameters. Once parameters are decided on the validation set, the accuracy will be obtained on test set.

The train, validation and test are partitioned in 60-20-20 shares.

Different settings of hyperparameters are performed on the training dataset.

```
##    param_C param_degree param_kernel  mean_test_score
## 1   0.001            3          rbf         0.554565
## 5    0.01            4          rbf         0.554565
## 2     100            4      sigmoid         0.770018
## 3      10            4      sigmoid         0.781782
## 9     100            4         poly         0.982309
## 8     100            2         poly         0.991089
## 6       1            2          rbf         0.994030
## 7     100            2          rbf         0.994030
## 0      10            3       linear         0.997015
## 4    1000            3       linear         0.997015
```

Now, the settings with high test scores are going to be performed on validation set.

```
##    param_C param_degree param_kernel  validation_score
## 0     100            2         poly          0.973451
## 1       1            2          rbf          0.991150
## 2     100            2          rbf          0.973451
## 3      10            3       linear          0.982301
## 4    1000            3       linear          0.982301
```

The best performing parameters on validation set are C = 1, degree = 3 and kernel = rbf. Now, the accuracy on the test set is going to be calculated with the best resulting parameters obtained from the validation set.

Finally, the accuracy of the SVM on test set is 98.25%.

MLP

Now, an MLP classifier is going to be constructed with different parameters. Once parameters are decided on the validation set, the accuracy will be obtained on test set.

Different settings of hyperparameters are performed on the training dataset.

```
##    param_solver param_learning_rate  ... param_activation mean_test_score
## 9           sgd          invscaling  ...         identity        0.861370
## 5           sgd          invscaling  ...             relu        0.920281
## 0           sgd            adaptive  ...             tanh        0.988191
## 3          adam            adaptive  ...             tanh        0.991089
## 7         lbfgs          invscaling  ...         identity        0.991089
## 1         lbfgs            adaptive  ...             tanh        0.994030
## 2           sgd            constant  ...         identity        0.994030
## 4         lbfgs          invscaling  ...             tanh        0.994030
## 8         lbfgs            constant  ...             relu        0.994030
## 6          adam          invscaling  ...             tanh        0.994074
##
## [10 rows x 6 columns]
```

Now, the settings with high test scores are going to be performed on validation set.

```
##   param_solver param_learning_rate  ... param_activation validation_score
## 0          sgd            adaptive  ...             tanh         0.991150
## 1         adam            adaptive  ...             tanh         0.982301
## 2        lbfgs          invscaling  ...         identity         0.982301
## 3        lbfgs            adaptive  ...             tanh         0.982301
## 4          sgd            constant  ...         identity         0.982301
## 5        lbfgs          invscaling  ...             tanh         0.982301
## 6        lbfgs            constant  ...             relu         0.973451
## 7         adam          invscaling  ...             tanh         0.991150
##
## [8 rows x 6 columns]
```

The best performing parameters on validation set are solver = "adam", learning_rate = invscaling, hidden layer size = 100, alpha = 0.1, activation = identity. Now, the accuracy on the test set is going to be calculated with the best resulting parameters obtained from the validation set.

Finally, the accuracy of the SVM on test set is 94.73%.

Conclusion

Apparently, supervised learning algorithms perform better than the unsupervised learning algorithm. The results of SVM and MLP are pretty similar to each other. Overall, it can be concluded that the classification task on fall detection could be done with supervised learning algorithms. However, if unsupervised learning algorithms are going to be implemented, a prior data analysis step should be made to catch possible outliers.

Appendix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
import warnings
warnings.filterwarnings('ignore')
np.set_printoptions(suppress=True)

# Import the data
# This code is reading a CSV file named "falldetection_dataset.csv"
#located in the specified file path
# and assigning it to the variable `train_data`.
train_data = pd.read_csv("path_to_dataset.csv", header=None)

labels = train_data[[0,1]]
train_data = train_data.drop([0, 1], axis = 1)

# `scaled_data = np.reshape(train_data, (566, 306))` is reshaping the
```

```python
#`train_data` array from a data frame to (566, 306) numpy array.
shaped_data = np.reshape(train_data, (566, 306))

# PART A [40 points]

# This code is performing Principal Component Analysis (PCA) on the
#`scaled_data` array, which is a numpy array of shape (566, 306) and then gives
#a plot of captured variances. Then, it projects the dataset onto principal
#components.

plt.clf()
pca = PCA(n_components=2)
pca.fit(shaped_data)

captured_variance = pca.explained_variance_ratio_
names = ["First Principal", "Second Principal"]
fig, ax = plt.subplots()
ax.bar(names, captured_variance)

ax.set_title('Scree Plot')
ax.set_xlabel('Principal Components')
ax.set_ylabel('Captured Variance')

plt.show()

projected_data = pca.transform(shaped_data)

# The scatter plot of the projected dataset is as follows,

x = projected_data[:, 0]
y = projected_data[:, 1]

fig, ax = plt.subplots()
ax.scatter(x, y)

ax.set_title('Scatter Plot of the Projected Dataset')
ax.set_xlabel('First Principal Component')
ax.set_ylabel('Second Principal Component')

plt.show()

# Now, k-means clustering is going to be applied to separate data into clusters.

labels.loc[labels[1] == "F", "binary"] = 0
labels.loc[labels[1] == "NF", "binary"] = 1

def get_accuracy(nclusters, labels, projected):
    accuracy = []
    for i in nclusters:
        kmeans = KMeans(n_clusters=i, random_state=15).fit(projected)
        cluster_labels = kmeans.labels_
        accuracy.append(round(np.sum(cluster_labels ==
        labels["binary"]) / len(cluster_labels),2))
```

```
    results = pd.DataFrame({"nclusters": nclusters, "accuracy": accuracy})
    return results

nclusters = [2, 3, 4, 5, 6, 7, 8, 9, 10]
results = get_accuracy(nclusters, labels, projected_data)

# When we set F to 0 and NF to 1, the best accuracy is achieved when the number
#of clusters is set to 2 with 55% accuracy. When we examine the clusters it
#generates,

kmeans = KMeans(n_clusters=2, random_state=15).fit(projected_data)
cluster_labels = kmeans.labels_

zeros = len(cluster_labels[cluster_labels == 0])
ones = len(cluster_labels[cluster_labels == 1])

print("Number of observations labeled as 0 is " + str(zeros)+ " & \n number of observations labeled as 

place_outlier = np.where(projected_data[:,0] == max(projected_data[:, 0]))
no_labels = labels.drop(int(place_outlier[0][0]))
no_outlier = projected_data[projected_data[:, 0] <= 2000]

kmeans = KMeans(n_clusters=2, random_state=15).fit(no_outlier)
cluster_labels = kmeans.labels_

print("The accuracy of the kmeans clustering with 2 clusters become " +
str(round(np.sum(cluster_labels == no_labels["binary"]) / len(cluster_labels)
,2)), ".")

# Part B [60 points]

# SVM

# Now, an SVM classifier is going to be constructed with different parameters.
# Once parameters are decided on the validation set, the accuracy will be obtained on test set.

X= train_data.values
y= labels[1] #target column's values attribute

len_x = len(X)

X_train = X[:int(len_x*60/100)]
y_train = y[:int(len_x*60/100)]

X_valid = X[int(len_x*60/100):int(len_x*80/100)]
y_valid = y[int(len_x*60/100):int(len_x*80/100)]

X_test = X[int(len_x*80/100):]
y_test = y[int(len_x*80/100):]


# Define a parameter grid with distributions of possible parameters to use
svc_param_grid = {
```

```
"C":[0.001, 0.01, 0.1, 1, 10, 100, 1000],
"degree": [2,3,4],
"kernel": ["linear", "poly", "rbf", "sigmoid"]
}


# Create a SVC classifier
svm = SVC(random_state=5)

# Instantiate RandomizedSearchCV() with rf and the parameter grid
svm_rs = RandomizedSearchCV(
    estimator=svm,
    param_distributions=svc_param_grid,
    n_iter=10,
    cv=5,
    scoring="accuracy"
)

# Train the model on the training set
svm_rs.fit(X_train, y_train)

# Different settings of hyperparameters are performed on the training dataset.

hyper_train_svm = pd.DataFrame(svm_rs.cv_results_)[["param_C", "param_degree",
"param_kernel", "mean_test_score"]].sort_values(by="mean_test_score")
hyper_train_svm

# Now, the settings with high test scores are going to be performed on
# validation set.

my_dt = hyper_train_svm[5:].reset_index()
my_dt = my_dt.drop("index", axis=1)
my_dt["validation_score"] = 0
my_dt.iloc[1,2]
accuracy = []
for i in range(0,5):
  params = {
  "C": my_dt.iloc[i][0],
  "degree": my_dt.iloc[i][1],
  "kernel": my_dt.iloc[i][2],
  }

  svm = SVC(**params)
  svm.fit(X_train, y_train)

  # Test the model on the test set
  y_pred = svm.predict(X_valid)

  # Calculate the accuracy of the classifier
  accuracy = accuracy_score(y_valid, y_pred)

  my_dt.loc[i, "validation_score"] = accuracy
```

```
my_dt[["param_C", "param_degree", "param_kernel", "validation_score"]]

# The best performing parameters on validation set are C = 1, degree = 3 and
# kernel = rbf. Now, the accuracy on the test set is going to be calculated.

params = {
"C": 1,
"degree": 3,
"kernel": "rbf"
}

svm = SVC(**params)
svm.fit(X_train, y_train)


# Test the model on the test set
y_pred = svm.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)

# MLP

# Now, an MLP classifier is going to be constructed with different parameters.

# Create an MLP classifier with random parameters
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
solver='adam', alpha=0.0001, learning_rate='constant', random_state=42)

# Define a parameter grid with distributions of possible parameters to use
mlp_param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (200,)],
    'activation': ['identity', 'logistic', 'tanh', 'relu'],
    'solver': ['lbfgs', 'sgd', 'adam'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'invscaling', 'adaptive']
}

mlp_rs = RandomizedSearchCV(
    estimator=mlp,
    param_distributions=mlp_param_grid,
    n_iter=10,
    cv=5,
    scoring="accuracy"
)

# Train the model on the training set
mlp_rs.fit(X_train, y_train)

# Different settings of hyperparameters are performed on the training dataset.

hyper_train_mlp = pd.DataFrame(mlp_rs.cv_results_)[["param_solver",
"param_learning_rate", "param_hidden_layer_sizes", "param_alpha",
```

```python
"param_activation", "mean_test_score"]].sort_values(by="mean_test_score")
hyper_train_mlp

# Now, the settings with high test scores are going to be performed on
#validation set.

my_dt = hyper_train_mlp[2:].reset_index()
my_dt = my_dt.drop("index", axis=1)
my_dt["validation_score"] = 0

for i in range(0,8):
  params = {
    'hidden_layer_sizes': my_dt.iloc[i][2],
    'activation': my_dt.iloc[i][4],
    'solver': my_dt.iloc[i][0],
    'alpha': my_dt.iloc[i][3],
    'learning_rate': my_dt.iloc[i][1]
  }

  mlp = MLPClassifier(**params)
  mlp.fit(X_train, y_train)

  # Test the model on the test set
  y_pred = mlp.predict(X_valid)

  # Calculate the accuracy of the classifier
  accuracy = accuracy_score(y_valid, y_pred)

  my_dt.loc[i, "validation_score"] = accuracy

# The best performing parameters on validation set are solver = "adam",
#learning_rate = invscaling, hidden layer size = 100, alpha = 0.1, activation =
#identity. Now, the accuracy on the test set is going to be calculated.

params = {
  'hidden_layer_sizes': (100,),
  'activation': "identity",
  'solver': "adam",
  'alpha': 0.01,
  'learning_rate': "invscaling"
}

mlp = MLPClassifier(**params)
mlp.fit(X_train, y_train)


# Test the model on the test set
y_pred = mlp.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("The code is run without errors")
```