|  |  |
| --- | --- |
| **unsupervised\_learning-1.py ---- This example shows the clustering of a 2D dataset (2 features)** | |
| # visualize training data  training\_data.plot.scatter(x = "mean\_challenge\_level", y = "mean\_total\_score", s = 70);  plt.show();  input\_data = training\_data[["mean\_challenge\_level", "mean\_total\_score"]] | For this example, each user is represented by two features/axes. So, it’s easy to visualize each user as a 2D point. |
| model = KMeans(n\_clusters=2).fit(input\_data); | You can add the following code to print the clustering output:  Centers = model.cluster\_centers\_;  Labels = model.labels\_;  print(Centers, Labels);  This will print Centers, as a CxF matric, where C is the number of clusters (**n\_clusters=2**) and F the number of features. Labels will be an array 1xU, where U is the number of users. Check sklearn website for each algorithm/model you use regarding the input and output parameters |
| for i, p in enumerate(numpy.asarray(input\_data)):  plt.plot(p[0], p[1], 'o', markersize=9, color = colors[model.labels\_[i]]);  plt.xlabel("challenge");  plt.ylabel("score");  plt.title("clustered data - inertia = " + str(round(model.inertia\_,2)));  plt.show(); | This visualizes each user colored based on the clustering output.  Inertia is an evaluation metric of the given model (kmeans). Check the sklearn website for the input/output variable of each algorithm you use. |
| # set heights of bars  bars1 = model.cluster\_centers\_[0];  bars2 = model.cluster\_centers\_[1];  tlabels = ["challenge", "score"]  # Set position of bar on X axis  r1 = numpy.arange(len(bars1))  r2 = [x + barWidth for x in r1]  # Make the plot  plt.bar(r1, bars1, color='b', width=barWidth, edgecolor='white', label='cluster1')  plt.bar(r2, bars2, color='r', width=barWidth, edgecolor='white', label='cluster2')  # Add xticks on the middle of the group bars  plt.xlabel('cluster data (centroids)', fontweight='bold')  plt.xticks([r + 0.5\*barWidth for r in range(len(bars1))], tlabels)    # Create legend & Show graphic  plt.legend()  plt.show() | The provided code works for **two clusters**, if you want to visualize the clustering output for N = 3 clusters you can change **n\_clusters = 3**and use the following code for visualization.  *#check the output and their dimensions*  Centers = model.cluster\_centers\_;  Labels = model.labels\_;  print(Centers, Labels);  *# visualization*  bars1 = kmeans.cluster\_centers\_[0];  bars2 = kmeans.cluster\_centers\_[1];  bars3 = kmeans.cluster\_centers\_[2];  # Set position of bar on X axis  r1 = numpy.arange(len(bars1))  r2 = [x + barWidth for x in r1]  r3 = [x + barWidth for x in r2]  # Make the plot  plt.bar(r1, bars1, color='b', width=barWidth, edgecolor='white', label='cluster1')  plt.bar(r2, bars2, color='r', width=barWidth, edgecolor='white', label='cluster2')  #plt.bar(r3, bars3, color='g', width=barWidth, edgecolor='white', label='cluster3') |
| **unsupervised\_learning-2.py ---- clustering of a multidimensional dataset (12 features)** | |
| # visualize training data  training\_data.plot.scatter(x = "mean\_challenge\_level", y = "mean\_total\_score", s = 70);  plt.show();  input\_data = training\_data[["mean\_challenge\_level", "mean\_total\_score"]] | For this example, each user is represented by two features (two axes – dimensions). Therefore, it’s possible to visualize each user as a 2D point to see the distribution of the users (data points). |
| model = KMeans(n\_clusters=2).fit(input\_data); | You can add the following code to print the clustering output:  Centers = model.cluster\_centers\_;  Labels = model.labels\_;  print(Centers, Labels);  This will print Centers, as a CxF matric, where C is the number of clusters and F the number of features. Labels will be an array 1xU, where U is the number of users. Check sklearn website for each algorithm/model you use regarding the input and output parameters |
| for i, p in enumerate(numpy.asarray(input\_data)):  plt.plot(p[0], p[1], 'o', markersize=9, color = colors[model.labels\_[i]]);  plt.xlabel("challenge");  plt.ylabel("score");  plt.title("clustered data - inertia = " + str(round(model.inertia\_,2)));  plt.show(); | This visualizes each user colored based on the clustering output.  Inertia is an evaluation metric of the given model (kmeans). Check the sklearn website for the input/output variable of each algorithm you use. |
| # set heights of bars  bars1 = model.cluster\_centers\_[0];  bars2 = model.cluster\_centers\_[1];  tlabels = ["challenge", "score"]  # Set position of bar on X axis  r1 = numpy.arange(len(bars1))  r2 = [x + barWidth for x in r1]  # Make the plot  plt.bar(r1, bars1, color='b', width=barWidth, edgecolor='white', label='cluster1')  plt.bar(r2, bars2, color='r', width=barWidth, edgecolor='white', label='cluster2')  # Add xticks on the middle of the group bars  plt.xlabel('cluster data (centroids)', fontweight='bold')  plt.xticks([r + 0.5\*barWidth for r in range(len(bars1))], tlabels)    # Create legend & Show graphic  plt.legend()  plt.show() | The provided code works for two clusters. If you want to visualize the clustering output for N = 3 clusters you can change ***n\_clusters = 3***and use the following code  *#check the output and their dimensions*  Centers = model.cluster\_centers\_;  Labels = model.labels\_;  print(Centers, Labels);  *# visualization*  bars1 = kmeans.cluster\_centers\_[0];  bars2 = kmeans.cluster\_centers\_[1];  bars3 = kmeans.cluster\_centers\_[2];  # Set position of bar on X axis  r1 = numpy.arange(len(bars1))  r2 = [x + barWidth for x in r1]  r3 = [x + barWidth for x in r2]  # Make the plot  plt.bar(r1, bars1, color='b', width=barWidth, edgecolor='white', label='cluster1')  plt.bar(r2, bars2, color='r', width=barWidth, edgecolor='white', label='cluster2')  plt.bar(r3, bars3, color='g', width=barWidth, edgecolor='white', label='cluster3')  # Add xticks on the middle of the group bars  plt.xlabel('cluster data (centroids)', fontweight='bold')  plt.xticks([r + 2\*barWidth for r in range(len(bars1))], tlabels)    # Create legend & Show graphic  plt.legend()  plt.show() |
| # multidimensional scaling  M = training\_data - numpy.asarray(training\_data).mean();  similarities = euclidean\_distances(M);  mds = MDS(n\_components=2, max\_iter=1000, random\_state=100, eps=1e-16, dissimilarity="precomputed")  pos = mds.fit(similarities).embedding\_  colors = ["r", "b"];  for i, p in enumerate(pos):  plt.plot(p[0], p[1], 'o', markersize=9, color = colors[model.labels\_[i]]);  plt.show(); | Multidimensional scaling is used to visualize the data and the clustering output – dimensionality reduction. Clustering can also be performed to the 2D data (the result of the MDS)  # kmeans on the “mapped” data (2D) – MDS output: pos  Input\_data = pos;  model2 = KMeans(n\_clusters=2).fit(input\_data);  print(model2.labels\_, model2.cluster\_centers\_);  #plot kmeans on 2D data  colors = ["r", "b"];  for i, p in enumerate(pos):  plt.plot(p[0], p[1], 'o', markersize=9, color = colors[model2.labels\_[i]]);  plt.show(); |

|  |  |
| --- | --- |
| **supervised\_learning-1.py: regression example to estimate a numerical continuous value of a feature (score)** | |
| svr = SVR(kernel='rbf', C=200)  svr.fit(X\_train, Y\_train);  svr\_prediction = svr.predict(X\_test);  svr\_mse = mean\_squared\_error(Y\_test, svr\_prediction)  svr\_rmse1 = numpy.sqrt(svr\_mse)  plt.plot(svr\_prediction, 'b', label = "predicted scores");  plt.plot(Y\_test, 'r', label = "real\_scores");  plt.legend()  plt.show(); | In this code, we train two different models using the same data. The reason to do this is to compare the models and select the best one.  Moreover, you should also experiment with the parameters for each algorithm you use, e.g., C = 200 for SVR, or the activation and solver parameters for the MLPRegressor.  In order to compare regression models, we can estimate the **mean squared error (mse),** and the **root-mean-square error (rmse).**  ***The smaller the error, the better the model.***  These two algorithms are appropriate for the specific problem. You should check the sklearn website to check all available models. |
| mlr = MLPRegressor(activation='logistic', max\_iter=5000, solver='lbfgs', hidden\_layer\_sizes=(3, 8), random\_state=42)  mlr.fit(X\_train, Y\_train)  mlr\_prediction = mlr.predict(X\_test);  mlr\_mse = mean\_squared\_error(Y\_test, mlr\_prediction)  mlr\_rmse1 = numpy.sqrt(mlr\_mse)  plt.plot(mlr\_prediction, 'b', label = "predicted scores");  plt.plot(Y\_test, 'r', label = "real\_scores");  plt.legend()  plt.show(); |

|  |  |
| --- | --- |
| **supervised\_learning-1b.py: classification example to estimate a class (based on score)** | |
| y\_temp = sessions[["gscore"]];  y\_temp.gscore[y\_temp.gscore < 13] = 0  y\_temp.gscore[y\_temp.gscore >= 13] = 1  Y = y\_temp.to\_numpy(); | We create two classes based on the score.  We could also have 3 classes or a different score threshold. |
| svm = SVC(kernel='linear')  svm.fit(X\_train, Y\_train);  svm\_prediction = svm.predict(X\_test);  print("SVM Accuracy:",metrics.accuracy\_score(Y\_test, svm\_prediction))  print(confusion\_matrix(Y\_test, svm\_prediction)); | In this code, we train two different models using the same data. The reason to do this is to compare the models and select the best one.  Moreover, you should also experiment with the parameters for each algorithm you use, e.g., C = 200 for SVR, or the activation and solver parameters for the MLPRegressor.  In order to compare regression models, we can estimate the **mean squared error (mse),** and the **root-mean-square error (rmse).**  ***The larger the accuracy, the better the model.***  Especially for unbalanced classes (much more 1s than 0s or vice versa), the confusion matrix gives better insights, i.e., if there are only 10% instance on class 1 and they are all classified incorrectly as class 0, the accuracy will be 90%, but the confusion matrix shows that the model detects everything as class 0.  Moreover, there are more evaluation metrics to visualize and compare.  These four algorithms are appropriate for the specific problem. You should check the sklearn website to check all available models. |
| # multilayer perceptron classification  mlp = MLPClassifier(activation='logistic', max\_iter=5000, solver='lbfgs', hidden\_layer\_sizes=(2, 5), random\_state=42)  mlp.fit(X\_train, Y\_train)  mlp\_prediction = mlp.predict(X\_test);  print("MLP Accuracy:",metrics.accuracy\_score(Y\_test, mlp\_prediction))  print(confusion\_matrix(Y\_test, mlp\_prediction)); |
| # Decision Tree calssification  clf = tree.DecisionTreeClassifier(random\_state=10);  clf.fit(X\_train, Y\_train)  clf\_prediction = clf.predict(X\_test)  print("DT Accuracy:",metrics.accuracy\_score(Y\_test, clf\_prediction))  print(confusion\_matrix(Y\_test, clf\_prediction)); |
| rf = RandomForestClassifier(n\_estimators=150, criterion = 'entropy')  rf.fit(X\_train, Y\_train)  rf\_prediction = rf.predict(X\_test)  print("RF Accuracy:",metrics.accuracy\_score(Y\_test, rf\_prediction))  print(confusion\_matrix(Y\_test, rf\_prediction)); |