CM146, Fall 2020
Problem Set 4: Kernel, SVM, K-Means, GMM
Due Dec 10, 2020 at 11:59 pm

1 Kernels [30 pts]

One way to construct complex kernels is to build from simple ones. In the following, we will prove a list of properties of kernels (Rule 1 – Rule 5) and then use these rules to show the Gaussian kernel $k(x, z) = \exp(\frac{-\|x-z\|^2}{\sigma^2})$ is a valid kernel.

(a) **(5 pts)** Rule 1: Suppose we have $x \in \mathbb{X}, z \in \mathbb{X}, \quad g : \mathbb{X} \to \mathbb{R}$. Prove that $k(x, z) = g(x) \times g(z)$ is a valid kernel by constructing a feature map $\Phi(\cdot)$ and show that $k(x, z) = \Phi(x)^T \Phi(z)$.
Solution: Suppose we have the feature map of $\Phi(\cdot)$ be a vector with a dimensionality 1 (which is more or less equivalent to mapping a vector to a real number). Then, the product of $k(x, z)$ is a real number, as a product between two vectors with a dimensionality of 1 is basically a product between two real numbers, and maps to a real number as well. Thus k(x, z) is a valid kernel.

(b) **(5 pts)** Rule 2: Suppose we have a valid kernel $k_1(x, z) = \Phi_1(x)^T \Phi_1(z)$. Prove that $k(x, z) = \alpha \cdot k_1(x, z) \ \ \forall \alpha \geq 0$ is also a valid kernel by constructing a new feature map $\Phi(\cdot)$ using $\Phi_1(\cdot)$ and show that $k(x, z) = \Phi(x)^T \Phi(z)$.
Solution: In this case, we can have a new $\Phi(\cdot)$ with a feature map equal to that of $\Phi_1(\cdot)$, except all feature are multiplied by $\sqrt{\alpha}$. In such a case, $\Phi(x) = \sqrt{\alpha} \times \Phi_1(x)$. This would mean that $k(x, z) = \Phi(x)^T \Phi(z) = \Phi_1(x)^T \Phi_1(z) \times \alpha = \alpha \cdot k_1(x, z)$.

(c) **(5 pts)** Rule 3: Suppose we have two valid kernels $k_1(x, z) = \Phi_1(x)^T \Phi_1(z)$ and $k_2(x, z) = \Phi_2(x)^T \Phi_2(z)$. Prove that $k(x, z) = k_1(x, z) + k_2(x, z)$ is also a valid kernel by constructing a new feature map $\Phi(\cdot)$ using $\Phi_1(\cdot)$ and $\Phi_2(\cdot)$ and show that $k(x, z) = \Phi(x)^T \Phi(z)$.
Solution: In this case, we can concatenate the two feature vectors into a new 1D vector such that
$\Phi(\cdot) = \begin{bmatrix} \Phi_1(\cdot) \\ \Phi_2(\cdot) \end{bmatrix}$
When we take the resulting product $k(x, z) = \Phi(x)^T \Phi(z)$, we would be summing up the each pairwise product of each element $\Phi(x)_n \times \Phi(z)_n$, which is equivalent to the original $\Phi_1(x)^T \Phi_1(z) + \Phi_2(x)^T \Phi_2(z)$. Thus, $k(x, z)$ is a valid kernel.

(d) **(5 pts)** Rule 4: Suppose we have two valid kernels $k_1(x, z) = \Phi_1(x)^T \Phi_1(z)$ and $k_2(x, z) = \Phi_2(x)^T \Phi_2(z)$. Prove that $k(x, z) = k_1(x, z) \times k_2(x, z)$ is a valid kernel by constructing a new feature map $\Phi(\cdot)$ using $\Phi_1(\cdot)$ and $\Phi_2(\cdot)$ and show that $k(x, z) = \Phi(x)^T \Phi(z)$.
Solution: In this case, the feature map of $\Phi(\cdot)$ would basically be a vector of $m \times n$ elements, where m is the number of features in $\Phi_1(\cdot)$ and n is the number of features in $\Phi_2(\cdot)$. The elements of $\Phi(\cdot)$ would be $(\Phi_1(\cdot))_a \times (\Phi_2(\cdot))_b$, where $(\Phi_1(\cdot))_a$ is an element in $\Phi_1(\cdot)$ with $0 \leq a < m$ and $(\Phi_2(\cdot))_b$ is an element in $\Phi_2(\cdot)$ with $0 \leq b < n$. Such a feature map would result in a kernel equivalent to $\sum_{i=0}^{m \times n} (\Phi(x))_i (\Phi(z))_i = \sum_{i=0}^{m} \sum_{j=0}^{n} (\Phi_1(x))_i (\Phi_2(x))_j (\Phi_1(z))_i (\Phi_2(z))_j$. This is equivalent to $k_1(x, z) \times k_2(x, z) = \Phi_1(x)^T \Phi_1(z) \times \Phi_2(x)^T \Phi_2(z)$, so $k(x, z) = \Phi(x)^T \Phi(z)$ is a valid kernel.

(e) **(5 pts)** Rule 5: Suppose we have a valid kernel $k_1(x,z) = \Phi_1(x)^T\Phi_1(z)$. Prove that $\exp(k_1(x,z))$ is also a valid kernel by applying Rules 1-4 in problem 1(a)-1(d).
Hint:

$$\exp(k_1(x,z)) = \lim_{i\to\infty} 1 + k_1(x,z) + ... + \frac{k_1^i(x,z)}{i!}$$
$$= 1 + \sum_{i=1}^{i=\infty} \frac{k_1^i(x,z)}{i!},$$

where $k_1^i(x,z)$ is $k_1(x,z)$ to the power of i.
Solution: From what is given, we can express the exponent of $k_1(x,z)$ as a sum of products. From part a, we know that a kernel outputting a constant real number is a valid kernel. so the first term by itself can have a valid kernel. From part d, we know that any product between feature maps of valid kernels can have valid kernels. From part c, we know that multiplying a kernel function by a constant can also have a valid kernel function. Combining parts d and c support that the rest of the terms in the summation can have valid kernels. Finally, from part c, we know that a sum of kernel functions can have a valid kernel function. So putting these all together, we the summation of valid kernel functions can also have a valid kernel function.

(f) **(5 pts)** Prove the Gaussian Kernel $k(x,z) = \exp(\frac{-\|x-z\|^2}{\sigma^2})$ is a valid kernel by applying Rules 1-5 in problem 1(a)-1(e).
Solution: First, let's look at the inside term of the exponential. The term $-\|x-z\|^2$ is basically can be rewritten as $(x-z)^T(x-z)$, a simple kernel function in it own mapping. Dividing this by a constant will still yield a valid kernel function (as from part b), and taking an exponential of it will also yield a valid kernel function (as from part e). Thus, the Gaussian kernel is a valid kernel.

2 SVM [25 pts]

Suppose we have the following six training examples. x_1, x_2, x_3 are positive instances and x_4, x_5, x_6 are negative instances. Note: we expect you to use a simple geometric argument to narrow down the search and derive the same solution SVM optimization would result in for the following two questions. You don't need to write a program to solve this problem.

$Example$	$feature_1$	$feature_2$	y
x_1	1	7	1
x_2	3	2	1
x_3	4	10	1
x_4	-1	-7	-1
x_5	1	-1	-1
x_6	3	-6	-1

(a) **(5 pts)** Suppose we are looking for a hard-SVM decision boundary $\boldsymbol{w}^T\boldsymbol{x}_n + b = 0$ passing through origin (i.e., $b = 0$). In other words, we minimize $||\boldsymbol{w}||_2$ subject to $y_n\boldsymbol{w}^T\boldsymbol{x}_n \geq 1, n = 1, \ldots, N$. Identify the `support vectors` (data points that are actually used in the calculation of w and margin) in this training dataset.

Solution: If we are to plot the points above, we would see that the points closest to the separating hyperplane (approximately around y = 0 by eye) would be (3, 2) and (1, -1). These points would be the most likely for $y\boldsymbol{w}^T\boldsymbol{x} \leq 1$.

(b) **(5 pts)** Following part (a), what is $\boldsymbol{w}^* \in \mathbb{R}^2$ in this case and what is the margin: $\frac{1}{\|w^*\|_2}$?
Solution: (using support vectors and having $w^T x_1 = 1$ and $w^T x_2 = -1$)
$\boldsymbol{w}^* = \begin{bmatrix} -1/5 \\ 4/5 \end{bmatrix}$ and $\frac{1}{\|w^*\|_2} = 1.471$

(c) **(15 pts)** Suppose we now allow the offset parameter b to be non-zero. In other words, we minimize $||\boldsymbol{w}||_2$ subject to $y_n\boldsymbol{w}^T\boldsymbol{x}_n + b \geq 1, n = 1, \ldots, N$. How would the classifier and the actual margin change in the previous question? What are $\boldsymbol{w}^*, b^*, \frac{1}{\|\boldsymbol{w}^*\|_2}$,? Compare your solutions with problem 2(b).
Solution:
(plug the support vectors into $y(\boldsymbol{w}^T\boldsymbol{x} + b) = 1$)
(using the system of equations $3w_1 + 2w_2 = 1 - b$ and $w_1 - w_2 = -1 - b$)
(using $w_1 = \frac{-1-3b}{5}$ and $w_2 = \frac{4+2b}{5}$)
$\boldsymbol{w}^* = \begin{bmatrix} 0 \\ 2/3 \end{bmatrix}$
$b^* = -1/3$
$\frac{1}{\|\boldsymbol{w}^*\|_2} = 2.25$

3 K-means [10 pts]

In this problem, we will first work through a numerical K-means example for a one-dimensional data and show that K-Means does not guarantee global optimal solution.

(a) **(5 pts)** Consider the case where $K = 3$ and we have 4 data points $x_1 = 1, x_2 = 2, x_3 = 5, x_4 = 7$. What is the optimal clustering for this data ? What is the corresponding value of the objective $\sum_{n=1}^N \sum_{k=1}^K \gamma_{nk}\|x_n - \mu_k\|_2^2$? ($x_n$ denotes the n^{th} data point, μ_k denotes the cluster mean of the k^{th} cluster and γ_{nk} is a Boolean indicator variable and is set to 1 only if n^{th} data point belongs to k^{th} cluster.)
Solution:
cluster1 = $\{x_1, x_2\}$, cluster2 = $\{x_3\}$, cluster3 = $\{x_4\}$
$\mu_1 = 0.5$, $\mu_2 = 5$, $\mu_3 = 7$
$\sum_{n=1}^N \sum_{k=1}^K \gamma_{nk}\|x_n - \mu_k\|_2^2 = 0.5$

(b) **(5 pts)** In K-Means, if we initialize the cluster centers as

$$c_1 = 1, c_2 = 2, c_3 = 6$$

what is the corresponding cluster assignment and the objective $\sum_{n=1}^N \sum_{k=1}^K \gamma_{nk}\|x_n - \mu_k\|_2^2$ at the first iteration? Does k-Mean algorithm improve the clusters at the second iteration?
Solution:
cluster1 = $\{x_1\}$, cluster2 = $\{x_2\}$, cluster3 = $\{x_3, x_4\}$
$\mu_1 = 1.5$, $\mu_2 = 2$, $\mu_3 = 6$
$\sum_{n=1}^N \sum_{k=1}^K \gamma_{nk}\|x_n - \mu_k\|_2^2 = 2$
For the second iteration, the clusters will likely not improve, as the objective function is at a local minimum.

4 Twitter analysis using SVMs, KMeans, GMM [35 pts]

In this project, you will be working with Twitter data. Specifically, we have supplied you with a number of tweets that are reviews/reactions to 4 different movies[1],

e.g., *"@nickjfrost just saw The Boat That Rocked/Pirate Radio and I thought it was brilliant! You and the rest of the cast were fantastic! < 3".*

The original data can be found here if you are interested in skimming through the tweets to get a sense of the data. We have preprocessed them for you and export them to a tweet_df.txt file

Specifically, we use a bag-of-words model to convert each tweet into a feature vector. A bag-of-words model treats a text file as a collection of words, disregarding word order. The first step in building a bag-of-words model involves building a "dictionary". A dictionary contains all of the unique words in the text file. For this project, we will be including punctuations in the dictionary too. For example, a text file containing *"John likes movies. Mary likes movies2!!"* will have a dictionary `{'John':0, 'Mary':1, 'likes':2, 'movies':3, 'movies2':4, '.':5, '!':6}`. Note that the `(key,value)` pairs are `(word, index)`, where the index keeps track of the number of unique words (size of the dictionary).

Given a dictionary containing d unique words, we can transform the n variable-length tweets into n feature vectors of length d by setting the i^{th} element of the j^{th} feature vector to 1 if the i^{th} dictionary word is in the j^{th} tweet, and 0 otherwise.

The preprocessed data contains 628 tweets on 4 different movies. Each line in the file contains exactly one tweet, so there are 628 lines in total. If a tweet praises or recommends a movie, it is classified as a positive review and labeled $+1$; otherwise it is classified as a negative review and labeled -1. The labels for positive or negative reviews as well as the labels for which movie is this tweet referring to are also included in the file.

You will learn to automatically classify such tweets as either positive or negative reviews. To do this, you will employ Support Vector Machines (SVMs), a popular choice for a large number of classification problems. You will also learn to automatically cluster such tweets in low dimensional space with Gaussian Mixture Model (GMM) and Kmeans.

Next, please download the preprocessed version of the tweets data tweet_df.txt and upload them to your google drive. For all the coding, please refer to the following colab notebook Fall2020-CS146-HW4.ipynb. Please save a local copy to your own drive first and only edit the TODO blocks in the notebook.

Documentation

- Support Vector Machine: link
- F1 score: link
- PCA: link
- KMeans: link

[1]Please note that these data were selected at random and thus the content of these tweets do not reflect the views of the course staff. :-)

- Gaussian Micture Model: link
- Adjusted Rand Index: link

4.1 Hyper-parameter Selection for a Linear-Kernel SVM [10 pts]

We will learn a classifier to separate the training data into positive and negative tweets. For the classifier, we will use SVMs with the linear kernel. We will use the `sklearn.svm.SVC` class and explicitly set only two of the initialization parameters: `kernel` set to 'linear', and `C`. As usual, we will use `SVC.fit(X,y)` to train our SVM,and `SVC.predict(X)` to make predictions.

SVMs have hyperparameters that must be set by the user. For linear kernel SVM, we will select the hyper-parameter using a simple train set and a development set. In the notebook, the train, dev, test split is already done for you. Please do NOT change that split on your own as we will evaluate all answers under the split we provided in the Notebook. We will train several different models with different hyper-parameters using the train set and select the hyper-parameter that leads to the 'best' performance in the dev set.

(a) **(7.5 pts)** Please use **F1-Score** as the performance measure. Train the linear SVM with different values of `C`, $C = 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^{2}, 10^{3}$. Plot the train f1 score and the dev f1 score against different choices of `C`. Include this plot in your writeup, and provide a 1-2 sentence description of your observations. What is the effect of C? What is the best value of `C`? What is the dev set f1-score when using the best `C`? **Please also copy-paste your code as part of the solution for this problem.**
Solution:

CODE: (comments in actual .py)

```
plotX = []
plotYTrain = []
plotYDev = []
C = 0.001
while C ≤ 1000 :
    plotX.append(C)
    linearSVM = SVC(C=C, kernel="linear")
    linearSVM.fit(X_train, y_train)
    y_trainPred = linearSVM.predict(X_train)
    trainF1Score = metrics.f1_score(y_train, y_trainPred)
    plotYTrain.append(trainF1Score)
    y_devPred = linearSVM.predict(X_dev)
    devF1Score = metrics.f1_score(y_dev, y_devPred)
    plotYDev.append(devF1Score)
    C = C * 10;
fig = plt.figure(figsize=(8,4))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_xscale('log')
ax1.set_yscale('linear')
ax2.set_xscale('log')
```

```
ax2.set_yscale(’linear’)
ax1.plot(plotX, plotYTrain)
ax1.title.set_text(”Train F1 Score”)
ax2.plot(plotX, plotYDev)
ax2.title.set_text(”Dev F1 Score”)
plt.show()
```

The main effect of C is that it modulated how much to weigh the empirical loss over the regularization of the weight vector of coefficients (in a square scale). The best value of C is 1 (when the f1-score of the dev set is the greatest). Any greater suggests overfitting due to f1-score since it doesn’t increase any further on the training set. The dev set f1-score when using C = 1 is 0.6745098039215687.

(b) **(2.5 pts)** Retraining the model with the best `C` on the train and dev set together. Report the F1-score on the test set. **Please also copy-paste your code as part of the solution for this problem.**
Solution:

CODE: (comments in actual .py)

```
linearSVM = SVC(C=1, kernel=”linear”)
linearSVM.fit(np.concatenate((X_train, X_dev), axis=0), np.concatenate((y_train, y_dev), axis=0))
y_testPred = linearSVM.predict(X_test)
testF1Score = metrics.f1_score(y_test, y_testPred)
print(f”testF1Score: testF1Score”)
```

The f1-score on the test set is 0.8735632183908046.

4.2 Low Dimensional Embedding Space Clustering [15 pts]

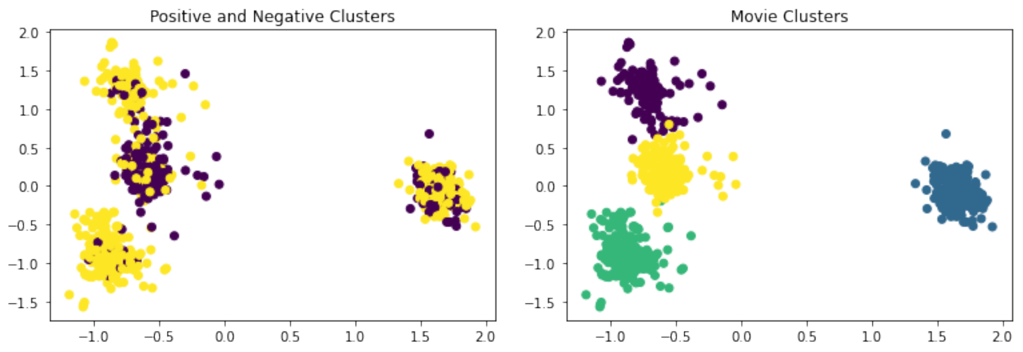
In this section, we will explore two unsupervised clustering algorithms: KMeans and GMM. The preprocessed twitter data lives in high (1811) dimensional space but it is hard for us to visualize more than three dimensional objects. Let’s project the data into a low dimensional embedding space with a commonly used algorithm: Principal Component Analysis (PCA). We have already provided the code to do that. Please run it and from now on work with `X_embedding` variable which is 628 by 2 instead of 628 by 1811. If you are interested in the math behind PCA here are some good reading materials: A One-Stop Shop for Principal Component Analysis, Wikipedia page for PCA. However, for this project you are not required to know the math behind it. Intuitively, this algorithm is extracting the most useful linear combination of the features such that it reduces the amount of information loss occurred when projecting from high (1811) dimensions to low (2) dimensions.

We will also evaluate the purity of the clusters returned from any unsupervised algorithms against the ground truth labels from the original `tweet_df.txt`.

Here we will use `sklearn.metrics.adjusted_rand_score`. On a high level, this metric is measuring what fraction of the examples are labeled correctly but normalized so that if the cluster assignment is near random then the adjusted rand score will be close to 0. If the assignment is near perfect, the adjusted rand score should be close to 1.

(a) **(3 pts)** Visualize the low dimensional data by calling function `plot_scatter`. First label/color the scatter plot by positive or negative reviews. Then label/color each tweet/dot by which movie is that review referring to. Do positive reviews and negative reviews form two nice clusters? Do 4 different movies form nice clusters? Include both plots in your writeup. **Please also copy-paste your code as part of the solution for this problem.**
Solution:

CODE:
plot_scatter(X_embedding, y, title="Positive and Negative Clusters")
plot_scatter(X_embedding, movies, title="Movie Clusters")



The positive/negative reviews don't seem to distinguish the two very well into clusters (yellow for -1, black for 1). For the movies, the feature maps do form nice clusters for each specific movie (green for 0, black for 1, blue for 2, and yellow for 3).

(b) **(6 pts)** Use the `sklearn.cluster.KMeans` class on `X_embedding` with `n_clusters` set to 4, `init` set to 'random', `n_init` set to 1, and `random_state` set to 2. Visualize the low dimensional data by labeling/coloring each tweet with Kmeans results. Calculate the adjusted rand score

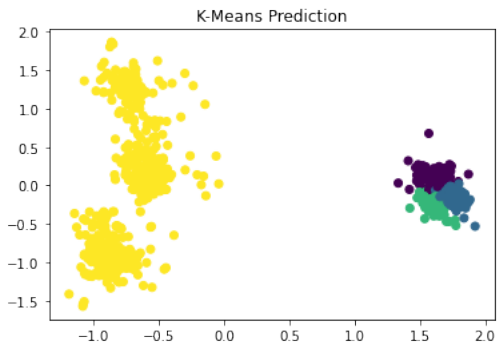
Use the `sklearn.mixture.GaussianMixture` class on `X_embedding` with `n_components` set to 4, `random_state` set to 0 `init_params` set to 'random'. Visualize the low dimensional data by labeling/coloring each tweet with GMM results. Calculate the adjusted rand score

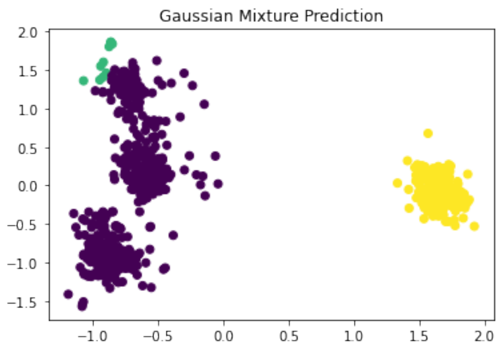
Visually, do you think they perform poorly or great? What are the adjusted rand scores for both Kmeans labels and GMM labels? Why might that be the case? Include both plots, the performance results, and a 1-2 sentence description/justification of what you saw in your writeup. **Please also copy-paste your code as part of the solution for this problem.**
Solution:

CODE:
k4means = KMeans(n_clusters=4, init='random', n_init=1, random_state=2)
k4means.fit(X_embedding)
k4meansPred = k4means.predict(X_embedding)
plot_scatter(X_embedding, k4meansPred, title="K-Means Prediction")

```
print(f”K-Means Adjusted Rand Score: metrics.adjusted_rand_score(k4meansPred, movies)”)
gmm = GaussianMixture(n_components=4, random_state=0, init_params=’random’)
gmm.fit(X_embedding)
gmmPred = gmm.predict(X_embedding)
plot_scatter(X_embedding, gmmPred, title=”Gaussian Mixture Prediction”)
print(f”Gaussian Mixture Adjusted Rand Score: metrics.adjusted_rand_score(gmmPred, movies)”)
```

Visually, I think that the models performed pretty poorly. The adjusted rand scores for Kmeans ad GMM labels are 0.25935622456493296 and 0.4189198834394529, respectively. This might be the case since we haven’t set any cluster centers (and had them be set randomly). In the resulting scatter plots from the predictions, we see that ”connected” clusters ended up being categorized as the same cluster (had we manually set the cluster centers, this probably wouldn’t have happened).





(c) **(6 pts)** Use the `sklearn.cluster.KMeans` class on `X_embedding` with `n_clusters` set to 4, `init` set to ’random’, `n_init` set to 100, and `random_state` set to 2. Visualize the low dimensional data by labeling/coloring each tweet with Kmeans over 100 different starting points results. Calculate the adjusted rand score

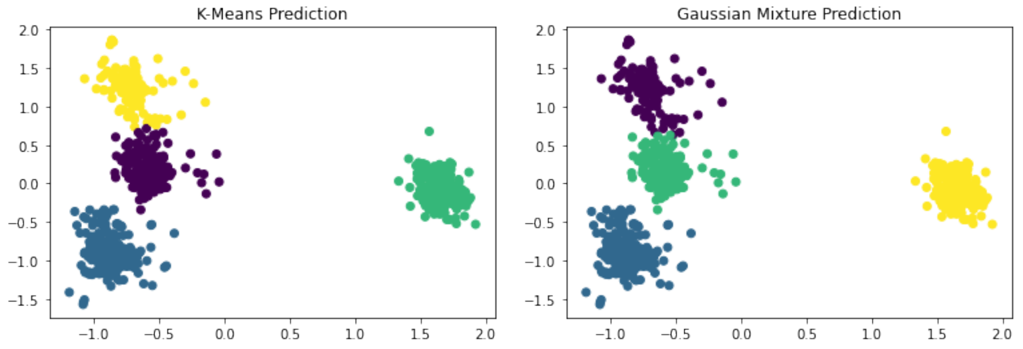
Use the `sklearn.mixture.GaussianMixture` class on `X_embedding` with `n_components` set to 4, `random_state` set to 0 `init_params` set to ’random’, and `n_init` set to 100. Visualize the low dimensional data by labeling/coloring each tweet with this random initialized GMM but with 100 different starting points results. Calculate the adjusted rand score

Do you see a huge performance gain over what we got from part (b)? Include both plots, the performance results, and a 1-2 sentence description/justification of what you saw in your writeup. **Please also copy-paste your code as part of the solution for this problem.**
Solution:

CODE:
```
k4means = KMeans(n_clusters=4, init=’random’, n_init=100, random_state=2)
k4means.fit(X_embedding)
k4meansPred = k4means.predict(X_embedding)
plot_scatter(X_embedding, k4meansPred, title=”K-Means Prediction”)
print(f”K-Means Adjusted Rand Score: metrics.adjusted_rand_score(k4meansPred, movies)”)
gmm = GaussianMixture(n_components=4, random_state=0, init_params=’random’, n_init=100)
gmm.fit(X_embedding)
gmmPred = gmm.predict(X_embedding)
plot_scatter(X_embedding, gmmPred, title=”Gaussian Mixture Prediction”)
```

print(f"Gaussian Mixture Adjusted Rand Score: metrics.adjusted_rand_score(gmmPred, movies)")

In this case, when we set n_init to 100, we see a significant improvement in accuracy when distinguishing the different movies. This is likely due to the fact that setting different init values a large number of times across the span of the data would make local minimums of the objective function less impactful, making finding the global minimum of the objective function to optimize performance much more likely. The adjusted rand scores for Kmeans ad GMM labels are 0.9824442429232366 and 0.982486480501798, respectively.


K-Means Prediction
Gaussian Mixture Prediction