1. (PCA using MSE and population covariance matrix[1]) Assume that $\mathbf{x}$ is a zero-mean p dimensional random vector ($\mathbb{E}[\mathbf{x}] = \mathbf{0}$) with covariance matrix: (10 pts)

$$\mathbf{R} = \mathbb{E}[\mathbf{x}\mathbf{x}^T]$$

We wish to estimate $\mathbf{x}$ with $M \leq p$ *principal directions* as:

$$\hat{\mathbf{x}} = \sum_{i=1}^{M} \alpha_i \mathbf{e}_i = \boldsymbol{\alpha}^T \mathbf{e}$$

where $\mathbf{e}_i$'s are the orthonormal eigenvectors of the covariance matrix $\mathbf{R}$ and $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_p]^T$. Show that the minimization of the mean squared error:

$$J = \mathbb{E}[\|\mathbf{x} - \hat{\mathbf{x}}\|^2]$$

with respect to $\alpha_1, \ldots, \alpha_m$ yields:

$$\alpha_i = \mathbf{e}_i^T \mathbf{x}, \ i = 1, 2, \ldots, M$$

as the *principal component*, that is, the projection of the data vector $\mathbf{x}$ onto the eigenvector $\mathbf{e}_i$.

2. Let $p(\mathbf{x}|\omega_i)$ be arbitrary densities with means μ_i and covariance matrices σ_i — not necessarily normal — for $i = 1, 2$. Let y= $\mathbf{w}^T\mathbf{x}$ be a projection, and let the induced one-dimensional densities $p(y|\omega_i)$ have means μ_i and variances σ_i^2. (15 pts)

   (a) Show that the criterion function

$$J_1(\mathbf{w}) = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

   is maximized by

$$\mathbf{w} = (\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

   (b) If $P(\omega_i)$ is the prior probability for ω_i, show that the criterion function

$$J_2(\mathbf{w}) = \frac{(\mu_1 - \mu_2)^2}{P(\omega_1)\sigma_1^2 + P(\omega_2)\sigma_2^2}$$

   is maximized by

$$\mathbf{w} = (P(\omega_1)\boldsymbol{\Sigma}_1 + P(\omega_2)\boldsymbol{\Sigma}_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

   (c) Explain which of $J(\mathbf{w}_1)$ and $J(\mathbf{w}_2)$ is "closer" to the criterion that is used by Fisher's LDA.

[1] Using the population covariance matrix instead of the scatter matrix simplifies the formulation here

3. **Time Series Classification Part 1: Feature Creation/Extraction**

   Important Note: You will NOT submit this part with Homework 5. It was the programming assignment of Homework 4. However, you may want to submit the code for Homework 4 with Homework 5 again, since it might need the feature creation code. .

   An interesting task in machine learning is classification of time series. In this problem, we will classify the activities of humans based on time series obtained by a Wireless Sensor Network.

   (a) Download the AReM data from: `https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+system+based+on+Multisensor+data+fusion+\%28AReM\%29` . The dataset contains 7 folders that represent seven types of activities. In each folder, there are multiple files each of which represents an instant of a human performing an activity.[2] Each file containis 6 time series collected from activities of the same person, which are called avg_rss12, var_rss12, avg_rss13, var_rss13, vg_rss23, and ar_rss23. There are 88 instances in the dataset, each of which contains 6 time series and each time series has 480 consecutive values.

   (b) Keep datasets 1 and 2 in folders bending1 and bending 2, as well as datasets 1, 2, and 3 in other folders as test data and other datasets as train data.

   (c) Feature Extraction

   Classification of time series usually needs extracting features from them. In this problem, we focus on time-domain features.

   i. Research what types of time-domain features are usually used in time series classification and list them (examples are minimum, maximum, mean, etc).

   ii. Extract the time-domain features minimum, maximum, mean, median, standard deviation, first quartile, and third quartile for all of the 6 time series in each instance. You are free to normalize/standardize features or use them directly.[3]

   Your new dataset will look like this:

Instance	$\min_1$	$\max_1$	mean_1	median_1 $\cdots$ 1st quart_6	3rd quart_6
1					
2					
3					
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$ ... $\vdots$	$\vdots$
88					

   where, for example, 1st quart_6, means the first quartile of the sixth time series in each of the 88 instances.

   iii. Estimate the standard deviation of each of the time-domain features you extracted from the data. Then, use Python's bootstrapped or any other method to build a 90% bootsrap confidence interval for the standard deviation of each feature.

[2] Some of the data files need very minor cleaning. You can do it by Excel or Python.

[3] You are welcome to experiment to see if they make a difference.

iv. Use your judgement to select the three most important time-domain features (one option may be min, mean, and max).

v. Assume that you want to use the training set to classify bending from other activities, i.e. you have a binary classification problem. Depict scatter plots of the features you specified in 3(c)iv extracted from time series 1, 2, and 6 of each instance, and use color to distinguish bending vs. other activities. (See p. 129 of the ISLR textbook).[4]

4. **Time Series Classification Part 2: Binary and Multiclass Classification**

(a) Binary Classification Using Logistic Regression[5]

i. Break each time series in your training set into two (approximately) equal length time series. Now instead of 6 time series for each of the training instances, you have 12 time series for each training instance. Repeat the experiment in 3(c)v, i.e depict scatter plots of the features extracted from both parts of the time series 1,2, and 12. Do you see any considerable difference in the results with those of 3(c)v? (5 pts)

ii. Break each time series in your training set into $l \in \{1, 2, \ldots, 20\}$ time series of approximately equal length and use logistic regression[6] to solve the binary classification problem, using time-domain features. Remember that breaking each of the time series does not change the number of instances. It only changes the number of features for each instance. Calculate the p-values for your logistic regression parameters in each model corresponding to each value of l and refit a logistic regression model using your pruned set of features.[7] Alternatively, you can use backward selection using sklearn.feature_selection or glm in R. Use 5-fold cross-validation to determine the best value of the pair (l, p), where p is the number of features used in recursive feature elimination. Explain what the right way and the wrong way are to perform cross-validation in this problem.[8] Obviously, use the right way! Also, you may encounter the problem of class imbalance, which may make some of your folds not having any instances of the rare class. In such a case, you can use *stratified cross validation.* Research what it means and use it if needed. (15 pts)
In the following, you can see an example of applying Python's Recursive

[4]You are welcome to repeat this experiment with other features as well as with time series 3, 4, and 5 in each instance.

[5]Some logistic regression packages have a built-in $\mathscr{L}_2$ regularization. To remove the effect of $\mathscr{L}_2$ regularization, set $\lambda = 0$ or set the budget $C \to \infty$ (i.e. a very large value).

[6]If you encountered instability of the logistic regression problem because of linearly separable classes, modify the Max-Iter parameter in logistic regression to stop the algorithm immaturely and prevent from its instability.

[7]R calculates the p-values for logistic regression automatically. One way of calculating them in Python is to call R within Python. There are other ways to obtain the p-values as well.

[8]This is an interesting problem in which the number of features changes depending on the value of the parameter l that is selected via cross validation. Another example of such a problem is Principal Component Regression, where the number of principal components is selected via cross validation.

Feature Elimination, which is a backward selection algorithm, to logistic regression.

```
# Recursive Feature Elimination
from sklearn import datasets
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load the iris datasets
dataset = datasets.load_iris()
# create a base classifier used to evaluate a subset of attributes
model = LogisticRegression()
# create the RFE model and select 3 attributes
rfe = RFE(model, 3)
rfe = rfe.fit(dataset.data, dataset.target)
# summarize the selection of the attributes
print(rfe.support_)
print(rfe.ranking_)
```

iii. Report the confusion matrix and show the ROC and AUC for your classifier on train data. Report the parameters of your logistic regression β_i's as well as the p-values associated with them. (10 pts)

iv. Test the classifier on the test set. Remember to break the time series in your test set into the same number of time series into which you broke your training set. Remember that the classifier has to be tested using the features extracted from the test set. Compare the accuracy on the test set with the cross-validation accuracy you obtained previously. (10 pts)

v. Do your classes seem to be well-separated to cause instability in calculating logistic regression parameters?

vi. From the confusion matrices you obtained, do you see imbalanced classes? If yes, build a logistic regression model based on case-control sampling and adjust its parameters. Report the confusion matrix, ROC, and AUC of the model. (10 pts)

(b) Binary Classification Using $\mathscr{L}_1$-penalized logistic regression

i. Repeat 4(a)ii using $\mathscr{L}_1$-penalized logistic regression,[9] i.e. instead of using p-values for variable selection, use $\mathscr{L}_1$ regularization. Note that in this problem, you have to cross-validate for both l, the number of time series into which you break each of your instances, and λ, the weight of $\mathscr{L}_1$ penalty in your logistic regression objective function (or C, the budget). Packages usually perform cross-validation for λ automatically.[10] (15 pts)

ii. Compare the $\mathscr{L}_1$-penalized with variable selection using p-values. Which one performs better? Which one is easier to implement? (5 pts)

(c) Multi-class Classification (The Realistic Case)

[9] For $\mathscr{L}_1$-penalized logistic regression, you may want to use normalized/standardized features

[10] Using the package Liblinear is strongly recommended.

i. Find the best l in the same way as you found it in 4(b)i to build an $\mathscr{L}_1$-penalized multinomial regression model to classify all activities in your training set.[11] Report your test error. Research how confusion matrices and ROC curves are defined for multiclass classification and show them for this problem if possible.[12] (10 pts)

ii. Repeat 4(c)i using a Naïve Bayes' classifier. Use both Gaussian and Multinomial pdfs and compare the results. (10 pts)

iii. Create p Principal Components from features extracted from features you extracted from l time series. Cross validate on the (l, p) pair to build a Naïve Bayes' classifier based on the PCA features to classify all activities in your data set. Report your test error and plot the scatterplot of the classes in your training data based on the first and second principal components you found from features extracted from l time series, where l is the value you found using cross-validation. Show confusion matrices and ROC curves. (10 pts)

iv. Which method is better for multi-class classification in this problem? (5 pts)

[11] New versions of scikit learn allow using $\mathscr{L}_1$-penalty for multinomial regression.

[12] For example, the pROC package in R does the job.