Assignment 5 - Kaggle Competition and Unsupervised Learning

Yuanjing Zhu

Netid: yz792

Note: this assignment falls under collaboration Mode 2: Individual Assignment – Collaboration Permitted. Please refer to the syllabus for additional information.

Instructions for all assignments can be found here, and is also linked to from the course syllabus.

Total points in the assignment add up to 90; an additional 10 points are allocated to presentation quality.

Learning objectives

Through completing this assignment you will be able to...

1. Apply the full supervised machine learning pipeline of preprocessing, model selection, model performance evaluation and comparison, and model application to a real-world scale dataset
2. Apply clustering techniques to a variety of datasets with diverse distributional properties, gaining an understanding of their strengths and weaknesses and how to tune model parameters
3. Apply PCA and t-SNE for performing dimensionality reduction and data visualization

1

[40 points] Kaggle Classification Competition

You've learned a great deal about supervised learning and now it's time to bring together all that you've learned. You will be competing in a Kaggle Competition along with the rest of the class! Your goal is to predict hotel reservation cancellations based on a number of potentially related factors such as lead time on the booking, time of year, type of room, special requests made, number of children, etc. While you will be asked to take certain steps along the way to your submission, you're encouraged to try creative solutions to this problem and your choices are wide open for you to make your decisions on how to best make the predictions.

IMPORTANT: Follow the link posted on Ed to register for the competition

You can view the public leaderboard anytime here

The Data. The dataset is provided as `a5_q1.pkl` which is a pickle file format, which allows you to load the data directly using the code below; the data can be downloaded from the Kaggle competition website. A data dictionary for the project can be found here and the original paper that describes the dataset can be found here. When you load the data, 5 matrices are provided `X_train_original`, `y_train`, and `X_test_original`, which are the original, unprocessed features and labels for the training set and the test features (the test labels are not provided - that's what you're predicting). Additionally, `X_train_ohe` and `X_test_ohe` are provided which are one-hot-encoded (OHE) versions of the data. The OHE versions OHE processed every categorical variable. This is provided for convenience if you find it helpful, but you're welcome to reprocess the original data other ways if your prefer.

Scoring. You will need to achieve a minimum acceptable level of performance to demonstrate proficiency with using these supervised learning techniques. Beyond that, it's an open competition and scoring in the top three places of the *private leaderboard* will result in **5 bonus points in this assignment** (and the pride of the class!). Note: the Kaggle leaderboard has a public and private component. The public component is viewable throughout the competition, but the private leaderboard is revealed at the end. When you make a submission, you immediately see your submission on the public leaderboard, but that only represents scoring on a fraction of the total collection of test data, the rest remains hidden until the end of the competition to prevent overfitting to the test data through repeated submissions. You will be be allowed to hand-select two eligible submissions for private score, or by default your best two public scoring submissions will be selected for private scoring.

Requirements:

(a) Explore your data. Review and understand your data. Look at it; read up on what the features represent; think through the application domain; visualize statistics from the paper data to understand any key relationships. **There is no output required for this question**, but you are encouraged to explore the data personally before going further.

(b) Preprocess your data. Preprocess your data so it's ready for use for classification and describe what you did and why you did it. Preprocessing may include: normalizing data, handling missing or erroneous values, separating out a validation dataset, preparing categorical variables through one-hot-encoding, etc. To make one step in this process easier, you're provided with a one-hot-encoded version of the data already.

- Comment on each type of preprocessing that you apply and both how and why you apply it.

(c) Select, train, and compare models. Fit at least 5 models to the data. Some of these can be experiments with different hyperparameter-tuned versions of the same model, although all 5 should not be the same type of model. There are no constraints on the types of models, but you're encouraged to explore examples we've discussed in class including:

1. Logistic regression
2. K-nearest neighbors
3. Random Forests
4. Neural networks
5. Support Vector Machines
6. Ensembles of models (e.g. model bagging, boosting, or stacking). `Scikit-learn` offers a number of tools for assisting with this including those for bagging, boosting, and stacking. You're also welcome to explore options beyond the `sklean` universe; for example, some of you may have heard of XGBoost which is a very fast implementation of gradient boosted decision trees that also allows for parallelization.

When selecting models, be aware that some models may take far longer than others to train. Monitor your output and plan your time accordingly.

Assess the classification performance AND computational efficiency of the models you selected:

- Plot the ROC curves and PR curves for your models in two plots: one of ROC curves and one of PR curves. For each of these two plots, compare the performance of the models you selected above and trained on the training data, evaluating them on the validation data. Be sure to plot the line representing random guessing on each plot. You should plot all of the model's ROC curves on a single plot and the PR curves on a single plot. One of the models should also be your BEST performing submission on the Kaggle public leaderboard (see below). In the legends of each, include the area under the curve for each model (limit to 3 significant figures). For the ROC curve, this is the AUC; for the PR curve, this is the average precision (AP).
- As you train and validate each model time how long it takes to train and validate in each case and create a plot that shows both the training and prediction time for each model included in the ROC and PR curves.
- Describe:
  - Your process of model selection and hyperparameter tuning
  - Which model performed best and your process for identifying/selecting it

(d) Apply your model "in practice". Make *at least* 5 submissions of different model results to the competition (more submissions are encouraged and you can submit up to 5 per day!). These do not need to be the same that you report on above, but you should select your *most competitive* models.

- Produce submissions by applying your model on the test data.
- Be sure to RETRAIN YOUR MODEL ON ALL LABELED TRAINING AND VALIDATION DATA before making your predictions on the test data for submission. This will help to maximize your performance on the test data.
- In order to get full credit on this problem you must achieve an AUC on the Kaggle public leaderboard above the "Benchmark" score on the public leaderboard.

Guidance:

1. **Preprocessing**. You may need to preprocess the data for some of these models to perform well (scaling inputs or reducing dimensionality). Some of this preprocessing may differ from model to model to achieve the best performance. A helpful tool for creating such preprocessing and model fitting pipelines is the sklearn `pipeline` module which lets you group a series of processing steps together.
2. **Hyperparameters**. Hyperparameters may need to be tuned for some of the model you use. You may want to perform hyperparameter tuning for some of the models. If you experiment with different hyperparameters that include many model runs, you may want to apply them to a small subsample of your overall data before running it on the larger training set to be time efficient (if you do, just make sure to ensure your selected subset is representative of the rest of your data).
3. **Validation data**. You're encouraged to create your own validation dataset for comparing model performance; without this, there's a significant likelihood of overfitting to the data. A common choice of the split is 80% training, 20% validation. Before you make your final predictions on the test data, be sure to retrain your model on the entire dataset.
4. **Training time**. This is a larger dataset than you've worked with previously in this class, so training times may be higher that what you've experienced in the past. Plan ahead and get your model pipeline working early so you can experiment with the models you use for this problem and have time to let them run.

Starter code

Below is some code for (1) loading the data and (2) once you have predictions in the form of confidence scores for those classifiers, to produce submission files for Kaggle.

In [ ]:
```python
import pandas as pd
import numpy as np
import pickle

#################################
# Load the data
#################################
data = pickle.load( open( "F:/Duke MIDS/705_ML/"
                         "Assignment/05/a5_q1.pkl", "rb" ) )

y_train = data['y_train']
X_train_original = data['X_train'] # Original dataset
X_train_ohe = data['X_train_ohe']  # One-hot-encoded dataset

X_test_original = data['X_test']
X_test_ohe = data['X_test_ohe']

#################################
# Produce submission
#################################

def create_submission(confidence_scores, save_path):
    '''Creates an output file of submissions for Kaggle

    Parameters
    ----------
    confidence_scores : list or numpy array
        Confidence scores (from predict_proba methods from classifiers) or
        binary predictions (only recommended in cases when predict_proba is
        not available)
    save_path : string
        File path for where to save the submission file.

    Example:
    create_submission(my_confidence_scores, './data/submission.csv')

    '''
    import pandas as pd

    submission = pd.DataFrame({"score":confidence_scores})
    submission.to_csv(save_path, index_label="id")
```

ANSWER

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.model_selection import PredefinedSplit
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.metrics import f1_score, roc_auc_score, \
    roc_curve, precision_recall_curve, average_precision_score
from time import time
```

(a) Explore the data

In [ ]:
```
X_train_original.head()
```

Out[ ]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day
0	Resort Hotel	342	2015	July	27	
2	Resort Hotel	7	2015	July	27	
3	Resort Hotel	13	2015	July	27	
4	Resort Hotel	14	2015	July	27	
5	Resort Hotel	14	2015	July	27	

5 rows × 29 columns

In [ ]:
```
X_train_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 95512 entries, 0 to 119389
Data columns (total 29 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   hotel                           95512 non-null  object 
 1   lead_time                       95512 non-null  int64  
 2   arrival_date_year               95512 non-null  int64  
 3   arrival_date_month              95512 non-null  object 
 4   arrival_date_week_number        95512 non-null  int64  
 5   arrival_date_day_of_month       95512 non-null  int64  
 6   stays_in_weekend_nights         95512 non-null  int64  
 7   stays_in_week_nights            95512 non-null  int64  
 8   adults                          95512 non-null  int64  
 9   children                        95510 non-null  float64
 10  babies                          95512 non-null  int64  
 11  meal                            95512 non-null  object 
 12  country                         95117 non-null  object 
 13  market_segment                  95512 non-null  object 
 14  distribution_channel            95512 non-null  object 
 15  is_repeated_guest               95512 non-null  int64  
 16  previous_cancellations          95512 non-null  int64  
 17  previous_bookings_not_canceled  95512 non-null  int64  
 18  reserved_room_type              95512 non-null  object 
 19  assigned_room_type              95512 non-null  object 
 20  booking_changes                 95512 non-null  int64  
 21  deposit_type                    95512 non-null  object 
 22  agent                           82431 non-null  float64
 23  company                         5453 non-null   float64
 24  days_in_waiting_list            95512 non-null  int64  
 25  customer_type                   95512 non-null  object 
 26  adr                             95512 non-null  float64
 27  required_car_parking_spaces     95512 non-null  int64  
 28  total_of_special_requests       95512 non-null  int64  
dtypes: float64(4), int64(15), object(10)
memory usage: 21.9+ MB
```

There are 10 categorical variables and 19 numeric variables

In [ ]:
```
X_train_original.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	
lead_time	95512.0	103.849768	106.722804	0.00	18.00	69.0	160.0	
arrival_date_year	95512.0	2016.157205	0.707470	2015.00	2016.00	2016.0	2017.0	2
arrival_date_week_number	95512.0	27.152902	13.611204	1.00	16.00	27.0	38.0	
arrival_date_day_of_month	95512.0	15.823038	8.786777	1.00	8.00	16.0	23.0	
stays_in_weekend_nights	95512.0	0.928491	0.999940	0.00	0.00	1.0	2.0	
stays_in_week_nights	95512.0	2.503288	1.918017	0.00	1.00	2.0	3.0	
adults	95512.0	1.855746	0.596925	0.00	2.00	2.0	2.0	
children	95510.0	0.103696	0.397763	0.00	0.00	0.0	0.0	
babies	95512.0	0.007748	0.093348	0.00	0.00	0.0	0.0	
is_repeated_guest	95512.0	0.031598	0.174929	0.00	0.00	0.0	0.0	
previous_cancellations	95512.0	0.087235	0.844491	0.00	0.00	0.0	0.0	
previous_bookings_not_canceled	95512.0	0.140035	1.532968	0.00	0.00	0.0	0.0	
booking_changes	95512.0	0.220621	0.653900	0.00	0.00	0.0	0.0	
agent	82431.0	86.893778	110.839209	1.00	9.00	14.0	229.0	
company	5453.0	188.237117	131.459182	6.00	62.00	178.0	269.0	
days_in_waiting_list	95512.0	2.316348	17.651287	0.00	0.00	0.0	0.0	
adr	95512.0	101.679313	50.906371	-6.38	69.29	94.5	126.0	5
required_car_parking_spaces	95512.0	0.062673	0.246274	0.00	0.00	0.0	0.0	
total_of_special_requests	95512.0	0.571310	0.792712	0.00	0.00	0.0	1.0	

Label distribution:

In [ ]:
```
y_train.value_counts()
```

Out[ ]:
```
0    60123
1    35389
Name: is_canceled, dtype: int64
```

In [ ]:
```
print(f"Percentage of positive class: {y_train.value_counts()[1]\
      /y_train.shape[0]*100:.2f}%")
print(f"Percentage of negative class: {y_train.value_counts()[0]\
    /y_train.shape[0]*100:.2f}%")
```

```
Percentage of positive class: 37.05%
Percentage of negative class: 62.95%
```

Label is slightly imbalanced, sampling method is needed

```
In [ ]: for i in X_train_original.columns:
            if X_train_original[i].isnull().sum()/X_train_original.shape[0]*100 > 0:
                print(f"Missing value number of {i}: {X_train_original[i].isnull().sum()}")
                print(f"Missing value percentage of {i}: \
                {X_train_original[i].isnull().sum()/X_train_original.shape[0]*100:.2f}%")
                print('='*50)
```

```
Missing value number of children: 2
Missing value percentage of children: 0.00%
==================================================
Missing value number of country: 395
Missing value percentage of country: 0.41%
==================================================
Missing value number of agent: 13081
Missing value percentage of agent: 13.70%
==================================================
Missing value number of company: 90059
Missing value percentage of company: 94.29%
==================================================
```

children, country, agent, and compay have missing values.

(b) Preprocess the data

- **missing values**

```
In [ ]: for i in X_train_ohe.columns:
            if X_train_ohe[i].isnull().sum()/X_train_ohe.shape[0]*100 > 0:
                print(f"Missing value number of {i}: {X_train_ohe[i].isnull().sum()}")
                print(f"Missing value percentage of {i}: \
                      {X_train_ohe[i].isnull().sum()/X_train_ohe.shape[0]*100:.2f}%")
                print('='*50)
```

```
Missing value number of children: 2
Missing value percentage of children: 0.00%
==================================================
```

After on-hot encoding, only children has missing values. I fill the missing values with 0.

```
In [ ]: X_train_ohe_2 = X_train_ohe.copy()
        X_train_ohe_2.fillna(0, inplace=True)
        assert X_train_ohe_2.isnull().sum().sum() == 0
```

- **Resampling**

The lable distribution is a little imbalanced. The percentage of negative class is 62.95% while the percentage of positive class is 37.05%. I use SMOTE to resample the data.

```
In [ ]: smote = SMOTE(random_state=42)
        X_train_resampled, y_train_resampled = smote.fit_resample(X_train_ohe_2, y_train)
```

- **train-test-split**

After resampling, I split the data into 80% training and 20% validation.

```
In [ ]: Xtrain, Xval, ytrain, yval = train_test_split(X_train_resampled,\
                                 y_train_resampled, test_size=0.2, random_state=42)
```

- **normalize**

I used StandardScaler to normalize the data for dimension reduction.

```
In [ ]: stdscalar = StandardScaler()
        stdscalar.fit(Xtrain)
        Xtrain = stdscalar.transform(Xtrain)
        Xval = stdscalar.transform(Xval)
```

```
In [ ]: X_train_plus_val = np.concatenate((Xtrain, Xval), axis=0)
        y_train_plus_val = np.concatenate((ytrain, yval), axis=0)
```

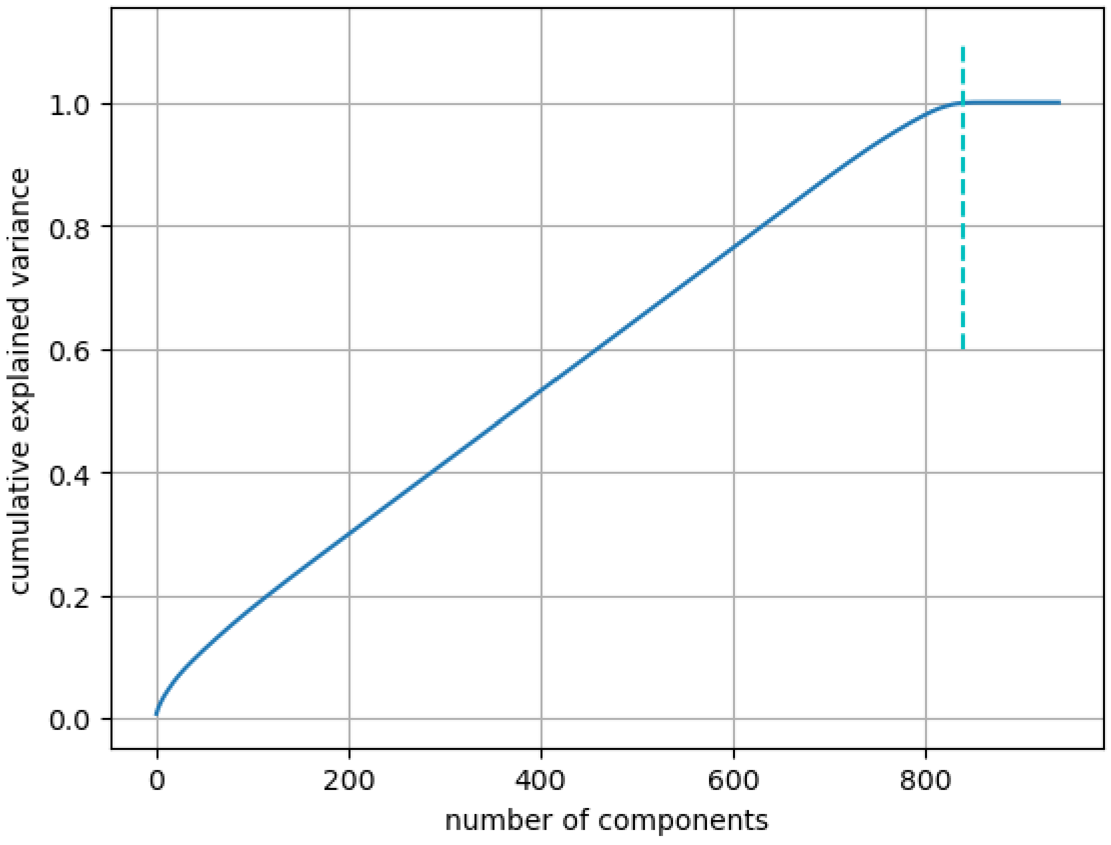
- **Dimensionality reduction**

First I plotted the cumulative explained variance ratio of all the features. I found that the first 800 features explained most of the variance. Then I applied a decision tree classifier on both the original data and the reduced data to check whether dimension reduction would affect the performance of the model.

```
In [ ]: pca = PCA()
        pca.fit(Xtrain)
```

```
Out[ ]: ▾ PCA
        PCA()
```

```
In [ ]: # plot the cumulative sum of explained variance ratio
        plt.plot(np.cumsum(pca.explained_variance_ratio_))
        plt.grid('on')
        plt.vlines(840, 0.6, 1.1, colors = "c", linestyles = "dashed")
        plt.xlabel('number of components')
        plt.ylabel('cumulative explained variance')
        plt.show()
```



```
In [ ]: pca= PCA(n_components=840)
        pca.fit(Xtrain)
        Xtrain_pca = pca.transform(Xtrain)
        Xval_pca = pca.transform(Xval)
```

```
In [ ]: # test the effect of PCA on the performance of the model
        # decision tree
        dt = DecisionTreeClassifier(random_state=42)
        dt.fit(Xtrain, ytrain)
        print(f"Decision Tree score on training set: {dt.score(Xtrain, ytrain):.4f}")
        print(f"Decision Tree score on validation set: {dt.score(Xval, yval):.4f}")

        dt = DecisionTreeClassifier(random_state=42)
        dt.fit(Xtrain_pca, ytrain)
        print(f"Decision Tree score on training set after PCA: \
              {dt.score(Xtrain_pca, ytrain):.4f}")
        print(f"Decision Tree score on validation set after PCA: \
              {dt.score(Xval_pca, yval):.4f}")
```

```
Decision Tree score on training set: 0.9967
Decision Tree score on validation set: 0.8756
Decision Tree score on training set after PCA: 0.9967
Decision Tree score on validation set after PCA: 0.8504
```

From the principal component analysis, I find that the first 800 principal components can explain most of the variance. Then I use the decision tree classifier to test the effect of dimensionality reduction. I find that the accuracy of the model is reduced from 87.56% to 85.04% on validation dataset after dimensionality reduction. So I decide **not to use pca in further analysis**.

(c) Model training

1. **Logistic regression**

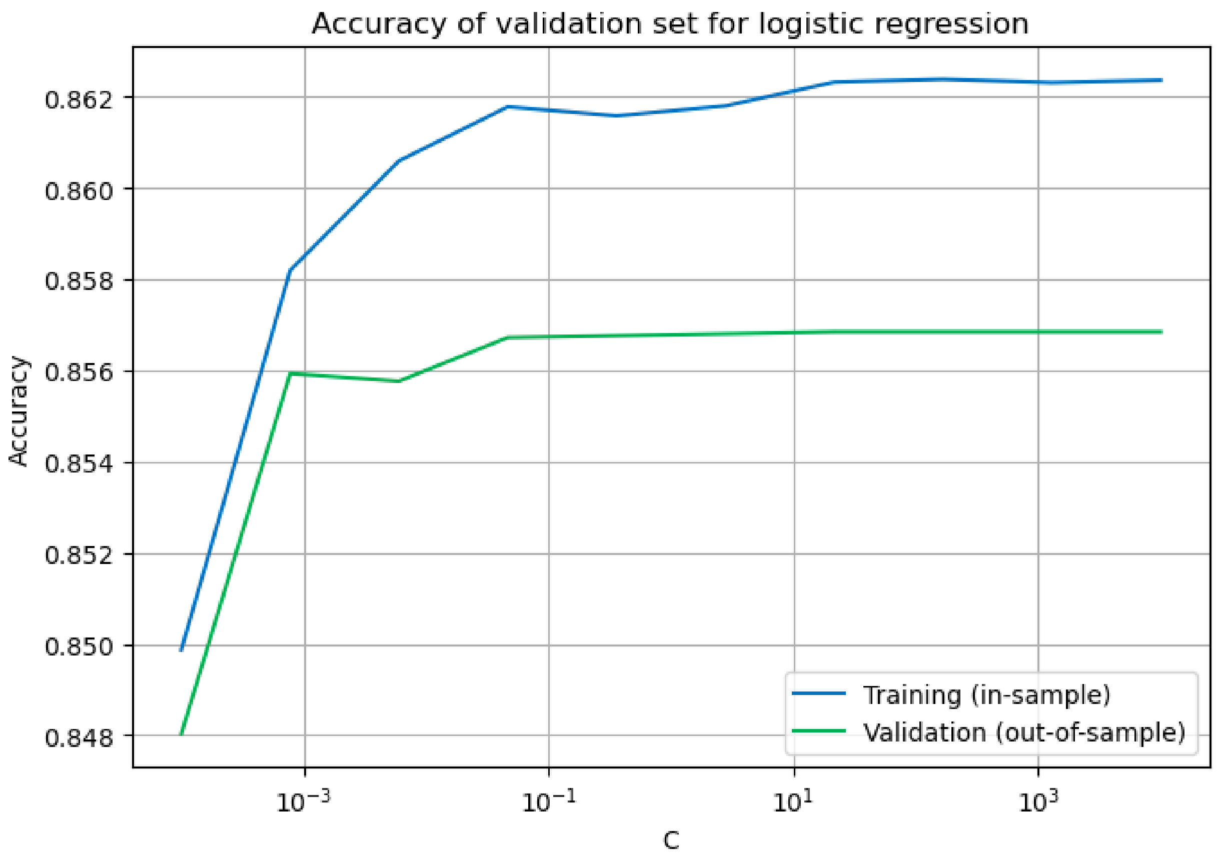
```
In [ ]: start = time()
        LR = LogisticRegression(solver='liblinear', random_state=42)
        LR.fit(Xtrain, ytrain)
        end = time()
        print(f"Time used: {end-start:.2f}s")
        print(f"Accuracy of validation set: {100*LR.score(Xval, yval):.2f}%")
```

```
Time used: 79.60s
Accuracy of validation set: 85.66%
```

```
In [ ]: # tuning C
        l_C = np.logspace(-4, 4, 10)
        acc_train_lr = []
        acc_val_lr = []
        for c in l_C:
            LR = LogisticRegression(solver='liblinear', random_state=42, C=c)
            LR.fit(Xtrain, ytrain)
            acc_train_lr.append(LR.score(Xtrain, ytrain))
            acc_val_lr.append(LR.score(Xval, yval))
```

```
In [ ]: print(f"Best C: {l_C[np.argmax(acc_val_lr)]}")
        print(f"Best accuracy: {100*np.max(acc_val_lr):.2f}%")
        # plot the acc of validation set
        plt.figure(figsize=(7,5), dpi= 100)
        plt.semilogx(l_C, acc_train_lr, color = "#0070C0", label='Training (in-sample)')
        plt.semilogx(l_C, acc_val_lr, color = "#00B050", label='Validation (out-of-sample)')
        plt.legend()
        plt.grid('on')
        plt.xlabel('C')
        plt.ylabel('Accuracy')
        plt.legend()
        plt.title("Accuracy of validation set for logistic regression")
        plt.tight_layout() # Use this to maximize the use of space in the figure
        plt.show()
```

```
Best C: 21.54434690031882
Best accuracy: 85.68%
```



In [ ]:
```
# the best Logistic Regression model
start = time()
best_LR = LogisticRegression(solver='liblinear',\
                             random_state=42, C=21.54).fit(Xtrain, ytrain)
end = time()
LR_train_time = end-start

# save the model
import pickle
pickle.dump(best_LR, open('best_LR.pkl', 'wb'))
```

In [ ]:
```
start = time()
LR_pred_prob = best_LR.predict_proba(Xval)[:,1]
end = time()
LR_pred_time = end-start
```

2. **KNN**

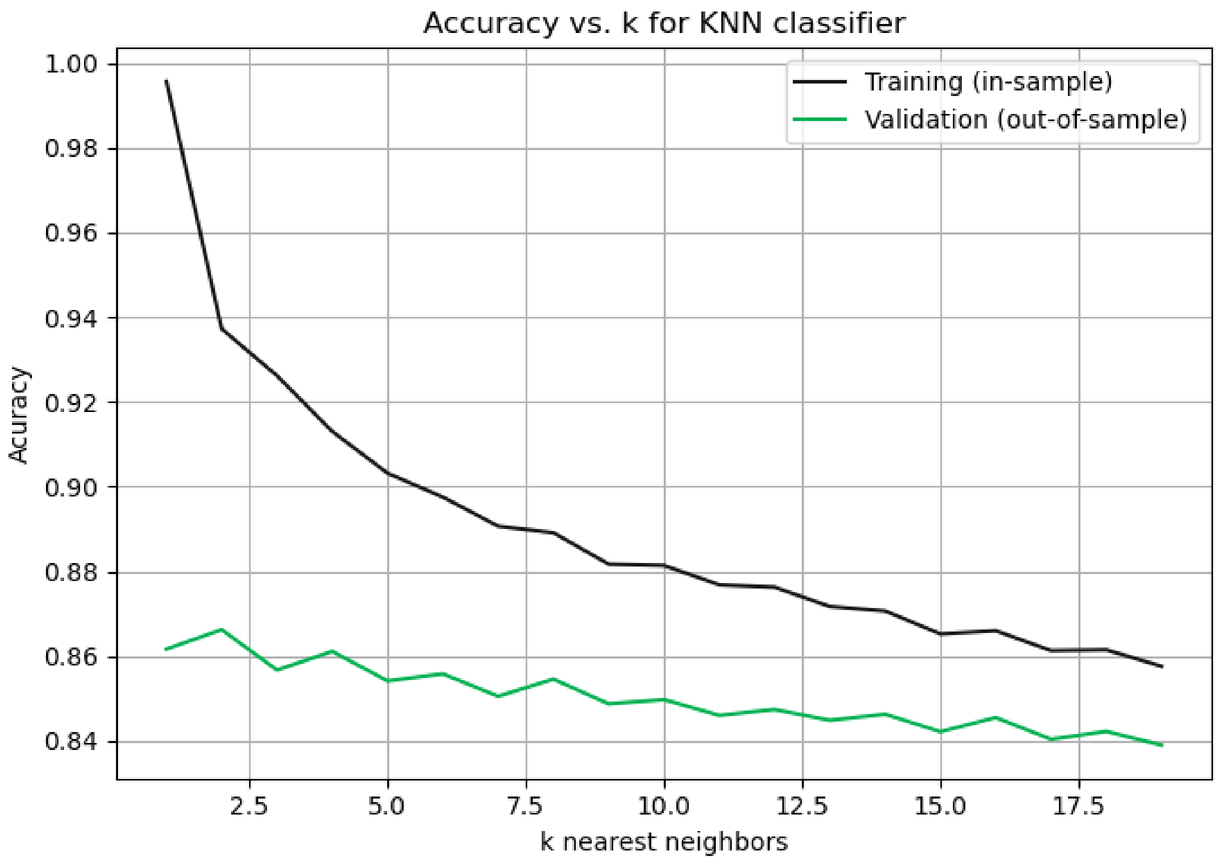
In [ ]:
```
# default setting
start = time()
knn = KNeighborsClassifier()
knn.fit(Xtrain, ytrain)
end = time()
print(f"Accuracy of validation set: {100*knn.score(Xval, yval):.2f}%")
print(f"Time used: {end-start:.2f}s")
```

```
Accuracy of validation set: 85.41%
Time used: 0.52s
```

In [ ]:
```
# tuning n_neighbors
l_n_neighbors = np.arange(1, 20, 1)
acc_train_knn = []
acc_val_knn = []
for n in l_n_neighbors:
    knn = KNeighborsClassifier(n_neighbors=n)
    knn.fit(Xtrain, ytrain)
    acc_train_knn.append(knn.score(Xtrain, ytrain))
    acc_val_knn.append(knn.score(Xval, yval))
```

In [ ]:
```
print(f"Best k: {l_n_neighbors[np.argmax(acc_val_knn)]}")
print(f"Best accuracy: {100*np.max(acc_val_knn):.2f}%")
# plot the error rate of training and test set
plt.figure(figsize=(7,5), dpi= 100)
plt.plot(l_n_neighbors, acc_train_knn, color = "#121619", \
         label='Training (in-sample)')
plt.plot(l_n_neighbors, acc_val_knn, color = '#00B050', \
         label='Validation (out-of-sample)')
plt.legend()
plt.grid('on')
plt.xlabel('k nearest neighbors')
plt.ylabel('Acuracy')
plt.legend()
plt.title("Accuracy vs. k for KNN classifier")
plt.tight_layout() # Use this to maximize the use of space in the figure
plt.show()
```

```
Best k: 2
Best accuracy: 86.62%
```



In [ ]:
```
# the best knn model
```

```
start = time()
best_knn = KNeighborsClassifier(n_neighbors=2).fit(Xtrain, ytrain)
end = time()
knn_train_time = end-start

# save the model
pickle.dump(best_knn, open('best_knn.pkl', 'wb'))
```

In [ ]:
```
start = time()
knn_pred_prob = best_knn.predict_proba(Xval)[:,1]
end = time()
knn_pred_time = end-start
```

3. **Random Forest**

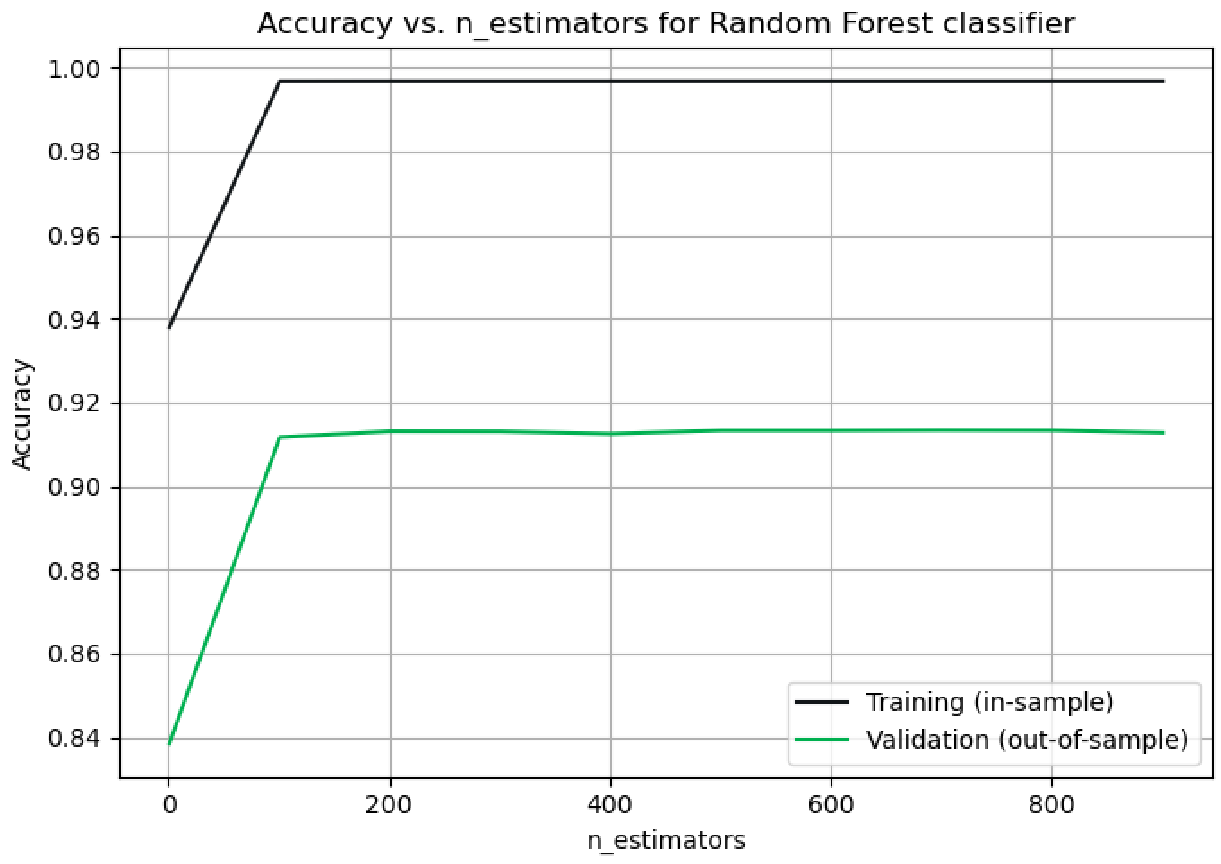
In [ ]:
```
# default setting
start = time()
rfc = RandomForestClassifier(random_state=42)
rfc.fit(Xtrain, ytrain)
end = time()
print(f"Accuracy of validation set: {100*rfc.score(Xval, yval):.2f}%")
print(f"Time used: {end-start:.2f}s")
```

```
Accuracy of validation set: 91.20%
Time used: 52.43s
```

In [ ]:
```
# n_estimators
l_n_estimators = np.arange(1, 1000, 100)
acc_train_rfc = []
acc_val_rfc = []
for n in l_n_estimators:
    rfc = RandomForestClassifier(n_estimators=n, random_state=42)
    rfc.fit(Xtrain, ytrain)
    acc_train_rfc.append(rfc.score(Xtrain, ytrain))
    acc_val_rfc.append(rfc.score(Xval, yval))
```

In [ ]:
```
print(f"Best n_estimators: {l_n_estimators[np.argmax(acc_val_rfc)]}")
print(f"Best accuracy: {100*np.max(acc_val_rfc):.2f}%")
# plot the error rate of training and test set
plt.figure(figsize=(7,5), dpi= 100)
plt.plot(l_n_estimators, acc_train_rfc, color = "#121619",\
          label='Training (in-sample)')
plt.plot(l_n_estimators, acc_val_rfc, color = '#00B050',\
          label='Validation (out-of-sample)')
plt.legend()
plt.grid('on')
plt.xlabel('n_estimators')
plt.ylabel('Accuracy')
plt.legend()
plt.title("Accuracy vs. n_estimators for Random Forest classifier")
plt.tight_layout() # Use this to maximize the use of space in the figure
plt.show()
```

```
Best n_estimators: 701
Best accuracy: 91.33%
```



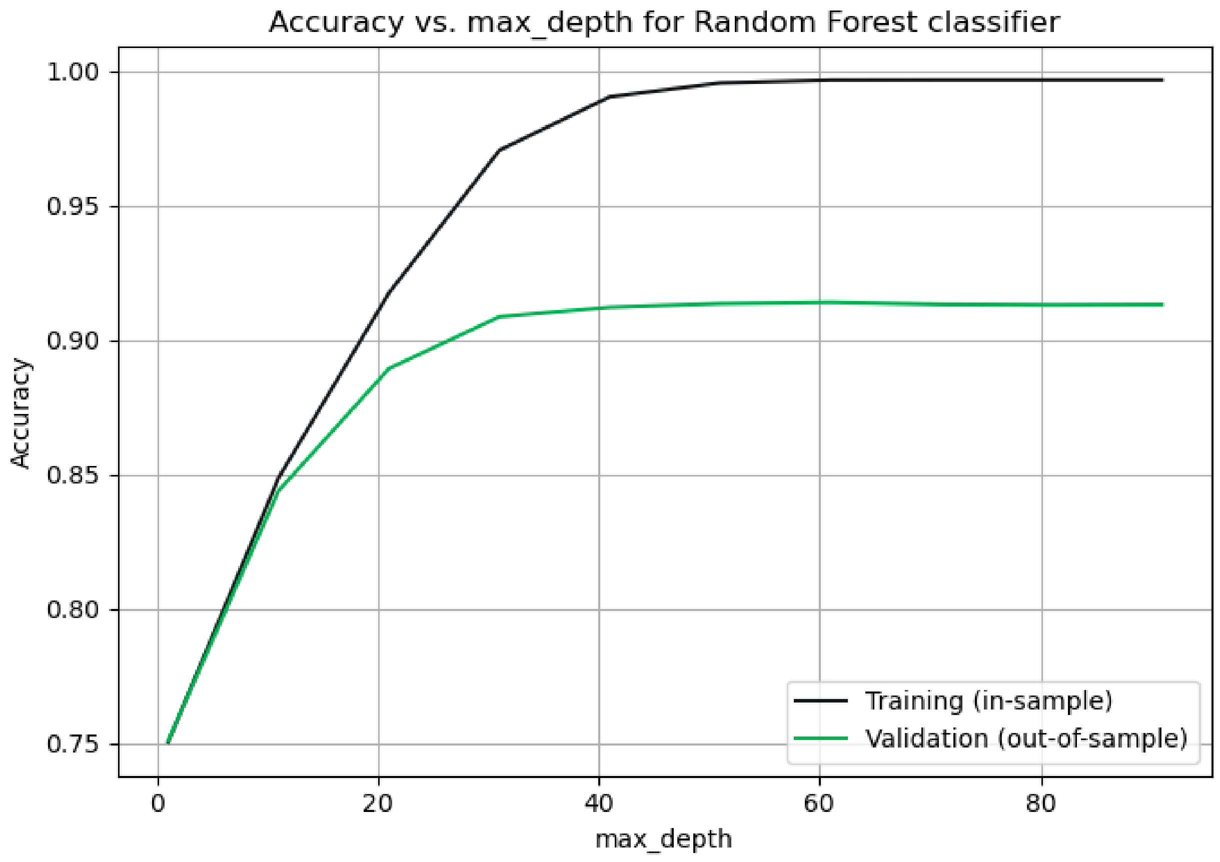
```
In [ ]: rfc_best = RandomForestClassifier(n_estimators=701, random_state=42)
        rfc_best.fit(Xtrain, ytrain)
        print(f"Accuracy of validation set: {100*rfc_best.score(Xval, yval):.2f}%")
```

```
Accuracy of validation set: 91.33%
```

```
In [ ]: # tunnig max_depth
        l_max_depth = np.arange(1, 100, 10)
        acc_train_rfc = []
        acc_val_rfc = []
        for n in l_max_depth:
            rfc = RandomForestClassifier(n_estimators=701, max_depth=n, random_state=42)
            rfc.fit(Xtrain, ytrain)
            acc_train_rfc.append(rfc.score(Xtrain, ytrain))
            acc_val_rfc.append(rfc.score(Xval, yval))
```

```
In [ ]: print(f"Best max_depth: {l_max_depth[np.argmax(acc_val_rfc)]}")
        print(f"Best accuracy: {100*np.max(acc_val_rfc):.2f}%")
        # plot the learning curve of training and test set
        plt.figure(figsize=(7,5), dpi= 100)
        plt.plot(l_max_depth, acc_train_rfc, color = "#121619",\
                  label='Training (in-sample)')
        plt.plot(l_max_depth, acc_val_rfc, color = '#00B050',\
                 label='Validation (out-of-sample)')
        plt.legend()
        plt.grid('on')
        plt.xlabel('max_depth')
        plt.ylabel('Accuracy')
        plt.legend(loc = "best")
        plt.title("Accuracy vs. max_depth for Random Forest classifier")
        plt.tight_layout() # Use this to maximize the use of space in the figure
        plt.show()
```

```
Best max_depth: 61
Best accuracy: 91.40%
```



In [ ]:
```
# the best rfc model
start = time()
best_rfc = RandomForestClassifier(n_estimators=701, \
                                  max_depth=61, random_state=42).fit(Xtrain, ytrain)
end = time()
rfc_train_time = end-start

# save the model
pickle.dump(best_rfc, open('best_rfc.pkl', 'wb'))
```

In [ ]:
```
start = time()
rfc_pred_prob = best_rfc.predict_proba(Xval)[:,1]
end = time()
rfc_pred_time = end-start
```

4. **Neural Networks**

In [ ]:
```
# default setting
start = time()
mlp = MLPClassifier(max_iter=500, random_state=42)
mlp.fit(Xtrain, ytrain)
end = time()
mlp_train_time = end-start
print(f"Accuracy of validation set: {100*mlp.score(Xval, yval):.2f}%")
print(f"Time used: {end-start:.2f}s")

# save the model
pickle.dump(mlp, open('mlp.pkl', 'wb'))
```

```
Accuracy of validation set: 89.27%
Time used: 1372.23s
```

In [ ]:
```
start = time()
mlp_pred_prob = mlp.predict_proba(Xval)[:,1]
end = time()
mlp_pred_time = end-start
```

Training time is too long on local machine and the accuracy is not as high as random forest classifier, so I decided not to use neural networks in further analysis.

5. **Support Vector Machine**

In [ ]:
```
start = time()
svc = SVC(random_state=42, probability=True)
svc.fit(Xtrain, ytrain)
end = time()
svc_train_time = end-start

print(f"Accuracy of validation set: {100*svc.score(Xval, yval):.2f}%")
print(f"Time used: {end-start:.2f}s")

# save the model
pickle.dump(svc, open('svc.pkl', 'wb'))
```

```
Accuracy of validation set: 86.59%
Time used: 19736.57s
```

In [ ]:
```
start = time()
svc_pred_prob = svc.predict_proba(Xval)[:,1]
end = time()
svc_pred_time = end-start
```

It requires a long time to train the model. So I didn't decide to use it as the final model.

In [ ]:
```

```

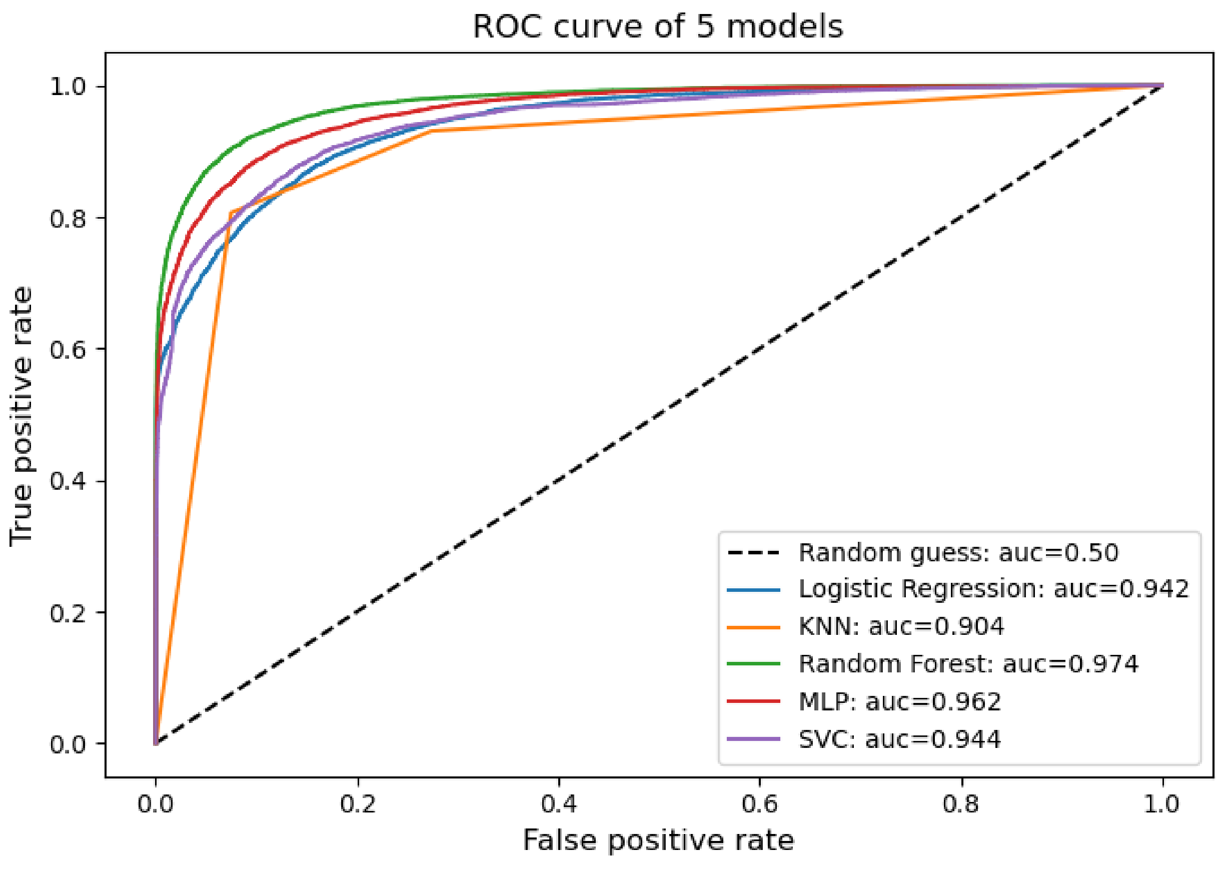
Assess the classification performance AND computational efficiency of the models you selected:

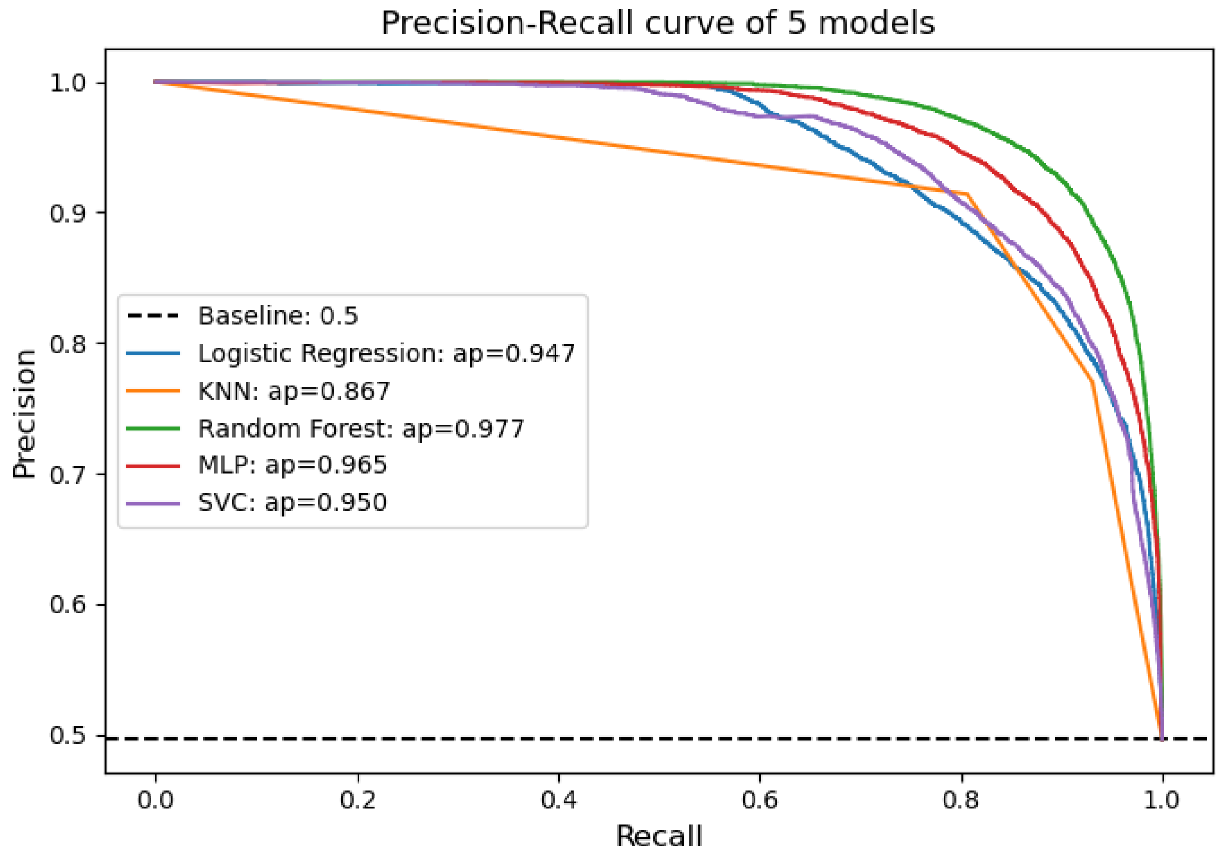
- Plot the ROC curves and PR curves for your models in two plots: one of ROC curves and one of PR curves. For each of these two plots, compare the performance of the models you selected above and trained on the training data, evaluating them on the validation data. Be sure to plot the line representing random guessing on each plot. You should plot all of the model's ROC curves on a single plot and the PR curves on a single plot. One of the models should also be your BEST performing submission on the Kaggle public leaderboard (see below). In the legends of each, include the area under the curve for each model (limit to 3 significant figures). For the ROC curve, this is the AUC; for the PR curve, this is the average precision (AP).

In [ ]:
```python
l_model = [best_LR, best_knn, best_rfc, mlp, svc]
l_model_name = ['Logistic Regression', 'KNN', 'Random Forest', 'MLP', 'SVC']
l_train_time = [LR_train_time, knn_train_time, \
                rfc_train_time, mlp_train_time, svc_train_time]
l_pred_time = [LR_pred_time, knn_pred_time, rfc_pred_time,\
                mlp_pred_time, svc_pred_time]
l_pred_prob = [LR_pred_prob, knn_pred_prob, rfc_pred_prob, \
                mlp_pred_prob, svc_pred_prob]
```

In [ ]:
```python
# iterate through the 4 models and store info for plotting
l_fpr = []
l_tpr = []
l_auc = []
l_precision = []
l_recall = []
l_ap = []
for pred_prob in l_pred_prob:
    fpr_, tpr_, _ = roc_curve(yval, pred_prob)
    l_fpr.append(fpr_)
    l_tpr.append(tpr_)
    auc_ = roc_auc_score(yval, pred_prob)
    l_auc.append(auc_)
    # store info for plotting pr curve
    precision, recall, _ = precision_recall_curve(yval, pred_prob)
    l_precision.append(precision)
    l_recall.append(recall)
    l_ap.append(average_precision_score(yval, pred_prob))
```

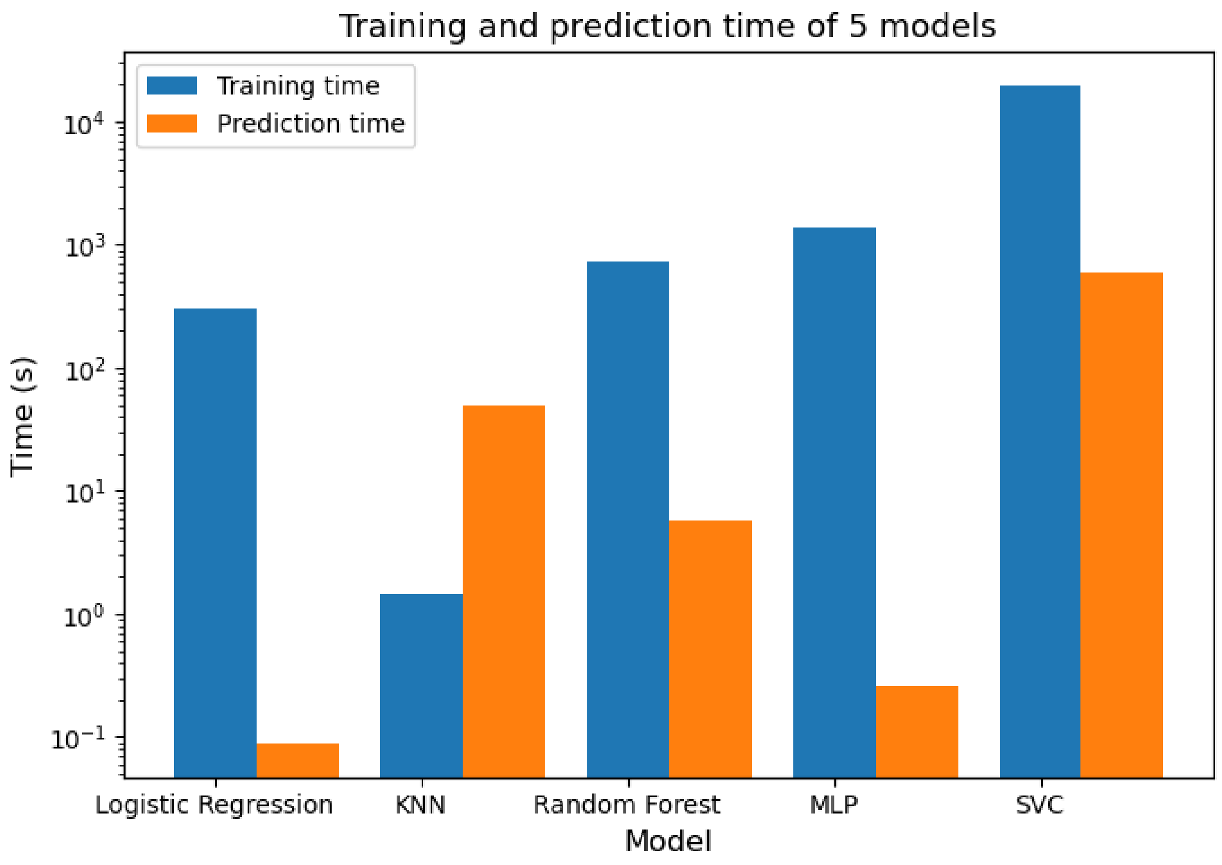
In [ ]:
```python
plt.figure(figsize=(7, 5), dpi=100)
plt.plot([0, 1], [0, 1], 'k--', label='Random guess: auc=0.50')
for fpr, tpr, auc, model_name in zip(l_fpr, l_tpr, l_auc,l_model_name):
    plt.plot(fpr, tpr, label=f"{model_name}: auc={auc:.3f}")
    plt.xlabel('False positive rate', fontsize=12)
    plt.ylabel('True positive rate', fontsize=12)
    plt.title('ROC curve of 5 models', fontsize=13)
    plt.legend()
plt.tight_layout()
plt.show()

# pr curve
plt.figure(figsize=(7, 5), dpi=100)
pr_baseline = (yval==1).sum() / len(yval)
plt.axhline(pr_baseline, ls='--', color='k', \
            label=f"Baseline: {round(pr_baseline,2)}")
for precision, recall, ap, model_name in \
                    zip(l_precision, l_recall, l_ap, l_model_name):
    plt.plot(recall, precision, label=f"{model_name}: ap={ap:.3f}")
    plt.xlabel('Recall', fontsize=12)
    plt.ylabel('Precision', fontsize=12)
    plt.title('Precision-Recall curve of 5 models', fontsize=13)
    plt.legend()
plt.tight_layout()
plt.show()
```





- As you train and validate each model time how long it takes to train and validate in each case and create a plot that shows both the training and prediction time for each model included in the ROC and PR curves.

```
In [ ]: # plot training and prediction time
        plt.figure(figsize=(7, 5), dpi=100)
        # change y to log scale
        # separate the bar chart into 2 parts
        plt.yscale('log')
        plt.bar(l_model_name, l_train_time, label='Training time', width=0.4)
        plt.bar([i + 0.4 for i in range(len(l_model_name))], \
                l_pred_time, label='Prediction time', width=0.4)
        plt.xlabel('Model', fontsize=12)
        plt.ylabel('Time (s)', fontsize=12)
        plt.title('Training and prediction time of 5 models', fontsize=13)
        plt.legend()
        plt.tight_layout()
        plt.show()
```



- Describe:
    - Your process of model selection and hyperparameter tuning
    - Which model performed best and your process for identifying/selecting it

First of all, I selected five models: logistic regression, KNN, random forest, neural networks, and support vector machine. I began with using default setting to see the performance of each model. Then I performed hyperparameter training on each model. Since I ran the training locally, I didn't use grid search or random search, just to save time. For logistic regression, I tuned the regularization parameter C. For KNN, I tuned the number of neighbors. For random forest, I tuned the number of trees and the maximum depth of each tree. For neural network and support vector machine, it took more time than I expected to train one set of hyperparameters, and the accuracy didn't improve much compared to random forest classifier. So I didn't tune the hyperparameters for these two models.

Among the five models, **random forest classifier** has the best performance on validation dataset. The AUC is 0.974, AP is 0.977, and the accuracy is 91.40%, which are the highest among the five. It is also above the benchmark score on the public leaderboard on Kaggle. The training time is about 10 minutes and the prediction takes 5 seconds, which is not long compared to other models. So I decided to use random forest classifier as the final model.

(d) Apply your model "in practice". Make *at least* 5 submissions of different model results to the competition (more submissions are encouraged and you can submit up to 5 per day!). These do not need to be the same that you report on above, but you should select your *most competitive* models.

- Produce submissions by applying your model on the test data.
- Be sure to RETRAIN YOUR MODEL ON ALL LABELED TRAINING AND VALIDATION DATA before making your predictions on the test data for submission. This will help to maximize your performance on the test data.
- In order to get full credit on this problem you must achieve an AUC on the Kaggle public leaderboard above the "Benchmark" score on the public leaderboard.

In [ ]:
```
# fill missing values with 0
X_test_ohe.fillna(0, inplace=True)
assert X_test_ohe.isnull().sum().sum() == 0

# standardize the test set
X_test_ohe = stdscalar.transform(X_test_ohe)

# load the saved model
best_rfc = pickle.load(open('best_rfc.pkl', 'rb'))
best_LR = pickle.load(open('best_LR.pkl', 'rb'))
best_knn = pickle.load(open('best_knn.pkl', 'rb'))
mlp = pickle.load(open('mlp.pkl', 'rb'))
svc = pickle.load(open('svc.pkl', 'rb'))

# fit on the whole training set
best_rfc.fit(X_train_plus_val, y_train_plus_val)
best_LR.fit(X_train_plus_val, y_train_plus_val)
best_knn.fit(X_train_plus_val, y_train_plus_val)
mlp.fit(X_train_plus_val, y_train_plus_val)
svc.fit(X_train_plus_val, y_train_plus_val)

# calculate prediction probability
rfc_fit_all_pred_prob = best_rfc.predict_proba(X_test_ohe)[:, 1]
LR_fit_all_pred_prob = best_LR.predict_proba(X_test_ohe)[:, 1]
knn_fit_all_pred_prob = best_knn.predict_proba(X_test_ohe)[:, 1]
mlp_fit_all_pred_prob = mlp.predict_proba(X_test_ohe)[:, 1]
svc_fit_all_pred_prob = svc.predict_proba(X_test_ohe)[:, 1]
```

```
# create submission file
create_submission(rfc_fit_all_pred_prob, 'submit.csv')
create_submission(LR_fit_all_pred_prob, 'submit_LR.csv')
create_submission(knn_fit_all_pred_prob, 'submit_knn.csv')
create_submission(mlp_fit_all_pred_prob, 'submit_mlp.csv')
create_submission(svc_fit_all_pred_prob, 'submit_svc.csv')
```

In [ ]:

2

[25 points] Clustering

Clustering can be used to reveal structure between samples of data and assign group membership to similar groups of samples. This exercise will provide you with experience applying clustering algorithms and comparing these techniques on various datasets to experience the pros and cons of these approaches when the structure of the data being clustered varies. For this exercise, we'll explore clustering in two dimensions to make the results more tangible, but in practice these approaches can be applied to any number of dimensions.

Note: For each set of plots across the five datasets, please create subplots within a single figure (for example, when applying DBSCAN - please show the clusters resulting from DBSCAN as a single figure with one subplot for each dataset). This will make comparison easier.

(a) Run K-means and choose the number of clusters. Five datasets are provided for you below and the code to load them below.

- Scatterplot each dataset
- For each dataset run the k-means algorithm for values of k ranging from 1 to 10 and for each plot the "elbow curve" where you plot dissimilarity in each case. Here, you can measure dissimilarity using the within-cluster sum-of-squares, which in sklean is known as "inertia" and can be accessed through the `inertia_` attribute of a fit KMeans class instance.
- For each dataset, where is the elbow in the curve of within-cluster sum-of-squares and why? Is the elbow always clearly visible? When it's not clear, you will have to use your judgment in terms of selecting a reasonable number of clusters for the data. *There are also other metrics you can use to explore to measure the quality of cluster fit (but do not have to for this assignment) including the silhouette score, the Calinski-Harabasz index, and the Davies-Bouldin, to name a few within sklearn alone. However, assessing the quality of fit without "preferred" cluster assignments to compare against (that is, in a truly unsupervised manner) is challenging because measuring cluster fit quality is typically poorly-defined and doesn't generalize across all types of inter- and intra-cluster variation.*
- Plot your clustered data (different color for each cluster assignment) for your best k-means fit determined from both the elbow curve and your judgment for each dataset and your inspection of the dataset.

(b) Apply DBSCAN. Vary the `eps` and `min_samples` parameters to get as close as you can to having the same number of clusters as your choices with K-means. In this case, the black points are points that were not assigned to clusters.

(c) Apply Spectral Clustering. Select the same number of clusters as selected by k-means.

(d) Comment on the strengths and weaknesses of each approach. In particular, mention:

- Which technique worked "best" and "worst" (as defined by matching how human intuition would cluster the data) on each dataset?
- How much effort was required to get good clustering for each method (how much parameter tuning needed to be done)?

Note: For these clustering plots in this question, do NOT include legends indicating cluster assignment; instead, just make sure the cluster assignments are clear from the plot (e.g. different colors for each cluster)

Code is provided below for loading the datasets and for making plots with the clusters as distinct colors

In [ ]:
```
#################################
# Load the data
#################################
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons

# Create / load the datasets:
n_samples = 1500
X0, _ = make_blobs(n_samples=n_samples, centers=2, n_features=2, random_state=0)
X1, _ = make_blobs(n_samples=n_samples, centers=5, n_features=2, random_state=0)

random_state = 170
X, y = make_blobs(n_samples=n_samples, random_state=random_state, cluster_std=1.3)
transformation = [[0.6, -0.6], [-0.2, 0.8]]
X2 = np.dot(X, transformation)
X3, _ = make_blobs(n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5], \
                   random_state=random_state)
X4, _ = make_moons(n_samples=n_samples, noise=.12)

X = [X0, X1, X2, X3, X4]
# The datasets are X[i], where i ranges from 0 to 4
```

In [ ]:
```
#################################
# Code to plot clusters
#################################
def plot_cluster(ax, data, cluster_assignments):
    '''Plot two-dimensional data clusters

    Parameters
    ----------
    ax : matplotlib axis
        Axis to plot on
    data : list or numpy array of size [N x 2]
        Clustered data
    cluster_assignments : list or numpy array [N]
        Cluster assignments for each point in data
    '''
    clusters = np.unique(cluster_assignments)
    n_clusters = len(clusters)
    for ca in clusters:
        kwargs = {}
        if ca == -1:
            # if samples are not assigned to a cluster
            # (have a cluster assignment of -1, color them gray)
            kwargs = {'color':'gray'}
            n_clusters = n_clusters - 1
        ax.scatter(data[cluster_assignments==ca, 0], \
                   data[cluster_assignments==ca, 1],s=5,alpha=0.5, **kwargs)
        ax.set_xlabel('feature 1')
        ax.set_ylabel('feature 2')
        ax.set_title(f'No. Clusters = {n_clusters}')
        ax.axis('equal')
```

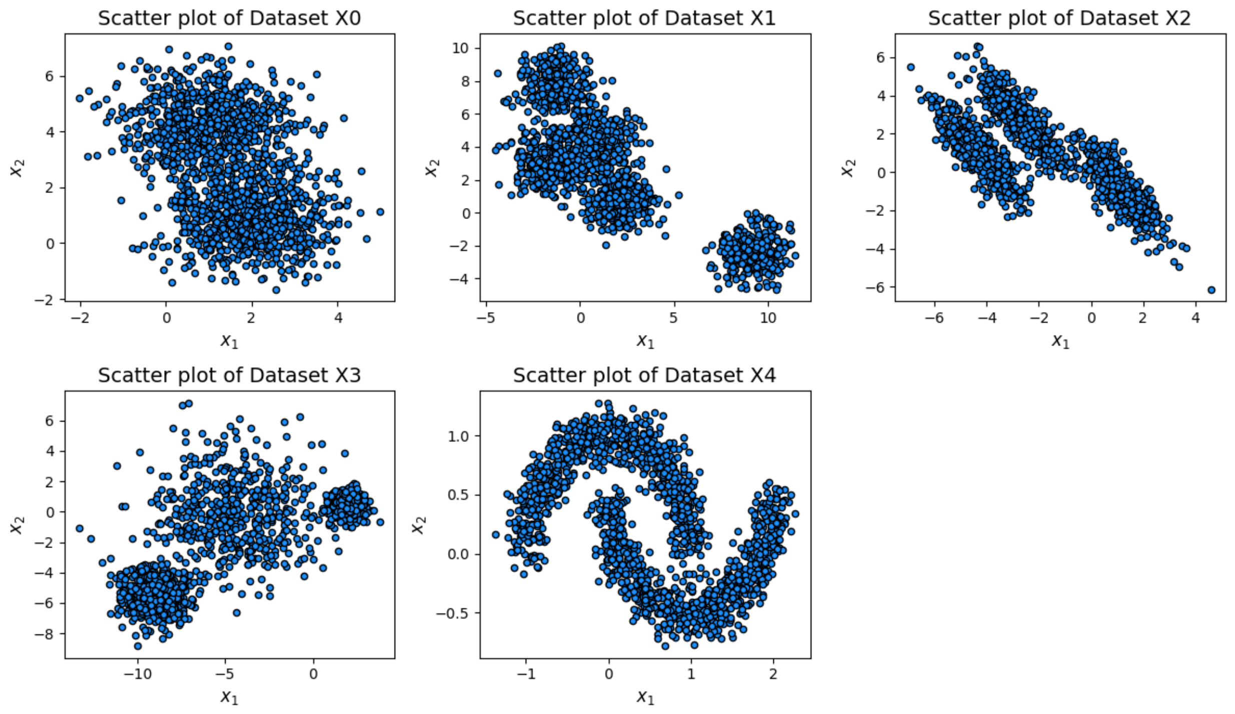
ANSWER

(a) Run K-means and choose the number of clusters.

```
(i) Scatterplot each dataset
```

In [ ]:
```
# plot the training data by class
def plot_scatter(axis, X, title):
    '''
    Plot a scatter plot of the data X
    Parameters:
    axis: a matplotlib axis
    X: a pandas dataframe of size [N x 2]
    title: a string for the title of the plot
    Return:
    None
    '''
    axis.scatter(X[:, 0], X[:, 1], edgecolors='black', s=20, c='dodgerblue')
    axis.set_xlabel('$x_1$', fontsize=12)
    axis.set_ylabel('$x_2$', fontsize=12)
    axis.set_title(title, fontsize=14)
```

In [ ]:
```
fig, axes = plt.subplots(2, 3, figsize=(12, 7))
for i in range(2):
    for j in range(3):
        if i*3+j == 5:
            break
        plot_scatter(axes[i, j], X[i*3+j], \
                     title="Scatter plot of Dataset X{}".format(i*3+j))
        axes[1, 2].axis('off')
        plt.tight_layout()
```

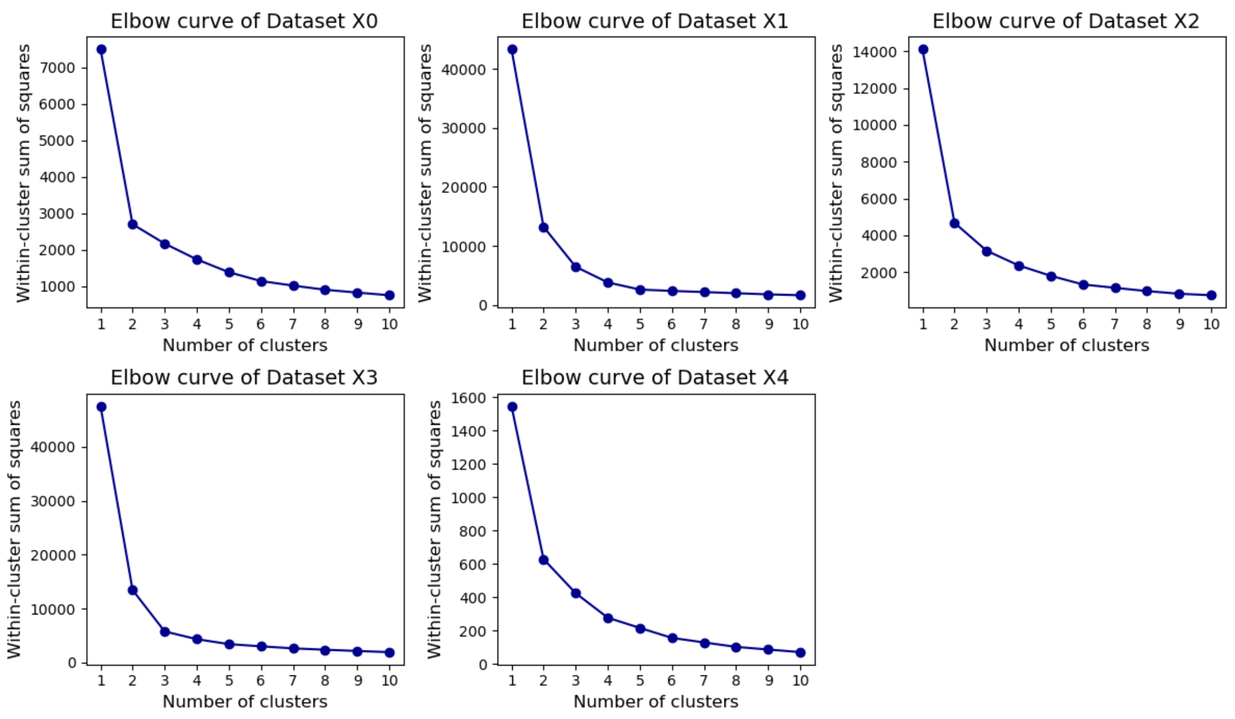


(ii) Run k-means algorithm for values of k ranging from 1 to 10 and for each plot the "elbow curve".

In [ ]:
```python
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```python
def plot_elbow(axis, X, title):
    '''
    Plot the elbow curve for KMeans clustering
    Parameters
    axis : matplotlib axis, Axis to plot on
    X : list or numpy array of size [N x 2], Data to cluster
    title : string, Title of the plot
    '''
    wscc = []
    for i in range(10):
        kmeans = KMeans(n_clusters=i+1, random_state=42).fit(X)
        wscc.append(kmeans.inertia_)

    # Plot the elbow curve
    axis.plot(range(1, 11), wscc, marker='o', linestyle='-', color='darkblue')
    axis.set_xticks(range(1, 11))
    axis.set_xlabel('Number of clusters', fontsize=12)
    axis.set_ylabel('Within-cluster sum of squares', fontsize=12)
    axis.set_title(title, fontsize=14)
```

In [ ]:
```python
# plot clustered data
fig, axes = plt.subplots(2, 3, figsize=(12, 7))
for i in range(2):
    for j in range(3):
        if i*3+j == 5:
            break
        plot_elbow(axes[i, j], X[i*3+j], \
                   title="Elbow curve of Dataset X{}".format(i*3+j))
        axes[1, 2].axis('off')
        plt.tight_layout()
```



```
(iii) For each dataset, where is the elbow in the curve of within-
cluster sum-of-squares and why? Is the elbow always clearly visible?
When it's not clear, you will have to use your judgment in terms of
selecting a reasonable number of clusters for the data.
```

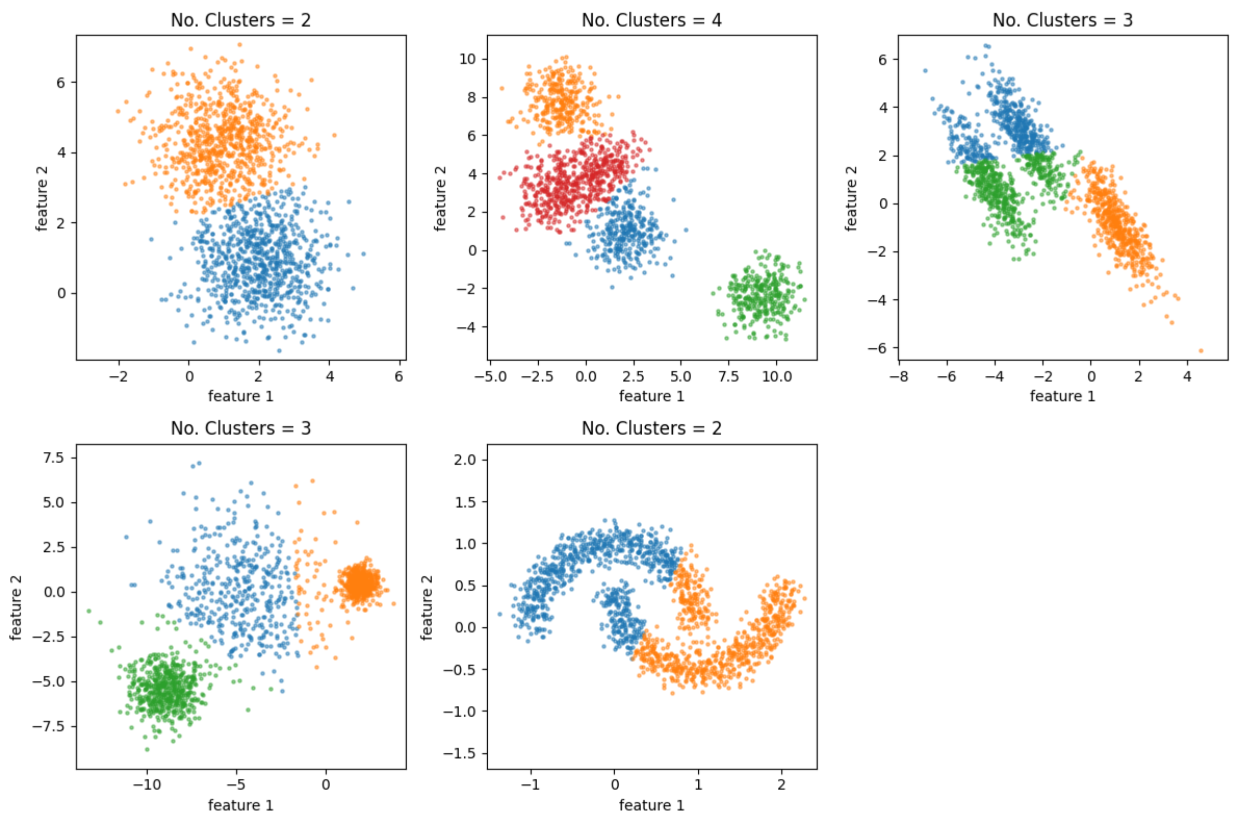
- For Dataset **X0**, the best k **is 2** from the elbow curve because the decrease in WCSS begins to level off after $k = 2$
- For Dataset **X1**, the best k is not very clear from the elbow curve, but I choose $k = 4$ using my judgement because adding more clusters to the model no longer leads to significant improvements in the clustering quality after $k = 4$
- For Dataset **X2**, the best k is not very clear from the elbow curve, but I choose $k = 3$ using my judgement. After the number of clusters reaches four ($k = 3$), further increases in the number of clusters do not result in significant decrease in WCSS.
- For Dataset **X3**, the best k **is 3** from the elbow curve since the level of decrease in WCSS is much less after $k = 3$
- For Dataset **X4**, the best k is not very clear from the elbow curve, but I choose $k = 2$ using my judgement. The decrease of WCSS begins to slow down after $k = 2$.

```
(iv) Plot your clustered data (different color for each cluster
assignment) for your best $k$-means fit determined from both the
elbow curve and your judgment for each dataset and your inspection of
the dataset.
```

```
In [ ]: def pred_y(X, n_clusters):
            '''
            Parameters
            X : list or numpy array of size [N x 2]
            n_clusters : int, Number of clusters to use for KMeans clustering

            Returns
            y_pred : list or numpy array [N]'''
            kmeans = KMeans(n_clusters=n_clusters, random_state=42).fit(X)
            return kmeans.predict(X)
```

```
In [ ]: # store cluster assignments for each dataset to a list
        l_cluster_assginments = []
        l_cluster_assginments.append(pred_y(X[0], 2))
        l_cluster_assginments.append(pred_y(X[1], 4))
        l_cluster_assginments.append(pred_y(X[2], 3))
        l_cluster_assginments.append(pred_y(X[3], 3))
        l_cluster_assginments.append(pred_y(X[4], 2))
```

```
In [ ]: # plot clustered data
        fig, axes = plt.subplots(2, 3, figsize=(12, 8))
        for i in range(2):
            for j in range(3):
                if i*3+j == 5:
                    break
                plot_cluster(axes[i,j], X[i*3+j], \
                             cluster_assignments=l_cluster_assginments[i*3+j])
                axes[1, 2].axis('off')
                plt.tight_layout()
```



After choosing the best k from both the elbow curve and my judgement, KMeans clustering performs well on dataset $X0$, $X1$, $X3$, but it struggles with correctly clustering datasets $X2$ and $X4$, leading to some confusion.

(b) Apply DBSCAN. Vary the `eps` and `min_samples` parameters to get as close as you can to having the same number of clusters as your choices with K-means. In this case, the black points are points that were not assigned to clusters.

In [ ]:
```
from sklearn.cluster import DBSCAN
```

Dataset $X0$

In [ ]:
```
def dbscan_learning_curve(eps_range, min_samples_range, X):
    '''
    Parameters
    eps_range : list, Range of eps to use
    min_samples_range : list, Range of min_samples to use
    X: dataset to fit
    '''
    l_n_clusters = []
    l_n_noise = []
    for eps in eps_range:
        for min_samples in min_samples_range:
            db = DBSCAN(eps=eps, min_samples=min_samples).fit(X)
            labels = db.labels_
            n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
            n_noise_ = list(labels).count(-1)
            l_n_clusters.append(n_clusters_)
            l_n_noise.append(n_noise_)
            print('eps = {}, min_samples = {}'.format(eps, min_samples))
            print("Estimated number of clusters: %d" % n_clusters_)
            print("Estimated number of noise points: %d" % n_noise_)
```

In [ ]:
```
dbscan_learning_curve(np.linspace(0.01, 1, 10), range(1, 80, 20), X[0])
```

```
eps = 0.01, min_samples = 1
Estimated number of clusters: 1482
Estimated number of noise points: 0
eps = 0.01, min_samples = 21
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 41
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 61
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.12, min_samples = 1
Estimated number of clusters: 388
Estimated number of noise points: 0
eps = 0.12, min_samples = 21
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.12, min_samples = 41
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.12, min_samples = 61
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23, min_samples = 1
Estimated number of clusters: 79
Estimated number of noise points: 0
eps = 0.23, min_samples = 21
Estimated number of clusters: 5
Estimated number of noise points: 1072
eps = 0.23, min_samples = 41
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23, min_samples = 61
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.34, min_samples = 1
Estimated number of clusters: 29
Estimated number of noise points: 0
eps = 0.34, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 362
eps = 0.34, min_samples = 41
Estimated number of clusters: 2
Estimated number of noise points: 967
eps = 0.34, min_samples = 61
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.45, min_samples = 1
Estimated number of clusters: 12
Estimated number of noise points: 0
eps = 0.45, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 129
eps = 0.45, min_samples = 41
Estimated number of clusters: 2
Estimated number of noise points: 380
```

```
eps = 0.45, min_samples = 61
Estimated number of clusters: 2
Estimated number of noise points: 742
eps = 0.56, min_samples = 1
Estimated number of clusters: 5
Estimated number of noise points: 0
eps = 0.56, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 54
eps = 0.56, min_samples = 41
Estimated number of clusters: 1
Estimated number of noise points: 145
eps = 0.56, min_samples = 61
Estimated number of clusters: 2
Estimated number of noise points: 304
eps = 0.67, min_samples = 1
Estimated number of clusters: 3
Estimated number of noise points: 0
eps = 0.67, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 22
eps = 0.67, min_samples = 41
Estimated number of clusters: 1
Estimated number of noise points: 70
eps = 0.67, min_samples = 61
Estimated number of clusters: 1
Estimated number of noise points: 118
eps = 0.78, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.78, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 9
eps = 0.78, min_samples = 41
Estimated number of clusters: 1
Estimated number of noise points: 29
eps = 0.78, min_samples = 61
Estimated number of clusters: 1
Estimated number of noise points: 59
eps = 0.89, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 2
eps = 0.89, min_samples = 41
Estimated number of clusters: 1
Estimated number of noise points: 11
eps = 0.89, min_samples = 61
Estimated number of clusters: 1
Estimated number of noise points: 25
eps = 1.0, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 1.0, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
```

```
eps = 1.0, min_samples = 41
Estimated number of clusters: 1
Estimated number of noise points: 4
eps = 1.0, min_samples = 61
Estimated number of clusters: 1
Estimated number of noise points: 11
```

When eps = 0.56, min_samples = 61, estimated number of clusters is 2, and the estimated number of noise points is the smallest (304).

Dataset $X1$

```
In [ ]: dbscan_learning_curve(np.linspace(0.01, 2, 10), range(1, 80, 10), X[1])
```

```
eps = 0.01, min_samples = 1
Estimated number of clusters: 1492
Estimated number of noise points: 0
eps = 0.01, min_samples = 11
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 21
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 31
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 41
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 51
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 61
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 71
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23111111111111113, min_samples = 1
Estimated number of clusters: 217
Estimated number of noise points: 0
eps = 0.23111111111111113, min_samples = 11
Estimated number of clusters: 18
Estimated number of noise points: 1216
eps = 0.23111111111111113, min_samples = 21
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23111111111111113, min_samples = 31
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23111111111111113, min_samples = 41
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23111111111111113, min_samples = 51
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23111111111111113, min_samples = 61
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23111111111111113, min_samples = 71
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.4522222222222225, min_samples = 1
Estimated number of clusters: 33
Estimated number of noise points: 0
eps = 0.4522222222222225, min_samples = 11
Estimated number of clusters: 4
Estimated number of noise points: 164
eps = 0.4522222222222225, min_samples = 21
Estimated number of clusters: 4
Estimated number of noise points: 492
```

```
eps = 0.45222222222222225, min_samples = 31
Estimated number of clusters: 8
Estimated number of noise points: 986
eps = 0.45222222222222225, min_samples = 41
Estimated number of clusters: 1
Estimated number of noise points: 1448
eps = 0.45222222222222225, min_samples = 51
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.45222222222222225, min_samples = 61
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.45222222222222225, min_samples = 71
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.6733333333333333, min_samples = 1
Estimated number of clusters: 11
Estimated number of noise points: 0
eps = 0.6733333333333333, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 33
eps = 0.6733333333333333, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 93
eps = 0.6733333333333333, min_samples = 31
Estimated number of clusters: 3
Estimated number of noise points: 174
eps = 0.6733333333333333, min_samples = 41
Estimated number of clusters: 4
Estimated number of noise points: 339
eps = 0.6733333333333333, min_samples = 51
Estimated number of clusters: 4
Estimated number of noise points: 517
eps = 0.6733333333333333, min_samples = 61
Estimated number of clusters: 5
Estimated number of noise points: 791
eps = 0.6733333333333333, min_samples = 71
Estimated number of clusters: 3
Estimated number of noise points: 1279
eps = 0.8944444444444445, min_samples = 1
Estimated number of clusters: 6
Estimated number of noise points: 0
eps = 0.8944444444444445, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 8
eps = 0.8944444444444445, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 18
eps = 0.8944444444444445, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 32
eps = 0.8944444444444445, min_samples = 41
Estimated number of clusters: 3
Estimated number of noise points: 59
eps = 0.8944444444444445, min_samples = 51
Estimated number of clusters: 3
Estimated number of noise points: 90
```

```
eps = 0.8944444444444445, min_samples = 61
Estimated number of clusters: 3
Estimated number of noise points: 161
eps = 0.8944444444444445, min_samples = 71
Estimated number of clusters: 4
Estimated number of noise points: 212
eps = 1.1155555555555556, min_samples = 1
Estimated number of clusters: 4
Estimated number of noise points: 0
eps = 1.1155555555555556, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 2
eps = 1.1155555555555556, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 4
eps = 1.1155555555555556, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 9
eps = 1.1155555555555556, min_samples = 41
Estimated number of clusters: 2
Estimated number of noise points: 11
eps = 1.1155555555555556, min_samples = 51
Estimated number of clusters: 2
Estimated number of noise points: 19
eps = 1.1155555555555556, min_samples = 61
Estimated number of clusters: 2
Estimated number of noise points: 30
eps = 1.1155555555555556, min_samples = 71
Estimated number of clusters: 3
Estimated number of noise points: 44
eps = 1.3366666666666667, min_samples = 1
Estimated number of clusters: 3
Estimated number of noise points: 0
eps = 1.3366666666666667, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 1
eps = 1.3366666666666667, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 1
eps = 1.3366666666666667, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 1
eps = 1.3366666666666667, min_samples = 41
Estimated number of clusters: 2
Estimated number of noise points: 2
eps = 1.3366666666666667, min_samples = 51
Estimated number of clusters: 2
Estimated number of noise points: 3
eps = 1.3366666666666667, min_samples = 61
Estimated number of clusters: 2
Estimated number of noise points: 6
eps = 1.3366666666666667, min_samples = 71
Estimated number of clusters: 2
Estimated number of noise points: 9
eps = 1.557777777777778, min_samples = 1
Estimated number of clusters: 2
Estimated number of noise points: 0
```

```
eps = 1.5577777777777778, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.5577777777777778, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.5577777777777778, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.5577777777777778, min_samples = 41
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.5577777777777778, min_samples = 51
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.5577777777777778, min_samples = 61
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.5577777777777778, min_samples = 71
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.778888888888889, min_samples = 1
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.778888888888889, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.778888888888889, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.778888888888889, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.778888888888889, min_samples = 41
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.778888888888889, min_samples = 51
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.778888888888889, min_samples = 61
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 1.778888888888889, min_samples = 71
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 2.0, min_samples = 1
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 2.0, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 2.0, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 2.0, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 0
```

```
eps = 2.0, min_samples = 41
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 2.0, min_samples = 51
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 2.0, min_samples = 61
Estimated number of clusters: 2
Estimated number of noise points: 0
eps = 2.0, min_samples = 71
Estimated number of clusters: 2
Estimated number of noise points: 0
```

When eps = 0.89, min_samples = 71, estimated number of clusters is 4, and the estimated number of noise points is 212.

Dataset $X2$

In [ ]:
```
dbscan_learning_curve(np.linspace(0.01, 1, 10), range(1, 40, 10), X[2])
```

```
eps = 0.01, min_samples = 1
Estimated number of clusters: 1479
Estimated number of noise points: 0
eps = 0.01, min_samples = 11
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 21
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 31
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.12, min_samples = 1
Estimated number of clusters: 378
Estimated number of noise points: 0
eps = 0.12, min_samples = 11
Estimated number of clusters: 7
Estimated number of noise points: 1413
eps = 0.12, min_samples = 21
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.12, min_samples = 31
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23, min_samples = 1
Estimated number of clusters: 92
Estimated number of noise points: 0
eps = 0.23, min_samples = 11
Estimated number of clusters: 6
Estimated number of noise points: 367
eps = 0.23, min_samples = 21
Estimated number of clusters: 7
Estimated number of noise points: 1063
eps = 0.23, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 1460
eps = 0.34, min_samples = 1
Estimated number of clusters: 28
Estimated number of noise points: 0
eps = 0.34, min_samples = 11
Estimated number of clusters: 4
Estimated number of noise points: 127
eps = 0.34, min_samples = 21
Estimated number of clusters: 3
Estimated number of noise points: 322
eps = 0.34, min_samples = 31
Estimated number of clusters: 3
Estimated number of noise points: 537
eps = 0.45, min_samples = 1
Estimated number of clusters: 9
Estimated number of noise points: 0
eps = 0.45, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 49
eps = 0.45, min_samples = 21
Estimated number of clusters: 3
Estimated number of noise points: 128
```

```
eps = 0.45, min_samples = 31
Estimated number of clusters: 3
Estimated number of noise points: 225
eps = 0.56, min_samples = 1
Estimated number of clusters: 4
Estimated number of noise points: 0
eps = 0.56, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 23
eps = 0.56, min_samples = 21
Estimated number of clusters: 3
Estimated number of noise points: 45
eps = 0.56, min_samples = 31
Estimated number of clusters: 3
Estimated number of noise points: 87
eps = 0.67, min_samples = 1
Estimated number of clusters: 4
Estimated number of noise points: 0
eps = 0.67, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 16
eps = 0.67, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 21
eps = 0.67, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 41
eps = 0.78, min_samples = 1
Estimated number of clusters: 4
Estimated number of noise points: 0
eps = 0.78, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 10
eps = 0.78, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 14
eps = 0.78, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 22
eps = 0.89, min_samples = 1
Estimated number of clusters: 3
Estimated number of noise points: 0
eps = 0.89, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 5
eps = 0.89, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 10
eps = 0.89, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 15
eps = 1.0, min_samples = 1
Estimated number of clusters: 3
Estimated number of noise points: 0
eps = 1.0, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 3
```

```
eps = 1.0, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 6
eps = 1.0, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 11
```

When eps = 0.56, min_samples = 31, estimated number of clusters is 3, and the estimated number of noise points is 87.

Dataset $X3$

In [ ]:
```
dbscan_learning_curve(np.linspace(0.01, 1, 10), range(1, 40, 10), X[3])
```

```
eps = 0.01, min_samples = 1
Estimated number of clusters: 1480
Estimated number of noise points: 0
eps = 0.01, min_samples = 11
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 21
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 31
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.12, min_samples = 1
Estimated number of clusters: 708
Estimated number of noise points: 0
eps = 0.12, min_samples = 11
Estimated number of clusters: 3
Estimated number of noise points: 1246
eps = 0.12, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 1421
eps = 0.12, min_samples = 31
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.23, min_samples = 1
Estimated number of clusters: 351
Estimated number of noise points: 0
eps = 0.23, min_samples = 11
Estimated number of clusters: 5
Estimated number of noise points: 821
eps = 0.23, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 1088
eps = 0.23, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 1175
eps = 0.34, min_samples = 1
Estimated number of clusters: 164
Estimated number of noise points: 0
eps = 0.34, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 616
eps = 0.34, min_samples = 21
Estimated number of clusters: 3
Estimated number of noise points: 750
eps = 0.34, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 1002
eps = 0.45, min_samples = 1
Estimated number of clusters: 90
Estimated number of noise points: 0
eps = 0.45, min_samples = 11
Estimated number of clusters: 9
Estimated number of noise points: 424
eps = 0.45, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 609
```

```
eps = 0.45, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 693
eps = 0.56, min_samples = 1
Estimated number of clusters: 53
Estimated number of noise points: 0
eps = 0.56, min_samples = 11
Estimated number of clusters: 5
Estimated number of noise points: 263
eps = 0.56, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 535
eps = 0.56, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 582
eps = 0.67, min_samples = 1
Estimated number of clusters: 40
Estimated number of noise points: 0
eps = 0.67, min_samples = 11
Estimated number of clusters: 4
Estimated number of noise points: 142
eps = 0.67, min_samples = 21
Estimated number of clusters: 4
Estimated number of noise points: 431
eps = 0.67, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 528
eps = 0.78, min_samples = 1
Estimated number of clusters: 27
Estimated number of noise points: 0
eps = 0.78, min_samples = 11
Estimated number of clusters: 4
Estimated number of noise points: 100
eps = 0.78, min_samples = 21
Estimated number of clusters: 4
Estimated number of noise points: 261
eps = 0.78, min_samples = 31
Estimated number of clusters: 2
Estimated number of noise points: 507
eps = 0.89, min_samples = 1
Estimated number of clusters: 21
Estimated number of noise points: 0
eps = 0.89, min_samples = 11
Estimated number of clusters: 3
Estimated number of noise points: 64
eps = 0.89, min_samples = 21
Estimated number of clusters: 3
Estimated number of noise points: 151
eps = 0.89, min_samples = 31
Estimated number of clusters: 5
Estimated number of noise points: 327
eps = 1.0, min_samples = 1
Estimated number of clusters: 11
Estimated number of noise points: 0
eps = 1.0, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 46
```

```
eps = 1.0, min_samples = 21
Estimated number of clusters: 3
Estimated number of noise points: 106
eps = 1.0, min_samples = 31
Estimated number of clusters: 3
Estimated number of noise points: 179
```

When eps = 0.89, min_samples = 11, estimated number of clusters is 3, and the estimated number of noise points is 64.

Dataset $X4$

In [ ]:
```
dbscan_learning_curve(np.linspace(0.01, 1, 10), range(1, 40, 5), X[4])
```

```
eps = 0.01, min_samples = 1
Estimated number of clusters: 1381
Estimated number of noise points: 0
eps = 0.01, min_samples = 6
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 11
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 16
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 21
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 26
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 31
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.01, min_samples = 36
Estimated number of clusters: 0
Estimated number of noise points: 1500
eps = 0.12, min_samples = 1
Estimated number of clusters: 7
Estimated number of noise points: 0
eps = 0.12, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 14
eps = 0.12, min_samples = 11
Estimated number of clusters: 2
Estimated number of noise points: 30
eps = 0.12, min_samples = 16
Estimated number of clusters: 2
Estimated number of noise points: 62
eps = 0.12, min_samples = 21
Estimated number of clusters: 2
Estimated number of noise points: 106
eps = 0.12, min_samples = 26
Estimated number of clusters: 2
Estimated number of noise points: 186
eps = 0.12, min_samples = 31
Estimated number of clusters: 9
Estimated number of noise points: 393
eps = 0.12, min_samples = 36
Estimated number of clusters: 10
Estimated number of noise points: 868
eps = 0.23, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.23, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.23, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 0
```

```
eps = 0.23, min_samples = 16
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.23, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.23, min_samples = 26
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.23, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.23, min_samples = 36
Estimated number of clusters: 1
Estimated number of noise points: 1
eps = 0.34, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.34, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.34, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.34, min_samples = 16
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.34, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.34, min_samples = 26
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.34, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.34, min_samples = 36
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.45, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.45, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.45, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.45, min_samples = 16
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.45, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.45, min_samples = 26
Estimated number of clusters: 1
Estimated number of noise points: 0
```

```
eps = 0.45, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.45, min_samples = 36
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.56, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.56, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.56, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.56, min_samples = 16
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.56, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.56, min_samples = 26
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.56, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.56, min_samples = 36
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.67, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.67, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.67, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.67, min_samples = 16
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.67, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.67, min_samples = 26
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.67, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.67, min_samples = 36
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.78, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
```

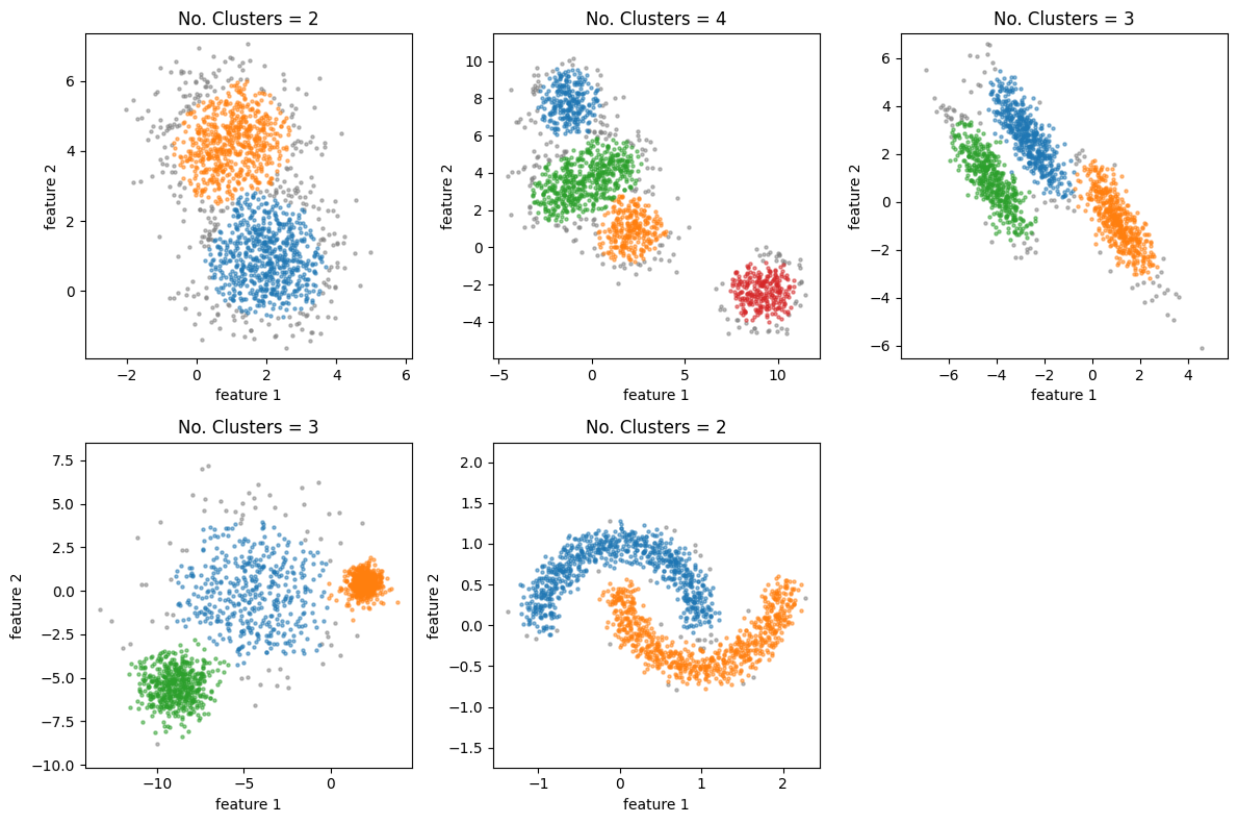
```
eps = 0.78, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.78, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.78, min_samples = 16
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.78, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.78, min_samples = 26
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.78, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.78, min_samples = 36
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 16
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 26
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 0.89, min_samples = 36
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 1.0, min_samples = 1
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 1.0, min_samples = 6
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 1.0, min_samples = 11
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 1.0, min_samples = 16
Estimated number of clusters: 1
Estimated number of noise points: 0
```

```
eps = 1.0, min_samples = 21
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 1.0, min_samples = 26
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 1.0, min_samples = 31
Estimated number of clusters: 1
Estimated number of noise points: 0
eps = 1.0, min_samples = 36
Estimated number of clusters: 1
Estimated number of noise points: 0
```

When eps = 0.12, min_samples = 11, estimated number of clusters is 3, and the estimated number of noise points is 30.

In [ ]:
```python
# final DBSCAN clustering for each dataset
db_x0 = DBSCAN(eps=0.56, min_samples=61).fit(X[0])
clusters_x0 = db_x0.fit_predict(X[0])

db_x1 = DBSCAN(eps=0.89, min_samples=71).fit(X[1])
clusters_x1 = db_x1.fit_predict(X[1])

db_x2 = DBSCAN(eps=0.56, min_samples=31).fit(X[2])
clusters_x2 = db_x2.fit_predict(X[2])

db_x3 = DBSCAN(eps=0.89, min_samples=11).fit(X[3])
clusters_x3 = db_x3.fit_predict(X[3])

db_x4 = DBSCAN(eps=0.12, min_samples=11).fit(X[4])
clusters_x4 = db_x4.fit_predict(X[4])
```

In [ ]:
```python
# plot the clustering results
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
plot_cluster(axes[0,0], X[0], cluster_assignments=clusters_x0)
plot_cluster(axes[0,1], X[1], cluster_assignments=clusters_x1)
plot_cluster(axes[0,2], X[2], cluster_assignments=clusters_x2)
plot_cluster(axes[1,0], X[3], cluster_assignments=clusters_x3)
plot_cluster(axes[1,1], X[4], cluster_assignments=clusters_x4)
axes[1, 2].axis('off')
plt.tight_layout()
plt.show()
```



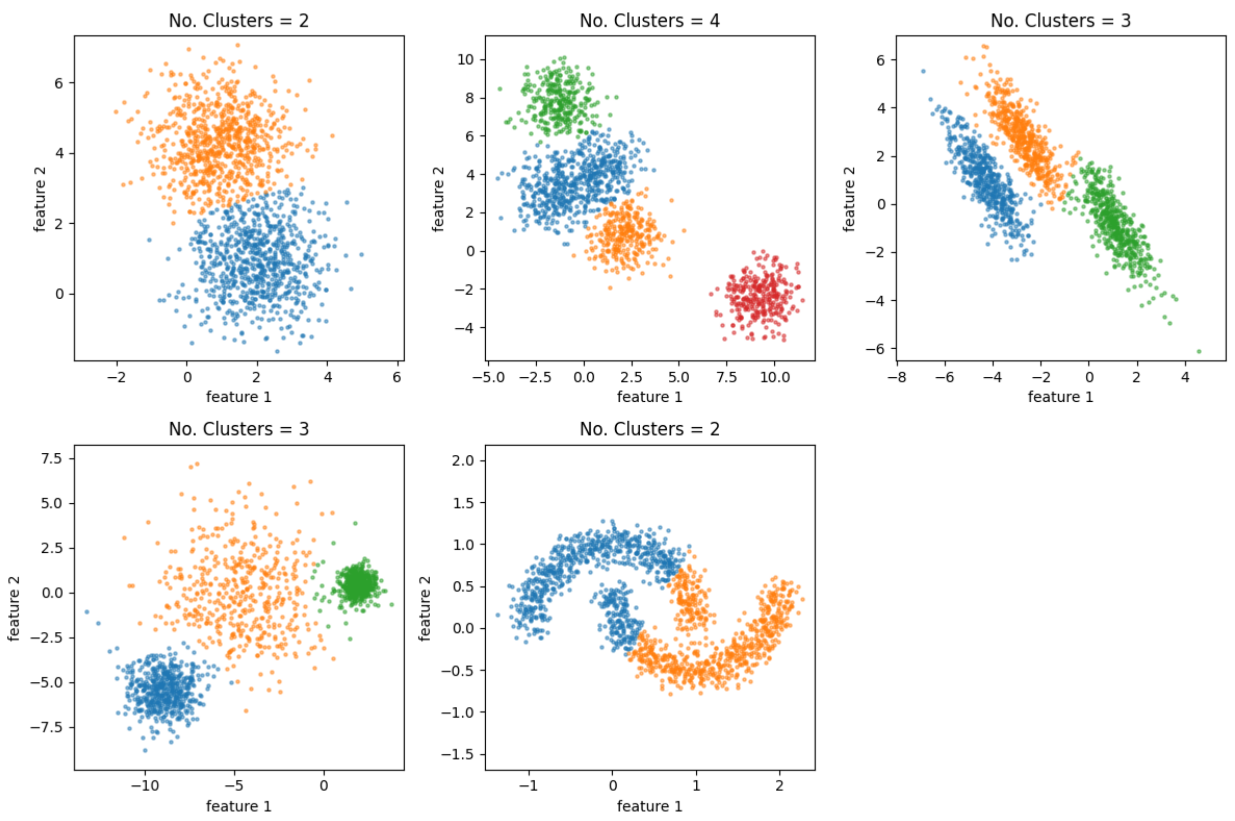
(c) Apply Spectral Clustering. Select the same number of clusters as selected by k-means.

```
In [ ]: from sklearn.cluster import SpectralClustering
```

```
In [ ]: def pred_y(X, n_clusters):
            '''
            Parameters
            X : list or numpy array of size [N x 2]
            n_clusters : int, Number of clusters to use for KMeans clustering

            Returns
            y_pred : list or numpy array [N]'''
            Sclustering = SpectralClustering(n_clusters=n_clusters, random_state=42).fit(X)
            return Sclustering.labels_
```

```
In [ ]: # store cluster assignments for each dataset to a list
        l_cluster_assginments = []
        l_cluster_assginments.append(pred_y(X[0], 2))
        l_cluster_assginments.append(pred_y(X[1], 4))
        l_cluster_assginments.append(pred_y(X[2], 3))
        l_cluster_assginments.append(pred_y(X[3], 3))
        l_cluster_assginments.append(pred_y(X[4], 2))
```

```
In [ ]: # plot clustered data
        fig, axes = plt.subplots(2, 3, figsize=(12, 8))
        for i in range(2):
            for j in range(3):
                if i*3+j == 5:
                    break
                plot_cluster(axes[i,j], X[i*3+j], \
                             cluster_assignments=l_cluster_assginments[i*3+j])
                axes[1, 2].axis('off')
                plt.tight_layout()
```



(d) Comment on the strengths and weaknesses of each approach. In particular, mention:

- Which technique worked "best" and "worst" (as defined by matching how human intuition would cluster the data) on each dataset?
- How much effort was required to get good clustering for each method (how much parameter tuning needed to be done)?

Comparing with KMeans, DBSCAN, and spectral clustering, DBSCAN worked best on the five datasets while KMeans worked the worst. DBSCAN achieved good performance on all datasets, spectral clustering had some trouble clustering dataset $X4$, and KMeans struggles on both dataset $X2$ and dataset $X4$.

- **KMeans**:
  - Strengths:
    1. It converges very quickly.
    2. It excels with clusters of equal variance.
    3. Ituitive, easy to understand and implement.
  - Weaknesses:
    1. It is sensitive to initialization of means.
    2. It requires specifying number of clusters.
    3. It struggles in situations with variation in cluster variance and correlation between features. It also struggles when there are nonlinear boundaries between clusters. Worked 'worst' on the five datasets.

- **DBSCAN**:

  - Strengths:
    1. No need to pre-define number of clusters.
    2. Can find arbitrarily shaped clusters. Worked 'best' on the five datasets.
    3. It is robust to outliers.
  - Weaknesses:
    1. It requires the most effort to train. Need to perform hyperparameter tuning (eps and min_samples).
    2. It cannot handle significant variation in cluster density.
    3. It is not entirely deterministic.
- **Spectral clustering**:

  - Strengths:
    1. It can capture the global structure of the data and often produces good clustering results
    2. No need to perform hyperparameter tuning, easy to implement
    3. It is robust to noise and outliers.
  - Weaknesses:
    1. It requires specifying number of clusters.
    2. It can be computationally expensive for large datasets

```
In [ ]:
```

3

[25 points] Dimensionality reduction and visualization of digits with PCA and t-SNE

(a) Reduce the dimensionality of the data with PCA for data visualization. Load the `scikit-learn` digits dataset (code provided to do this below). Apply PCA and reduce the data (with the associated cluster labels 0-9) into a 2-dimensional space. Plot the data with labels in this two dimensional space (labels can be colors, shapes, or using the actual numbers to represent the data - definitely include a legend in your plot).

(b) Create a plot showing the cumulative fraction of variance explained as you incorporate from 1 through all D principal components of the data (where D is the dimensionality of the data).

- What fraction of variance in the data is UNEXPLAINED by the first two principal components of the data?
- Briefly comment on how this may impact how well-clustered the data are.

You can use the `explained_variance_` attribute of the PCA module in `scikit-learn` to assist with this question

(c) Reduce the dimensionality of the data with t-SNE for data visualization. T-distributed stochastic neighborhood embedding (t-SNE) is a nonlinear dimensionality reduction technique that is particularly adept at embedding the data into lower 2 or 3 dimensional spaces. Apply t-SNE using the `scikit-learn` implementation to the digits dataset and plot it in 2-dimensions (with associated cluster labels 0-9). You may need to adjust the parameters to get acceptable performance. You can read more about how to use t-SNE effectively here.

(d) Briefy compare/contrast the performance of these two techniques.

- Which seemed to cluster the data best and why?
- Notice that while t-SNE has a `fit` method and a `fit_transform` method, these methods are actually identical, and there is no `transform` method. Why is this? What implications does this imply for using this method?

Note: Remember that you typically will not have labels available in most problems.

Code is provided for loading the data below.

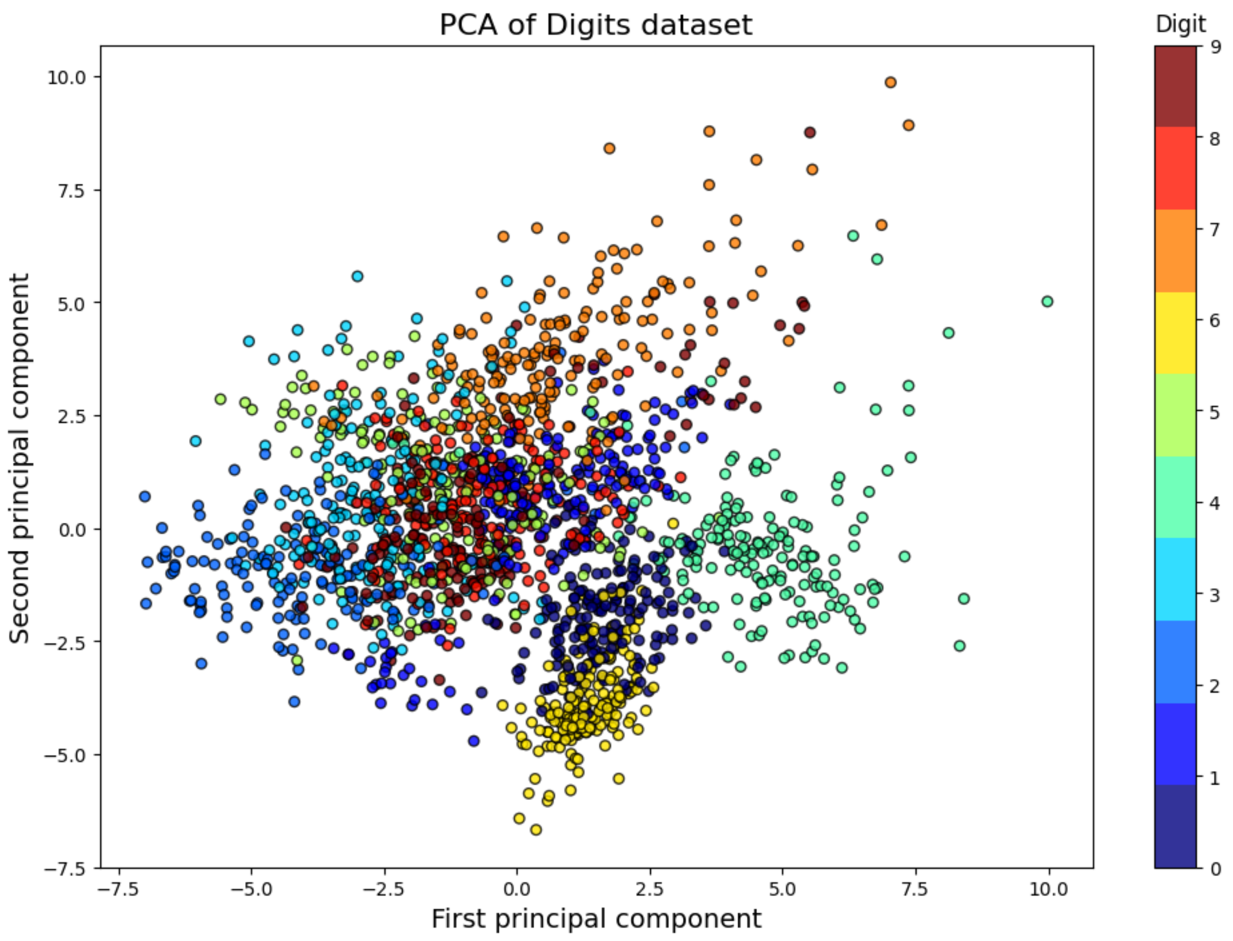
In [ ]:
```
##################################
# Load the data
##################################
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Load dataset
digits = datasets.load_digits()
n_sample = digits.target.shape[0]
n_feature = digits.images.shape[1] * digits.images.shape[2]
X_digits = np.zeros((n_sample, n_feature))
for i in range(n_sample):
    X_digits[i, :] = digits.images[i, :, :].flatten()
y_digits = digits.target
```

ANSWER

(a) Reduce the dimensionality of the data with PCA for data visualization. Apply PCA and reduce the data (with the associated cluster labels 0-9) into a 2-dimensional space. Plot the data with labels in this two dimensional space (labels can be colors, shapes, or using the actual numbers to represent the data - definitely include a legend in your plot).

In [ ]:
```
# standardize the data
from sklearn.preprocessing import StandardScaler
X_digits_std = StandardScaler().fit_transform(X_digits)
```

In [ ]:
```
# fit PCA and get transformed X
pca = PCA(n_components=2, random_state=42).fit(X_digits_std)
X_pca = pca.transform(X_digits_std)

# plotting
plt.figure(figsize=(12,8), dpi=100)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_digits, \
    cmap=plt.cm.get_cmap('jet', 10), s=30, alpha=0.8, edgecolors='black')
plt.colorbar(ticks=range(10))
plt.xlabel('First principal component', fontsize=14)
plt.ylabel('Second principal component', fontsize=14)
plt.text(12, 11, 'Digit', fontsize=12)
plt.title('PCA of Digits dataset', fontsize=16)
plt.show()
```

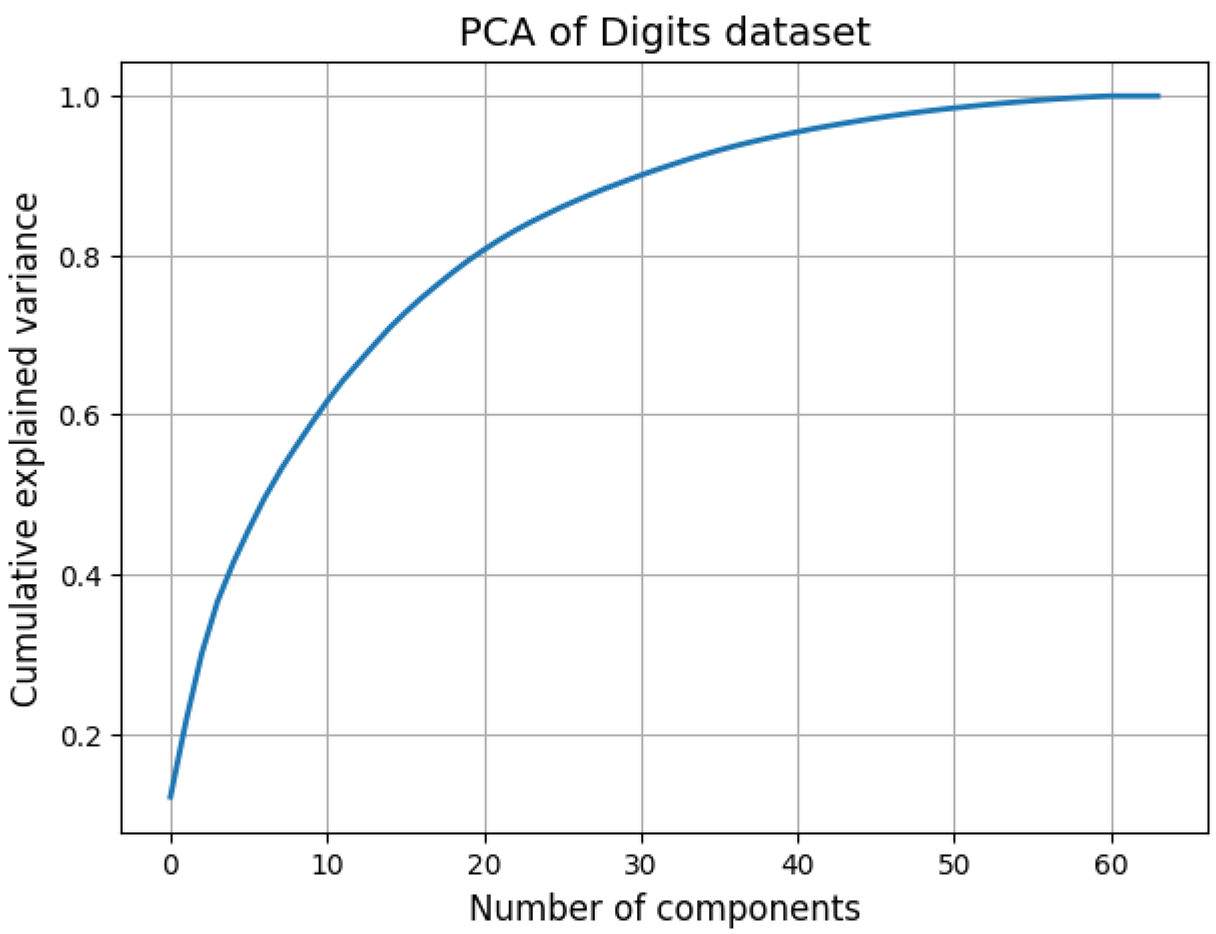


(b) Create a plot showing the cumulative fraction of variance explained as you incorporate from 1 through all D principal components of the data (where D is the dimensionality of the data).

- What fraction of variance in the data is UNEXPLAINED by the first two principal components of the data?
- Briefly comment on how this may impact how well-clustered the data are.

You can use the `explained_variance_` attribute of the PCA module in `scikit-learn` to assist with this question

```
In [ ]: pca = PCA().fit(X_digits_std)
        plt.figure(figsize=(7, 5), dpi=100)
        plt.grid('on')
        plt.plot(np.cumsum(pca.explained_variance_ratio_), linewidth=2)
        plt.xlabel('Number of components', fontsize=12)
        plt.ylabel('Cumulative explained variance', fontsize=12)
        plt.title('PCA of Digits dataset', fontsize=14)
        plt.show()
```



```
In [ ]: pca_two = PCA(n_components=2, random_state=42).fit(X_digits_std)
        var_exp_1, var_exp_2 = pca_two.explained_variance_ratio_
        print(f"Fraction of variance that is UNEXPLAINED by the first two \
            principal components: {(1 - var_exp_1 - var_exp_2)*100:.2f}%")
```

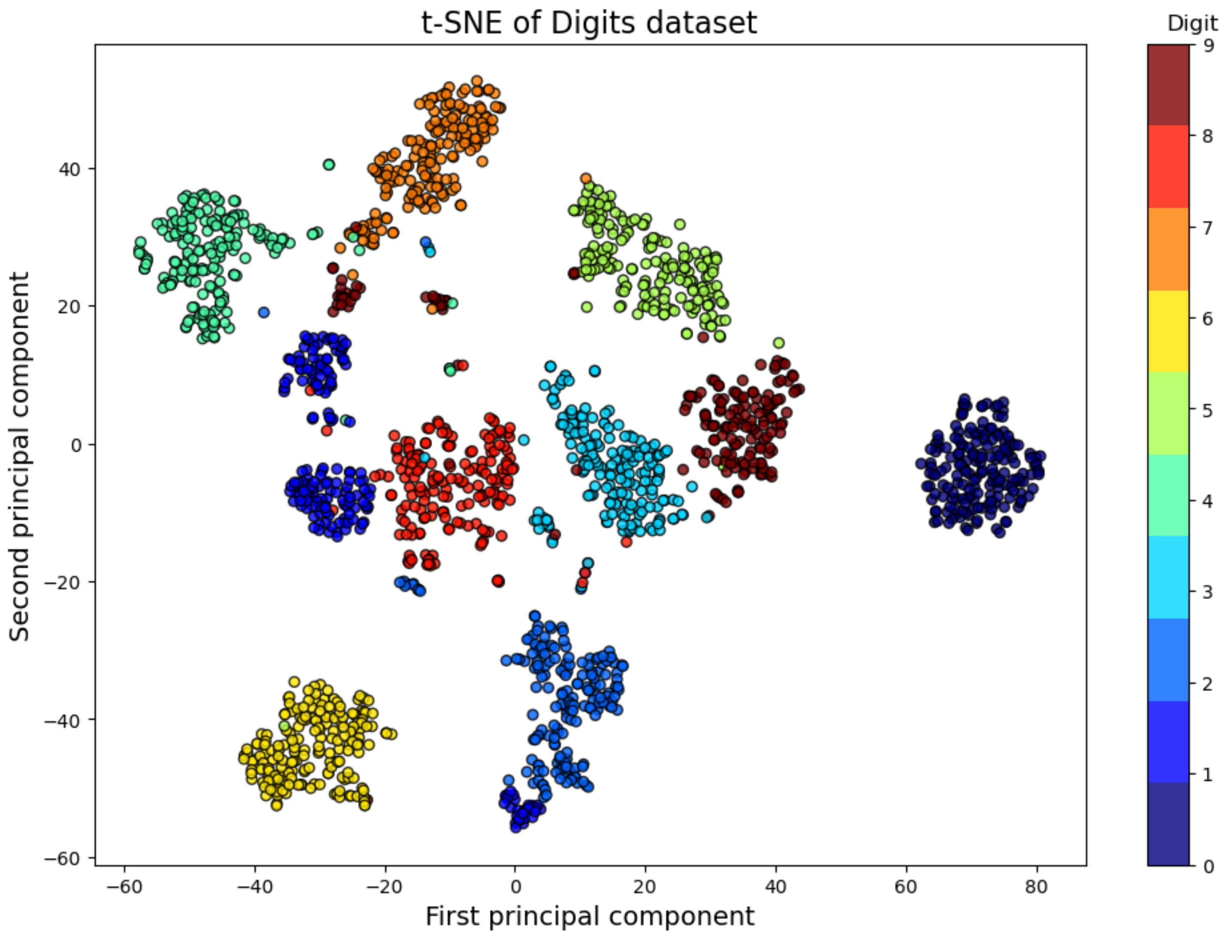
```
Fraction of variance that is UNEXPLAINED by the first two principal components: 78.4
1%
```

From the plot of cumulative fraction of variance explained, we can see that as the number of components increase, more variance is explained. The fraction of variance that is unexplained by the first two components reach 78.41%, which means that the first two principal components may not be sufficient to capture all the important characteristics of the data. This can result in clusters that are less well-separated and less distinct, as the remaining variability may be contributing to overlaps between the clusters

(c) Reduce the dimensionality of the data with t-SNE for data visualization. T-distributed stochastic neighborhood embedding (t-SNE) is a nonlinear dimensionality reduction technique that is particularly adept at embedding the data into lower 2 or 3 dimensional spaces. Apply t-SNE using the `scikit-learn` implementation to the digits dataset and plot it in 2-dimensions (with associated cluster labels 0-9). You may need to adjust the parameters to get acceptable performance. You can read more about how to use t-SNE effectively here.

```
In [ ]: X_digits_tsne = TSNE(n_components=2, random_state=42).fit_transform(X_digits_std)
```

```
# plotting
plt.figure(figsize=(12,8), dpi=100)
plt.scatter(X_digits_tsne[:, 0], X_digits_tsne[:, 1], c=y_digits, \
            cmap=plt.cm.get_cmap('jet', 10), s=30, alpha=0.8, edgecolors='black')
plt.colorbar(ticks=range(10))
plt.xlabel('First principal component', fontsize=14)
plt.ylabel('Second principal component', fontsize=14)
plt.text(100, 60, 'Digit', fontsize=12)
plt.title('t-SNE of Digits dataset', fontsize=16)
plt.show()
```



(d) Briefy compare/contrast the performance of these two techniques.

- Which seemed to cluster the data best and why?
- Notice that while t-SNE has a `fit` method and a `fit_transform` method, these methods are actually identical, and there is no `transform` method. Why is this? What implications does this imply for using this method?

From the two-dimensional plots, it seems that t-SNE performs better than PCA in clustering the data. There is less overlapping between clusters after t-SNE dimentionality reduction. This is becaue t-SNE uses a probabilistic approach to embed the data in a lower-dimensional space while preserving the local structure of the data. Therefore, it is particularly well-suited for visualizing complex relationships between variables and clusters that may be difficult to separate in high-dimensional space. On the other hand, PCA is a linear method, so it may struggle with non-linear relationships between the variables.

The reason why t-SNE has *fit* and *fit_transform* method without *transform* method is due to the fact that when mapping high-dimensional data to a lower-dimensional space, the mapping is not a linear transformation, and the reduced representation of the data cannot be computed directly from a pre-defined transformation matrix.
The t-SNE algorithm uses the *fit* method to start an iterative optimization process that updates the locations of the data points until it reaches a minimum. The *fit_transform* method performs the same optimization process as *fit*, but it also returns the reduced representation of the data in a lower-dimensional space. Since there is no pre-defined transformation matrix for t-SNE, it doesn't make sense to have a separate *transform* method to apply a pre-defined transformation to new data points.