A5 - Assignment_5_vReport

March 26, 2022

1 Assignment 5 - Kaggle Competition and Unsupervised Learning

1.1 ***Tego Chang***

Netid: cc703

Names of students you worked with on this assignment: **Nansu Wang**

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.

2 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

3 1

3.1 [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.

3.1.1 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.

3.1.2 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. One of these 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 10 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.

3.1.3 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.

3.1.4 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.

[1]:
```python
import pandas as pd
import numpy as np
import pickle

################################
# Load the data
################################
data = pickle.load( open( "./data/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

EDA

The options can be taken in this stage include:

- Manual feature engineering
  - To check if certain predictors are not quite related to response (mostly through visualization, e.g., scatter plot).
- Filter methods
  - to check if certain predictors are highly related to each other.

Check missing data

We found there are two entries missed the data in the *children* variable. Since the number of missing entries is small, we apply simple imputation.

[3]:
```
print(X_train_ohe.columns[X_train_ohe.isna().sum()!=0])
print(X_train_ohe.children.isna().sum())
print(X_train_ohe.children.value_counts())
```

```
Index(['children'], dtype='object')
2
0.0    88660
1.0     3859
2.0     2928
3.0       63
Name: children, dtype: int64
```

[4]:
```
print(X_test_ohe.columns[X_train_ohe.isna().sum()!=0])
print(X_test_ohe.children.isna().sum())
print(X_test_ohe.children.value_counts())
```

```
Index(['children'], dtype='object')
2
0.0     22136
1.0      1002
2.0       724
3.0        13
10.0        1
Name: children, dtype: int64
```

[5]:
```
from sklearn.impute import SimpleImputer

imp = SimpleImputer(strategy="most_frequent")
X_train_ohe_imputed = imp.fit_transform(X_train_ohe) # return a numpy array
X_test_ohe_imputed = imp.fit_transform(X_test_ohe) # return a numpy array
```

[6]:
```
X_train_ohe_imputed = pd.DataFrame(X_train_ohe_imputed, columns = X_train_ohe.
 ↪columns, index = X_train_ohe.index)
X_train_ohe_imputed.children.isna().sum()
```

[6]:
```
0
```

```
[7]: X_test_ohe_imputed = pd.DataFrame(X_test_ohe_imputed, columns = X_test_ohe.
      ↪columns, index = X_test_ohe.index)
     X_test_ohe_imputed.children.isna().sum()
```

```
[7]: 0
```

```
[8]: X_train_ohe_imputed.head(10)
     X_train_ohe_imputed.columns
```

```
[8]: Index(['lead_time', 'arrival_date_year', 'arrival_date_week_number',
            'arrival_date_day_of_month', 'stays_in_weekend_nights',
            'stays_in_week_nights', 'adults', 'children', 'babies',
            'is_repeated_guest',
            …
            'company_530.0', 'company_531.0', 'company_534.0', 'company_539.0',
            'company_541.0', 'company_543.0', 'customer_type_Contract',
            'customer_type_Group', 'customer_type_Transient',
            'customer_type_Transient-Party'],
           dtype='object', length=940)
```

```
[9]: X_train_ohe_imputed[X_train_ohe.children.isna()]['children']
```

```
[9]: 40667    0.0
     40679    0.0
     Name: children, dtype: float64
```

```
[10]: X_test_ohe_imputed[X_test_ohe.children.isna()]['children']
```

```
[10]: 40600    0.0
      41160    0.0
      Name: children, dtype: float64
```

```
[11]: X_train_ohe_imputed.shape
```

```
[11]: (95512, 940)
```

```
[12]: X_test_ohe_imputed.shape
```

```
[12]: (23878, 940)
```

```
[13]: X_train_ohe_imputed.index
      X_train_ohe.index
      y_train.index
```

```
[13]: Int64Index([     0,      2,      3,      4,      5,      6,      8,      9,
                  10,     12,
                 …
```

```
 119376, 119378, 119380, 119381, 119382, 119383, 119384, 119387,
 119388, 119389],
dtype='int64', length=95512)
```

Sample 10% data

The step is not mandatory, and we sample the data to shorten the model training time in the following. However, once confirmed the process, we sampled the complete dataset.

Important Note: once the data is sample, the index has been shuffle. Thus, the index of y_train must be adjusted with the sampled X index accordingly.

[14]:
```
type(X_train_ohe_imputed)
```

[14]:
```
pandas.core.frame.DataFrame
```

[15]:
```
X_train_ohe_imputed_s = X_train_ohe_imputed.sample (frac = 1)
```

[16]:
```
y_train_s = y_train[X_train_ohe_imputed_s.index]
```

[18]:
```
X_train_ohe_imputed_s = np.asanyarray(X_train_ohe_imputed_s)
y_train_s = np.asanyarray(y_train_s)
```

[19]:
```
print ("Training sample dataset shape of X, Y: ", X_train_ohe_imputed_s.shape,␣
 ↪y_train_s.shape)
print ("Training dataset shape of X, Y: ", X_train_ohe_imputed.shape, y_train.
 ↪shape)
```

```
Training sample dataset shape of X, Y:  (95512, 940) (95512,)
Training dataset shape of X, Y:  (95512, 940) (95512,)
```

Training testing split

We split the dataset into training, validation, and testing for hyperparameters tuning and model performance evaluation.

[20]:
```
from sklearn.model_selection import train_test_split

X_train1, X_test1, y_train1, y_test1 = train_test_split(X_train_ohe_imputed_s,␣
 ↪y_train_s, test_size=0.20, random_state=321)
# X_train1, X_test1, y_train1, y_test1 = train_test_split(X_train_ohe_imputed,␣
 ↪y_train, test_size=0.20, random_state=321)
X_train2, X_val1, y_train2, y_val1 = train_test_split(X_train1, y_train1,␣
 ↪test_size=0.20, random_state=321)

print ("Training dataset 1 shape of X, Y: ", X_train1.shape, y_train1.shape)
print ("Testing dataset 1 shape of X, Y: ", X_test1.shape, y_test1.shape)

print ("Training dataset 2 shape of X, Y: ", X_train2.shape, y_train2.shape)
print ("Validation dataset 1 shape of X, Y: ", X_val1.shape, y_val1.shape)
```

```
Training dataset 1 shape of X, Y:  (76409, 940) (76409,)
Testing dataset 1 shape of X, Y:  (19103, 940) (19103,)
Training dataset 2 shape of X, Y:  (61127, 940) (61127,)
Validation dataset 1 shape of X, Y:  (15282, 940) (15282,)
```

Normalize your data: We normalized the dataset for the following process or models:

- dimension reduction: PCA
- feature subset selection: selectKBest
- model: KNN, SVM

[21]:
```
from sklearn import preprocessing

std_train1 = preprocessing.StandardScaler()
X_train1_std = std_train1.fit_transform(X_train1)

std_train2 = preprocessing.StandardScaler()
X_train2_std = std_train2.fit_transform(X_train2)

std_val1 = preprocessing.StandardScaler()
X_val1_std = std_val1.fit_transform(X_val1)

std_test1 = preprocessing.StandardScaler()
X_test1_std = std_test1.fit_transform(X_test1)

### Below For Final Training and Prediction ###

std_train0 = preprocessing.StandardScaler()
X_train_ohe_imputed_s_std = std_train0.fit_transform(X_train_ohe_imputed_s)

std_test0 = preprocessing.StandardScaler()
X_test_ohe_imputed_std = std_test0.fit_transform(X_test_ohe_imputed)
```

[22]:
```
minmax = preprocessing.MinMaxScaler()
X_train_ohe_imputed_s_mimx = minmax.fit_transform(X_train_ohe_imputed_s)
X_test_ohe_imputed_mimx = minmax.fit_transform(X_test_ohe_imputed)

X_train1_mimx = minmax.fit_transform(X_train1)
```

Dimension Reduction

Here we applied dimension reduction through PCA to see if will enhance the model performance. However, based on our tuned hyper parameters, it seems not help.

[23]:
```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=800, random_state=0)
pca.fit(X_train_ohe_imputed_s_std)
```

```
[23]: PCA(n_components=800, random_state=0)
```

```
[24]: sum(pca.explained_variance_ratio_) # 800, 94.5%; 850, 98.7%
```

```
[24]: 0.9451850498811352
```

```
[25]: X_train1_PCA = pca.transform(X_train1_std)
      X_train2_PCA = pca.transform(X_train2_std)
      X_val1_PCA = pca.transform(X_val1_std)
      X_test1_PCA = pca.transform(X_test1_std)

      ####
      X_train_ohe_imputed_s_PCA = pca.transform(X_train_ohe_imputed_s)
      X_test_ohe_imputed_PCA = pca.transform(X_test_ohe_imputed)
```

```
/Users/tegochang/opt/miniconda3/lib/python3.9/site-packages/sklearn/base.py:443:
UserWarning: X has feature names, but PCA was fitted without feature names
  warnings.warn(
```

```
[26]: print(X_train_ohe_imputed_s_PCA.shape)
      print(X_test_ohe_imputed_PCA.shape)

      print (X_train1_PCA.shape, X_train2_PCA.shape, X_val1_PCA.shape, X_test1_PCA.
       ↪shape)
```

```
(95512, 800)
(23878, 800)
(76409, 800) (61127, 800) (15282, 800) (19103, 800)
```

Feature selection Here we applied different feature subset selection mechanism, we fine-tuned the hyper parameters, K, manually. The result turned out that SelectKBest with $K = 300$ works the best for our models.

```
[27]: from sklearn.feature_selection import SelectKBest # RF: 0.939

      selector = SelectKBest(k = 300) # for classification: f_classif (default), chi2;
       ↪ for continuous response: mutual_info_regression
      selector.fit(X_train1_std, y_train1) # for chi2: minmax, for f_classif: standard
      # X_train_ohe.columns[selector.get_support()]

      df_features = pd.DataFrame({"Names":   X_train_ohe.columns[selector.
       ↪get_support()],
                                  "P Value": selector.pvalues_[selector.
       ↪get_support()],
                                  "Scores":  selector.scores_[selector.
       ↪get_support()]})
      df_features = df_features.sort_values(by='Scores', ascending = False)
```

```
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning:
Features [ 44  50  57  81  97 106 110 133 141 150 155 158 182 298 366 383 430
443
 445 451 453 462 468 494 498 504 525 528 532 541 542 570 587 588 617 626
 641 644 647 668 680 684 704 706 723 724 756 766 772 785 798 802 816 822
 831 836 856 859 884 885 898 900 911 914 916 923 925 933 934] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
UserWarning)
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/feature_selection/_univariate_selection.py:113: RuntimeWarning:
invalid value encountered in true_divide
  f = msb / msw
```

[28]:
```
df_features.Names.head(50)
```

[28]:
```
100             deposit_type_Non Refund
99              deposit_type_No Deposit
67                          country_PRT
0                             lead_time
13          total_of_special_requests
78                market_segment_Groups
12         required_car_parking_spaces
89                 assigned_room_type_A
102                           agent_1.0
83           distribution_channel_TA/TO
77                market_segment_Direct
81          distribution_channel_Direct
9                       booking_changes
14                     hotel_City Hotel
15                   hotel_Resort Hotel
298             customer_type_Transient
299       customer_type_Transient-Party
92                 assigned_room_type_D
48                          country_GBR
47                          country_FRA
41                          country_DEU
7               previous_cancellations
6                     is_repeated_guest
107                           agent_7.0
76             market_segment_Corporate
123                          agent_28.0
116                          agent_19.0
```

```
80     distribution_channel_Corporate
113                        agent_14.0
124                        agent_29.0
84               reserved_room_type_A
44                        country_ESP
93               assigned_room_type_E
4                              adults
205                       agent_250.0
200                       agent_241.0
198                       agent_236.0
8      previous_bookings_not_canceled
132                        agent_40.0
138                        agent_58.0
91               assigned_room_type_C
109                         agent_9.0
251                      company_40.0
10               days_in_waiting_list
35                        country_BEL
222                       agent_326.0
11                                adr
62                        country_NLD
94               assigned_room_type_F
104                         agent_3.0
Name: Names, dtype: object
```

[29]:
```
X_train1_std = X_train1_std[:, selector.get_support()]
X_train2_std = X_train2_std[:, selector.get_support()]
X_val1_std = X_val1_std[:, selector.get_support()]
X_test1_std = X_test1_std[:, selector.get_support()]

# ###

# X_train_std_partial = X_train_ohe_imputed_s_std[:, selector.get_support()]
# X_test_std_partial = X_test_ohe_imputed_std[:, selector.get_support()]
```

[30]:
```
selector2 = SelectKBest(k = 300) # for classification: f_classif (default),␣
 ↪chi2; for continuous response: mutual_info_regression
selector2.fit(X_train_ohe_imputed_s_std, y_train_s) # for chi2: minmax, for␣
 ↪f_classif: standard
# KEY is to use y_train_s, not y_train as sample already shuffle
# X_train_ohe.columns[selector.get_support()]

df_features = pd.DataFrame({"Names":   X_train_ohe.columns[selector2.
 ↪get_support()],
                            "P Value": selector2.pvalues_[selector2.
 ↪get_support()],
                                "Scores":  selector2.scores_[selector2.
↪get_support()]})
df_features = df_features.sort_values(by='Scores', ascending = False)
```

```
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning:
Features [ 50  57 106 141 150 155 298 366 383 430 443 451 462 468 494 498 504
528
 532 541 587 588 617 626 668 680 723 724 798 816 822 836 859 900 911 916
 923 925] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
UserWarning)
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/feature_selection/_univariate_selection.py:113: RuntimeWarning:
invalid value encountered in true_divide
  f = msb / msw
```

[31]:
```
df_features.Names.head(50)
```

[31]:
```
100           deposit_type_Non Refund
99            deposit_type_No Deposit
66                        country_PRT
0                           lead_time
12          total_of_special_requests
77               market_segment_Groups
11        required_car_parking_spaces
89                 assigned_room_type_A
102                         agent_1.0
82          distribution_channel_TA/TO
76               market_segment_Direct
80         distribution_channel_Direct
8                     booking_changes
13                   hotel_City Hotel
14                 hotel_Resort Hotel
298           customer_type_Transient
92                 assigned_room_type_D
299     customer_type_Transient-Party
47                        country_GBR
46                        country_FRA
40                        country_DEU
6              previous_cancellations
5                   is_repeated_guest
108                         agent_7.0
75            market_segment_Corporate
79      distribution_channel_Corporate
124                        agent_28.0
117                        agent_19.0
```

```
114                            agent_14.0
43                            country_ESP
125                            agent_29.0
83                     reserved_room_type_A
93                     assigned_room_type_E
205                           agent_250.0
3                                  adults
198                           agent_236.0
200                           agent_241.0
7         previous_bookings_not_canceled
133                            agent_40.0
140                            agent_58.0
91                     assigned_room_type_C
250                          company_40.0
110                             agent_9.0
9                     days_in_waiting_list
61                            country_NLD
33                            country_BEL
10                                    adr
94                     assigned_room_type_F
222                           agent_326.0
84                     reserved_room_type_D
Name: Names, dtype: object
```

[32]:
```
### Below For Final Training and Prediction ###
X_train_std = X_train_ohe_imputed_s_std[:, selector2.get_support()]
X_test_std = X_test_ohe_imputed_std[:, selector2.get_support()]
```

1. Logistic Regression

- We have tuned the regularation parameter, C, as shown in the following cells.
- (Future work) to tune learning rate: SGDClassifier (loss = ‘log’)

[42]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, average_precision_score,␣
↪roc_curve, auc, log_loss, f1_score
from tqdm import tqdm
```

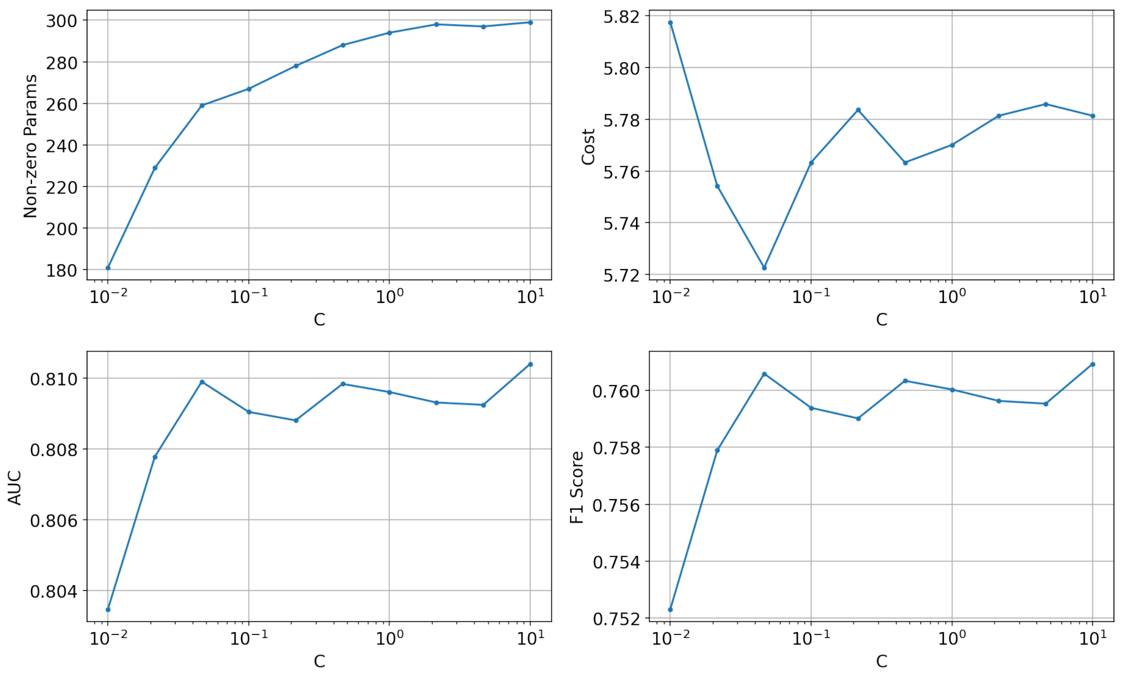
Start the process of tuning the regularation parameter of Logistic regression

[43]:
```
rc_trend = []

for idx, rc in tqdm(enumerate(np.linspace(-2, 1, 10))):
    model_lr = LogisticRegression(penalty='l1', C=10**rc, solver="liblinear")
    model_lr.fit (X_train2_std, y_train2)
    y_pred = model_lr.predict(X_val1_std)
    fpr, tpr, thresholds = roc_curve(y_val1, y_pred)

    rc_trend.append([10**rc,
                     np.sum(model_lr.coef_ != 0),
                     log_loss(y_val1, y_pred),
                     auc(fpr, tpr),
                     f1_score(y_val1, y_pred)])

    pass

rc_trend = np.asanyarray(rc_trend)
```

```
10it [12:23, 74.31s/it]
```

[44]:
```
import matplotlib.pyplot as plt
# Globally, we'll make the font size larger for increased readability:
%config InlineBackend.figure_format = 'retina' # Optional - best for macs
plt.rcParams.update({'font.size': 14})

plt.figure(figsize=(13, 8))

# Number of nonzero parameters
plt.subplot(221)
plt.plot(rc_trend[:, 0], rc_trend[:, 1], marker='.')
plt.grid(True)
plt.xlabel('C')
plt.ylabel('Non-zero Params')
plt.xscale('log')

# Logistic regression cost
plt.subplot(222)
plt.plot(rc_trend[:, 0], rc_trend[:, 2], marker='.')
plt.grid(True)
plt.xlabel('C')
plt.ylabel('Cost')
plt.xscale('log')
plt.tick_params(axis='y', which='minor')

# Area under the ROC curve (AUC)
plt.subplot(223)
plt.plot(rc_trend[:, 0], rc_trend[:, 3], marker='.')
plt.grid(True)
plt.xlabel('C')
plt.ylabel('AUC')
plt.xscale('log')

# F1 Score
```

```
plt.subplot(224)
plt.plot(rc_trend[:, 0], rc_trend[:, 4], marker='.')
plt.grid(True)
plt.xlabel('C')
plt.ylabel('F1 Score')
plt.xscale('log')

plt.tight_layout()
plt.show()
```



[45]:
```
df_rc = pd.DataFrame(rc_trend)
print ("The C value to result in the highest AUC is: ", df_rc.iloc[df_rc.iloc[:
 ↪,3].idxmax(), 0])
```

```
The C value to result in the highest AUC is:  10.0
```

[46]:
```
model_lr = LogisticRegression(penalty='l1', C=df_rc.iloc[df_rc.iloc[:,3].
 ↪idxmax(), 0], solver="liblinear")
model_lr.fit(X_train1_std, y_train1)

y_pred_lr = model_lr.predict_proba(X_test1_std)
# model_lr.score(X_test1_std, y_test1)

fpr, tpr, thresholds = roc_curve(y_test1, y_pred_lr[:,1])
```

```
print ("AUC for the tuned logistic regression is {:.3f}".format(auc(fpr, tpr)))
```

```
AUC for the tuned logistic regression is 0.914
```

Logistic Regression: Re-train and apply to test dataset

This model is one of the submission, but it's not the best of our model.

[47]:
```
model_lr = LogisticRegression(penalty='l1', C=df_rc.iloc[df_rc.iloc[:,3].
 ↪idxmax(), 0], solver="liblinear")
model_lr.fit(X_train_std, y_train_s)

y_pred_lr = model_lr.predict_proba(X_test_std)
```

[48]:
```
create_submission(y_pred_lr[:,1], './result/submissionN_tuned_LR.csv')
```

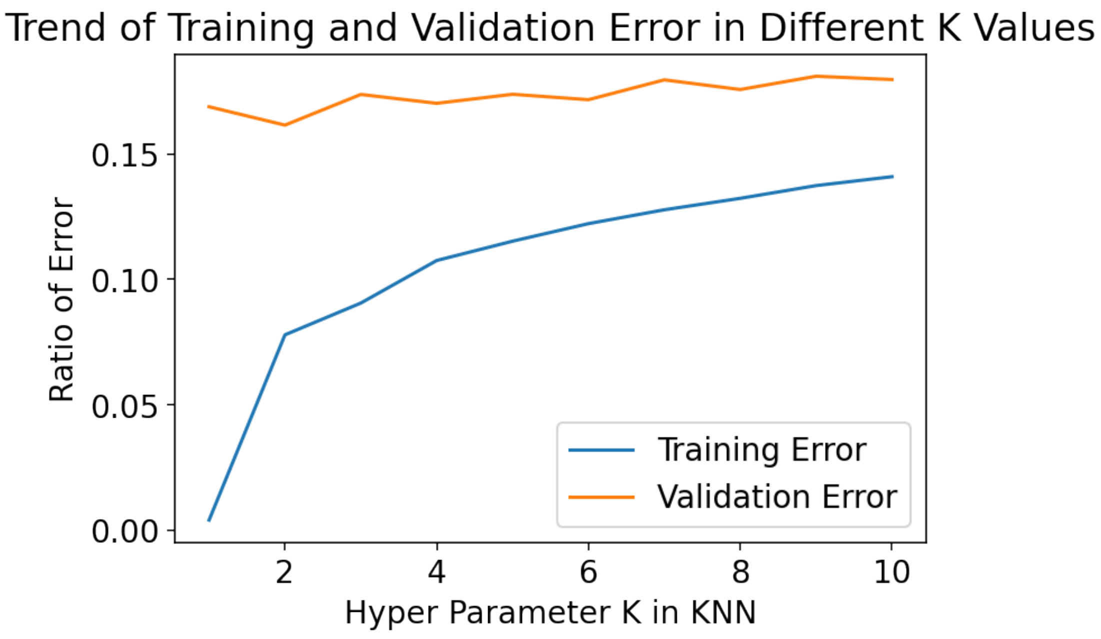
2. K-nearest neighbors

Start the process of tuning the hyperparameter K:

[49]:
```
from sklearn.neighbors import KNeighborsClassifier

obs = []

for k in tqdm(range(1, 11)):
    # train knn model
    model_knn = KNeighborsClassifier(n_neighbors=k)
    model_knn.fit(X_train2_std, y_train2)

    # calculate training error
    train_error = 1 - model_knn.score(X_train2_std, y_train2)

    # calculate validation error
    val_error = 1 - model_knn.score(X_val1_std, y_val1)

#     y_test_hat = model_knn.predict(X_val1)
#     test_error = np.sum(y_val1 != y_test_hat) / len(y_val1)

    obs.append([k, train_error, val_error])
    pass

df_knn_error = pd.DataFrame(obs, columns=["K", "Training Error", "Validation␣
 ↪Error"])
# df_knn_error.head()

line_error = df_knn_error.plot.line(x="K", y=["Training Error", "Validation␣
 ↪Error"])
```

```
plt.xlabel("Hyper Parameter K in KNN")
plt.ylabel("Ratio of Error")

plt.title("Trend of Training and Validation Error in Different K Values")
plt.show()
```

```
100%|
                | 10/10 [13:42<00:00, 82.23s/it]
```



[50]:
```
df_knn_error.iloc[df_knn_error["Validation Error"].idxmin(), :][0].astype(int)
```

[50]:
```
2
```

[51]:
```
model_knnOpt = KNeighborsClassifier(n_neighbors=df_knn_error.
 ↪iloc[df_knn_error["Validation Error"].idxmin(), :][0].astype(int))
model_knnOpt.fit(X_train1_std, y_train1)

y_pred_knn = model_knnOpt.predict_proba(X_test1_std)
# model_knnOpt.score(X_test1_std, y_test1)

fpr, tpr, thresholds = roc_curve(y_test1, y_pred_knn[:,1])

print ("AUC for the tuned KNN is {:.3f}".format(auc(fpr, tpr)))
```

```
AUC for the tuned KNN is 0.872
```

KNN: Re-train and apply to test dataset

This model is one of the submission, but it's not the best of our model.

```
[52]: model_knnOpt = KNeighborsClassifier(n_neighbors=df_knn_error.
       ↪iloc[df_knn_error["Validation Error"].idxmin(), :][0].astype(int))
      model_knnOpt.fit(X_train_std, y_train_s)

      y_pred_knn = model_knnOpt.predict_proba(X_test_std)
```

```
[53]: create_submission(y_pred_knn[:, 1], './result/submissionN_tuned_KNN.csv')
```

3. Random Forests

- Random forest with default parameters
- Trimmed random forest
  - version 1 with more hyperparameters tuning.
  - version 2 with few hyperparaneters tuning.

```
[54]: from sklearn.ensemble import RandomForestClassifier
```

```
[55]: model_baseRF = RandomForestClassifier(n_jobs = -1, random_state=0)
      model_baseRF.fit(X_train1_std, y_train1)

      y_pred_rf = model_baseRF.predict_proba(X_test1_std)

      fpr, tpr, thresholds = roc_curve(y_test1, y_pred_rf[:,1])

      print ("AUC for the baseline random forest is {:.3f}".format(auc(fpr, tpr)))
```

```
AUC for the baseline random forest is 0.956
```

```
[58]: # first ok solution for reference
      y_pred_rf[:5]
```

```
[58]: array([[1.  , 0.  ],
             [0.34, 0.66],
             [0.36, 0.64],
             [0.99, 0.01],
             [0.69, 0.31]])
```

Through PCA

PCA doesn't have better performance compared with selectKBest.

```
[59]: model_baseRF = RandomForestClassifier(n_jobs = -1, random_state=0)
      model_baseRF.fit(X_train1_PCA, y_train1)

      y_pred_rf = model_baseRF.predict_proba(X_test1_PCA)

      fpr, tpr, thresholds = roc_curve(y_test1, y_pred_rf[:,1])
```

```
print ("AUC for the baseline random forest is {:.3f}".format(auc(fpr, tpr)))
```

```
AUC for the baseline random forest is 0.804
```

Non-trimmed Random Forest: Re-train and apply to test dataset

This model is one of the submission, and it's **the best** of our model.

Through selectKbest

[60]:
```
model_baseRF = RandomForestClassifier(n_jobs = -1, random_state=0)
model_baseRF.fit(X_train_std, y_train_s)

y_pred_rf = model_baseRF.predict_proba(X_test_std)
y_pred_rf[:5]
```

[60]:
```
array([[0.  , 1.  ],
       [0.9 , 0.1 ],
       [0.68, 0.32],
       [0.97, 0.03],
       [0.44, 0.56]])
```

[61]:
```
create_submission(y_pred_rf[:,1], './result/submission3_baseRF_fit.csv')
```

Trimmed Random Forest

Start two versions of hyper parameters tuning.

Version 1

[62]:
```
from sklearn.model_selection import PredefinedSplit

# For RandomSeachCV, we will need to combine training and validation sets then
#  specify which portion is training and which is validation
# Also, for the final performance evaluation, train on all of the training AND␣
↪validation data
X_train_plus_val = np.concatenate((X_train2, X_val1), axis=0)
y_train_plus_val = np.concatenate((y_train2, y_val1), axis=0)

# Create a predefined train/test split for RandomSearchCV (to be used later)
validation_fold = np.concatenate((-1*np.ones(len(y_train2)), np.
↪zeros(len(y_val1))))
train_val_split = PredefinedSplit(validation_fold)
```

[63]:
```
from sklearn.model_selection import RandomizedSearchCV

# n_estimators = [int(x) for x in np.linspace(100, 500, 50)] # define allow␣
↪n_estimators, how many trees in a forest
```

```
n_estimators = [int(x) for x in np.linspace(50, 70, 5)] # define allow␣
 ↪n_estimators, how many trees in a forest
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(30, 60, 6)]
min_samples_split = [int(x) for x in np.linspace(2, 30, 5)]
min_samples_leaf = [int(x) for x in np.linspace(2, 30, 5)]
bootstrap = [True, False] # bootstrap for dataset sample
```

```
[65]: #turn the arrays into a grid dictionary
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

model_rf = RandomForestClassifier(n_jobs = -1, random_state=0)
# train-val split on 50 combinations of random forest parameters
random_search_model = RandomizedSearchCV(estimator = model_rf,
                                         param_distributions = random_grid,
                                         n_iter = 20,
                                         cv = train_val_split,
#                                          verbose=2,
                                         random_state=0,
                                         n_jobs = -1)
random_search_model.fit(X_train_plus_val, y_train_plus_val)
random_search_model.best_params_ #get the best combination of parameters
```

```
[65]: {'n_estimators': 50,
 'min_samples_split': 16,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 42,
 'bootstrap': False}
```

```
[66]: # model_optRF = RandomForestClassifier(n_jobs = -1,
#                                        random_state = 0,
#                                        n_estimators= 56,
#                                        min_samples_split = 2,
#                                        min_samples_leaf = 2,
#                                        max_features = 'auto',
#                                        max_depth=66,
#                                        bootstrap = False)

model_optRF = RandomForestClassifier(n_jobs = -1,
                                     random_state = 0,
                                     n_estimators= random_search_model.
 ↪best_params_['n_estimators'],
                                     min_samples_split = random_search_model.
 ↪best_params_['min_samples_split'],
                                     min_samples_leaf = random_search_model.
 ↪best_params_['min_samples_leaf'],
                                     max_features = random_search_model.
 ↪best_params_['max_features'],
                                     max_depth= random_search_model.
 ↪best_params_['max_depth'],
                                     bootstrap = random_search_model.
 ↪best_params_['bootstrap'])

model_optRF.fit(X_train1_std, y_train1)

y_pred_rf = model_optRF.predict_proba(X_test1_std)
# model_optRF.score(y_test1, y_pred_rf)

fpr, tpr, thresholds = roc_curve(y_test1, y_pred_rf[:,1])

print ("AUC for the tuned random forest is {:.3f}".format(auc(fpr, tpr)))
```

```
AUC for the tuned random forest is 0.952
```

Trimmed Random Forest: Re-train and apply to test dataset

This model is one of the submission, and it's **the second best** of our model.

[67]:
```
# model_optRF = RandomForestClassifier(n_jobs = -1,
#                                      random_state = 0,
#                                      n_estimators= 56,
#                                      min_samples_split = 2,
#                                      min_samples_leaf = 2,
#                                      max_features = 'auto',
#                                      max_depth=66,
#                                      bootstrap = False)

model_optRF = RandomForestClassifier(n_jobs = -1,
                                     random_state = 0,
                                     n_estimators= random_search_model.
 ↪best_params_['n_estimators'],
                                     min_samples_split = random_search_model.
 ↪best_params_['min_samples_split'],
                                     min_samples_leaf = random_search_model.
 ↪best_params_['min_samples_leaf'],
                                     max_features = random_search_model.
 ↪best_params_['max_features'],
                                     max_depth= random_search_model.
↪best_params_['max_depth'],
                                     bootstrap = random_search_model.
↪best_params_['bootstrap'])

model_optRF.fit(X_train_std, y_train_s)

y_pred_rf = model_optRF.predict_proba(X_test_std)
```

[68]:
```
create_submission(y_pred_rf[:,1], './result/submission4_trimmedRF.csv')
```

Fined-tuned Random Forest version 2

- number of estimators
  - usually bigger the forest the better, there is small chance of overfitting here
- max depth of each tree (default none, leading to full tree)
  - reduction of the maximum depth helps fighting with overfitting
- max features per split (default sqrt(d))
  - you might one to play around a bit as it significantly alters behaviour of the whole tree. sqrt heuristic is usually a good starting point but an actual sweet spot might be somewhere else

[69]:
```
# n_estimators = [int(x) for x in np.linspace(100, 500, 50)] # define allow␣
↪n_estimators, how many trees in a forest
n_estimators = [int(x) for x in np.linspace(50, 300, 5)] # define allow␣
↪n_estimators, how many trees in a forest
max_features = np.arange(0.01, 0.3, 6)
max_depth = [int(x) for x in np.linspace(30, 60, 6)]
```

[70]:
```
#turn the arrays into a grid dictionary
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth}

model_rf = RandomForestClassifier(n_jobs = -1, random_state=0)
# train-val split on 50 combinations of random forest parameters
random_search_model = RandomizedSearchCV(estimator = model_rf,
                                         param_distributions = random_grid,
                                         n_iter = 20,
                                         cv = train_val_split,
#                                          verbose=2,
                                         random_state=0,
                                         n_jobs = -1)
random_search_model.fit(X_train_plus_val, y_train_plus_val)
random_search_model.best_params_ #get the best combination of parameters
```

[70]:
```
{'n_estimators': 300, 'max_features': 0.01, 'max_depth': 54}
```

```
[71]: # model_optRF = RandomForestClassifier(n_jobs = -1,
      #                                      random_state = 0,
      #                                      n_estimators= 62,
      # #                                      min_samples_split = 2,
      # #                                      min_samples_leaf = 2,
      #                                      max_features = 0.07,
      #                                      max_depth=34,
      #                                      bootstrap = False)

      model_optRF = RandomForestClassifier(n_jobs = -1,
                                           random_state = 0,
                                           n_estimators= random_search_model.
       ↪best_params_['n_estimators'],
      #                                      min_samples_split = random_search_model.
       ↪best_params_['min_samples_split'],
      #                                      min_samples_leaf = random_search_model.
       ↪best_params_['min_samples_leaf'],
                                           max_features = random_search_model.
       ↪best_params_['max_features'],
                                           max_depth= random_search_model.
       ↪best_params_['max_depth']
      #                                      bootstrap = random_search_model.
       ↪best_params_['bootstrap']
                                          )

      model_optRF.fit(X_train1_std, y_train1)

      y_pred_rf = model_optRF.predict_proba(X_test1_std)
      # model_optRF.score(y_test1, y_pred_rf)

      fpr, tpr, thresholds = roc_curve(y_test1, y_pred_rf[:,1])

      print ("AUC for the tuned random forest is {:.3f}".format(auc(fpr, tpr)))
```

```
AUC for the tuned random forest is 0.953
```

Trimmed Random Forest: Re-train and apply to test dataset

This model is one of the submission, and it's **the second or third best** of our model.

```
[72]: # model_optRF = RandomForestClassifier(n_jobs = -1,
      #                                      random_state = 0,
      #                                      n_estimators= 62,
      # #                                      min_samples_split = 2,
      # #                                      min_samples_leaf = 2,
      #                                      max_features = 0.07,
      #                                      max_depth=34,
      #                                      bootstrap = False)
```

```
model_optRF = RandomForestClassifier(n_jobs = -1,
                                     random_state = 0,
                                     n_estimators= random_search_model.
 ↪best_params_['n_estimators'],
#                                      min_samples_split = random_search_model.
 ↪best_params_['min_samples_split'],
#                                      min_samples_leaf = random_search_model.
 ↪best_params_['min_samples_leaf'],
                                     max_features = random_search_model.
 ↪best_params_['max_features'],
                                     max_depth= random_search_model.
 ↪best_params_['max_depth']
#                                      bootstrap = random_search_model.
 ↪best_params_['bootstrap']
                                    )

model_optRF.fit(X_train_std, y_train_s)

y_pred_rf = model_optRF.predict_proba(X_test_std)
```

[73]:
```
create_submission(y_pred_rf[:,1], './result/submissionN_trimmedRF_v2.csv')
```

4. Neural Networks

Two Neural Netowrks model are built here. The first one is without any tuning. The second we tuned the learning rate.

[74]:
```
from sklearn.neural_network import MLPClassifier

model_baseNN = MLPClassifier()
model_baseNN.fit(X_train1_std, y_train1)
y_pred_nn = model_optRF.predict_proba(X_test1_std)

fpr, tpr, thresholds = roc_curve(y_test1, y_pred_nn[:,1])

print ("AUC for the base NN is {:.3f}".format(auc(fpr, tpr)))
```

```
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
AUC for the base NN is 0.745
```

Tuned Neural Networks

Start the process of tuning the hyperparameter learning rate:

[76]:
```
from sklearn.utils.fixes import loguniform

model_NN = MLPClassifier()

param_dist = {
    "learning_rate_init": loguniform(1e-5, 1e0)
#     "alpha": loguniform(1e-8, 1e2)
}

n_iter_search = 20
random_search_nn = RandomizedSearchCV(model_NN,
                                      param_distributions=param_dist,
                                      n_iter=n_iter_search,
                                      cv = train_val_split,
                                      n_jobs = -1
)
result = random_search_nn.fit(X_train_plus_val,y_train_plus_val)
result.best_params_
```

```
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[76]:
```
{'learning_rate_init': 0.00011886251747369423}
```

[77]:
```
model_optNN = MLPClassifier(learning_rate_init = result.
 ↪best_params_['learning_rate_init'],
#                              hidden_layer_sizes = (200,),
#                              alpha = result.best_params_['alpha'],
                            solver = 'sgd',
#                              batch_size = 250,
#                              tol = 1e-5,
#                              early_stopping = False,
#                              activation = 'relu',
#                              n_iter_no_change = 1000,
#                              max_iter = 500
                           )

model_optNN.fit(X_train1_std, y_train1)
y_pred_nn = model_optNN.predict_proba(X_test1_std)
```

```
fpr, tpr, thresholds = roc_curve(y_test1, y_pred_nn[:,1])

print ("AUC for the optimum NN is {:.3f}".format(auc(fpr, tpr)))
```

```
AUC for the optimum NN is 0.930
```

```
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

Tuned Neural Network: Re-train and apply to test dataset

This model is one of the submission, but it's not the best of our model.

[78]:
```
model_optNN = MLPClassifier(learning_rate_init = result.
 ↪best_params_['learning_rate_init'],
#                             hidden_layer_sizes = (200,),
#                             alpha = result.best_params_['alpha'],
                            solver = 'sgd',
#                             batch_size = 250,
#                             tol = 1e-5,
#                             early_stopping = False,
#                             activation = 'relu',
#                             n_iter_no_change = 1000,
#                             max_iter = 500
                           )

model_optNN.fit(X_train_std, y_train_s)
y_pred_nn = model_optNN.predict_proba(X_test_std)
```

```
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[79]:
```
create_submission(y_pred_nn[:, 1], './result/submissionN_tunedNN.csv')
```

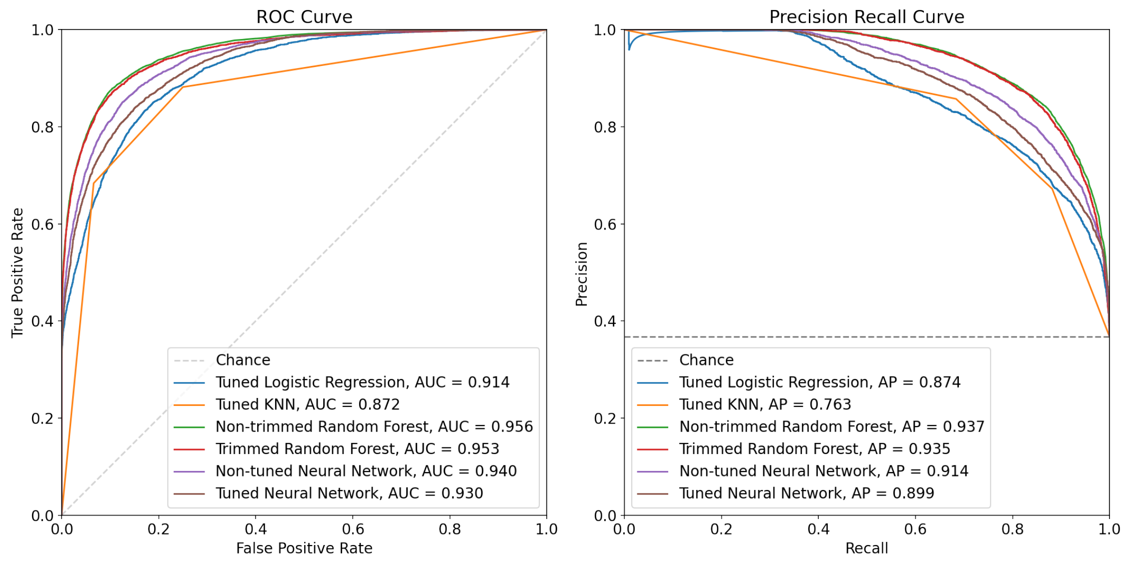
ROC, PR curve (No SVM version)

SVM is too heavy, so we plotted the ROC and PR curves without it first.

[80]:
```
# plot ROC, PR curve

fig, axs = plt.subplots(1,2,figsize=(15,10))
# fig, ax = plt.subplots(figsize=(8, 8))
```

```
models = [model_lr, model_knnOpt, model_baseRF, model_optRF,
          model_NN, model_optNN]
labels = ['Tuned Logistic Regression','Tuned KNN', 'Non-trimmed Random Forest',␣
 ↪'Trimmed Random Forest',
          'Non-tuned Neural Network', 'Tuned Neural Network']

# Plot the chance diagonal and PR random chance lines
axs[0].plot((0,1),(0,1),color='lightgrey',linestyle='--', label='Chance')
pr_baseline = sum(y_test1 == 1) / len(y_test1)
axs[1].plot((0,1),(pr_baseline, pr_baseline),color='grey',linestyle='--',␣
 ↪label='Chance')

for i, model in enumerate(models):
    print(f'Training model: {labels[i]}')
    # Fit the model to the data
    model.fit(X_train1_std, y_train1)

    # Produce confidence scores from the test data
    scores = model.predict_proba(X_test1_std)[:,1]

    # Compute the ROC and PR curve data
    fpr, tpr, _ = roc_curve(y_test1, scores)
    roc_auc = auc(fpr,tpr)

    precision, recall, _ = precision_recall_curve(y_test1, scores)
    ap = average_precision_score(y_test1, scores)

    # Plot the ROC curve
    axs[0].plot(fpr, tpr, label=labels[i] + ', AUC = {:.3f}'.format(roc_auc))

    # Plot the PR curve
    axs[1].plot(recall, precision, label=labels[i] + ', AP = {:.3f}'.format(ap))

for ax in axs:
    ax.axis('square')
    ax.set(xlim=(0,1), ylim=(0,1))
    ax.legend()
axs[0].set(xlabel='False Positive Rate', ylabel='True Positive Rate',␣
 ↪title='ROC Curve')
axs[1].set(xlabel='Recall', ylabel='Precision', title='Precision Recall Curve')
plt.tight_layout()
plt.show()
```

```
Training model: Tuned Logistic Regression
Training model: Tuned KNN
Training model: Non-trimmed Random Forest
Training model: Trimmed Random Forest
Training model: Non-tuned Neural Network

/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(

Training model: Tuned Neural Network

/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```



5. SVM

Tuneable hyper parameter: - c - kernel

We applied the bagging ensemble method to shorten the training time of SVM.

As SVM is too heavy, we only sampled 10% of the data and just select 10 features for training.

Sample 10% data

The step is not mandatory, and we sample the data to shorten the model training time in the following. However, once confirmed the process, we sampled the complete dataset.

Important Note: once the data is sample, the index has been shuffle. Thus, the index of y_train must be adjusted with the sampled X index accordingly.

```
[81]: type(X_train_ohe_imputed)
```

[81]: pandas.core.frame.DataFrame

[82]:
```
X_train_ohe_imputed_s = X_train_ohe_imputed.sample (frac = 0.1)
```

[83]:
```
y_train_s = y_train[X_train_ohe_imputed_s.index]
```

[84]:
```
# X_train_ohe_imputed_s.index
# y_train_s.index
```

[85]:
```
X_train_ohe_imputed_s = np.asanyarray(X_train_ohe_imputed_s)
y_train_s = np.asanyarray(y_train_s)
```

[86]:
```
print ("Training sample dataset shape of X, Y: ", X_train_ohe_imputed_s.shape,␣
 ↪y_train_s.shape)
print ("Training dataset shape of X, Y: ", X_train_ohe_imputed.shape, y_train.
 ↪shape)
```

```
Training sample dataset shape of X, Y:  (9551, 940) (9551,)
Training dataset shape of X, Y:  (95512, 940) (95512,)
```

Training testing split

We split the dataset into training, validation, and testing for hyperparameters tuning and model performance evaluation.

[87]:
```
from sklearn.model_selection import train_test_split

X_train1, X_test1, y_train1, y_test1 = train_test_split(X_train_ohe_imputed_s,␣
 ↪y_train_s, test_size=0.20, random_state=321)
# X_train1, X_test1, y_train1, y_test1 = train_test_split(X_train_ohe_imputed,␣
 ↪y_train, test_size=0.20, random_state=321)
X_train2, X_val1, y_train2, y_val1 = train_test_split(X_train1, y_train1,␣
 ↪test_size=0.20, random_state=321)

print ("Training dataset 1 shape of X, Y: ", X_train1.shape, y_train1.shape)
print ("Testing dataset 1 shape of X, Y: ", X_test1.shape, y_test1.shape)

print ("Training dataset 2 shape of X, Y: ", X_train2.shape, y_train2.shape)
print ("Validation dataset 1 shape of X, Y: ", X_val1.shape, y_val1.shape)
```

```
Training dataset 1 shape of X, Y:  (7640, 940) (7640,)
Testing dataset 1 shape of X, Y:  (1911, 940) (1911,)
Training dataset 2 shape of X, Y:  (6112, 940) (6112,)
Validation dataset 1 shape of X, Y:  (1528, 940) (1528,)
```

[88]:
```
std_train1 = preprocessing.StandardScaler()
X_train1_std_SVM = std_train1.fit_transform(X_train1)
```

```
std_train2 = preprocessing.StandardScaler()
X_train2_std_SVM = std_train2.fit_transform(X_train2)

std_val1 = preprocessing.StandardScaler()
X_val1_std_SVM = std_val1.fit_transform(X_val1)

std_test1 = preprocessing.StandardScaler()
X_test1_std_SVM = std_test1.fit_transform(X_test1)
```

[89]:
```
selector_SVM = SelectKBest(k = 10) # for classification: f_classif (default),␣
↪chi2; for continuous response: mutual_info_regression
selector_SVM.fit(X_train1_std_SVM, y_train1) # for chi2: minmax, for f_classif:␣
↪standard

df_features_SVN = pd.DataFrame({"Names":   X_train_ohe_imputed.
↪columns[selector_SVM.get_support()],
                                "P Value": selector_SVM.pvalues_[selector_SVM.
↪get_support()],
                                "Scores":  selector_SVM.scores_[selector_SVM.
↪get_support()]})
df_features_SVN.sort_values(by='Scores', ascending = False)
```

```
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/feature_selection/_univariate_selection.py:112: UserWarning:
Features [ 38  39  40  44  45  46  49  50  52  53  54  56  57  58  62  63  69
73
  76  80  81  91  93  97  98 101 103 104 106 110 114 118 120 123 124 125
 129 132 133 141 142 144 146 147 148 150 152 154 155 156 158 161 165 168
 172 177 179 182 183 184 186 187 189 191 196 200 201 203 206 208 209 211
 220 225 235 246 247 295 298 303 305 312 315 319 329 331 332 341 345 346
 348 352 353 356 360 363 366 369 371 374 379 382 383 384 388 392 393 395
 402 405 410 411 414 418 420 430 440 443 444 446 448 450 451 453 454 455
 457 459 460 461 462 466 467 468 469 470 471 474 476 480 481 482 485 490
 494 496 497 498 501 504 505 506 507 509 511 512 515 518 520 521 523 524
 525 526 527 528 531 532 533 534 535 537 538 540 541 542 543 546 548 550
 552 553 554 555 556 557 558 560 562 563 565 566 567 568 569 570 571 575
 576 579 580 583 584 585 587 588 589 590 592 596 598 600 603 610 613 614
 615 616 617 619 620 623 626 627 628 629 633 634 641 643 644 645 646 647
 648 649 650 651 652 657 658 659 661 664 666 667 668 672 674 678 679 680
 682 689 691 692 693 696 698 700 702 704 705 707 709 712 719 720 721 723
 724 726 729 730 731 732 733 734 736 737 738 739 740 741 742 745 746 747
 749 750 751 752 755 756 759 762 764 766 767 768 769 771 772 773 775 776
 777 779 780 781 782 783 784 785 787 789 790 791 793 794 798 800 801 802
 803 804 805 806 809 810 811 816 817 818 820 821 822 823 824 827 829 831
 832 834 835 836 837 838 841 842 843 844 845 846 848 853 854 855 856 857
 859 860 862 863 866 869 870 871 873 874 875 878 879 880 881 882 884 886
 888 890 892 893 894 895 896 897 898 899 900 901 903 904 905 907 908 910
 911 913 914 916 918 919 920 921 922 923 924 925 928 929 930 931 932 933
 934 935] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
UserWarning)
/Users/tegochang/opt/miniconda3/lib/python3.9/site-
packages/sklearn/feature_selection/_univariate_selection.py:113: RuntimeWarning:
invalid value encountered in true_divide
  f = msb / msw
```

[89]:

```
                        Names        P Value       Scores
8      deposit_type_Non Refund   0.000000e+00  2216.511690
7      deposit_type_No Deposit   0.000000e+00  2164.048613
3                  country_PRT  2.241736e-204   989.678006
0                    lead_time  1.366178e-155   739.886889
2    total_of_special_requests   1.491414e-94   437.685694
4        market_segment_Groups   7.224118e-92   424.651989
6          assigned_room_type_A   1.352556e-80   370.176593
9                    agent_1.0   3.122757e-69   315.655452
1  required_car_parking_spaces   2.345449e-65   297.151273
5   distribution_channel_TA/TO   2.642743e-57   258.858634
```

[90]:

```python
X_train1_std_SVM = X_train1_std_SVM[:, selector_SVM.get_support()]
X_train2_std_SVM = X_train2_std_SVM[:, selector_SVM.get_support()]
X_val1_std_SVM = X_val1_std_SVM[:, selector_SVM.get_support()]
X_test1_std_SVM = X_test1_std_SVM[:, selector_SVM.get_support()]
```

Bagging with SVM

[91]:

```python
from sklearn import svm
```

[92]:

```python
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier

model_BagSVM = BaggingClassifier(base_estimator=SVC(probability=True),
                                 n_estimators=10,
                                 random_state=0).fit(X_train1_std_SVM, y_train1)
y_pred_svm = model_BagSVM.predict_proba(X_test1_std_SVM)
fpr, tpr, thresholds = roc_curve(y_test1, y_pred_svm[:,1])

print ("AUC for the base SVM is {:.3f}".format(auc(fpr, tpr)))
```

```
AUC for the base SVM is 0.815
```

4 2

4.1 [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.

(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 know as "inertia" and can be accessed through the `inertia_` attribute of a fit KMeans class instance. - For each datasets, where is the elbow in the curve of within-cluster sum-of-squares and why? Is the elbow always clearly visible? When its not clear, you will have to use your judgement 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 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 judgement 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

[1]:
```
##################################
# Load the data
##################################
import os
import numpy as np
```

```python
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
```

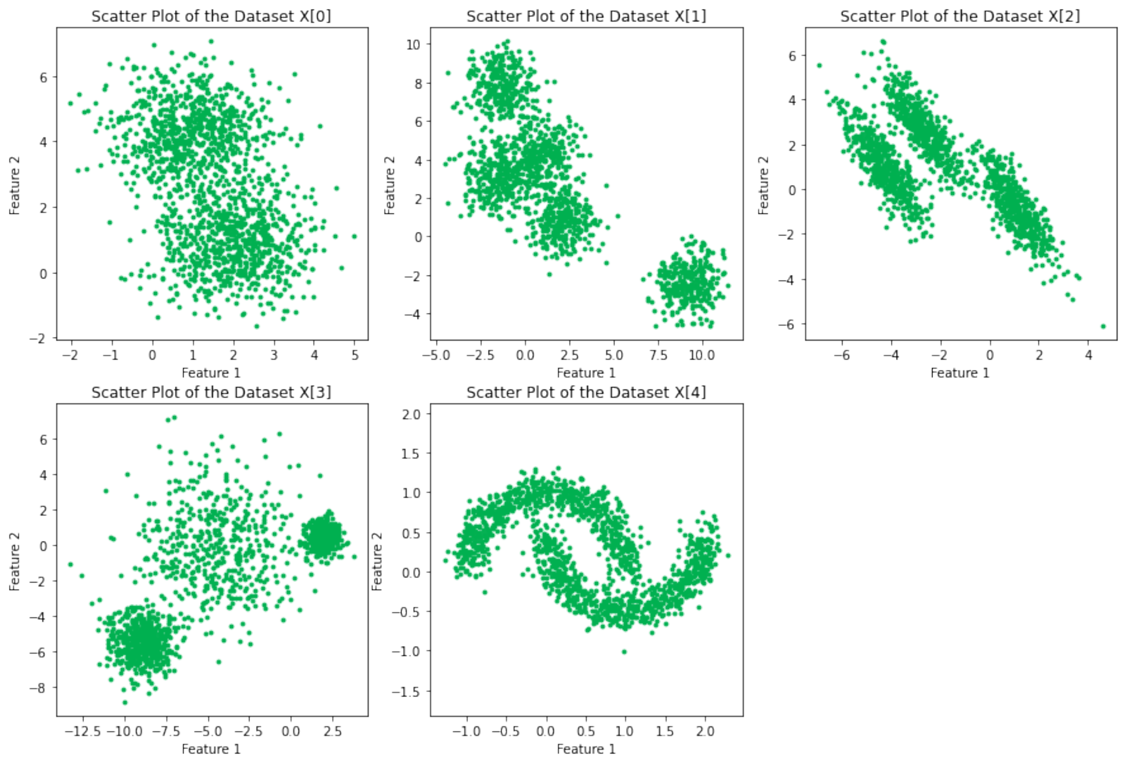
[2]:

```python
################################
# 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)
```

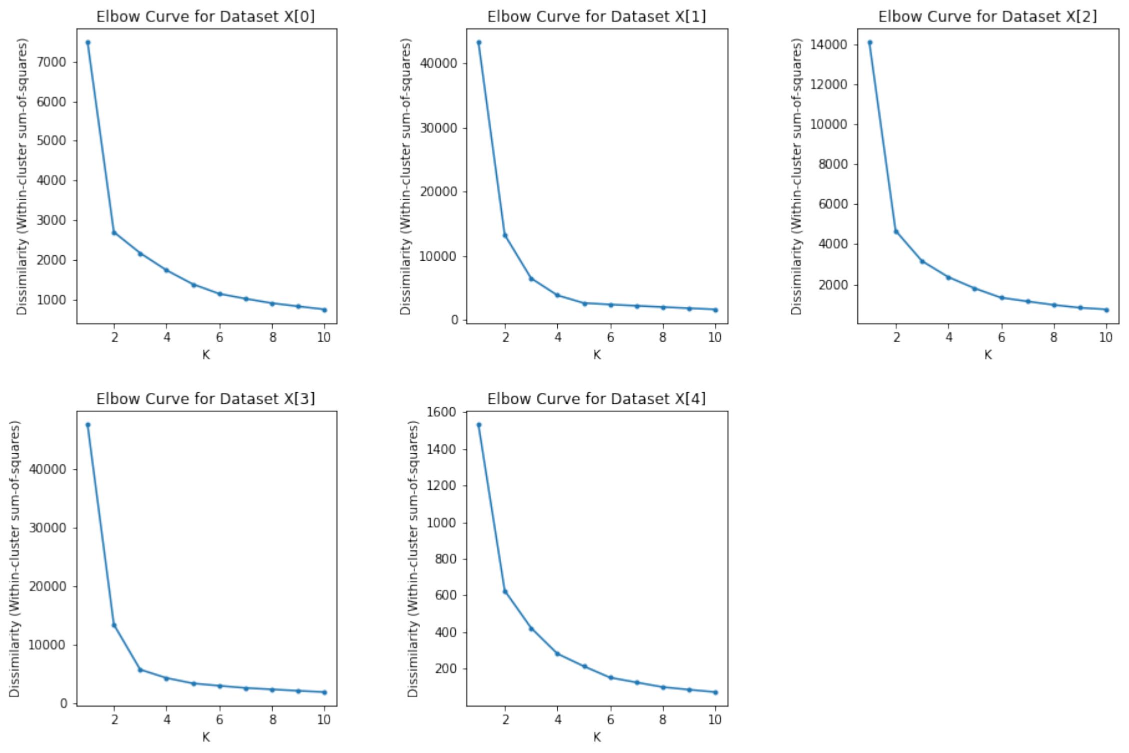
```
        ax.set_xlabel('feature 1')
        ax.set_ylabel('feature 2')
        ax.set_title(f'No. Clusters = {n_clusters}')
        ax.axis('equal')
```

ANSWER

[3]:
```
# scatter plot the data
color0 = '#121619' # Custom color: Dark grey
color1 = '#00B050' # Custom color: Green

fig, axs = plt.subplots(2,3,figsize=(15,10))

for i in range(len(X)):
    axs[i//3, i%3].scatter(X[i][:, 0], X[i][:, 1], color=color1, s=8)
    axs[i//3, i%3].set_xlabel("Feature 1")
    axs[i//3, i%3].set_ylabel("Feature 2")
    axs[i//3, i%3].set_title("Scatter Plot of the Dataset X[{0}]".format(i))
#      {:.3f}".format(auc(fpr, tpr)))
#      axs[i//3, i%3].legend()

axs[1, 2].remove()
plt.axis('equal')
plt.show()
```



[4]:
```
from sklearn.cluster import KMeans

K_array = np.arange(1,11)

fig, axs = plt.subplots(2,3,figsize=(15,10))
fig.subplots_adjust(hspace=0.3, wspace=0.5)

disSim = []

for i in range(len(X)):
    disSim.append([])
    for k in range(1, 11):
        kms = KMeans(n_clusters = k, random_state = 0).fit(X[i])
        disSim[i].append(kms.inertia_)
        pass
    axs[i//3, i%3].plot(K_array, disSim[i], marker='.')
    axs[i//3, i%3].set_xlabel("K")
    axs[i//3, i%3].set_ylabel("Dissimilarity (Within-cluster sum-of-squares)")
    axs[i//3, i%3].set_title("Elbow Curve for Dataset X[{0}]".format(i))
    pass

axs[1, 2].remove()
plt.show()
```



Based on the below calculations of the values of slope change in each K value, we shall say the K shall be set to 2 for all 5 dataset. However, based on our judgement through visualization, except for X[0] and X[4], we shall not set K = 2 for the other three datasets. K = 4 for X[1], K = 3 for X[2], and K = 3 for X[3].

```
[5]: # abs_m = []

     # for i in range(len(X)):
     #     abs_m.append([i])
     #     for m in range(len(disSim[i])-1):
     #         abs_m[i].append([m+1])
     #         abs_m[i][m+1].append(abs(disSim[i][m] - disSim[i][m+1]))
```

```
[6]: abs_m = []

     for m in range(len(disSim[0])-1):
         abs_m.append([m+1])
         abs_m[m].append(abs(disSim[0][m] - disSim[0][m+1]))

     abs_m_delta = []

     for m_delta in range(len(disSim[0])-2):
         abs_m_delta.append([m_delta+2])
         abs_m_delta[m_delta].append(abs(abs_m[m_delta][1] - abs_m[m_delta+1][1]))

     abs_m_delta = np.asanyarray(abs_m_delta)

     print ("The elbow point for the dataset X[0] is when K = ", abs_m_delta[np.
      ↪argmax(abs_m_delta[:,1]), 0].astype(int))
```

```
The elbow point for the dataset X[0] is when K =  2
```

```
[7]: abs_m = []

     for m in range(len(disSim[1])-1):
         abs_m.append([m+1])
         abs_m[m].append(abs(disSim[1][m] - disSim[1][m+1]))

     abs_m_delta = []

     for m_delta in range(len(disSim[1])-2):
         abs_m_delta.append([m_delta+2])
         abs_m_delta[m_delta].append(abs(abs_m[m_delta][1] - abs_m[m_delta+1][1]))

     abs_m_delta = np.asanyarray(abs_m_delta)
```

```
print ("The elbow point for the dataset X[1] is when K = ", abs_m_delta[np.
 ↪argmax(abs_m_delta[:,1]), 0].astype(int))
```

```
The elbow point for the dataset X[1] is when K =  2
```

[8]:
```
abs_m = []

for m in range(len(disSim[2])-1):
    abs_m.append([m+1])
    abs_m[m].append(abs(disSim[2][m] - disSim[2][m+1]))

abs_m_delta = []

for m_delta in range(len(disSim[2])-2):
    abs_m_delta.append([m_delta+2])
    abs_m_delta[m_delta].append(abs(abs_m[m_delta][1] - abs_m[m_delta+1][1]))

abs_m_delta = np.asanyarray(abs_m_delta)

print ("The elbow point for the dataset X[2] is when K = ", abs_m_delta[np.
 ↪argmax(abs_m_delta[:,1]), 0].astype(int))
```

```
The elbow point for the dataset X[2] is when K =  2
```

[9]:
```
abs_m = []

for m in range(len(disSim[3])-1):
    abs_m.append([m+1])
    abs_m[m].append(abs(disSim[3][m] - disSim[3][m+1]))

abs_m_delta = []

for m_delta in range(len(disSim[3])-2):
    abs_m_delta.append([m_delta+2])
    abs_m_delta[m_delta].append(abs(abs_m[m_delta][1] - abs_m[m_delta+1][1]))

abs_m_delta = np.asanyarray(abs_m_delta)

print ("The elbow point for the dataset X[3] is when K = ", abs_m_delta[np.
 ↪argmax(abs_m_delta[:,1]), 0].astype(int))
```

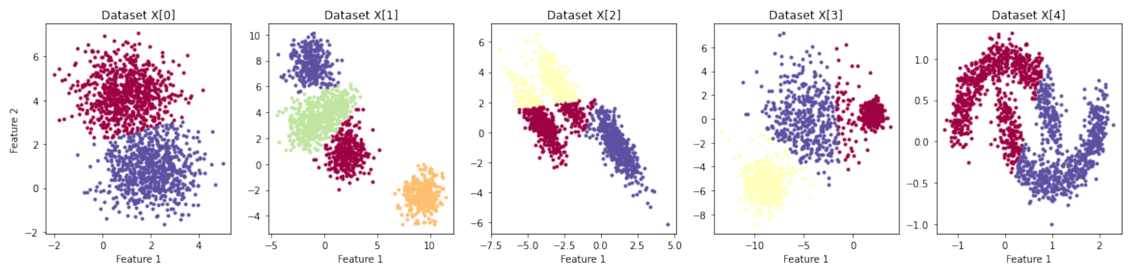
```
The elbow point for the dataset X[3] is when K =  2
```

[10]:
```
abs_m = []

for m in range(len(disSim[4])-1):
    abs_m.append([m+1])
    abs_m[m].append(abs(disSim[4][m] - disSim[4][m+1]))

abs_m_delta = []

for m_delta in range(len(disSim[4])-2):
    abs_m_delta.append([m_delta+2])
    abs_m_delta[m_delta].append(abs(abs_m[m_delta][1] - abs_m[m_delta+1][1]))

abs_m_delta = np.asanyarray(abs_m_delta)

print ("The elbow point for the dataset X[4] is when K = ", abs_m_delta[np.
↪argmax(abs_m_delta[:,1]), 0].astype(int))
```

```
The elbow point for the dataset X[4] is when K =  2
```

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

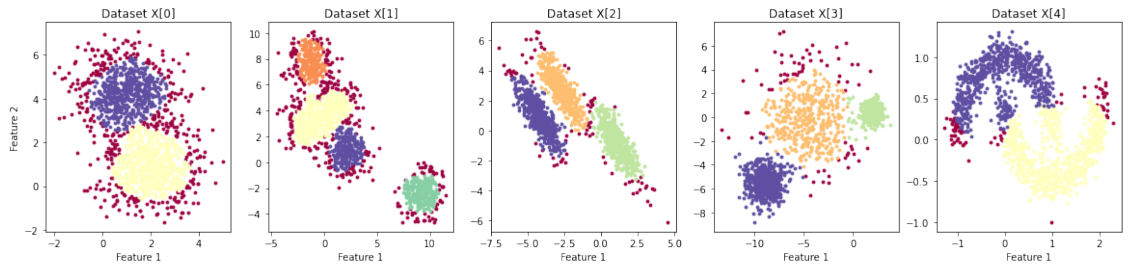
[11]:
```
# scatter plot the data
# color0 = '#121619' # Custom color: Dark grey
# color1 = '#00B050' # Custom color: Green



fig, axs = plt.subplots(1,5,figsize=(20,4))
# fig.subplots_adjust(hspace=0.3, wspace=0.5)

K_decision = [2, 4, 3, 3, 2]

for i in range(len(X)):
    kms = KMeans(n_clusters = K_decision[i], random_state = 0).fit(X[i])
    colors = plt.cm.Spectral(np.linspace(0, 1, len(np.unique(kms.labels_)))) #␣
↪spectral color map divided into K colors

    for k, col in zip(range(len(np.unique(kms.labels_))), colors):
        group_members = (kms.labels_ == k)
        axs[i].scatter(X[i][group_members, 0], X[i][group_members, 1],␣
↪color=col, s=8)
        axs[i].set_xlabel("Feature 1")
        axs[0].set_ylabel("Feature 2")
        axs[i].set_title("Dataset X[{0}]".format(i))

# plt.axis('equal')
plt.show()
```



[12]:
```
from sklearn.cluster import DBSCAN

for i in range(len(X)):
    print ("For dataset X[{0}]".format(i))
    for e in np.arange(5, 100, 10):
        for s in np.arange(1, 401, 10):
            dbsc = DBSCAN(eps=e/10, min_samples=s).fit(X[i])
            if (len(np.unique(dbsc.labels_)) == K_decision[i]+1) & (-1 in␣
 ↪list(np.unique(dbsc.labels_))):
                print ("The qualified eps for {0} clusters: {1}".
 ↪format(K_decision[i], e/10))
                print ("The qualified min_samples for {0} clusters: {1}".
 ↪format(K_decision[i], s))
                break
```

```
For dataset X[0]
The qualified eps for 2 clusters: 0.5
The qualified min_samples for 2 clusters: 51
For dataset X[1]
The qualified eps for 4 clusters: 0.5
The qualified min_samples for 4 clusters: 21
The qualified eps for 4 clusters: 1.5
The qualified min_samples for 4 clusters: 191
For dataset X[2]
The qualified eps for 3 clusters: 0.5
The qualified min_samples for 3 clusters: 21
For dataset X[3]
The qualified eps for 3 clusters: 1.5
The qualified min_samples for 3 clusters: 51
The qualified eps for 3 clusters: 2.5
The qualified min_samples for 3 clusters: 181
For dataset X[4]
The qualified eps for 2 clusters: 0.5
The qualified min_samples for 2 clusters: 251
```

[13]:
```
fig, axs = plt.subplots(1,5,figsize=(20,4))
# fig.subplots_adjust(hspace=0.3, wspace=0.5)

eps_sample_setup = [[0.5, 51], [0.5, 21], [0.5, 21], [1.5, 51], [0.5, 251]]

for i in range(len(X)):
    dbsc = DBSCAN(eps=eps_sample_setup[i][0],␣
 ↪min_samples=eps_sample_setup[i][1]).fit(X[i])
    colors = plt.cm.Spectral(np.linspace(0, 1, len(np.unique(dbsc.labels_)))) #␣
 ↪spectral color map divided into K colors

    for k, col in zip(range(len(np.unique(dbsc.labels_))), colors):
        group_members = (dbsc.labels_ == k-1)
        axs[i].scatter(X[i][group_members, 0], X[i][group_members, 1],␣
 ↪color=col, s=8)
        axs[i].set_xlabel("Feature 1")
        axs[0].set_ylabel("Feature 2")
        axs[i].set_title("Dataset X[{0}]".format(i))

# plt.axis('equal')
plt.show()
```

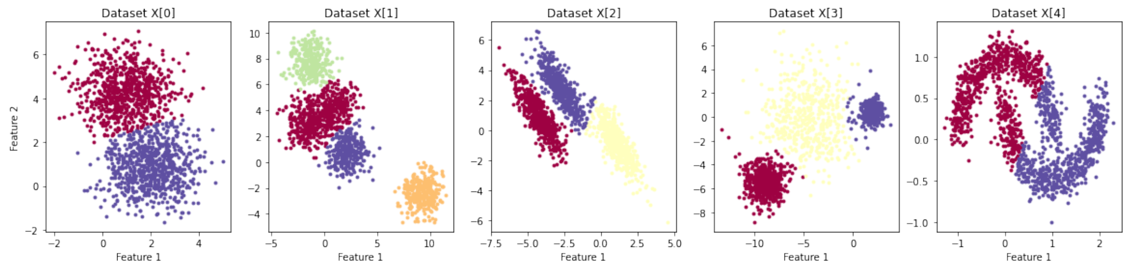


[14]:
```
from sklearn.cluster import SpectralClustering

sc = SpectralClustering(n_clusters=4,
                        random_state=0,
                        assign_labels='discretize',).fit(X[1])
np.unique(sc.labels_)
np.sum(sc.labels_==0)
```

[14]:
```
585
```

[15]:
```
# scatter plot the data
# color0 = '#121619' # Custom color: Dark grey
# color1 = '#00B050' # Custom color: Green
```

```
fig, axs = plt.subplots(1,5,figsize=(20,4))
# fig.subplots_adjust(hspace=0.3, wspace=0.5)

K_decision = [2, 4, 3, 3, 2]

for i in range(len(X)):
    sc = SpectralClustering(n_clusters = K_decision[i], random_state = 0).
 ↪fit(X[i])
    colors = plt.cm.Spectral(np.linspace(0, 1, K_decision[i])) # spectral color␣
 ↪map divided into K colors

    for k, col in zip(range(K_decision[i]), colors):
        group_members = (sc.labels_ == k)
        axs[i].scatter(X[i][group_members, 0], X[i][group_members, 1],␣
 ↪color=col, s=8)
        axs[i].set_xlabel("Feature 1")
        axs[0].set_ylabel("Feature 2")
        axs[i].set_title("Dataset X[{0}]".format(i))

# plt.axis('equal')
plt.show()
```

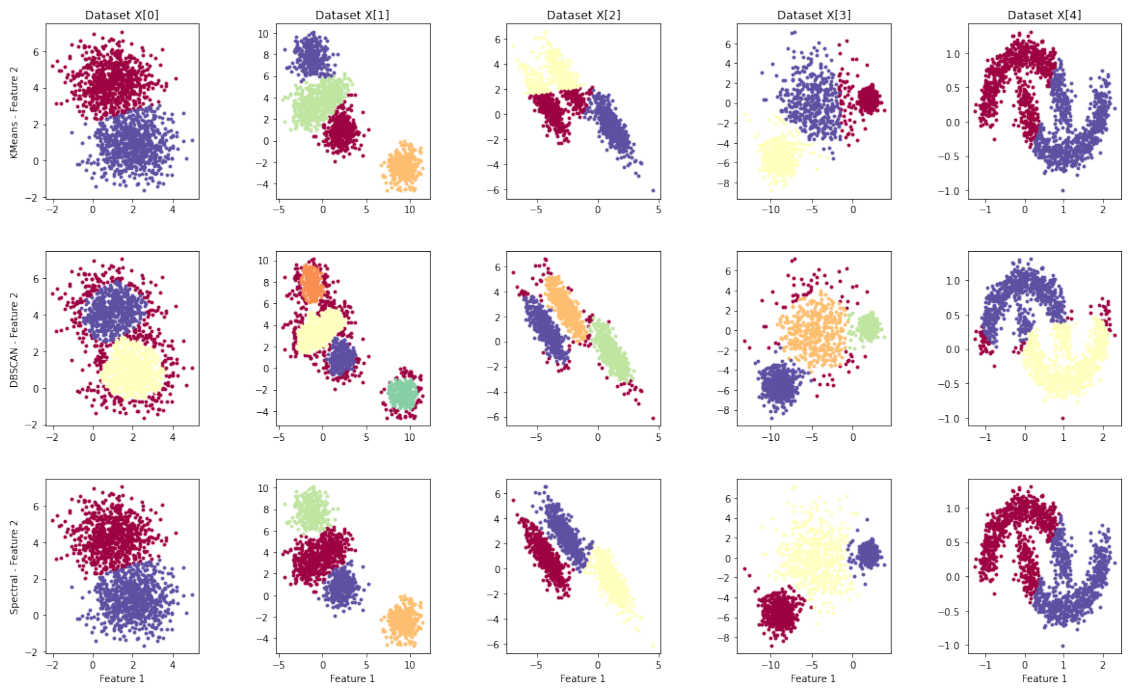


Based on the below combination plot of the three cluster algorithms and the five datasets, we found that:

- X[0] and X[1]: Three algorithms perform almost the same. However, DBSCAN will identify outliers in the dataset, which is a different behavior compared with the other two (soft clustering).
- X[2] and X[3]: K-Means performs the worst as it failed to handle clusters with correlation between features. Spectral Clustering has some misdetection between cluster 2 and 3. Thus, DBSCAN performs the best in this dataset.
- X[4]: Both K-Means and Spectral clustering failed to identify the non-linear cluster boundary, DBSCAN outperforms the other two with only some mis-identifications.

Overall, DBSCAN performs better in the five datasets we investigated compared with the other two. However, in the following analysis, we shall still base on the characteristics on each clustering algoritm to decide each to apply. There suppose will not have a algorithm dominating all the applications.

[16]:
```
fig, axs = plt.subplots(3,5,figsize=(20,12))
fig.subplots_adjust(hspace=0.3, wspace=0.5)

K_decision = [2, 4, 3, 3, 2]
eps_sample_setup = [[0.5, 51], [0.5, 21], [0.5, 21], [1.5, 51], [0.5, 251]]
# model = [KMeans, DBSCAN, SpectralClustering]
# model_var = [kms, dbsc, sc]

for i in range(len(X)):
    kms = KMeans(n_clusters = K_decision[i], random_state = 0).fit(X[i])
    dbsc = DBSCAN(eps=eps_sample_setup[i][0],␣
↪min_samples=eps_sample_setup[i][1]).fit(X[i])
    sc = SpectralClustering(n_clusters = K_decision[i], random_state = 0).
↪fit(X[i])

    for k in range(len(np.unique(kms.labels_))):
        kms_group_members = (kms.labels_ == k)
        k_colors = plt.cm.Spectral(np.linspace(0, 1, len(np.unique(kms.
↪labels_)))) # spectral color map divided into K colors
        axs[0, i%5].scatter(X[i][kms_group_members, 0], X[i][kms_group_members,␣
↪1], color=k_colors[k], s=8)
        axs[0, 0].set_ylabel("KMeans - Feature 2")

    for k in range(len(np.unique(dbsc.labels_))):
        dbsc_group_members = (dbsc.labels_ == k-1)
        dbsc_colors = plt.cm.Spectral(np.linspace(0, 1, len(np.unique(dbsc.
↪labels_)))) # spectral color map divided into K colors
        axs[1, i%5].scatter(X[i][dbsc_group_members, 0],␣
↪X[i][dbsc_group_members, 1], color=dbsc_colors[k], s=8)
        axs[1, 0].set_ylabel("DBSCAN - Feature 2")

    for k in range(len(np.unique(sc.labels_))):
        sc_group_members = (sc.labels_ == k)
        sc_colors = plt.cm.Spectral(np.linspace(0, 1, len(np.unique(sc.
↪labels_)))) # spectral color map divided into K colors
        axs[2, i%5].scatter(X[i][sc_group_members, 0], X[i][sc_group_members,␣
↪1], color=sc_colors[k], s=8)
        axs[2, 0].set_ylabel("Spectral - Feature 2")

    axs[2, i%5].set_xlabel("Feature 1")
    axs[0, i%5].set_title("Dataset X[{0}]".format(i))

# plt.axis('equal')
plt.show()
```



5 3

5.1 [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 t-SNE doesn't have a `fit` method, but only a `fit_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.

[17]:
```
#################################
# 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

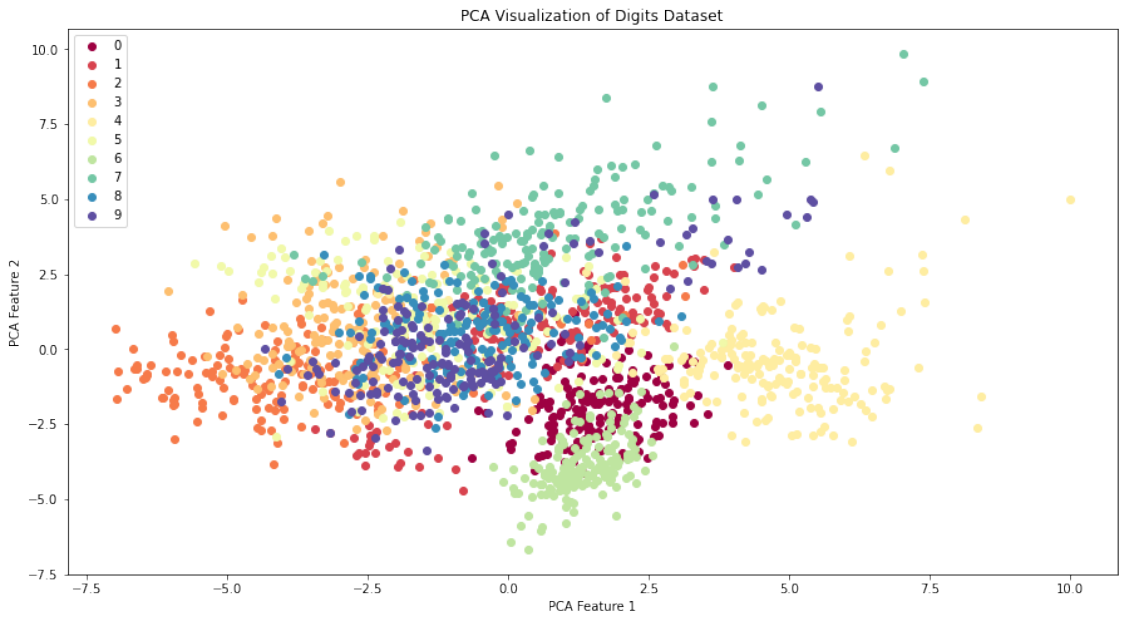
[19]:
```
from sklearn import preprocessing

std_q3 = preprocessing.StandardScaler()
X_digits_std = std_q3.fit_transform(X_digits)
```

[20]:
```
pca_q3 = PCA(n_components=2, random_state=0)
pca_q3.fit(X_digits_std)
```

[20]:
```
PCA(n_components=2, random_state=0)
```

[21]:
```
X_digits_std_pca = pca_q3.transform(X_digits_std)
```

[22]:
```
colors = plt.cm.Spectral(np.linspace(0, 1, len(np.unique(y_digits)))) #␣
 ↪spectral color map divided into K colors

fig, ax = plt.subplots(figsize=(15,8))

for i in range(len(np.unique(y_digits))):
    digit_member = (y_digits == i)
    ax.scatter(X_digits_std_pca[digit_member, 0],␣
 ↪X_digits_std_pca[digit_member, 1],
               color = colors[i], label= y_digits[i])

ax.set_xlabel("PCA Feature 1")
ax.set_ylabel("PCA Feature 2")
ax.set_title("PCA Visualization of Digits Dataset")
ax.legend()
plt.show()
```

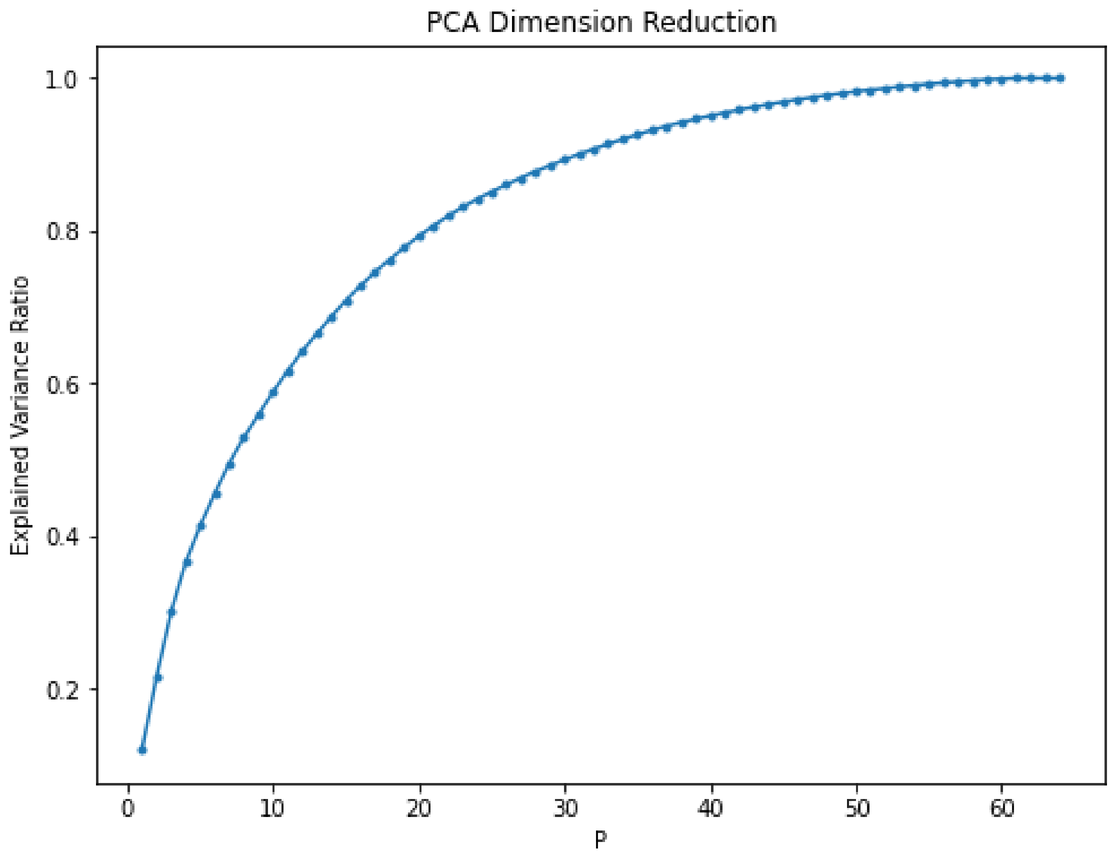


The higher ratio of unexplained variance means that the data is less distinguishable in the defined PCA dimensions. This would make the clustering being less well-performed. On the other hand, the higher the explained variance ratio means the data is more distinguishable given the defined PCA demensions, which makes clustering more well-performed.

[23]:
```
pca_q3b1 = PCA(n_components=2, random_state=0)
pca_q3b1.fit(X_digits_std)
print ("The fraction of variance that is unexplained by the first two principal␣
 ↪components is: {:.3f}".format(1-np.sum(pca_q3b1.explained_variance_ratio_)))
```

```
The fraction of variance that is unexplained by the first two principal
components is: 0.784
```

[24]:
```
fig, ax = plt.subplots(figsize = (8,6))

p_array = []
var_array = []

for p in range(1, X_digits.shape[1]+1):
    p_array.append(p)
    pca_q3b = PCA(n_components=p, random_state=0)
    pca_q3b.fit(X_digits_std)
    var_array.append(np.sum(pca_q3b.explained_variance_ratio_))

ax.plot(p_array, var_array, marker = '.')
ax.set_xlabel("P")
ax.set_ylabel("Explained Variance Ratio")
ax.set_title("PCA Dimension Reduction")

plt.show()
```

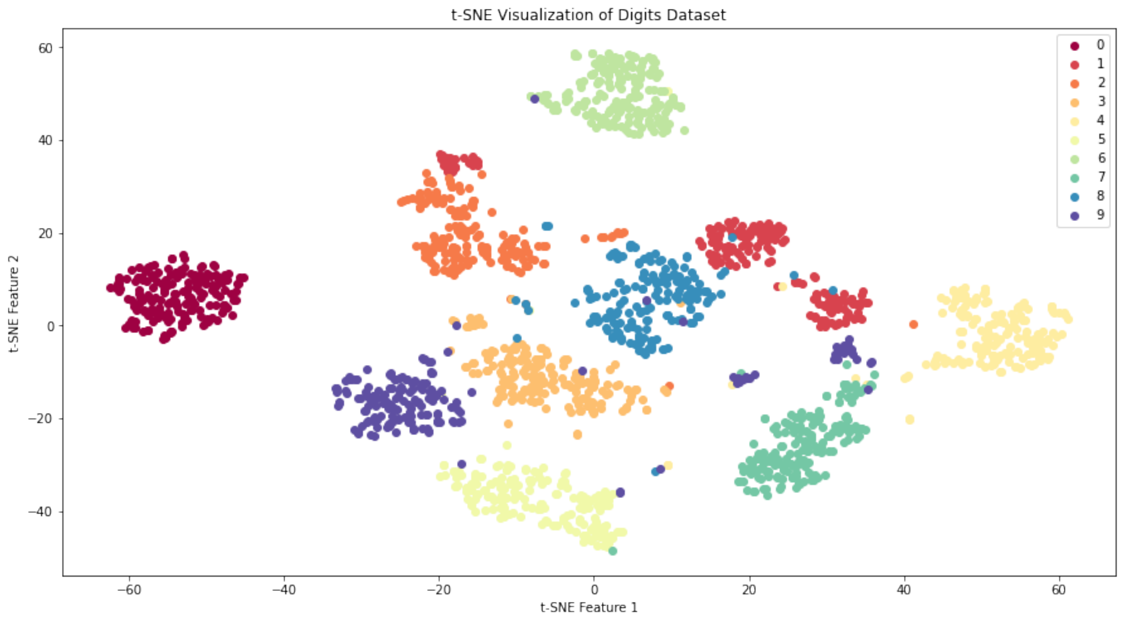


[25]:
```
from sklearn.manifold import TSNE

X_digits_std_tsne = TSNE(n_components=2, learning_rate='auto',
                    init='random').fit_transform(X_digits_std)
X_digits_std_tsne.shape
```

[25]:
```
(1797, 2)
```

[26]:
```
colors = plt.cm.Spectral(np.linspace(0, 1, len(np.unique(y_digits)))) #␣
 ↪spectral color map divided into K colors
```

```
fig, ax = plt.subplots(figsize=(15,8))

for i in range(len(np.unique(y_digits))):
    digit_member = (y_digits == i)
    ax.scatter(X_digits_std_tsne[digit_member, 0],␣
 ↪X_digits_std_tsne[digit_member, 1],
               color = colors[i], label= y_digits[i])

ax.set_xlabel("t-SNE Feature 1")
ax.set_ylabel("t-SNE Feature 2")
ax.set_title("t-SNE Visualization of Digits Dataset")
ax.legend()
plt.show()
```



t-SNE seems to work a lot better compared with PCA in data visualization after dimension reduction. The reason is that there's still a lot room of unexplained variance ratio for PCA and t-SNE is specifically tuned for displayed in 2D or 3D.

t-SNE doesn't have a *fit* function means that it can not be applicable to other dataset except for the current dataset it is applied to. This implie that even you clustered well on one dataset through t-SNE, you can not apply to local characteristics of this dataset to other dataset to replicate the same performance of clustering.