

Assignment 5

Kaggle Competition and Unsupervised Learning

Kyle Bradbury

2025-03-05

Table of contents

Instructions

Instructions for all assignments can be found [here](#). Note: this assignment falls under collaboration Mode 2: Individual Assignment – Collaboration Permitted. Please refer to the syllabus for additional information. Please be sure to list the names of any students that you worked with on this assignment. Total points in the assignment add up to 90; an additional 10 points are allocated to professionalism and 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

Exercise 1 - Kaggle Classification Competition

[40 points]

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 note

Follow the link posted on Ed to register for the competition. You can view the public leaderboard anytime at the Kaggle website (see the Ed post).

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 (see Ed Discussions for the link). 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 **3, 2, and 1 bonus points in this assignment, respectively** (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 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:

1.1. 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.

1.2. 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.

1.3. 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 `sklearn` 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

1.4. 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 than 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.

```
import pandas as pd
import numpy as np
import pickle

#####
# Load the data
#####
data = pd.read_pickle("./data/a5_q1.pkl")

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")

```

Exercise 2 - Clustering

[25 points]

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.

2.1. 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 sklearn 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.

2.2. 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. The same code plots as gray/black any points that were not assigned to clusters.

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

2.4. 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)?

Important 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

```
#####
# Load the data
#####
import os
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

```

```

#####
# 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')

```

Exercise 3 - Dimensionality reduction and visualization of digits with PCA and t-SNE

[25 points]

3.1. Reduce the dimensionality of the data with PCA for data visualization. Load the `scikit-learn` digits dataset (code provided to do this below). Consider whether any pre-processing may need to be applied (do the data need to be normalized?). 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).

3.2. 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*

3.3. 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](#).

3.4. Briefly 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.

```
#####
# 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
```