Due date: **October 24, 2023**

Instructions:

- The homework is due by 11:59PM EST on the due date. Please upload a PDF version of your assignment on ZoneCours.
- The homework is worth 20% of the course's final grade.
- Assignments are to be done individually.

- Please provide your code answers in the code block under each question and verbal answers in text boxes assigned in the notebook (where applicable).
- Please run the notebook before the submission so that the outputs are displayed.
- Please make sure that your results are reproducible. You may use random seeds from `random` and `numpy` packages. For `scikit-learn` modules, you may use the `random_state` argument.

In [ ]:
```
# enter you full name and HEC ID
full_name = "Charles Julien"
HEC_ID = "11289334"
```

Classification (7pt)

We will use a synthetic dataset. It is available here.

Once the data are accessible from your current working directory, you can load them using the following code:

`data = np.load("a22_devoir_q2-classification.npz")`

`X = data["X"]`

`y = data["y"]`

In [ ]:
```
# download and load the data
!wget -P /content https://github.com/davoodwadi/MATH60629A.A2023-MACHINE-LEARNING-I/raw/

import numpy as np
data = np.load('a22_devoir_q2-classification.npz')
X = data["X"]
y = data["y"]
```

```
--2023-10-24 20:07:58--  https://github.com/davoodwadi/MATH60629A.A2023-MACHINE-LEARNING
-I/raw/main/data/a22_devoir_q2-classification.npz
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/davoodwadi/MATH60629A.A2023-MACHINE-LEARNING
-I/main/data/a22_devoir_q2-classification.npz [following]
--2023-10-24 20:07:58--  https://raw.githubusercontent.com/davoodwadi/MATH60629A.A2023-M
ACHINE-LEARNING-I/main/data/a22_devoir_q2-classification.npz
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.
199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
```

```
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2890 (2.8K) [application/octet-stream]
Saving to: '/content/a22_devoir_q2-classification.npz'

a22_devoir_q2-class 100%[===================>]   2.82K  --.-KB/s    in 0s

2023-10-24 20:07:58 (41.2 MB/s) - '/content/a22_devoir_q2-classification.npz' saved [289
0/2890]
```

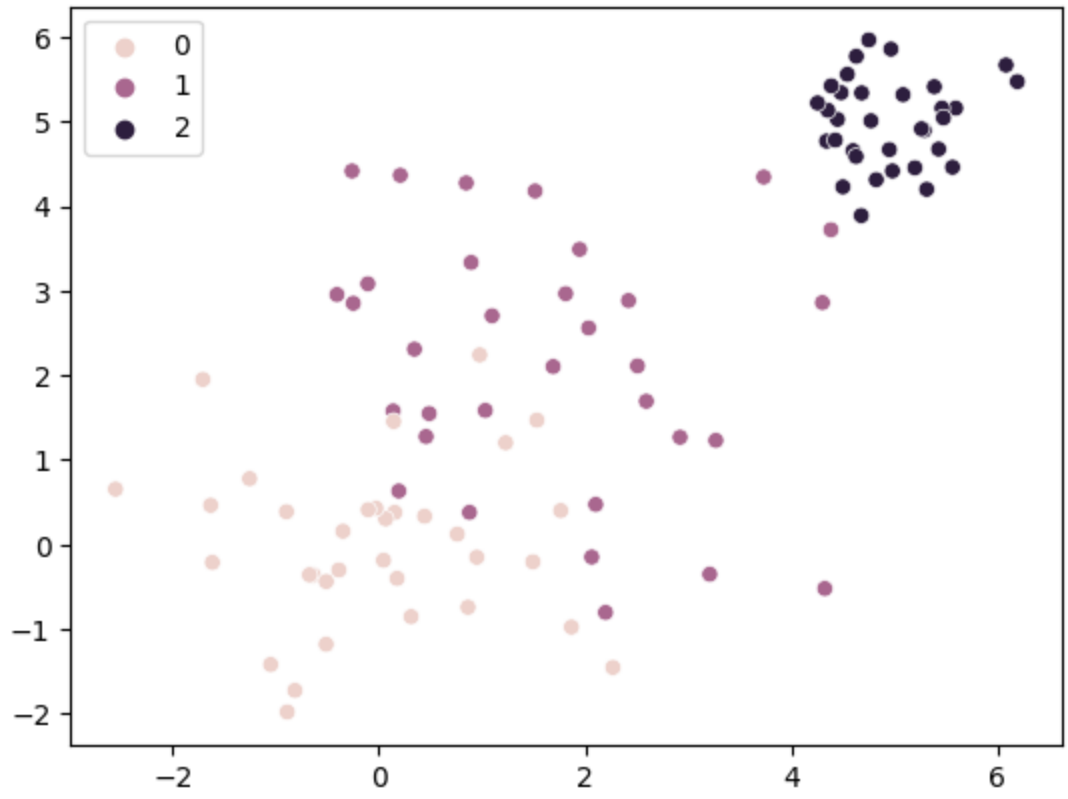
Exploratory Data Analysis

In [ ]:
```
X.shape
```

Out[ ]:
```
(100, 2)
```

texte du lien1. (**0.5pt**) Plot the features, X, and color each datum based on its class. (*Hint:* You may use the `hue` argument in the `seaborn` package.)

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(data=X, x=X[:, 0], y=X[:, 1], hue=y)
plt.show()
```



1. (**0.5pt**) Based on the plot, is a classifier with a linear margin a good choice for this classification task?

answer

Yes the data points could be well seperated with a linear margin although not perfectly.

3. (1pt) Split you data into *training*, *validation*, and *test* sets.

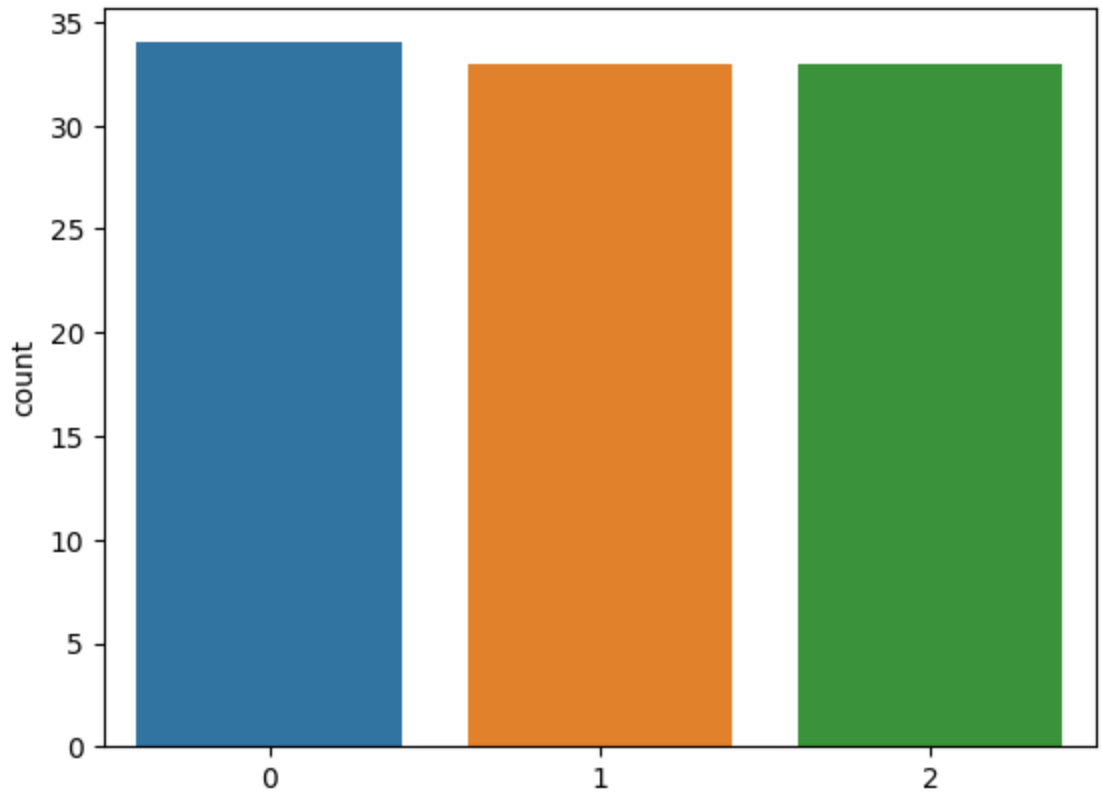
- Use 60% of the data for training, 20% for validation, and 20% for testing

- If using `train_test_split` function from `scikit-learn`, use `random state=1234`

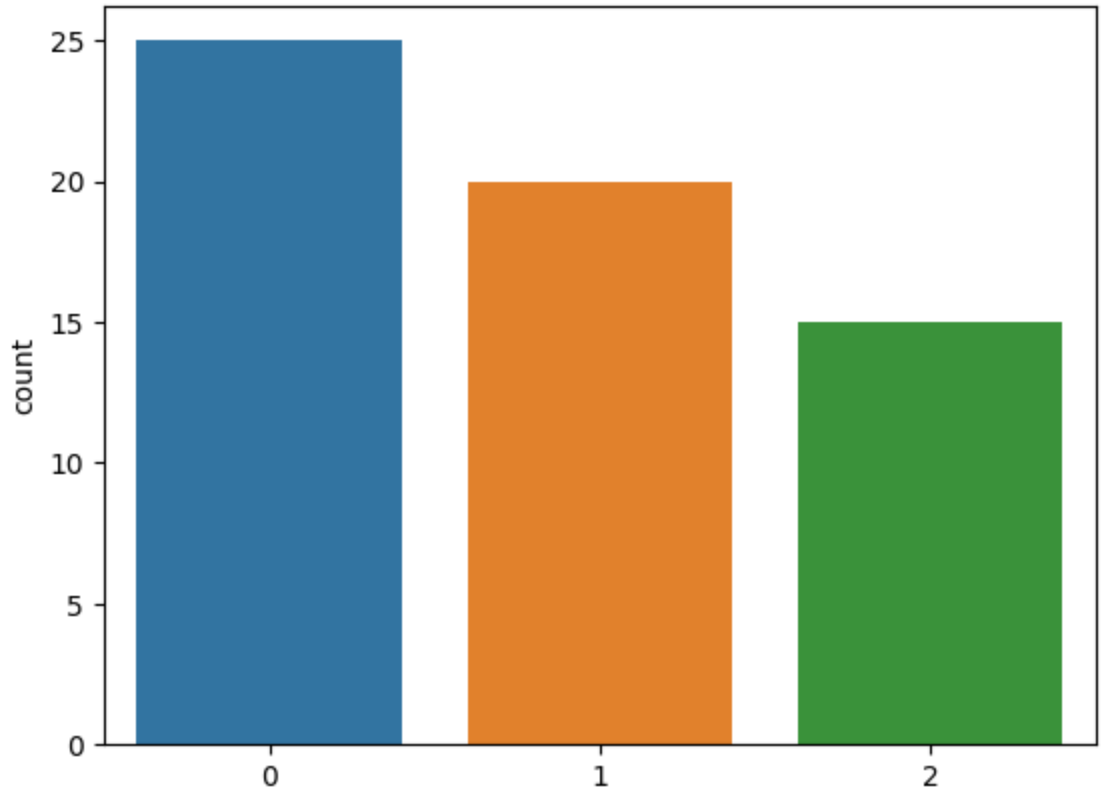
In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, rand
```

1. **(1pt)** Explore the distribution of the classes in the entire dataset, training set, validation set, and test set.

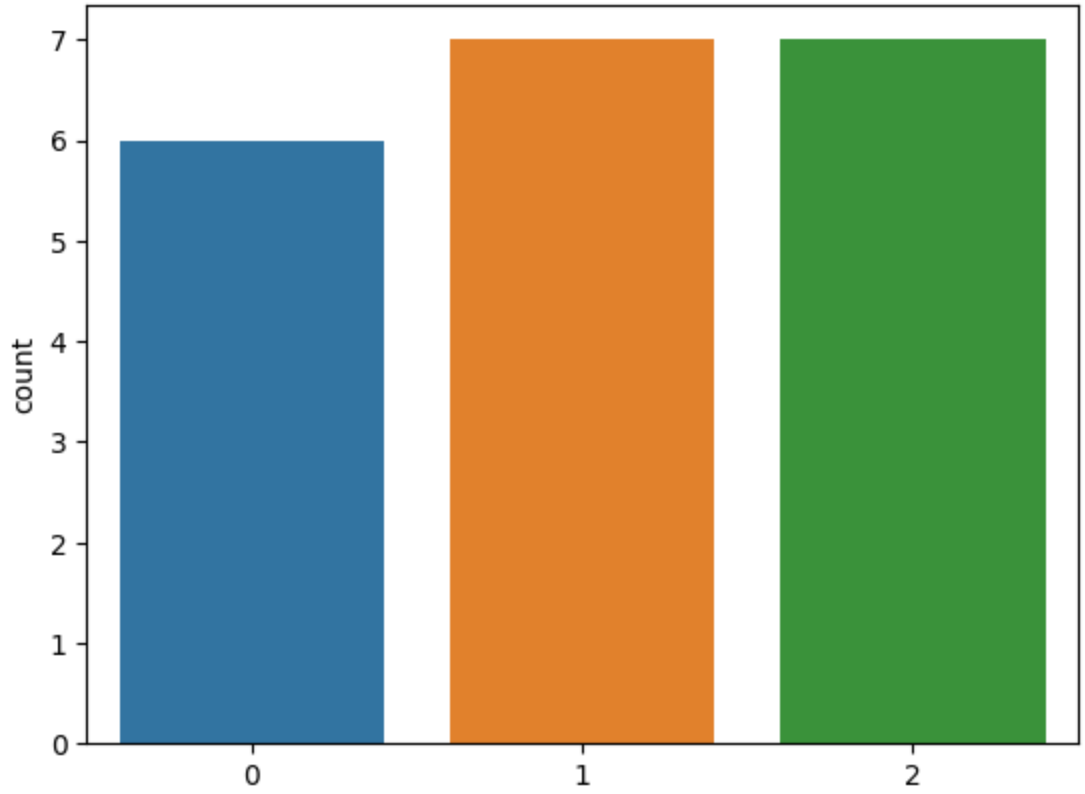
In [ ]:
```
# distribution of the entire dataset
sns.countplot(x=y)
plt.show()
```



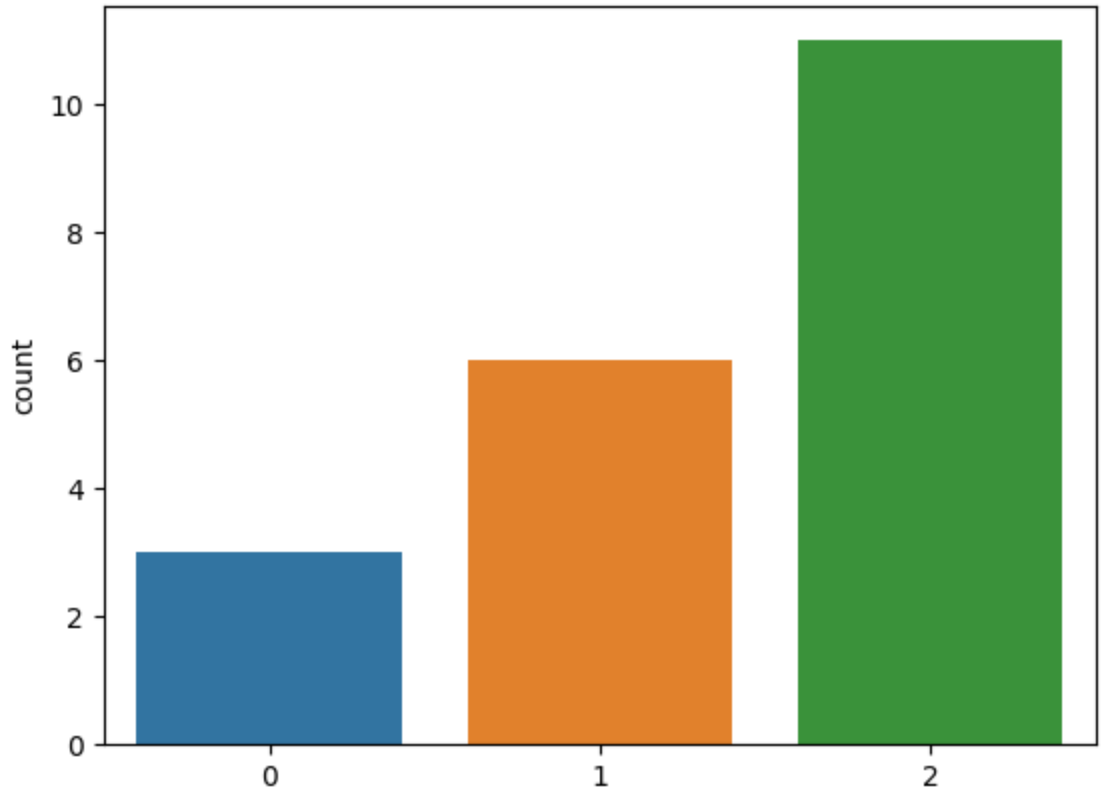
In [ ]:
```
# distribution of the training set
sns.countplot(x=y_train)
plt.show()
```



In [ ]:
```
# distribution of the validation set
sns.countplot(x=y_val)
plt.show()
```



In [ ]:
```
# distribution of the test set
sns.countplot(x=y_test)
plt.show()
```



1. **(0.5pt)** Is each subset balanced?

answer

No they are not. The entire dataset is balanced betweem the three classes, however, the trainning has an over representation of class 0 while the test set has an under representation of class 0. The oppposite effect is observed with class 2.

1. **(0.5pt)** How can the imbalance of each subset affect prediction accuracy?

answer

The imbalance will most probably decrease the accuracy of the model. The model will be more inclined to predict class 0 since it is the most prominant class in the trainning set. However, when we will test for the generalization of the model, the performance will be bad since class 0 is under represented. If we do not use stratification in our split, then the performance of the model will vary significantly depending on how the data is split into training and testing datasets. This is not desirable since we want our model to be as stable as possible.

1. **0.5** If the training/validation/test sets are not balanced, adjust your code to recreate balanced training/validation/test sets.

Hint: You may use the `stratify` argument in the `train_test_split` function to help with the balance of the splits.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, rand
```

SVM for classfication

1. **(1pt)** Train a linear SVM on the training set for each one of these C hyperparameter values: {0:001; 0:01; 0:1; 1; 10}. Find the best hyperparameter on the validation set.

In [ ]:
```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

C_values = [0.001, 0.01, 0.1, 1, 10]
best_C = None
best_accuracy = -1

for C in C_values:
    svm = LinearSVC(C=C,max_iter=10000)
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    print(f"Accuracy for C={C}: {accuracy}")
    f1 = f1_score(y_val, y_pred, average='weighted')
    print(f"F1 score for C={C}: {f1:.2f}\n")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_C = C

print(f"Best hyperparameter: C={best_C}, based on accuracy score")
```

```
Accuracy for C=0.001: 0.5
F1 score for C=0.001: 0.43

Accuracy for C=0.01: 0.65
F1 score for C=0.01: 0.63

Accuracy for C=0.1: 0.8
F1 score for C=0.1: 0.80

Accuracy for C=1: 0.9
F1 score for C=1: 0.90

Accuracy for C=10: 0.9
F1 score for C=10: 0.90

Best hyperparameter: C=1, based on accuracy score
```

1. **(1pt)** Using the best hyperparameter C, evaluate the accuracy, precision, recall, and F1-score on the test set.

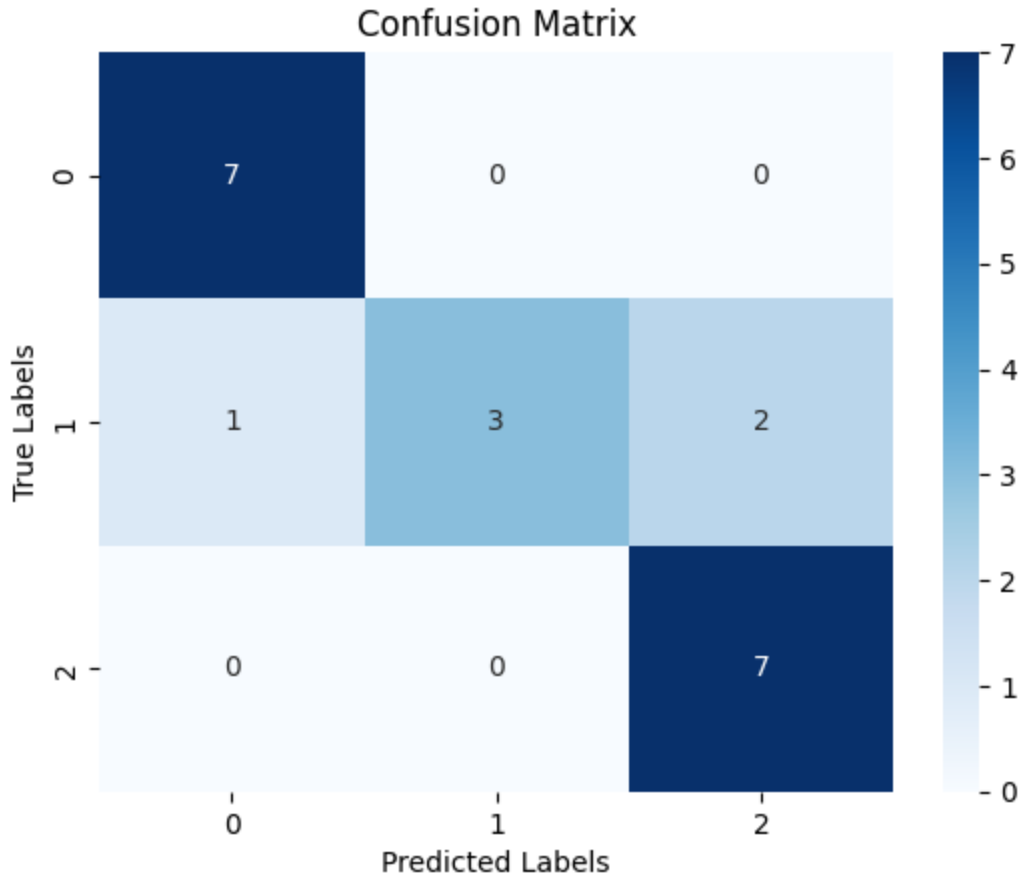
In [ ]:
```
svm = LinearSVC(C=1)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average = 'weighted')
recall = recall_score(y_test, y_pred, average = 'weighted')
f1 = f1_score(y_test, y_pred, average = 'weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

```
Accuracy: 0.85
Precision: 0.88
```

```
Recall: 0.85
F1-score: 0.83
```

1. **(0.5pt)** Plot the confusion matrix on the test set and explain the reason for your false negatives/positives.

In [ ]:
```python
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap="Blues")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```



Observations

The only class with missclassification is class 1. This is not very surprising since class 0 and 2 are very distant from one another in a euclidean way as seen previously in the first plot. This means they are easy to seperate. However, class 1 was harder to seperate from the other class in a linear fashion.

Regression (13pt)

You will now train k-NN and neural network models for the task of predicting the rating of a text review.

The data to download are **here**.

Each datum is a review in text format of an Amazon product. In the data file, each line corresponds to a datum. Each datum contains a target (y) followed by a short text (x). The target variable is the rating given

by a user to a product. It is an integer value between 1 and 5. The text is the review.

To pre-process the data you will first have to separate the targets from the features. *Hint: you can use the* `split('\t')` *function. There are also functions in* `pandas` *that will allow you to easily load this dataset.*

We will model this task as a regression problem (you can use mean squared error to measure performance).

1. **(0.5pt)** If we had decided to model this task as a classification problem with 5 classes, what are some of the metrics we could have used to measure performance?

answer

Since this is an ordinal regression, evaluating this model with a classification metric may not be the best idea. Some standard classification metrics that could be used are accuracy or F1 score.

1. List one **(0.5pt)** advantage and one **(0.5pt)** disadvantage of instead modelling the problem as a classification problem with 5 classes.

answer

Advantage:

Classification won't extrapolate out of the rating scale, it will not give 7.5 as an answer for a 1 to 5 integer scale.

Disadvantage:

By modeling as a classification with accuracy as a performance metric, you do not penalise more for large error. It is a hit or miss. While in regression it is a spectrum. Predicting 1 for a 5 yields a higher loss than 1 for a 2. One could argue that client satisfaction is more of a continous variable and would be better evaluated with a regression.

In [ ]:
```
# download and load the dataset
!wget -P /content https://raw.githubusercontent.com/davoodwadi/MATH60629A.A2023-MACHINE-

import pandas as pd
df = pd.read_csv('reviews.tsv', sep='\t', names=['y', 'X'])
```

```
--2023-10-21 15:33:25--  https://raw.githubusercontent.com/davoodwadi/MATH60629A.A2023-M
ACHINE-LEARNING-I/main/data/reviews.tsv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.
199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:44
3... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5569484 (5.3M) [text/plain]
Saving to: '/content/reviews.tsv'

reviews.tsv         100%[===================>]   5.31M  --.-KB/s    in 0.06s

2023-10-21 15:33:26 (95.4 MB/s) - '/content/reviews.tsv' saved [5569484/5569484]
```

1. **(0.5pt)** Divide the datasets into training (80% of the data), validation (10%), and test sets (10%). For this set the random seed to 1234 (`random state=1234`)

In [ ]:
```
train, test = train_test_split(df, test_size=0.2, random_state=1234, stratify=df.y)
val, test = train_test_split(test, test_size=0.5, random_state=1234, stratify=test.y)
```

1. **(2pt)** Now you must obtain a bag-of-words representation of the features. sklearn provides several functions for doing so.

To limit the required training time, please use a maximum of 2,000 words in your vocabulary (`max features=2000`) and the list of english stop words from sklearn (`stop words="english"`). Words on this list will be automatically removed from the data since they are, a priori, less predictive for the task at hand. Use the default value for all other function parameters.

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer

#Replace nan in X
print(train.X.isna().sum(), 'nan values, vectorizer does not seem to like them much')
train.X.fillna(' ', inplace=True)
val.X.fillna(' ', inplace=True)
test.X.fillna(' ', inplace=True)

#Vectorize
vectorizer = CountVectorizer(max_features=2000, stop_words="english")
vectorizer.fit(train.X)
X_train = vectorizer.transform(train.X)
X_val = vectorizer.transform(val.X)
X_test = vectorizer.transform(test.X)

#For convention
y_train = train.y
y_val = val.y
y_test = test.y
```

```
6 nan values, vectorizer does not seem to like them much
```

k-nearest neighbours

1. **(0.5pt)** Which of the following three distance functions ‘cosine’, ‘euclidean’, and ‘manhattan’ do you deem more appropriate for this problem? Please justify.

answer I believe cosine distance is the most appropriate since we do not really care about the magnitude of the vectors (length of review) but more about its direction (degree of satisfaction)

1. **(1pt)** Train an appropriate k-NN model for this task. We ask that you train models with 1, 10, 50, 100, and 1000 neighbours.

In [ ]:
```
from sklearn.neighbors import KNeighborsRegressor

k_values = [1, 10, 50, 100, 1000]
models = []

# Loop over the k values and create a model for each value
for k in k_values:
    model = KNeighborsRegressor(n_neighbors=k, metric='cosine')
    model.fit(X_train, y_train)
    models.append(model)
```

1. **(0.5pt)** What is the performance of each model on the training and validation sets?

In [ ]:
```
from sklearn.metrics import mean_squared_error

# Predict the labels of your train/val data for each model
y_preds_train = [model.predict(X_train) for model in models]
y_preds_val = [model.predict(X_val) for model in models]

# Calculate the MSE of each model
mses_train = []
for y_pred in y_preds_train:
    mse = mean_squared_error(y_train, y_pred)
    mses_train.append(mse)

mses_val = []
for y_pred in y_preds_val:
    mse = mean_squared_error(y_val, y_pred)
    mses_val.append(mse)

# Print the MSEs
print('Performance on training set: ')
for k, mse in zip(k_values, mses_train):
    print(f'MSE with {k} neighbors: {mse:.3f}')

print('\nPerformance on validation set: ')
for k, mse in zip(k_values, mses_val):
    print(f'MSE with {k} neighbors: {mse:.3f}')
```

```
Performance on training set:
MSE with 1 neighbors: 0.001
MSE with 10 neighbors: 0.706
MSE with 50 neighbors: 0.811
MSE with 100 neighbors: 0.835
MSE with 1000 neighbors: 0.893

Performance on validation set:
MSE with 1 neighbors: 1.472
MSE with 10 neighbors: 0.870
MSE with 50 neighbors: 0.850
MSE with 100 neighbors: 0.860
MSE with 1000 neighbors: 0.904
```

Observations

1. **(0.5pt)** What value of the hyperparameter provides the best results?

answer

k = 50 provides the best performance based on the validation set.

In [ ]:
```
# Measuring the performance of the model on the test set for future reference
knn = KNeighborsRegressor(n_neighbors=50, metric='cosine')
knn.fit(X_train, y_train)
mse_knn = mean_squared_error(y_test, knn.predict(X_test))
mse_knn
```

Out[ ]:
```
0.8495211999999999
```

Neural Networks

Upon instantiating your neural networks, fix the random seed to 1234 (that is `random state=1234`).

1. **(2pt)** You will now train a series of neural networks using different hyperparameters. Use the option

`early_stopping=True` and find the hyperparameters that obtain the best results on the validation dataset (to give you an idea, I imagine that you will train around 50 different models). I suggest that you explore the following three hyperarameters: *learning_rate*, *size of the network*, and *the strength of the L2 regularization term*.

In [ ]:
```
from sklearn.neural_network import MLPRegressor
import itertools

results = {'mse' : [],
           'learning_rate' : [],
           'alpha' : [],
           'hidden_layer_sizes' : []}

#hyperparams to validate
learning_rate = ['constant', 'adaptive']
alpha = [(10**-i) for i in range(-2,2)]
hidden_layer_sizes = [(h,)*w for h in range(4,16,4) for w in [1,2,4,8]]

for l, a, s in itertools.product(learning_rate, alpha, hidden_layer_sizes):

    mlp = MLPRegressor(early_stopping = True,
                       random_state = 1234 ,
                       solver = 'sgd',
                       learning_rate = l,
                       alpha = a,
                       hidden_layer_sizes = s)

    mlp.fit(X_train, y_train)
    mse = mean_squared_error(y_val, mlp.predict(X_val))
    results['mse'].append(mse)
    results['alpha'].append(a)
    results['learning_rate'].append(l)
    results['hidden_layer_sizes'].append(s)
    df = pd.DataFrame(results)

df_sorted = df.sort_values(by="mse", ascending=True)
df_sorted.head(10)
```

```
0.9363252704425735 constant 100 (4,)
0.9412784610098193 constant 100 (4, 4)
0.9747640000908341 constant 100 (4, 4, 4, 4)
0.974773646022279 constant 100 (4, 4, 4, 4, 4, 4, 4, 4)
0.9171841038505749 constant 100 (8,)
0.9440164480618698 constant 100 (8, 8)
0.9751603537073529 constant 100 (8, 8, 8, 8)
0.9754920397813527 constant 100 (8, 8, 8, 8, 8, 8, 8, 8)
0.9024164066513919 constant 100 (12,)
0.9532594237594496 constant 100 (12, 12)
0.9752578238851406 constant 100 (12, 12, 12, 12)
0.9793410422054107 constant 100 (12, 12, 12, 12, 12, 12, 12, 12)
0.7725904520473399 constant 10 (4,)
0.7481983533439661 constant 10 (4, 4)
0.9745185134933669 constant 10 (4, 4, 4, 4)
0.9747736460049331 constant 10 (4, 4, 4, 4, 4, 4, 4, 4)
0.7758489890285463 constant 10 (8,)
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
0.7530849711393574 constant 10 (8, 8)
0.6981292397036188 constant 10 (8, 8, 8, 8)
0.9703279494862577 constant 10 (8, 8, 8, 8, 8, 8, 8, 8)
0.761931530972199 constant 10 (12,)
0.736173319709316 constant 10 (12, 12)
```

```
0.7052152609860535 constant 10 (12, 12, 12, 12)
0.9697382283234852 constant 10 (12, 12, 12, 12, 12, 12, 12, 12)
0.7866066835098195 constant 1 (4,)
0.7769879157617086 constant 1 (4, 4)
0.9734837058570719 constant 1 (4, 4, 4, 4)
0.9747736460031553 constant 1 (4, 4, 4, 4, 4, 4, 4, 4)
0.7819881884830208 constant 1 (8,)
0.7687273121940293 constant 1 (8, 8)
0.7122606692139137 constant 1 (8, 8, 8, 8)
0.7231666917436805 constant 1 (8, 8, 8, 8, 8, 8, 8, 8)
0.7778811484963396 constant 1 (12,)
0.7831038519252563 constant 1 (12, 12)
0.7392940070567674 constant 1 (12, 12, 12, 12)
0.7425461068318764 constant 1 (12, 12, 12, 12, 12, 12, 12, 12)
0.7906497165316945 constant 0.1 (4,)
0.788045210103453 constant 0.1 (4, 4)
0.9728713800546668 constant 0.1 (4, 4, 4, 4)
0.9747736460029769 constant 0.1 (4, 4, 4, 4, 4, 4, 4, 4)
0.7881955930257397 constant 0.1 (8,)
0.7739548540729217 constant 0.1 (8, 8)
0.7215833903499672 constant 0.1 (8, 8, 8, 8)
0.7282974135151167 constant 0.1 (8, 8, 8, 8, 8, 8, 8, 8)
0.7888768904451026 constant 0.1 (12,)
0.7887391844519199 constant 0.1 (12, 12)
0.7560606247908829 constant 0.1 (12, 12, 12, 12)
0.7600477797469757 constant 0.1 (12, 12, 12, 12, 12, 12, 12, 12)
0.9363252704425735 adaptive 100 (4,)
0.9412784610098193 adaptive 100 (4, 4)
0.9747640000908341 adaptive 100 (4, 4, 4, 4)
0.974773646022279 adaptive 100 (4, 4, 4, 4, 4, 4, 4, 4)
0.9171841038505749 adaptive 100 (8,)
0.9440164480618698 adaptive 100 (8, 8)
0.9751603537073529 adaptive 100 (8, 8, 8, 8)
0.9754920397813527 adaptive 100 (8, 8, 8, 8, 8, 8, 8, 8)
0.9024164066513919 adaptive 100 (12,)
0.9532594237594496 adaptive 100 (12, 12)
0.9752047796294839 adaptive 100 (12, 12, 12, 12)
0.9774420870408574 adaptive 100 (12, 12, 12, 12, 12, 12, 12, 12)
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
0.7725904520473399 adaptive 10 (4,)
0.7481983533439661 adaptive 10 (4, 4)
0.9745185134933669 adaptive 10 (4, 4, 4, 4)
0.9747736460049331 adaptive 10 (4, 4, 4, 4, 4, 4, 4, 4)
0.7713698129540271 adaptive 10 (8,)
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
0.7530849711393574 adaptive 10 (8, 8)
0.6981292397036188 adaptive 10 (8, 8, 8, 8)
0.9703279494862577 adaptive 10 (8, 8, 8, 8, 8, 8, 8, 8)
0.7591374770879843 adaptive 10 (12,)
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
0.736173319709316 adaptive 10 (12, 12)
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.p
y:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and th
e optimization hasn't converged yet.
  warnings.warn(
```

```
0.7037041335449019 adaptive 10 (12, 12, 12, 12)
0.9697382283234852 adaptive 10 (12, 12, 12, 12, 12, 12, 12, 12)
0.7866066835098195 adaptive 1 (4,)
0.7769879157617086 adaptive 1 (4, 4)
0.9734427069312876 adaptive 1 (4, 4, 4, 4)
0.9747736460031553 adaptive 1 (4, 4, 4, 4, 4, 4, 4, 4)
0.7819881884830208 adaptive 1 (8,)
0.7687273121940293 adaptive 1 (8, 8)
0.7122606692139137 adaptive 1 (8, 8, 8, 8)
0.7231666917436805 adaptive 1 (8, 8, 8, 8, 8, 8, 8, 8)
0.7783618998282026 adaptive 1 (12,)
0.7831038519252563 adaptive 1 (12, 12)
0.7392940070567674 adaptive 1 (12, 12, 12, 12)
0.7425461068318764 adaptive 1 (12, 12, 12, 12, 12, 12, 12, 12)
0.7906497165316945 adaptive 0.1 (4,)
0.788045210103453 adaptive 0.1 (4, 4)
0.9727608698897742 adaptive 0.1 (4, 4, 4, 4)
0.9747736460029769 adaptive 0.1 (4, 4, 4, 4, 4, 4, 4, 4)
0.7881955930257397 adaptive 0.1 (8,)
0.7739548540729217 adaptive 0.1 (8, 8)
0.7215833903499672 adaptive 0.1 (8, 8, 8, 8)
0.7282974135151167 adaptive 0.1 (8, 8, 8, 8, 8, 8, 8, 8)
0.7888768904451026 adaptive 0.1 (12,)
0.7887391844519199 adaptive 0.1 (12, 12)
0.7560606247908829 adaptive 0.1 (12, 12, 12, 12)
0.7600477797469757 adaptive 0.1 (12, 12, 12, 12, 12, 12, 12, 12)
```

Note: I did leave the Stochastic Optimizer: Maximum iterations (200) to its default value of 200 due to the amount of time required to train this model.

In [ ]:
```
#Here are the results if you do not want to run the code above
d = {'mse': {18: 0.6981292397036188, 66: 0.6981292397036188, 70: 0.7037041335449019, 22:
res = pd.DataFrame(d)
res.head()
```

Out[ ]:

	mse	learning_rate	alpha	hidden_layer_sizes
18	0.698129	constant	10.0	(8, 8, 8, 8)
66	0.698129	adaptive	10.0	(8, 8, 8, 8)
70	0.703704	adaptive	10.0	(12, 12, 12, 12)
22	0.705215	constant	10.0	(12, 12, 12, 12)
78	0.712261	adaptive	1.0	(8, 8, 8, 8)

1. **(0.5pt)** Find the MSE on the test set for the best hyperparameter.

In [ ]:
```
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(early_stopping = True,
                   random_state = 1234 ,
                   solver = 'sgd',
                   learning_rate = 'adaptive',
                   alpha = 10,
                   hidden_layer_sizes = (8,8,8,8))

mlp.fit(X_train, y_train)
mse_nn = mean_squared_error(y_test, mlp.predict(X_test))
print(mse_nn)
```

```
0.6922215355519573
```

1. (0.5pt) What did you find out about the importance of the various hyperparameters?

answer

I found out that larger models are not always better even if you increase regularization to prevent overfitting. Also, I observed that the constant learning rate and adaptive often found the same minimum when they had the same model. I had to remove the invscaling learning rate because it took too long to converge. Fine tuning these parameters takes time and can impact performance alot.

Comparison

1. **(0.5pt)** Based on the neural network that you trained, if you were to keep a single feature (that is a single word), which one would it be and why?

In [ ]:
```
!pip install eli5
```

```
Collecting eli5
  Downloading eli5-0.13.0.tar.gz (216 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 216.2/216.2 kB 4.
5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: attrs>17.1.0 in /usr/local/lib/python3.10/dist-packages
 (from eli5) (23.1.0)
Requirement already satisfied: jinja2>=3.0.0 in /usr/local/lib/python3.10/dist-packages
 (from eli5) (3.1.2)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.10/dist-packages
 (from eli5) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from el
i5) (1.11.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from eli
5) (1.16.0)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.10/dist-pack
ages (from eli5) (1.2.2)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from
 eli5) (0.20.1)
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.10/dist-package
s (from eli5) (0.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-package
s (from jinja2>=3.0.0->eli5) (2.1.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages
 (from scikit-learn>=0.20->eli5) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pa
ckages (from scikit-learn>=0.20->eli5) (3.2.0)
Building wheels for collected packages: eli5
  Building wheel for eli5 (setup.py) ... done
  Created wheel for eli5: filename=eli5-0.13.0-py2.py3-none-any.whl size=107719 sha256=6
a33f5e02fc5ca95d45f8572d944ef7021c847798dcc7729328a01e754eed737
  Stored in directory: /root/.cache/pip/wheels/b8/58/ef/2cf4c306898c2338d51540e0922c8e0d
6028e07007085c0004
Successfully built eli5
Installing collected packages: eli5
Successfully installed eli5-0.13.0
```

In [ ]:
```
import eli5
from eli5.sklearn import PermutationImportance
perm = PermutationImportance(mlp).fit(X_test.toarray(), y_test)
eli5.show_weights(perm)
```

Out[ ]:

Weight	Feature
0.0299 ± 0.0151	x763
0.0244 ± 0.0119	x1025

0.0227 ± 0.0163	x1029
0.0192 ± 0.0068	x479
0.0162 ± 0.0112	x703
0.0103 ± 0.0081	x822
0.0082 ± 0.0076	x790
0.0079 ± 0.0035	x1770
0.0076 ± 0.0028	x168
0.0074 ± 0.0086	x1243
0.0070 ± 0.0033	x1066
0.0067 ± 0.0043	x533
0.0067 ± 0.0073	x1965
0.0062 ± 0.0042	x470
0.0061 ± 0.0029	x915
0.0060 ± 0.0033	x989
0.0056 ± 0.0023	x523
0.0052 ± 0.0058	x497
0.0052 ± 0.0038	x1583
0.0051 ± 0.0018	x209
... 1980 more ...	

In [ ]:
```
feature_names = vectorizer.get_feature_names_out()
feature_names[763]
```

Out[ ]:
```
'great'
```

answer

Features permutations allows us to identify the most important feature in our model. This is done by replacing a feature data points by random noise in the test set and measuring the impact on the score of the model. This saves us the trouble of refitting the model each time. The most important feature found using this method is the word 'great'. This makes sense intuitively since it is a common word that is easily associated with the sentiment of the customer.

1. **(0.5pt)** What is the final performance of each model (k-NN, and neural network)?

In [ ]:
```
print(f'The performance of K-NN is :{mse_knn}')
print(f'The performance of Neural Network is :{mse_nn}')
```

```
The performance of K-NN is :0.8495211999999999
The performance of Neural Network is :0.6922215355519573
```

1. **(1pt)** Find all the examples for which the prediction of the k-NN and the neural network differ by more than 2.0.

In [ ]:
```
print('There is',sum(abs(mlp.predict(X_test)-knn.predict(X_test))>2), 'example where pre
test[abs(mlp.predict(X_test)-knn.predict(X_test))>2].loc[1125,'X']
```

```
There is 1 example where predictions differ by more than 2:
```

Out[ ]:
```
'I initially enjoyed playing this game due to the co-op nature and how intriguing/random
it is but as I\'ve played it more and more, I\'ve begun to dislike it more and more.  Th
ere is so much happening with the game but the payoff is rarely worth it.  I\'ll play it
if my group is playing it but otherwise, I want nothing to do with it.Basic Play - You s
elect a character, your choice or at random, and get all equipment/money they are entitl
ed to.  Select a random Ancient One to battle.  You spend the rest of the game navigatin
g the board looking for clues, buying items, getting skills/spells/allies, fighting enem
ies and going into portals to close/seal them.  Your character card has stats you can ch
ange at the beginning of every turn.  For example, you can increase your ability to move
around the board faster but this decreases your ability to sneak past enemies.  You can
increase your luck but this decreases your lore.  Each character has their own individua
```

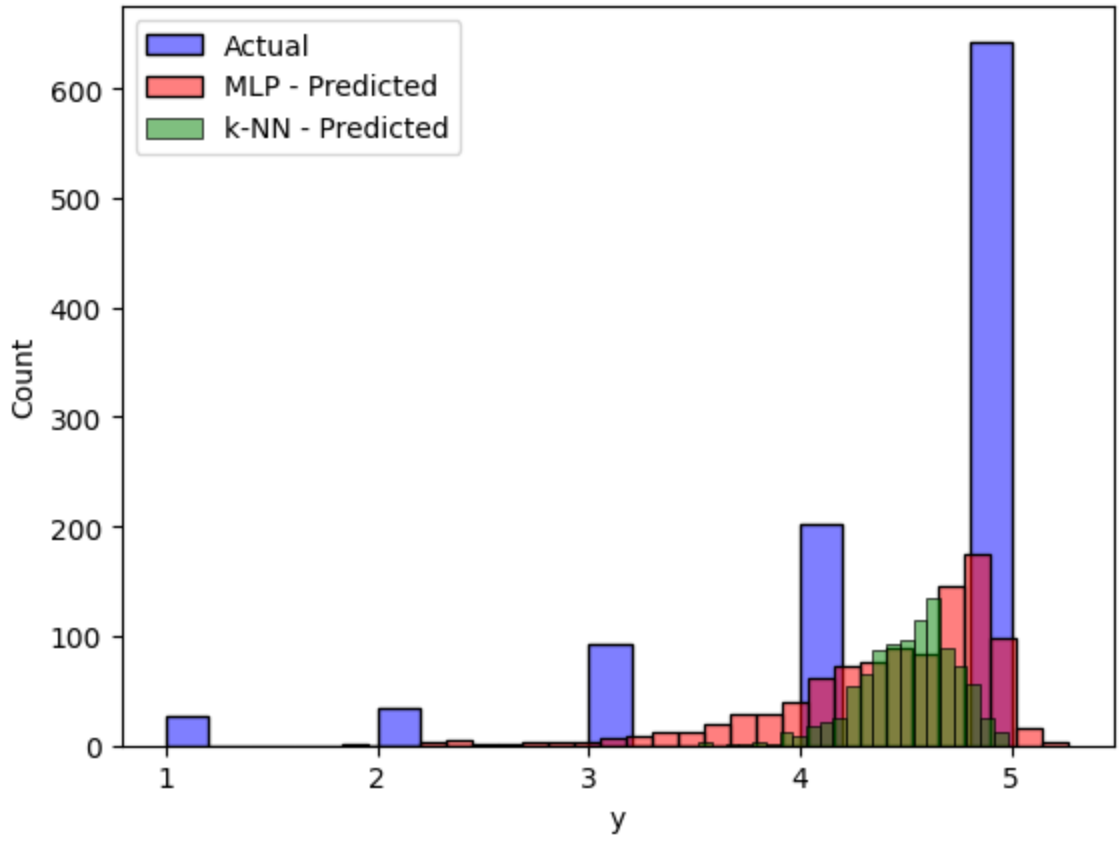
l max/minimums for each stat, starting stuff, and special ability that makes them uniqu e. Almost everything you do in this game will be done with dice. Rolling a 5 or 6 is a success while a 1-4 is a fail. You might enter a room and have to make a luck check, so you\'ll see what your character\'s luck rating is and get that many dice to roll. One 5 or 6 and you\'re good but if you get none than you fail. There are cards and other thin gs that can modify your abilities. Enemies require that you do all damage at once to ki ll them. So if an enemy has 3 life and you get 2 hits, he takes no damage and hits you and next round you will have to get 3 hits still to kill it. You can\'t team up in comb at with another player, even though this is a co-op game, to kill an enemy. All battles are one on one, except the final battle, if it happens. If you die at any point in the game, you simply re-spawn and lose half your items. It is possible to be devoured, in w hich case your character is permanently dead and you select a new character and come bac k onto the board with all their starting stuff (it is often better to be devoured then k illed). There are a ton of locations to visit in the game, all of them have a set of 7 cards that you pick one from at random to see what happens to you. Some locations have special text on them where you can choose to do what the text says rather than draw a ra ndom card, such as the item shops or medical areas. All players will go and then at the end of every round, a portal gate will open, if possible, and bring out monsters, if pos sible, and make monsters move, if possible. You win the game if you close all gates (ce rtain restrictions apply), seal 6 or more gates (by using 5 clue tokens or a special ite m), or kill the ancient one if he awakens. You lose if the ancient one awakens (which c an happen multiple ways) and you fail to defeat him.Components - This game just has way too many pieces. The board is huge, you will need a large table to play this on, especi ally if you get any of the expansions that make the board even larger. The board itself is nice to look at and well made. There are 16 large character cards and 8 ancient one cards to select from each game. They look nice and have unique stories for each of the characters on the back. There are 196 token pieces, 189 different cards your characters can acquire (split into 11 piles), 179 ancient one cards (split into 3 piles, the locati on pile will be split into 9 different piles), 60 monster tokens, 16 gate tokens and at least 30 more random tokens/markers/dice. On the plus side, everything looks nice and i t\'s unlikely anything will get worn out or damaged. On the negative side, you need a t on of room to place all this stuff, and then you need more room for each person playing the game. Also, the cards your characters can acquire are the smaller, annoying, hard t o shuffle style of cards (such as Ticket to Rides train car cards). The rulebook is lar ge and it is 24 pages. It is somewhat rough to get through and you\'ll have to check ba ck multiple times during your first couple playthroughs, probably even in some later pla ythroughs. There are no gameplay sheets to hand out to players, which you\'ll want and need, so make sure you go online and find a friendly gameplay sheet to print out for pla yers, you can find them at BoardGameGeek. The insert for the game is ok. I don\'t thin k they provided any baggies so you\'ll want to get some to put all the different pieces in.Length - This game takes a long time, especially your first couple playthroughs. Exp ect setup to take you about 15-30 minutes since you have to select characters, enemies, distribute starting items/cash/skills/life/sanity/etc, then you have to shuffle each dec k of cards, play a starting mythos card and place the gate and enemies. The actual game itself will run at least 4 hours for your initial play-through. It still regularly take s my group 2 1/2 - 3 hours to play a game. I\'m the guy that actually keeps the game mo ving because I want to get through it. They tried playing it once without me and it too k them over 5 hours because they couldn\'t stay focused.Re-playability - This is one of the big positives of the game. There is a lot of re-playability due to all the characte rs, ancients ones (that have various effects on monsters and gameplay), and just the ran dom nature of the game. Game win/loss conditions change depending on the number of play ers, along with how many monster spawns and other various rules.Luck - Almost everything you do in this game requires you to roll dice. There is a ton of luck. I\'ve watched 1 1 dice come up with no successes and 3 dice get 3 successes. Every location card you dr aw has something random happen at the location that could be good or bad. You might hav e your lore skill maxed out (which leaves you with little to no luck) and draw a helpfu l/hurtful card that requires lore to pass. Of course, you might draw a card that requir es luck to pass and you don\'t get to roll at all if you have no luck. Also, most cards take a die or two away from whatever skill check you are making. Monsters only move if the mythos card drawn at the end of each turn say so. So if you go into a room and a mo nster blocks the exit, you have to either fight or sneak past. Some monsters are extrem ely difficult to kill and some are difficult to sneak past, some have both. So if one o f those tough to sneak/fight monsters is blocking your room, enjoy being stuck in there for the next couple turns. There are also random cards that just cause you to die/get d evoured, so enjoy that as well.Interaction - There is surprisingly little interaction ne eded for this game. You might give someone an item once a game or use a special power t

o help someone but for the most part, everyone just kind of does there own thing. You c an\'t team up to fight monsters or close/seal gates. Most of the interaction will just be game banter.Thoughts pros - Hard to think of a lot at this point. There is a ton of theme. Can play 1-8 players. Components are all good quality and look nice. A lot of things you can do, characters and bad guys to pick. The co-operative aspect is nice, I guess.Thoughts cons - Can someone show me where the fun is? It takes a while to slog th orough and is just way to random. It\'s rough to teach to new players, you\'re better o ff just playing and hoping they pick up as they go along. The rules are annoying to rem ember. There are many rules that change depending on the number of people playing, so y ou have to look those up each game. Even after many playthroughs, using cheat sheets, I \'m pretty sure we still have a rule or two wrong. The co-op aspect isn\'t really used that often. Occasionally someone will give an item away but most of the time it\'s just spent deciding which gate someone is going to go into. Everyone pretty much just does t heir own thing; and I\'ve sat in one location for the majority of the game doing nothing and we still win most of the time. I\'ve also found that there is very little strategy required. The game gives you all of these options but I\'ve found it breaks down to 4 d ifferent things. 1. If you have 5 clues or the item that seals a gate, go into a gate. 2. If you can\'t close a gate and have money, go to the special item store and try to buy the item that automatically seals a gate (I think there are 3 or 4 of them in the sp ecial item deck). 3. If you have no money, run around and pick up clues. 4. If you ha ve no money and there are no clues, just sit in a room where you have a good shot at ear ning clues/money. For example, of the 7 cards for the newspaper location, some give you money with no rolling, some give a luck (-1) check for a clue and one gives you a lore (-1) check for 3 clues. So you just sit in that room and earn money or clues and then u se whatever you earned. Using this strategy, I\'ve only lost 2 of about 10+ we\'ve play ed. There have been a couple last turn thrillers where we won, but other than that, not much fun here. There are some locations that have almost no purpose. It is extremely r are that someone goes to the common item shop and buys a common item, or goes to the mag ic shop to buy magic, or goes to the skill area to buy a skill. You\'re better off just trying to draw the special item that lets you seal a gate. Some location cards won\'t b e drawn because there is no purpose to going there, I\'ve never seen anyone draw a locat ion card for the hospital, magic shop, common item shop, church, boarding house or curio sitie shoppe. There is no reason to visit these spots outside of their special text opt ions. Certain mythos cards have text on them titled "environment" that cause effects th at alter the game (all players get +1 luck/-1 lore, or no spells can be cast, etc) and t hese can be annoying to remember sometimes. Combat also isn\'t that interesting. Most of the time, your character will fit one of two molds, decked out for combat or don\'t b other fighting anything that has more than one life. Also, enemies are very uneven, the re only seem to be 2 difficulties, ridiculously easy or ridiculously hard. There are ve ry few that fall in between. You\'ll probably have one player who just runs around kill ing enemies. Sometimes you\'ll let yourself get killed or get devoured because it\'s be neficial. I shouldn\'t get rewarded for dying.I wasn\'t sure whether I wanted to give t his a 3 or 2 stars, but what decided it for me was that I\'ll enjoy playing a 3 star gam e once and a while, but this game, I just want it to be over as soon as possible. I don \'t like Pandemic, but I will play that over this any time. This game just takes too lo ng to setup (I\'d rather setup Axis & Allies) and play for the little payoff it gives an d there are too many cards/pieces. I don\'t mind longer games but there needs to be ent ertainment or a payoff in the gameplay and there is neither in this game. The game isn \'t hard, it\'s just random. Everything is too random and you\'re at the mercy of the d ice. There is next to no strategy or thinking required, you just hope something random doesn\'t happen to screw you. There are too many fiddly rules and too many pieces to fi ddle with, even some monsters have to be removed from the group of monsters because they \'re only used if a specific ancient one is in play. You\'ve got to remember what speci al power the ancient one gives enemies plus environments and events. People that love t his game seem to love it for the theme. I guess that\'s ok, but for me, theme, presenta tion etc... mean very little if the core gameplay isn\'t good. I\'d much rather play ge neric themed game with ugly looking components that has interesting/good gameplay than t his game. There is just nowhere near enough good things about this game to make up for the many negatives of the game. I think I\'d rather play Risk than this game. At least I can spread myself thin in Risk and get killed to end the suffering, this just goes on and on. I\'m tired of playing this game, I\'m either not going to play it from now on o r just play my 3DS or read a book while playing, at least then I\'ll feel like I\'m doin g something entertaining. Overall, I have to give this game a 3/10.'

1. **(0.5pt)** Plot the distribution of the actual labels, and the predicted labels from k-NN and neural

network.

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# actual labels
sns.histplot(y_test, color='blue', alpha=0.5, label='Actual')

# predicted labels
sns.histplot(mlp.predict(X_test), color='red', alpha=0.5, label='MLP - Predicted')

# predicted labels
sns.histplot(knn.predict(X_test), color='green', alpha=0.5, label='k-NN - Predicted')

plt.legend()
plt.show()
```



1. **(0.5pt)** Explain the reason behind the difference in prediction from k-NN and the actual labels.

answer

First of all, we used regression in a context where the y-label were classes therefore we would expect to see the predictions follow a continuous distribution and the actual label a discrete distribution. Thus, the difference in distribution is expected.

Secondly, the dataset is higly skewed, there is a lot more of 4-5 ratings than 1-2. This will make our model less inclined to predict lower values. Hence why we do not have many prediction under 3 stars.

Thirdly, each feature (words) have equal weight in knn. This may be a reason that explains the differences between the actual label and the prediction of the model. Indeed some words might have a better predictive power than others and should have a higher weight.

1. **(0.5pt)** Explain the reason behind the difference in prediction from the neural network and the actual labels.

answer

The two first reason given above also apply here.

Furthermore outliers in the dataset can impact neural network in a negative way.

Finaly, there is always randomness that cannot be captured by our model.

In [ ]: