Classifying dried beans using machine learning classification methods

Section 1: Introduction

Dried beans are cheap, nutritious, and last for a long time. Which makes them a staple in a broke university student's diet. But there are many different types of beans that can be hard to tell apart. In this report, machine learning classification methods will be trained on a dataset of dried bean data. The goal is to train a supervised K nearest neighbor classifier that can determine which species a dried bean is given its physical characteristics.

The report is organized as follows: Section 1 is the introduction. Section 2 formulates the problem and presents the dataset. Section 3 discusses the machine learning methods that will be used. Section 4 discusses the results obtained with the chosen model and the final test results. Finally, section 5 reviews and concludes the report.

Section 2: Problem Formulation

The dataset used is the dried bean dataset [1]. The dataset contains 13611 instances of data from dried bean images, categorized into 7 different species of beans.

The features present in the dataset are the size, perimeter, axis length, and convex area of the bean pictures, measured in pixels in the original pictures. Along with some normalized and preprocessed features: aspect ratio, eccentricity, solidity, roundness, and compactness are measured as continuous ratios and thus do not have units.

All features that are measured in pixels depend on the camera used and the picture size. As such, it will not be used as data for the machine learning model; only some normalized distinctive features will be used.

The labels that would be used for classification are the seven different species of beans present in the dataset. The model should be able to identify a bean species given some information about its shape.

Section 3: Methods

Data:

As stated above, there are features in the dataset that are not applicable or irrelevant to the classification problem. And having too many features might lead to computational complexity. So the chosen features are:

- Aspect ratio: The ratio between the length of the long axis and short axis of the bean.
- Convexity (Solidity): The ratio of the convex area compared to the area of the entire bean.
- Roundness: A measure of how round the bean is, calculated as $\frac{4\pi * area}{perimeter^2}$
- Extent: The ratio of bean area compared to the bounding box of the bean.

These features are chosen because they best represent the distinct physical characteristics of the species of beans, are readily present in the dataset, and are easy to work with since they are continuous values.

As these features are ratios and have very different levels of variation, they need to be scaled and normalized before a machine learning model can be applied.

KNN model:

After visualizing the data, it is observed that there are clear correlations between the chosen features and the species of beans. These form clear, visible clusters and regions of data points that all belong to the same label. These features are therefore distinct enough for a simple KNN model to work effectively. Which is why it was chosen.

The loss function of KNN is the 1/0 loss. Which means if the label is correct, the score is 1, else it is zero. The loss function punishes misclassification.

SVC model:

Because our dataset has multiple features, it is reasonable to use a C-Vector Support Machine (SVC) model for classification of the dataset since the model is known for being effective in datasets with a high number of features [2].

Using the SVC model, a kernel function for classification must also be chosen. Since in the visualization process there is no clear linear clustering of datapoints with the same label, a polynomial function will be chosen for its additional flexibility.

For the loss function, the Hinge loss function was chosen because it is simple, fast, and punishes misclassification heavily on classification problems.

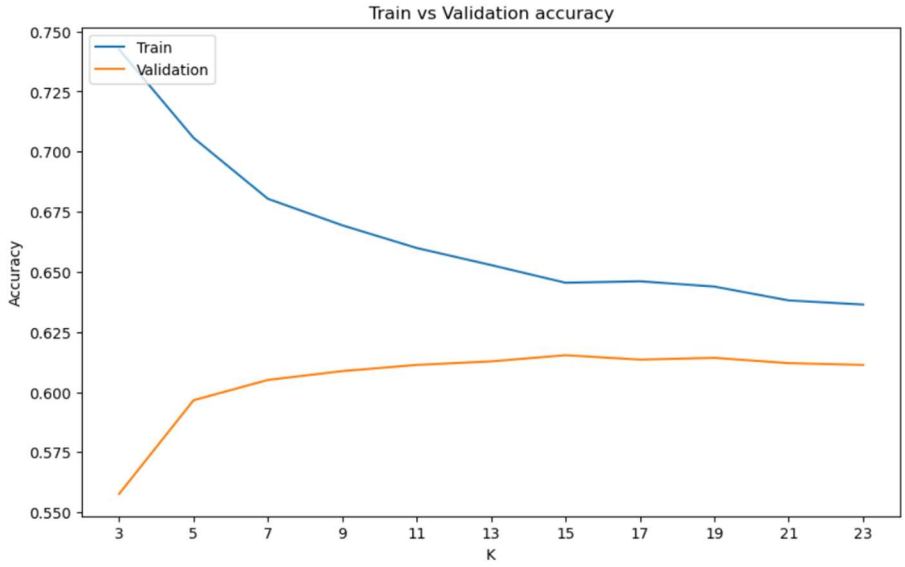
Splitting Data:

Since the number of the different species of beans are not uniform, the data must be split so that each species has approximately the same number of data points, or else the model will be biased against some beans species. The training set, validation set, and test set have been set to an 60%, 20%, and 20% split respectively. With over 13000 data points, this split should have enough datapoints to allow the model to train effectively, while leaving a large enough validation and testing set to see clear, unbiased results.

Section 4: Results

KNN model results:

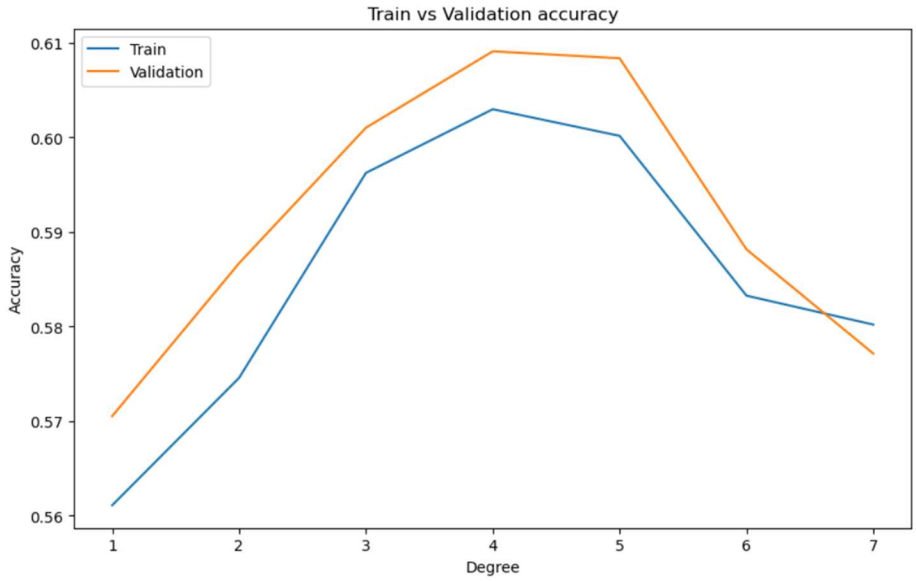
Training the model with variable k neighbors on the training set, then assessing the accuracy of the model on the training set itself and the predetermined test dataset results in the following graph.



It is seen in the graph that the model achieved the highest accuracy of 0.614 or 61.4% when using K = 15 nearest neighbors. The training accuracy decreases while validation accuracy increases when K increases, this is expected and shows that the model wasn't overfitted. It is interesting but not particularly surprising to find that a larger K yields better results, given that the dataset used has many intersecting labels.

SVC model results:

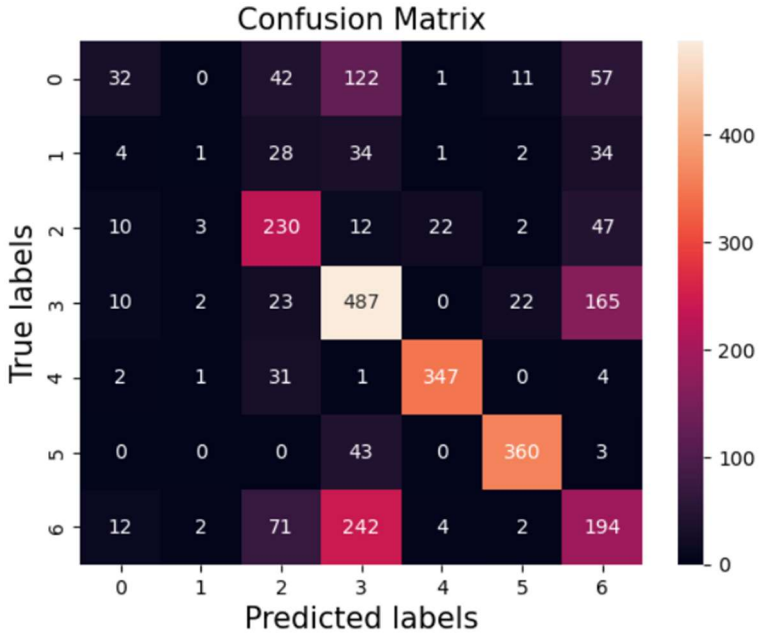
Training the model with variable polynomial degree of its kernel function on the training set, then assessing the accuracy of the model on the training set itself and the predetermined test dataset results in the following graph.



From the graph, it is clear that the model performs best at 0.609 or 60.9% when the polynomial kernel function has a degree of 4. What is more interesting is that the accuracy decreases for both training and validation dataset with a higher polynomial degree, which hints at the model trying to overfit with the dataset. Note that this method is only 0.5% less accurate than the best KNN models, which I find surprising since many labels in the dataset intersects.

Test results:

Using these two results, the final chosen method will be K-nearest neighbors with K = 15. Using this model, the final test accuracy is 60.63% with the following confusion matrix.



Section 5: Conclusion

As the resulting accuracy along with the final accuracy are all around 60% at best, there are a lot of room for improvement. Our original hypotheses, that beans species can be classified using shapes and physical characteristics of the dried beans might be flawed.

Looking at the final confusion matrix and throughout the training process, it is clear that the model does relatively well in classifying the beans in label 4 and 5, which correspond to the bean species Horoz beans and Seker beans, due to them having relatively distinct physical shapes. While it struggles in separating label 3 (Dermason beans) and 6 (Sira beans), having a large amount of both false positives and false negatives.

As the model is currently, it is not satisfactory to classify dried beans as the model misclassify around 40% of the time.

To be able to separate and classify the beans more effectively and efficiently, more features like colors are likely needed to differentiate between beans of similar shapes and size. Furthermore, because the feature engineering done in this report are relatively basic and simple, a more extensive and better feature engineering and processing will likely result in a better model with more accuracy.

References:

[1] https://archive.ics.uci.edu/dataset/602/dry+bean+dataset or https://www.kaggle.com/datasets/muratkokludataset/dry-bean-dataset

[2] Duan, K.-B., & Keerthi, S. S. (2005). Which Is the Best Multiclass SVM Method? An Empirical Study. Lecture Notes in Computer Science, 278–285. doi:10.1007/11494683_28

Beans

October 11, 2023

[18]:
```
import warnings

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns  #data visualization library
from imblearn.over_sampling import RandomOverSampler
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix  # evaluation␣
↪metrics

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, normalize
```

[38]:
```
df = pd.read_excel("Dry_Bean_Dataset.xlsx")
df.drop(columns=['Area','Perimeter','MajorAxisLength', 'MinorAxisLength',␣
↪"Eccentricity", 'ConvexArea', 'EquivDiameter', 'roundness',\
                 'ShapeFactor1', 'ShapeFactor2', 'ShapeFactor3',␣
↪'ShapeFactor4'],inplace=True)

le = LabelEncoder()
le.fit(df.Class)
df['Class'] = le.transform(df.Class)
le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
print(le_name_mapping)

df.head(5)
```

```
{'BARBUNYA': 0, 'BOMBAY': 1, 'CALI': 2, 'DERMASON': 3, 'HOROZ': 4, 'SEKER': 5,
'SIRA': 6}
```

[38]:

	AspectRation	Extent	Solidity	Compactness	Class
0	1.197191	0.763923	0.988856	0.913358	5
1	1.097356	0.783968	0.984986	0.953861	5
2	1.209713	0.778113	0.989559	0.908774	5

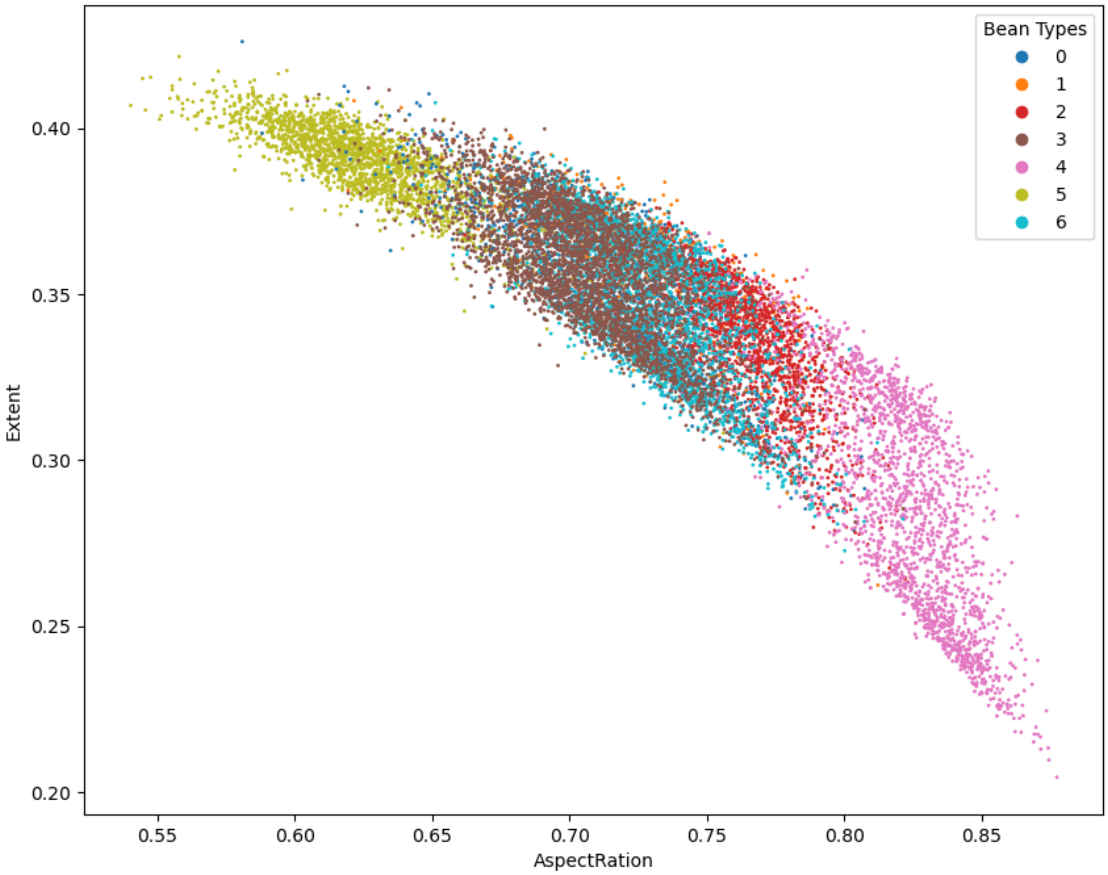
```
3      1.153638  0.782681  0.976696     0.928329      5
4      1.060798  0.773098  0.990893     0.970516      5
```

[20]:
```
y = df.Class.to_numpy()

df_data = df.drop(columns = ["Class"])
d = normalize(df_data)
#d = df_data
scaled_df = pd.DataFrame(d, columns=df_data.columns)

X = scaled_df.to_numpy()
scaled_df.head(10)
```

[20]:

	AspectRation	Extent	Solidity	Compactness
0	0.611824	0.390402	0.505354	0.466771
1	0.570578	0.407629	0.512150	0.495966
2	0.614617	0.395335	0.502764	0.461720
3	0.595007	0.403680	0.503746	0.478800
4	0.555495	0.404839	0.518889	0.508218
5	0.600311	0.397633	0.507242	0.473520
6	0.593302	0.394443	0.509133	0.482898
7	0.598889	0.395982	0.508344	0.475518
8	0.598987	0.396410	0.507835	0.475581
9	0.610431	0.398377	0.503667	0.463669

[21]:
```
plt.figure(figsize=(10, 8))
# convert the labels to numbers, each will be assigned a separate color based␣
 ↪on the cmap specified
sc = plt.scatter(X[:, 0], X[:, 1], s=1, c=y, cmap='tab10')
plt.xlabel("AspectRation")
plt.ylabel("Extent")
plt.legend(*sc.legend_elements(), title='Bean Types')
plt.show()
```



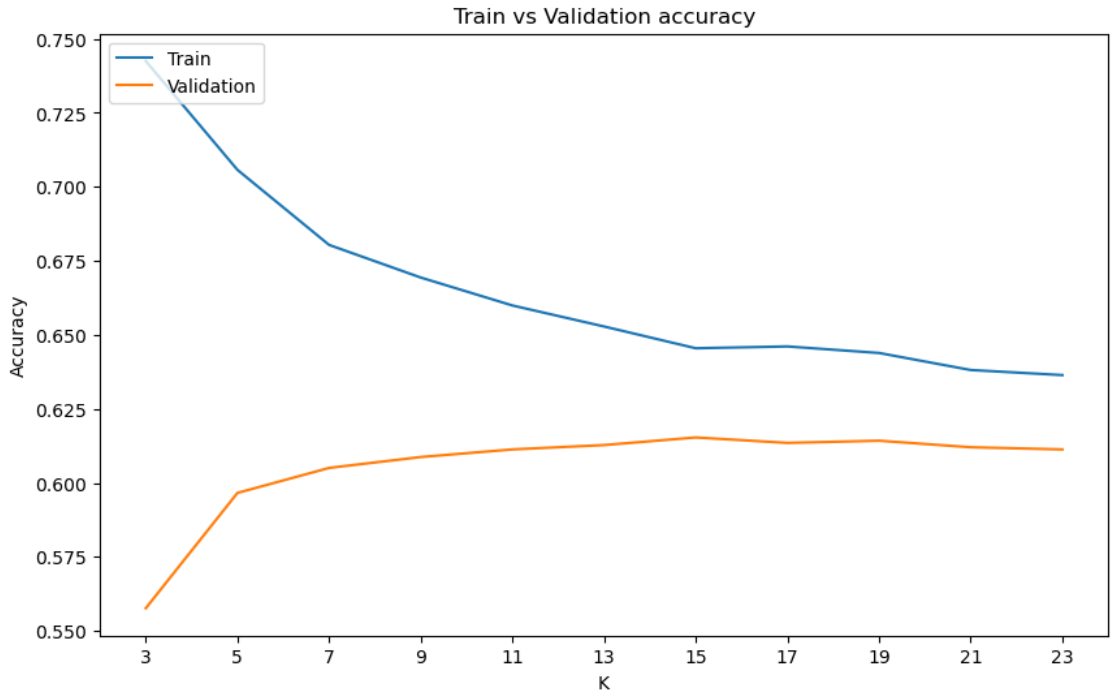
[24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
 ↪25, random_state=42, stratify=y_train)

print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
print(np.unique(y_train, return_counts=True))
print(np.unique(y_val, return_counts=True))
print(np.unique(y_test, return_counts=True))
```

```
(8166, 4)
(2722, 4)
(2723, 4)
(array([0, 1, 2, 3, 4, 5, 6]), array([ 793,  313,  978, 2128, 1156, 1216,
1582]))
(array([0, 1, 2, 3, 4, 5, 6]), array([264, 105, 326, 709, 386, 405, 527]))
(array([0, 1, 2, 3, 4, 5, 6]), array([265, 104, 326, 709, 386, 406, 527]))
```

[29]:
```
ks = range(3, 25, 2)

kneighbors_train_errors = []
kneighbors_val_errors = []

for i, k in enumerate(ks):

    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(X_train, y_train)

    y_pred_train = neigh.predict(X_train)
    train_error = accuracy_score(y_train, y_pred_train)
    y_pred_val = neigh.predict(X_val)
    val_error = accuracy_score(y_val, y_pred_val)

    kneighbors_train_errors.append(train_error)
    kneighbors_val_errors.append(val_error)

print(kneighbors_train_errors)
print(kneighbors_val_errors)
```

```
[0.7424687729610581, 0.7057310800881704, 0.6803820720058781, 0.6693607641440118,
0.659931422973304, 0.6528288023512123, 0.6454812637766348, 0.6460935586578497,
0.6438892970854764, 0.6381337252020574, 0.6364192995346559]
[0.5576781778104335, 0.5966201322556943, 0.6050698016164585, 0.6087435709037472,
0.6113152094048494, 0.6127847171197649, 0.615356355620867, 0.6135194709772226,
0.6142542248346804, 0.6120499632623071, 0.6113152094048494]
```

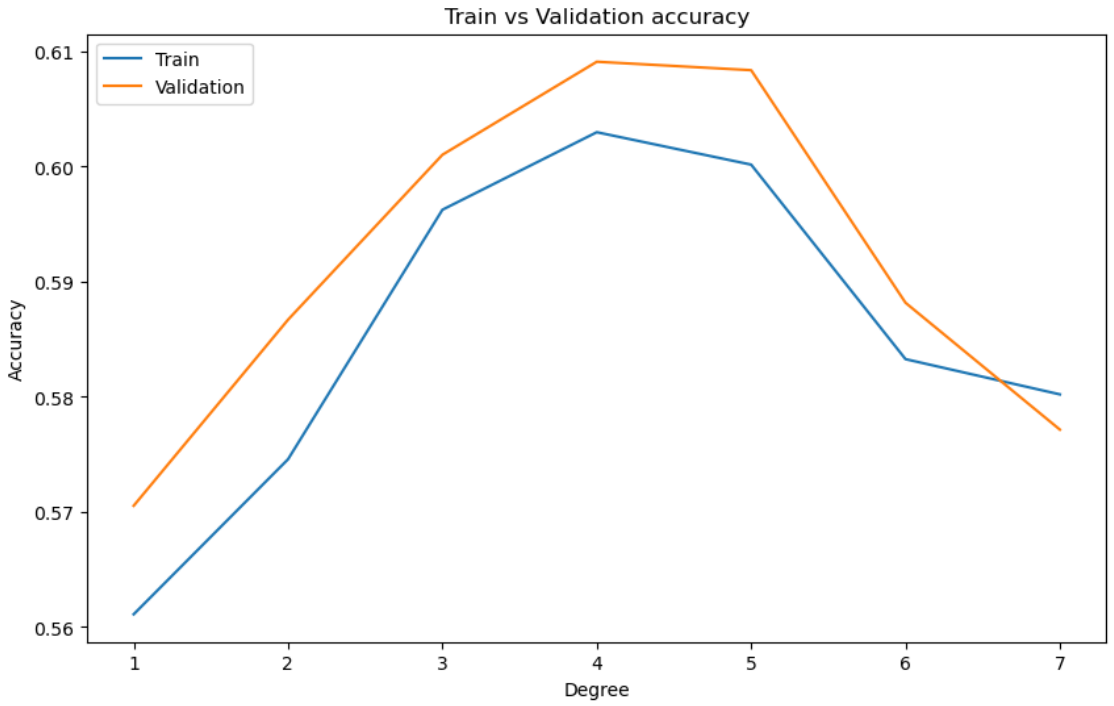
[30]:
```
plt.figure(figsize=(10, 6))

plt.plot(ks, kneighbors_train_errors, label = 'Train')
plt.plot(ks, kneighbors_val_errors,label = 'Validation')
plt.xticks(ks)
plt.legend(loc = 'upper left')

plt.xlabel('K')
plt.ylabel('Accuracy')
plt.title('Train vs Validation accuracy')
plt.show()
```



[33]:
```
ds = range(1, 8)

svc_train_errors = []
svc_val_errors = []

for i, d in enumerate(ds):

    svc = SVC(kernel='poly', class_weight='balanced', degree=d, random_state=42)
    svc.fit(X_train, y_train)

    y_pred = svc.predict(X_train)
    train_error = accuracy_score(y_train, y_pred)
    y_val_pred = svc.predict(X_val)
    val_error = accuracy_score(y_val, y_val_pred)

    svc_train_errors.append(train_error)
    svc_val_errors.append(val_error)

print(svc_train_errors)
print(svc_val_errors)
```

```
[0.5611070291452364, 0.5745775165319618, 0.5962527553269654, 0.6029879990203282,
0.6001714425667402, 0.5832721038452119, 0.5802106294391379]
[0.5705363703159442, 0.5867009551800147, 0.6010286554004408, 0.6091109478324761,
0.6083761939750184, 0.5881704628949302, 0.577149155033064]
```

[37]:
```
plt.figure(figsize=(10, 6))

plt.plot(ds, svc_train_errors, label = 'Train')
plt.plot(ds, svc_val_errors,label = 'Validation')
plt.xticks(ds)
plt.legend(loc = 'upper left')

plt.xlabel('Degree')
plt.ylabel('Accuracy')
plt.title('Train vs Validation accuracy')
plt.show()
```



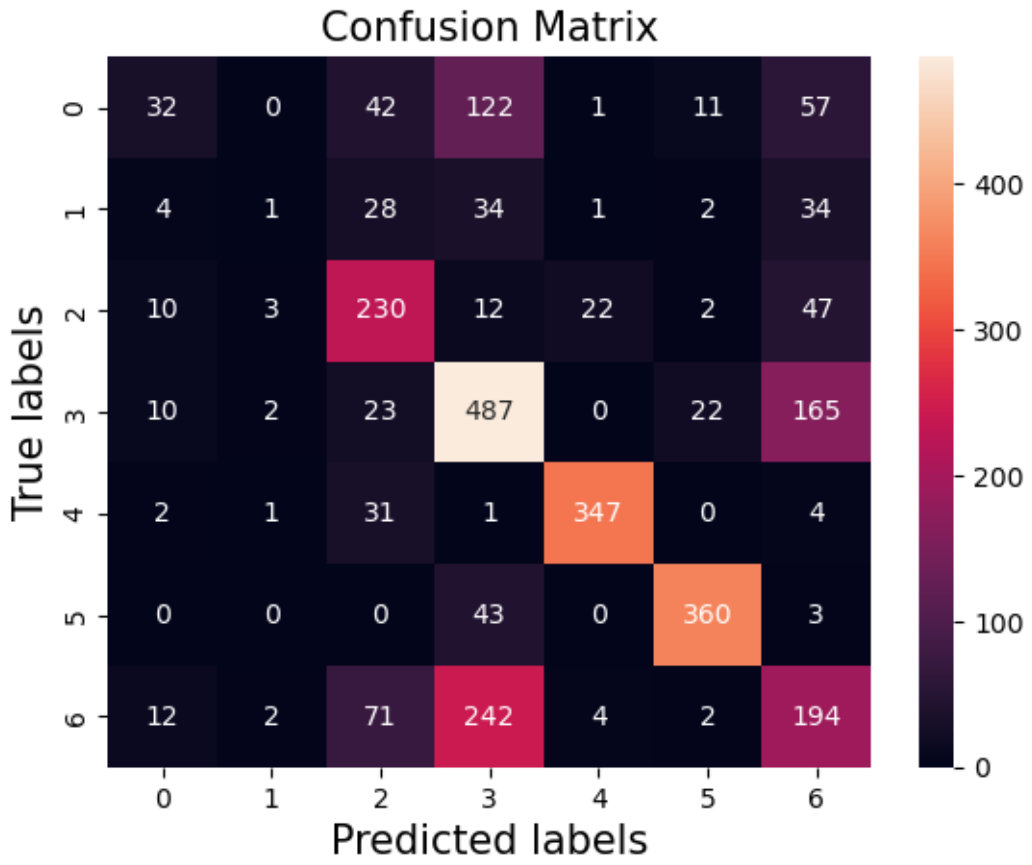
[35]:
```
neigh = KNeighborsClassifier(n_neighbors=15)
neigh.fit(X_train, y_train)

y_pred_test = neigh.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(test_accuracy)
```

```
0.6063165626147631
```

[36]:
```
conf_mat = confusion_matrix(y_test, y_pred_test)
```

```
# Visualize the confusion matrix you computed
ax= plt.subplot()

sns.heatmap(conf_mat, annot=True, fmt='g', ax=ax)

ax.set_xlabel('Predicted labels',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
ax.set_title('Confusion Matrix',fontsize=15)
```

[36]: Text(0.5, 1.0, 'Confusion Matrix')



[ ]: