Movie Genre Classification Project

Carlos, Jun, Shotaro, Tammita

1. Introduction

The objective of this project is to apply different machine learning techniques to classify a movie's genre. The provided datasets contain movie posters and some elementary metadata. In this project, we used labeled training data to train our classifier to predict the genres of unlabeled movies in the test data. Our predictions are then uploaded to Kaggle and our classifier is assigned a score based on the number of correct classifications.

2. Dataset Information

The training and test datasets are given for the project as CSV files. The training data file contains 3094 movies with their metadata of IMDB ID, title, release year, runtime (in minutes), IMDB rating, and genre (label). For simplicity, movies are labeled as one of four classes:

0. Action and/or adventure
1. Romance and/or comedy
2. Drama and/or crime
3. Biography and/or documentary

The test dataset is comprised of 343 movies with the same metadata as the training dataset, but without genre labels.

3. Data Exploration and Visualization

To explore and visualize the data, we created a scatter plot, shown in Figure 1, of the runtime vs. IMDB score of the movies in the training dataset. We colored each data point based on its labeled genre. There is some correlation seen in the plot. Genre 3 movies are generally grouped to the upper left and the remaining genres are grouped more to the right. Genre 0 contains more movies with higher runtimes and lower scores than the other genres. These patterns mean that we can use the runtimes and scores to predict movie genre.

We also created scatter plots for the starting year vs. runtime and starting year vs. IMDB score, but in both cases there was no clear correlation between the two selected features. Starting year may not a good indicator of genre, so we did not did not make as much use of it as we did the other features when creating our models.

Figure 1: Scatter Plot of Runtime vs. IMDB Score

4. Preprocessing Steps Taken

Preprocessing is a technique used to transform raw data into a cleaner form that can be fed to an algorithm and used for further processing. We preprocessed the tabular data by reordering the rows of the data with respect to IMDB IDs and, for the training set, one-hot encoded the genres column. For example, genre 0 is encoded as the vector [1 0 0 0], genre 1 as [0 1 0 0], and so on. We preprocessed the posters by sorting them with respect to IMDB IDs. Then, we used cv2 to read the posters as grayscale images and resized them into 64x64 pixel images. Since the pixels have values between 0 to 255, we normalized the images by dividing by 255. We reshaped the output in this way so that Convolutional Neural Network can be implemented on our data. After preprocessing the data, we implemented the following algorithms included in Section 5, varied the parameters, tested and compared the results of each, and decided which models to use for our final optimized design.

5. Model Selection / Comparison of Models Tried

5.1. K-Nearest Neighbors

The K-nearest neighbors algorithm is a non-parametric method used for classification. It falls under supervised learning since the algorithm uses class labels of the training data. It is simple yet powerful because there is no training involved but can yield great accuracy. To classify a test point, the algorithm finds K points from the training data that are closest to the test point, and the point is classified by a majority vote of its neighbors. If K = 1, the point is simply assigned to the class of its single nearest neighbor. The optimal K value should make stable decision boundaries with few errors. It should not be too large or too small, since a large K value would make the decision boundary a straight line (too smooth) while a small K value may cause overfitting. In either of those cases, implementing the test data to the algorithm will most likely lead to a high percentage of error.

To find an optimal K that gives the best accuracy, our model generated the list of odd-number Ks from 1 to 49 and applied them inside the 10-fold cross-validation loop. Furthermore, we experimented

with different inputs, such as only using columns 4 and 5 of the training data, which are the runtime and the IMDB score. We decided to use runtime and IMDB score in implementing KNN because they showed some correlation with movie genre, as seen in Figure 1. As a result, the best performed K was 23 with accuracy of 0.5471 with the inputs being columns 4 and 5 of the training data, deciding our K-NN algorithm as 23-NN.

While running KNN, we came with an idea to cross validate all our models at once (More on this in Section 6), and while we were testing our accuracy, we experimented with the K value for the KNN. During these tests, we found that the variance of accuracy was high for 23-NN so we sought to find a better K value. Through plugging in several K values, we found that 5-NN was also as good or even better than using 23-NN depending on the run. As a result, we optimized our code so that depending on the accuracy of 23-NN and 5-NN, the algorithm would choose the K value that gave us the higher of the two. The predicted results in our code were named neigh_predict1 and neigh_predict2. The code for our KNN is shown below in Figure 2.

```
from sklearn.neighbors import KNeighborsClassifier

genre_list =  train_genre.tolist()
#genre_list = genres.tolist()


#for cross val
neigh1 = KNeighborsClassifier(n_neighbors=5)
neigh1.fit(train_array_knn,genre_list)
neigh_predict1 = neigh1.predict(test_array_knn)

neigh2 = KNeighborsClassifier(n_neighbors=23)
neigh2.fit(train_array_knn,genre_list)
neigh_predict2 = neigh2.predict(test_array_knn)
```

Figure 2: KNN Code

5.2 SVM

Support Vector Machines (SVM) are supervised learning models with learning algorithms that analyze data used for classification. At first we assumed our data would be somewhat linearly separable so we used a linear classifier that predicts the label of a data vector based on dimensional hyperplanes that divide the dimensionality. With a margin of 2/||w|| between two hyperlanes, the best hyperplanes are the ones with the largest margin. Because there is a possibility that our data may not be linearly separable, we also trained non-linear classifiers such as polynomial, gaussian, and sigmoid.

To determine the best classifier, we implemented a 10-fold cross validation with the training data. The results showed that the linear classifier and the gaussian were the best, giving an accuracy of about 0.56 to 0.61, respectively, depending on the run. Furthermore, we tested to see what columns of the training data would be best for the algorithm. Through trial and error, we found that using columns 3, 4, and 5 (starting year, runtime, and score) resulted in the highest accuracy. From then on, we experimented with the parameters to optimize the algorithm, such as changing the C and gamma. We manually adjusted the parameters for each run and saw that for the linear classifier, C = 0.8 yielded the highest accuracy, and for the gaussian classifier, gamma = 0.009 and C = 1.8 yielded the highest accuracy.

Similar to what we implemented for KNN, we chose the classifier with the greater accuracy of the two. The predicted results in our code were named y_svm1 and y_svm2. The code for our SVM is shown below in Figure 3.

```
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
import numpy as np

#for cross val
genre_list = train_genre.tolist()

#for submission
#genre_list = genres.tolist()

#linear
svclassifier2 = SVC(kernel='linear', C=0.8)

#polynomial
#svclassifier = SVC(kernel='poly', degree=8)

#gaussian
svclassifier = SVC(kernel='rbf', gamma=0.009, C=1.8)

#svclassifier = LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=1e-4)

#sigmoid
#svclassifier = SVC(kernel='sigmoid')

#svm_scores = cross_val_score(svclassifier, csv_data[:,(3,4,5)], genre_list, cv=10, scoring='accuracy')
#print(svm_scores)

#for cross val
svclassifier.fit(train_array, genre_list)
y_svm1 = svclassifier.predict(test_array)

svclassifier2.fit(train_array, genre_list)
y_svm2 = svclassifier2.predict(test_array)
```

Figure 3: SVM Code

5.3 Convolutional Neural Network

Convolutional neural network (CNN) is a deep learning neural network used in computer vision and image classification. It is composed of an input layer (where the input is an image), output layer, and hidden layers, some of which are convolutional. In this project, we used a convolutional neural network to predict the genre of a movie from its poster.

CNN involves 4 main steps: convolution, pooling, flattening, and full connection. The first layer that our input, an unlabeled movie poster, is passed into is always convolutional. In this layer, a filter is created that performs convolution as it moves along the input image. Boundaries and simple colors are identified in this step. A nonlinear layer containing an activation function succeeds this convolutional layer. In our design, we set the activation function to the ReLU function.

Following the nonlinear layer is the pooling layer, which reduces the image volume. As some features have already been identified in previous convolutional layers, pooling allows us to compress the image since we do not need to process those details anymore. We used max pooling (selecting maximum pixel value in a certain region) in our design. After this, we performed flattening, which is the process of converting the resulting 2D array into a 1D vector. Finally, we used the Dense function to fully connect the neural network.

We went through several iterations of the CNN algorithm in order to find out which one worked best for our data using cross validation. First, we worked with the CNN code that was left to us by the

TA's. We began by changing the number of layers. This left us with varying performances, and we weren't really sure how to interpret which setup was best for us. Therefore, we decided to test different setups and measure how our CNN's would do over different Epochs and Batch sizes. Below we have pictures of our 4 main CNN's that we had used. The first is our most basic setup, with the least number of layers and no dropouts (Figure 4). The second is our basic setup with dropouts (Figure 5). The third and fourth are variations of our later designs (Figures 6 and 7). For the first two we had accuracy rates on cross validation was about 30 to 35 percent for epochs between 6 and 8, and around 20 percent accuracy for epochs greater or lower than this range. For our later two CNN's we had accuracy rates on cross validation of about 42 and 50 percent for epochs around 6 and 8, and around 35 to 40 percent accuracy for epochs above or below that range.

```
def createModel():
    model2 = Sequential()

    model2.add(InputLayer(input_shape=[64,64,1]))
    model2.add(Conv2D(32, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    model2.add(Flatten())  # this converts our 3D feature
    model2.add(Dense(64))
    model2.add(Activation('relu'))
    model2.add(Dense(4, activation='softmax'))


    return model2
```

Figure 4

```
[80] from keras import Sequential
     from keras.layers import InputLayer, Conv2D, MaxPool2D, Flatten, Dense, Dropout, SpatialDropout1D, SpatialDropout2D, MaxPooling2D, Activation



     def createModel():
         model2 = Sequential()

         model2.add(InputLayer(input_shape=[64,64,1]))
         model2.add(Conv2D(32, (3, 3)))
         model2.add(Activation('relu'))
         model2.add(MaxPooling2D(pool_size=(2, 2)))
         Dropout(.2)

         model2.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
         model2.add(Dense(64))
         model2.add(Activation('relu'))
         Dropout(.2)
         model2.add(Dense(4, activation='softmax'))


         return model2
```

Figure 5

```
def createModel():
    model2 = Sequential()
    model2.add(InputLayer(input_shape=[64,64,1]))
    model2.add(Conv2D(filters=32,kernel_size=(3,3),padding='same',activation='relu'))
    model2.add(MaxPool2D(pool_size=(2,2),padding='same'))
    model2.add(Conv2D(filters=32,kernel_size=(3,3),padding='same',activation='relu'))
    model2.add(MaxPool2D(pool_size=(2,2),padding='same'))
    model2.add(Conv2D(filters=64,kernel_size=(3,3),padding='same',activation='relu'))
    model2.add(Dropout(0.2))
    model2.add(MaxPool2D(pool_size=(2,2),padding='same'))
    model2.add(Dropout(0.2))

    model2.add(Flatten())
    model2.add(Dense(128, activation='relu'))
    model2.add(Dropout(0.2))
    model2.add(Dense(128, activation='relu'))
    model2.add(Dropout(0.2))
    model2.add(Dense(4, activation='softmax'))

    return model2
```

Figure 6

```
from keras import Sequential
from keras.layers import InputLayer, Conv2D, MaxPool2D, Flatten, Dense, Dropout, SpatialDropout1D, SpatialDropout2D, MaxPooling2D,


def createModel():
    model2 = Sequential()

    model2.add(InputLayer(input_shape=[64,64,1]))
    model2.add(Conv2D(32, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))
    model2.add(Conv2D(32, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    model2.add(MaxPooling2D(pool_size=(2, 2)))
    Dropout(.2)

    model2.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    model2.add(Dense(64))
    model2.add(Activation('relu'))
    Dropout(.2)
    model2.add(Dense(4, activation='softmax'))


    return model2
```

Figure 7

As we can see from the stated data above, the performances of these later CNN's were surprisingly strong. We were able to get almost 50 percent accuracy with some tweaking and were able to get somewhat strong results overall. The first two were rather barebones and didn't really perform quiet as well as we wanted, but this is to be expected as the CNN's were not yet altered for the data that we had. The main issue that we had with our CNN's was that the results greatly varied when we tested with cross validation and on kaggle itself. This was of course one of the greatest challenges because one of the main goals of a CNN was to build one that could be accurate for as many unique real world test data as possible. And while we did experiment with epochs, batch size, the number of layers, the quantity and rate of dropouts, and more, we were still not able to improve our variation or accuracy overall. However as an initial step our setup of the CNN's was good enough for what we wanted and a good starting point to be at, especially since our 4th CNN listed above was able to give us strong results nearing 50 percent accuracy on validation many times.

Moving forward, we tried to improve our CNN's by training them with augmented data. We realized that the biggest challenge of this project was the lack of data. Therefore, we decided to try and test and train our CNN by augmenting data and trying to further develop the CNN even with the limited data that we had. Below we have attached code for the augmented data and the results that came from it. Figure 8 - Figure 12 are the same CNN's in the same order, attached with their respective loss and accuracy curves. We tried the same code and saw the impact that it had on our results. In Figure 13 we see our setup for creating augmented data and then training our CNN with this tested data.

```
def createModel():
    model2 = Sequential()

    model2.add(InputLayer(input_shape=[64,64,1]))
    model2.add(Conv2D(32, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    model2.add(Flatten())  # this converts our 3D feature
    model2.add(Dense(64))
    model2.add(Activation('relu'))
    model2.add(Dense(4, activation='softmax'))


    return model2
```

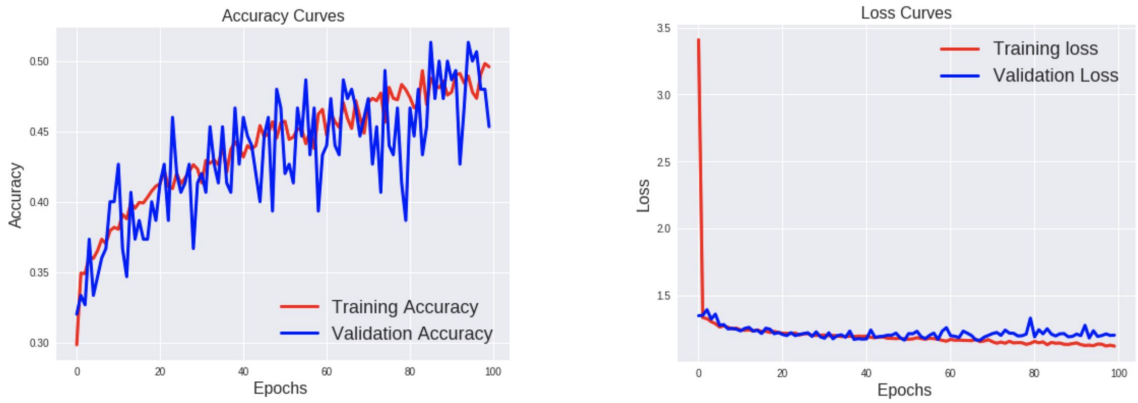

Figure 8

```
[80] from keras import Sequential
     from keras.layers import InputLayer, Conv2D, MaxPool2D, Flatten, Dense, Dropout, SpatialDropout1D, SpatialDropout2D, MaxPooling2D, Activation


     def createModel():
         model2 = Sequential()

         model2.add(InputLayer(input_shape=[64,64,1]))
         model2.add(Conv2D(32, (3, 3)))
         model2.add(Activation('relu'))
         model2.add(MaxPooling2D(pool_size=(2, 2)))
         Dropout(.2)

         model2.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
         model2.add(Dense(64))
         model2.add(Activation('relu'))
         Dropout(.2)
         model2.add(Dense(4, activation='softmax'))


         return model2
```

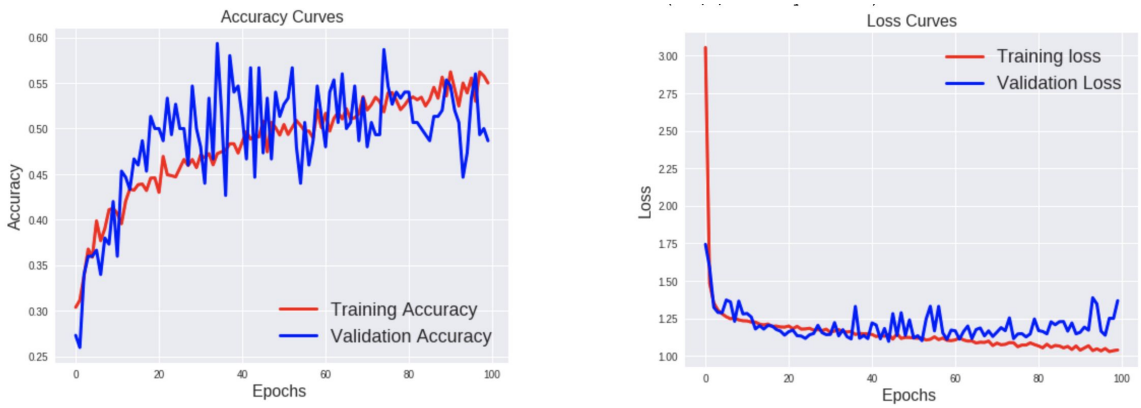

Figure 9

```
def createModel():
    model2 = Sequential()
    model2.add(InputLayer(input_shape=[64,64,1]))
    model2.add(Conv2D(filters=32,kernel_size=(3,3),padding='same',activation='relu'))
    model2.add(MaxPool2D(pool_size=(2,2),padding='same'))
    model2.add(Conv2D(filters=32,kernel_size=(3,3),padding='same',activation='relu'))
    model2.add(MaxPool2D(pool_size=(2,2),padding='same'))
    model2.add(Conv2D(filters=64,kernel_size=(3,3),padding='same',activation='relu'))
    model2.add(Dropout(0.2))
    model2.add(MaxPool2D(pool_size=(2,2),padding='same'))
    model2.add(Dropout(0.2))

    model2.add(Flatten())
    model2.add(Dense(128, activation='relu'))
    model2.add(Dropout(0.2))
    model2.add(Dense(128, activation='relu'))
    model2.add(Dropout(0.2))
    model2.add(Dense(4, activation='softmax'))

    return model2
```

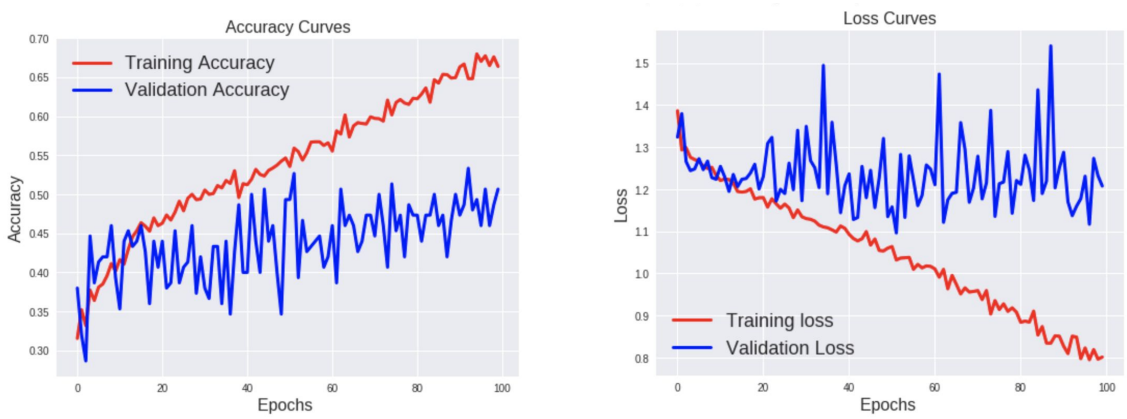

Figure 10

```
from keras import Sequential
from keras.layers import InputLayer, Conv2D, MaxPool2D, Flatten, Dense, Dropout, SpatialDropout1D, SpatialDropout2D, MaxPooling2D,


def createModel():
    model2 = Sequential()

    model2.add(InputLayer(input_shape=[64,64,1]))
    model2.add(Conv2D(32, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))
    model2.add(Conv2D(32, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    model2.add(MaxPooling2D(pool_size=(2, 2)))
    Dropout(.2)

    model2.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    model2.add(Dense(64))
    model2.add(Activation('relu'))
    Dropout(.2)
    model2.add(Dense(4, activation='softmax'))


    return model2
```

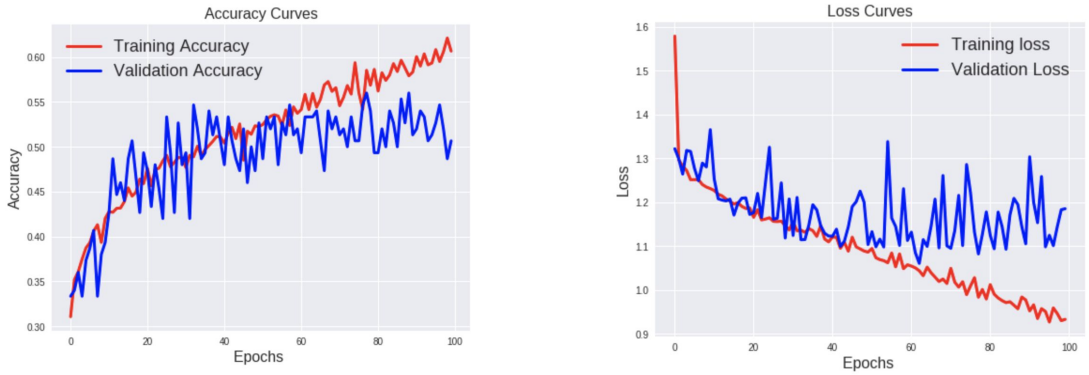

Figure 11

```
def createModel():
    model2 = Sequential()
    model2.add(InputLayer(input_shape=[64,64,1]))
    model2.add(Conv2D(32, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    model2.add(Conv2D(32, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    model2.add(Conv2D(64, (3, 3)))
    model2.add(Activation('relu'))
    model2.add(MaxPooling2D(pool_size=(2, 2)))

    model2.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    model2.add(Activation('relu'))
    model2.add(Dropout(0.5))
    model2.add(Dense(4, activation='softmax'))
```

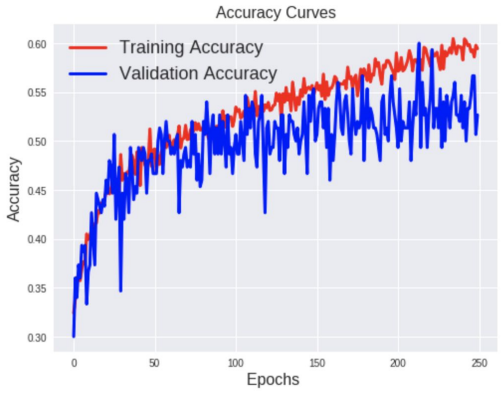

Figure 12

```
#SECOND OPTIMIZATION RUN
from keras.preprocessing.image import ImageDataGenerator
from keras import Sequential
from keras.layers import InputLayer, Conv2D, MaxPool2D, Flatten, Dense, Dropout, SpatialDropout1D, SpatialDropout2D, MaxPooling2D
import keras


model3 = createModel()

model3.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

batch_size = 256
epochs = 100
train_datagen = ImageDataGenerator(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
test_datagen = ImageDataGenerator()
train_generator = train_datagen.flow(x=cross_trainx,y=cross_trainy,batch_size=32)
validation_generator = test_datagen.flow(x=cross_testx,y=y_test,batch_size=32)

# Fit the model on the batches generated by datagen.flow().
history2 = model3.fit_generator(datagen.flow(x=cross_trainx, y=cross_trainy, batch_size=batch_size),steps_per_epoch=int(np.ceil(cross_trainx.shape[0] / float(batch
                                epochs=epochs,
                                validation_data=(cross_testx, keras.utils.np_utils.to_categorical(test_genre)),
                                workers=4)

model3.evaluate(cross_testx, keras.utils.np_utils.to_categorical(test_genre))
```

Figure 13

As we can see the graphs overall trained much better as compared to the unaltered training data that we had. Before our data was averaging about 35 percent for the first 2 CNN's, and about 45 to 50 percent for the later 2 graphs. We can see that the accuracy increase overall for all of the graphs and we can see and overall increase performance in terms of peak accuracy as some of the graphs are able to hit a consistent 50 - 55 which was not able to happen before. In addition, and arguably the most important part is that the graphs are able to have less variation than before. This makes sense because of the fact that the augmented data allows for "more" training data. The data that is being fed is altered and thus we are able to have data that is not quiet the same being fed in. Therefore the model is able to avoid overfitting to the given data and essentially work with more parameters and more new

information in order to become better at classifying information it has not seen yet. We do however noticed that there seems to be a bottleneck in performance. We can see that for most graphs there is a peak in performance around .5 to .55 accuracy rate. This we realized must be do to a limit of our optimization. In order to increase performance we would need to increase our optimization tools and techniques in order to improve from this point. We tried running with more epochs, at around 250, but the performance still was around the same range. Therefore we concluded that we would have to improve our model even more. We weren't quite sure how to continue this and while we did research some ideas, and tried different parameters to tweak, the results were not improving. We concluded that in order to improve our CNN it would take ideas and tools that we would need to learn as we become more familiar with CNN. Overall, our CNN underwent numerous performance increase during our project. As we being to become more familiar with CNN's and their characteristics, we know we will be able to create even more optimal networks and bring better results.

5.4 Naive Bayes Classifier

Naive Bayes is a text classification algorithm based on Bayes' Theorem and the assumption that the input features are independent. We used the Naive Bayes algorithm to label the genre of the movies in the partitioned training data (when using cross validation) according to the words in the movies' titles.

The first step of the algorithm is to convert the dataset into a frequency table that records the number of times a word appears in the training data. Then, a likelihood table is created and the Naive Bayesian equation is used to calculate the posterior probability for each class. The class prediction for an inputted data point is the class with the highest posterior probability.

To implement this, we used the Pipeline class in Scikit-Learn to tokenize the titles, compute the number of each word's occurrences (word counts), and compute term frequencies by dividing each word's count in a title by the number of words in that title. We then trained a multinomial Naive Bayes model. The multinomial variant is used for discrete counts and is the variant most used for text classification. The code for our Naive Bayes with cross validation is shown below in Figure 14.

```
import random
import numpy as np

test_list = list(range(3094))
#print(test_list)
train_list = list(range(3094))
#print(train_list)

test_rand = random.sample(test_list, 343)
train_rand = list(set(train_list) - set(test_rand))

#print(test_rand)
#print(train_rand)

test_genre = csv_data[test_rand,-1]
train_genre = csv_data[train_rand,-1]
#print(test_genre)
#print(train_genre)

test_title_cross = csv_data[test_rand,2]
train_title_cross = csv_data[train_rand,2]

test_genre = genres[test_rand].tolist()
train_genre = genres[train_rand].tolist()

from sklearn.pipeline import Pipeline
text_clf = Pipeline([
   ('vect', CountVectorizer()),
   ('tfidf', TfidfTransformer()),
   ('clf', MultinomialNB()),
   ])
text_clf.fit(train_title_cross, train_genre)
bayes_predict = text_clf.predict(test_title_cross)
```

Figure 14: Naive Bayes Code

The results from cross validation yielded an accuracy of about 0.41 with a variance of about 0.02. Since Naive Bayes does not perform well compared to other models, we decided not to utilize it in our final model. A possible reason why Naive Bayes did not perform well for our project may be that there are not many common words in the provided movie titles. If this is the case, then the movie titles from the given dataset are not accurate predictors of genre.

5.5. Final Design: Combination of KNN, SVM and CNN

We utilized the KNN, SVM, and CNN models discussed above in our final predictor model. For each test data point, CNN outputted a 4-dimensional vector of floating point numbers between 0 and 1, while K-NN and SVM outputted the actual genre label number. To combine these three results, we one-hot encoded K-NN's and SVM's results, multiplied them with a weight between 0 and 1, and added them to CNN's output. We chose the weights by manually going through and typing in different values until the accuracy of the summed result was generally optimized. In the end, we ended up multiplying 0.68 to the one-hot encoded K-NN's and SVM's results. The code for summing the predicted results is shown below in Figure 15.

```
# COMBINE CNN, KNN, and SVM
one_hot1_final = to_categorical(neigh_predict1_final)*0.68
one_hot2_final = to_categorical(y_svm1_final)*0.68
one_hot3_final = to_categorical(y_svm2_final)*0.68
one_hot4_final = to_categorical(neigh_predict2_final)*0.68

k_predict_final = one_hot1_final + one_hot2_final + predict_final
print(k_predict_final)
k_predict_final.shape
```

Figure 15: Code for Summing of the Results

This process takes into account the predictions from all three models and thus, with appropriate parameters and weights, can yield a more accurate prediction than that of any one of the models alone. From cross validating the summed result, we were able to tell that the accuracy increased. Individually, the accuracy of SVM and KNN were around 0.55 with a variance of about 0.05 and the accuracy of CNN was around 0.43 with a variance of about 0.04. The accuracy of the summed result was around 0.58 with a variance of 0.04. However, these results were only from the cross validation. When we ran the test data and submitted to Kaggle, we had an accuracy of only about 0.55 as the highest.

6. Cross Validation Testing

One of the most important parts of this project was cross validating so that we could test our accuracy and also make sure that we would not overfit. We set up our code so that each model we used had some kind of way to cross validate or check for accuracy using only the training data.

For KNN and SVM, we used the cross_val_score function to examine our accuracy. Furthermore, our KNN cross validation included a for loop that optimized the K. Figure 16 shows the code that we used for KNN and Figure 17 shows the code that we used for SVM.

```
#Using cross validation to find the best k for KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
import numpy as np

genre_list = genres.tolist()
#genre_list = train_genre.tolist()


# creating odd list of K for KNN
klist = list(range(1,50))

# subsetting just the odd ones
neighbors = filter(lambda x: x % 2 != 0, klist)

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, csv_data[:,(4,5)], genre_list, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

print("k best =",cv_scores.index(max(cv_scores)))
print("accuracy =",cv_scores[23])
```

Figure 16: Code for KNN Cross Validation

```
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
import numpy as np

#for cross val
genre_list = train_genre.tolist()

#for submission
#genre_list = genres.tolist()

#linear
svclassifier2 = SVC(kernel='linear', C=0.8)

#polynomial
#svclassifier = SVC(kernel='poly', degree=8)

#gaussian
svclassifier = SVC(kernel='rbf', gamma=0.009, C=1.8)

#svclassifier = LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=1e-4)

#sigmoid
#svclassifier = SVC(kernel='sigmoid')

svm_scores = cross_val_score(svclassifier, csv_data[:,(3,4,5)], genre_list, cv=10, scoring='accuracy')
print(svm_scores)
```

Figure 17: Code for SVM Cross Validation

For CNN, Naive Bayes, and determine the accuracy of the summed results we created an one iteration cross validation algorithm. We took the training data and created a training set and a testing set, in which the number of points in the testing set can be modified by typing the number we want. We then ran the training set for the models and used the accuracy_score function to compare the predicted classes to the classes in the test set. Figure 18 below shows the code we used to implement this specific cross validation method.

```
import random
import numpy as np

test_list = list(range(3094))
#print(test_list)
train_list = list(range(3094))
#print(train_list)

test_rand = random.sample(test_list, 150)
train_rand = list(set(train_list) - set(test_rand))

#print(test_rand)
#print(train_rand)

test_genre = csv_data[test_rand,-1]
train_genre = csv_data[train_rand,-1]

row_idx1 = np.array(test_rand)
col_idx1 = np.array([3,4,5])
test_array = csv_data[row_idx1[:, None], col_idx1]
#print(test_array)

row_idx2 = np.array(train_rand)
col_idx2 = np.array([3,4,5])
train_array = csv_data[row_idx2[:, None], col_idx2]
#print(train_array.shape)

row_idx3 = np.array(train_rand)
col_idx3 = np.array([4,5])
train_array_knn = csv_data[row_idx3[:, None], col_idx3]
#print(train_array_knn.shape)

row_idx4 = np.array(test_rand)
col_idx4 = np.array([4,5])
test_array_knn = csv_data[row_idx4[:, None], col_idx4]
```

```
#Overall accuracy
from sklearn.metrics import accuracy_score

genre_pred = np.argmax(k_predict,axis=1)
#genre_pred = y_svm

genre_pred_list = genre_pred.tolist()
test_genre_list = test_genre.tolist()

accuracy_score(test_genre_list, genre_pred_list)
```

Figure 18: Code for the Specific Cross Validation Method

7. Results and Conclusion

During our cross validation process, our accuracy, using the summed results, was around 0.58. We were able to observe that taking the weighted sum of the predicted classes of SVM, KNN, and CNN improved the accuracy instead of just using one of the models referenced above. For every run, we compared the accuracy of the individual models and the summed result. We observed that there was at least an increase of 0.03 in accuracy using the summed result over those of the individual models. However, the variance was noticeably large, which lead us to believe that the algorithm that we were using was not optimized enough. We also suspected that the models we were using were varying each time we ran through the code. Especially with CNN being volatile and with low accuracy, the overall result seemed to be affected significantly.

When testing with the actual test data set, we first submitted the predicted results that we obtained from using KNN. This resulted in an accuracy of 0.526. We then experimented with CNN, which at first resulted in 0.38 and 0.409. Because of CNN outputting such a low scores, we decided to work on the CNN using just the training data, which led to us implementing the one iteration cross validation technique. During the process, we came up with an idea to combine the weighted scores of the KNN and CNN, which resulted in 0.474. From then on, we implemented SVM and Naive Bayes so that we could experiment with several models to find the best algorithm. Through trial and error and researching about CNN, we were able to increase the CNN model's accuracy to around 0.47. With this in mind we implemented the summed results method to achieve an accuracy of 0.555. Even though this was below what we were averaging with just using cross validation, we found that this increase in accuracy was significant for our team. We continued to optimize our parameters and were able to further increase the accuracy using cross validation; however, we ended up averaging about 0.54 for the rest of the submission.

Overall, we were able to learn and implement many essential models used in supervised classification problems. We were able to utilize the KNN and SVM algorithms that we learned in class and apply them to this project using several different methods. The KNN algorithm came in handy when we first started this project because the implementation of it was straightforward. It was a good place to start because it was easy for us to visualize how the algorithm was classifying the data. It also led us to use cross validation to optimize parameters, which increased our accuracy. Similarly, SVM was very useful in that it allowed us to learn how the data could be classified. From linear classification to gaussian classification, we became familiar with several of the parameters that affected our results, such as the gamma and C. In the end, through experimenting, we were able to increase the overall accuracy of both KNN and SVM.

In addition, we learned and implemented machine learning models used in image (CNN algorithm) and text (Naive Bayes algorithm) classifications that were not covered in class. We used CNN on the provided posters and Naive Bayes on the movie titles. In both cases, we preprocessed the required data, applied the algorithms, and used cross validation to determine the accuracy of the models. We also tuned the parameters and created graphs to see which parameters worked best. Naive Bayes did not perform as well as KNN, SVM, or CNN, so we did not use it in our final model.

After implementing each algorithm, we tried taking the weighted sums of the predicted classes of SVM, KNN, and CNN and saw that doing so improved the accuracy instead of using just one model. Thus, our final optimized design consists of the SVM, KNN, and CNN models trained from the provided training dataset. The final predictions are then determined by taking the weighted sum of the genre predictions of each model and, for each testing data point, selecting the genre with the greatest weight.

8. Code

```
[ ]  !unzip train_posters.zip
     !unzip test_posters.zip
```

```
[ ]  import pandas as pd
     import numpy as np
     from sklearn.preprocessing import LabelEncoder
     from keras.utils import to_categorical
     import os
     import cv2
     import matplotlib.pyplot as plt

     # Read csv data
     # Reorder the labels to match the order of the images

     csv_data = pd.read_csv('train_data.csv').as_matrix()

     csv_data = csv_data[csv_data[:,1].argsort()] # csv_data[:,1].argsort() returns indices that sort movies by imdb #
     genres = csv_data[:,-1]


     test_data_csv = pd.read_csv('test_data.csv').as_matrix()

     # One-hot encode genres column

     train_labels = to_categorical(np.array(genres))
     #print("Label for first training example: {}".format(genres[0]))
     #print("One-hot encoded label for first training example: {}".format(train_labels[0]))
```

```
[ ]  import random
     import numpy as np

     test_list = list(range(3094))
     #print(test_list)
     train_list = list(range(3094))
     #print(train_list)

     test_rand = random.sample(test_list, 150)
     train_rand = list(set(train_list) - set(test_rand))

     #print(test_rand)
     #print(train_rand)

     test_genre = csv_data[test_rand,-1]
     train_genre = csv_data[train_rand,-1]

     row_idx1 = np.array(test_rand)
     col_idx1 = np.array([3,4,5])
     test_array = csv_data[row_idx1[:, None], col_idx1]
     #print(test_array)

     row_idx2 = np.array(train_rand)
     col_idx2 = np.array([3,4,5])
     train_array = csv_data[row_idx2[:, None], col_idx2]
     #print(train_array.shape)

     row_idx3 = np.array(train_rand)
     col_idx3 = np.array([4,5])
     train_array_knn = csv_data[row_idx3[:, None], col_idx3]
     #print(train_array_knn.shape)

     row_idx4 = np.array(test_rand)
     col_idx4 = np.array([4,5])
     test_array_knn = csv_data[row_idx4[:, None], col_idx4]
```

```
[ ]  train_data = 'train_posters'
     test_data = 'test_posters'

     def preprocess_training_data():
        train_images = []
        image_num = 0

        for ind,i in enumerate(csv_data[:,1]):

           path = os.path.join(train_data,str(i) + ".jpg")
           img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
           img = cv2.resize(img, (64,64))

           train_images.append(np.array(img)/255)
        return train_images


     def preprocess_test_data():
        test_images = []
        for ind,i in enumerate(test_data_csv[:,1]):

           path = os.path.join(test_data,str(i) + ".jpg")
           img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
           img = cv2.resize(img, (64,64))
           test_images.append(np.array(img)/255)

        return test_images

     preprocessed_train = preprocess_training_data()
     preprocessed_test = preprocess_test_data()


     x_train = np.array(preprocessed_train).reshape(-1,64,64,1)
     y_train = train_labels

     x_test = np.array(preprocessed_test).reshape(-1,64,64,1)
     x_test.shape
```

```
[ ]  # Display training example #1371 Death Note
     # Display movie poster and associated label and title

     # Feel free to change the train_ind and see how the preprocessing affect the images
     train_ind = 1371
     plt.imshow(preprocessed_train[train_ind])
     #print(csv_data[:,1][train_ind])
     #print(y_train[train_ind])
     print(csv_data[:,(3,4,5)])
```

```
[ ]  # Displaying test example #200
     # Remember there is no genre label or title associated with this image
     # We are trying to predict the labels!

     plt.imshow(preprocessed_test[200])
     print("Test example #200")
```

```
[ ]  import numpy as np

     cross_testx = x_train[test_rand]
     cross_trainx = np.delete(x_train, test_rand, 0)
     cross_trainy =np.delete(y_train, test_rand, 0)
```

```
[ ]  from keras import Sequential
     from keras.layers import InputLayer, Conv2D, MaxPool2D, Flatten, Dense, Dropout, SpatialDropout2D
     import numpy as np

     model = Sequential()

     model.add(InputLayer(input_shape=[64,64,1]))
     model.add(Conv2D(filters=32,kernel_size=(3,3),padding='same',activation='relu'))
     model.add(MaxPool2D(pool_size=(2,2),padding='same'))
     model.add(Conv2D(filters=32,kernel_size=(3,3),padding='same',activation='relu'))
     model.add(MaxPool2D(pool_size=(2,2),padding='same'))
     model.add(Conv2D(filters=64,kernel_size=(3,3),padding='same',activation='relu'))
     model.add(Dropout(0.2))
     model.add(MaxPool2D(pool_size=(2,2),padding='same'))
     model.add(Dropout(0.2))



     model.add(Flatten())
     model.add(Dense(128, activation='relu'))
     model.add(Dropout(0.2))
     model.add(Dense(128, activation='relu'))
     model.add(Dropout(0.2))
     model.add(Dense(4, activation='softmax'))

     model.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])

     #for cross val
     model.fit(x=cross_trainx, y=cross_trainy, epochs=4, batch_size=64)
     model.fit(x=cross_trainx, y=cross_trainy, epochs=4, batch_size= 128)
     model.fit(x=cross_trainx, y=cross_trainy, epochs=4, batch_size=256)
     model.fit(x=cross_trainx, y=cross_trainy, epochs=4, batch_size=128)


     #for submission
     #model.fit(x=x_train, y=y_train, epochs=4, batch_size=64)
     #model.fit(x=x_train, y=y_train, epochs=4, batch_size= 128)
     #model.fit(x=x_train, y=y_train, epochs=4, batch_size=256)
     #model.fit(x=x_train, y=y_train, epochs=4, batch_size=128)

     model.summary()

     model.save("movie_classifier.h5py")

     predict = model.predict(cross_testx, batch_size=32, verbose=0, steps=None)
     #predict = model.predict(x_test, batch_size=32, verbose=0, steps=None)
```

```
[ ]  from sklearn.svm import LinearSVC
     from sklearn.svm import SVC
     from sklearn import datasets, linear_model
     from sklearn.model_selection import cross_val_score
     import numpy as np

     #for cross val
     genre_list = train_genre.tolist()

     #for submission
     #genre_list = genres.tolist()

     #linear
     svclassifier2 = SVC(kernel='linear', C=0.8)

     #polynomial
     #svclassifier = SVC(kernel='poly', degree=8)

     #gaussian
     svclassifier = SVC(kernel='rbf', gamma=0.009, C=1.8)

     #svclassifier = LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=1e-4)

     #sigmoid
     #svclassifier = SVC(kernel='sigmoid')

     #svm_scores = cross_val_score(svclassifier, csv_data[:,(3,4,5)], genre_list, cv=10, scoring='accuracy')
     #print(svm_scores)

     #for cross val
     svclassifier.fit(train_array, genre_list)
     y_svm1 = svclassifier.predict(test_array)

     svclassifier2.fit(train_array, genre_list)
     y_svm2 = svclassifier2.predict(test_array)

     #for submission
     #svclassifier.fit(csv_data[:,(3,4,5)], genre_list)
     #y_svm1 = svclassifier.predict(test_data_csv[:,(3,4,5)])

     #svclassifier.fit(csv_data[:,(4,5)], genre_list)
     #y_svm2 = svclassifier.predict(test_data_csv[:,(4,5)])
```

```
[ ]  from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score

     genre_list =  train_genre.tolist()
     #genre_list = genres.tolist()


     #for cross val
     neigh1 = KNeighborsClassifier(n_neighbors=5)
     neigh1.fit(train_array_knn,genre_list)
     neigh_predict1 = neigh1.predict(test_array_knn)

     neigh2 = KNeighborsClassifier(n_neighbors=23)
     neigh2.fit(train_array_knn,genre_list)
     neigh_predict2 = neigh2.predict(test_array_knn)

     #for submission
     #neigh1 = KNeighborsClassifier(n_neighbors=5)
     #neigh1.fit(csv_data[:,(4,5)],genre_list)
     #neigh_predict1 = neigh1.predict(test_data_csv[:,(4,5)])

     #neigh2 = KNeighborsClassifier(n_neighbors=17)
     #neigh2.fit(csv_data[:,(4,5)],genre_list)
     #neigh_predict2 = neigh2.predict(test_data_csv[:,(4,5)])

     # COMBINE CNN, KNN, and SVM
     #one_hot1 = to_categorical(neigh_predict1)*0.68
     #one_hot2 = to_categorical(y_svm1)*0.68
     #one_hot3 = to_categorical(y_svm2)*0.68
     #one_hot4 = to_categorical(neigh_predict2)*0.68
```

```
genre_pred1 = y_svm1.tolist()
genre_pred2 = y_svm2.tolist()
test_genre_list = test_genre.tolist()

acc1 = accuracy_score(test_genre_list, genre_pred1)
acc2 = accuracy_score(test_genre_list, genre_pred2)

if acc1 >= acc2:
  print("one_hot2")
  one_hot2 = to_categorical(y_svm1)*0.68
else:
  print("one_hot3")
  one_hot2 = to_categorical(y_svm2)*0.68


genre_pred3 = neigh_predict1.tolist()
genre_pred4 = neigh_predict2.tolist()

acc3 = accuracy_score(test_genre_list, genre_pred3)
acc4 = accuracy_score(test_genre_list, genre_pred4)

if acc3 >= acc4:
  print("one_hot1")
  one_hot1 = to_categorical(neigh_predict1)*0.68
else:
  print("one_hot4")
  one_hot1 = to_categorical(neigh_predict2)*0.68


k_predict = predict+one_hot1+one_hot1
```

```
[ ]  #Overall accuracy
     from sklearn.metrics import accuracy_score

     genre_pred = np.argmax(k_predict,axis=1)
     #genre_pred = y_svm

     genre_pred_list = genre_pred.tolist()
     test_genre_list = test_genre.tolist()

     accuracy_score(test_genre_list, genre_pred_list)
```

```
[ ]  model.summary()

     model.save("movie_classifier.h5py")

     predict_final = model.predict(x_test, batch_size=32, verbose=0, steps=None)
```

```
[ ]  #for submission
     y_svm1_final = svclassifier.predict(test_data_csv[:,(3,4,5)])

     y_svm2_final = svclassifier2.predict(test_data_csv[:,(3,4,5)])
```

```
[ ]  neigh_predict1_final = neigh1.predict(test_data_csv[:,(4,5)])

     neigh_predict2_final = neigh2.predict(test_data_csv[:,(4,5)])

     # COMBINE CNN, KNN, and SVM
     one_hot1_final = to_categorical(neigh_predict1_final)*0.68
     one_hot2_final = to_categorical(y_svm1_final)*0.68
     one_hot3_final = to_categorical(y_svm2_final)*0.68
     one_hot4_final = to_categorical(neigh_predict2_final)*0.68

     k_predict_final = one_hot2_final + one_hot4_final + predict_final
     print(k_predict_final)
     k_predict_final.shape
```

```
[ ]  # write test predictions in csv file

     from google.colab import files
     import numpy as np
     import csv
     k_predict_class = np.argmax(k_predict_final,axis=1)
     with open('wasteof20hrs.csv','w') as f:
        writer = csv.writer(f)
        for val in k_predict_class:
           writer.writerows(str(val))
     f.close()

     files.download("wasteof20hrs.csv") # download csv file of test predictions
```

```
[ ]  #Using cross validation to find the best k for KNN
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn import datasets, linear_model
     from sklearn.model_selection import cross_val_score
     import numpy as np

     genre_list = genres.tolist()
     #genre_list = train_genre.tolist()


     # creating odd list of K for KNN
     klist = list(range(1,50))

     # subsetting just the odd ones
     neighbors = filter(lambda x: x % 2 != 0, klist)

     # empty list that will hold cv scores
     cv_scores = []

     # perform 10-fold cross validation
     for k in neighbors:
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, csv_data[:,(4,5)], genre_list, cv=10, scoring='accuracy')
        cv_scores.append(scores.mean())

     print("k best =",cv_scores.index(max(cv_scores)))
     print("accuracy =",cv_scores[23])
```