Report

109550005 張可晴

Introduction

In this project, I would use 5 classifiers which are kNN, random forest, support vector machine, linear discriminant analysis, and convolutional neural network respectively to predict 3 datasets. For public image dataset, I would choose Fashion-MNIST. As for public non-image dataset, I would use Mobile Price Range Prediction from Kaggle. Finally, I utilize the article from different categories of BBC News as self-made dataset.

Algorithms

Among the five classifiers I used in this project, four of them, which are kNN, random forest, SVM, and linear discriminant analysis, are from scikit-learn, while CNN model comes from keras. kNN classifier implement k-nearest neighbors vote algorithm, which predict the data based on the voting from k nearest neighbors. Secondly, random forest classifier will fit some decision tree classifiers on part of dataset and improve the performance by averaging those classifiers. In addition, SVM classifier aim to create a line or hyperplane which separate data into classes. As for LDA, it projects data from higher dimensionality to lower one in order to maximize the variability between the classes and reduce the variability within the classes. Finally, CNN is one kind of artificial neural network, while at least one of the layers must be convolution.

Analysis

A public image dataset: Fashion-MINST

I use the dataset provided from keras. Fashion-MNIST has already established training and testing dataset, so I directly use this specified splitting. It has the same data size with MNIST handwritten digits that are 60000 training data and 10000 testing data, while there are 10 classes for Fashion-MNIST. Moreover, 10 classes are T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

Comparison with different hyperparameter

For kNN, I firstly do comparison between different k value. To be more specific, I compare different k value among odd number to prevent from having the same number of neighbors. From the below table, it is obvious that accuracy, precision, recall, and f1-score are all highest when k is 5. From these results, we can come to 2 conclusions. On the one hand, we can conclude that when k is 5, kNN can make the best prediction to the Fashion-MNIST dataset. In other words, when k is less than 5, the kNN model would become so complex that overfitting might occur. By contrast, when k is more than 5, underfitting might happen. On the other hand, it is apparent that all metrics would have the best result when k is equal to 5, which means that the dataset is balanced. The conclusion makes sense since all the classes have the same numbers of data. However, as you can see, value of AUROC is not the best when k is equal to 5. Yet, I would still prefer to choose the best kNN model when k is equal to 5. Although AUROC is a good metric when the dataset is balanced like Fashion-MNIST, accuracy is also the good one in the same situation, plus being simple and easy to interpret. As a result, I would choose k=5 as the best kNN model and use this value for the following discussion.

	Accuracy	Precision	Recall	F1-score	AUROC
k=1	84.97	85.27	84.97	85.03	91.65
k=3	85.41	85.75	85.41	85.39	95.84
k=5	85.54	85.78	85.54	85.46	96.85
k=7	85.40	85.70	85.40	85.34	97.31
k=9	85.19	85.53	85.19	85.13	97.58

kNN comparison with different k

In terms of SVM, I compare different kernels, which are linear kernel, polynomial kernel, and Gaussian kernel. Linear kernel is just a special case of RBF kernel, so it's unlikely that the former would perform better than the latter. Also, from "Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel", it concluded that if complete model selection using the Gaussian kernel has been conducted, there is no need to consider linear SVM. Hence, it is no surprise that Gaussian kernel would outperform linear one. As for polynomial kernel, it turns out to be worse one, which surprises me since I expect that it would outperform linear one according to "A Comparison Study of Different Kernel Functions for SVM-based Classification of Multi-temporal Polarimetry SAR Data". Based on that paper, since polynomial kernel is more advanced than the linear one, it should release more reliable results with higher accuracies. In my humble opinion, the results that linear kernel outperform polynomial one might result from the hyperparameter of polynomial that I don't find out the best one. To sum up, RBF kernel performs best with every aspect of metrics than linear and polynomial ones, so I will use RBF kernel for the following discussion.

	Accuracy	Precision	Recall	F1-score	AUROC
Linear	84.63	84.57	84.63	84.56	98.45
polynomial	77.13	79.69	77.13	77.61	97.58
RBF	88.29	88.24	88.29	88.24	99.05

SVM comparison with different kernel

As for random forest, n_estimator represents the number of trees in the forest.

Intuitively, the more the trees are in the forest, the better the random forest classifier performs. By observing the results shown below, we can easily find that random forest classifier with n_estimator being 1000 outperforms the ones with n_esitmater being 100 and 300. Therefore, I would use the best one for the following discussion.

	Accuracy	Precision	Recall	F1-score	AUROC
n_estimator=100	87.64	87.53	87.64	97.49	98.93
n_estimator=300	87.78	87.67	87.78	87.64	99.00
n_estimator=1000	87.86	87.75	87.86	87.71	99.03

Random forest comparison with different n_estimator

Speaking of CNN model, I decide to compare different activation functions that are Linear and ReLu. CNN model utilizes multiple layers to make model learn better, while if the activation function is linear one, neural network would just collapse to 1 layer because any linear combination of linear functions is a linear function itself. Instead, CNN with ReLu, which is a non-linear function, will not let the above situation happen, thus performing better than CNN with linear activation function. The table below just show that ReLu outperforms linear activation function. Hence, I use CNN with ReLu function for the following discussion.

	Accuracy	Precision	Recall	F1-score	AUROC
Linear	91.53	91.73	91.53	91.55	99.52
ReLu	92.53	92.73	92.53	92.54	99.59

CNN comparison with different activation function

Comparison with different classifier

It is no surprise that CNN model outperforms others. According to “A Comparison Between Support Vector Machine (SVM) and Convolutional Neural Network (CNN) Models For Hyperspectral Image Classification”, it says that CNN utilizes rich spatial features at multiple scales to describe the spatial structure of the data for classifying each pixel in the image. As a result, the rich complementary spatial details obtained at different scales helps boost the performance of the proposed classification method. By contrast, SVM has difficulty extracting features, thus lowering its performance. Besides, CNN models needs lots of training data to learn well, while SVM just need small dataset. For Fashion-MNIST, it’s large enough to meet the requirement from CNN models. Therefore, CNN model performs better. As for comparison with SVM and random forest, SVM performs better than random forest. The reason may be that SVM is more suitable for Fashion-MNIST. There is no correct answer that which one is better for SVM and random forest. The key to the answer is to give it a try since it highly depends on the characteristic of the dataset. Finally, kNN performs worse than SVM, which makes no surprise. Since kNN is rather simple algorithm and more suitable to few dimensions, it cannot perform as better as SVM can when predicting Fashion-MNIST dataset, which has 784 features.

	Accuracy	Precision	Recall	F1-score	AUROC
kNN	85.54	85.78	85.54	85.46	96.85
SVM	88.29	88.24	88.29	88.24	99.05
Random forest	87.86	87.75	87.86	87.71	99.03
CNN	92.53	92.73	92.53	92.54	99.59

comparison with different classifier

Comparison with different training dataset size

In this part, I randomly abandon 20% training data to observe the performance with different amounts of training data. From the results listed below, we can easily observe that if the size of training data is smaller, performance of all the classifiers will become worse due to lack of learning.

	Accuracy with original size	Accuracy with smaller size	F1-score with original size	F1-score with smaller size
kNN	85.54	85.06	85.46	85.00
SVM	88.29	86.96	88.24	86.93
Random forest	87.86	86.24	87.71	86.13

comparison with different training dataset size

Comparison the results with and without dimensionality reduction (PCA)

Since there is limited space to put all the results of metrics in the below table, I just choose accuracy and F1-score that are more representative and easier to interpret, while other metrics' results reveal same information as accuracy and F1-score to us. It's apparent that there is no much enhancement after doing PCA on Fashion-MNIST. kNN merely have a little better after doing PCA, while SVM and random forest even perform worse after doing so. It's all because that Fashion-MNIST doesn't belong to high dimensional data. It has only 784 feature, while it has 60000 training data. In this way, do the dimensional reduction would lower model's performance. In terms of the results of kNN, it might be because that kNN is very sensitive to bad features, thus doing PCA on the dataset can get rid of some of them and enhance the performance.

	Accuracy without PCA	Accuracy with PCA	F1-score without PCA	F1-score with PCA
kNN	85.54	85.72	85.46	86.65
SVM	88.29	87.30	88.24	87.27
Random forest	87.86	86.59	87.71	86.46

comparison with and without PCA

Comparison with data augmentation

In this part, I will let the dataset randomly shift, fill points outside the input boundaries, horizontally and randomly flip the data, and do ZCA whitening for eliminating redundancy. From the below results, we can find that data augmentation does improve performance.

	Accuracy	Precision	Recall	F1-score	AUROC
Without	92.53	92.73	92.53	92.54	99.59
With	92.77	92.98	92.77	92.82	99.63

comparison with and without data augmentation

Comparison with results of paper

The referenced paper is "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". From below table, my kNN model had better performance than the paper's, while my SVM model perform worse than paper's However, I don't find how the authors implement their models, so I cannot give a reasonable explanation to such result.

	Accuracy from paper	Accuracy by my own
kNN	85.2	85.54
SVM	89.7	88.29

comparison with paper

A public non-image dataset: Mobile Price Range Prediction

This dataset comes from Kaggle (https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification). We need to predict price range of mobiles based on given 21 features. There are 1800 training data provided from Kaggle, so I use these training data as this part's dataset. Since there is no testing data, I would apply 5-fold cross validation to evaluate the model's performance.

Comparison with different hyperparameter

For kNN classifier, the results show that when k is equal to 9, the performance will be the best. All the other models with lower value of k might be too complex for this dataset to perform as well as k is 9.

	Accuracy	Precision	Recall	F1-score	AUROC
k=1	87.39	87.56	87.39	87.38	91.61
k=3	88.83	88.95	88.83	88.81	97.39
k=5	89.89	90.03	89.89	89.87	98.53
k=7	90.39	90.44	90.39	90.37	98.75
k=9	90.83	90.93	90.83	90.84	98.84

kNN comparison with different k

Speaking of SVM, it is surprising for me that the model with linear kernel outperforms others because this result contradicts the statement I find on the Internet. I have also found others having the same results that model with linear kernel performs best. The reasons why this dataset perform better when its kernel is linear one might be that this dataset is linearly separable, plus the hyperparameter of the model with Gaussian kernel having other better choice that I cannot find out.

	Accuracy	Precision	Recall	F1-score	AUROC
Linear	96.94	96.97	96.94	96.95	99.88
polynomial	96.28	96.33	96.28	96.28	99.81
RBF	95.06	95.20	95.06	95.03	99.74

SVM comparison with different kernel

As for random forest classifier, the best model appears when n_estimator is 300. The reason for this result might be that model with n_estimator being 100 is underfitting, while the one with n_estimator being 1000 is overfitting. Aa a result, model with n_estimator being 300 have the best performance among them.

	Accuracy	Precision	Recall	F1-score	AUROC
n_estimator=100	88.28	88.48	88.28	88.23	97.74
n_estimator=300	88.61	88.84	88.61	88.56	98.05
n_estimator=1000	88.17	88.38	88.17	88.11	98.14

Random forest comparison with different n_estimator

Comparison with different classifier

Evidently, SVM outperforms other classifiers. For this dataset, we can easily observe that kNN perform relatively better than last one, the reason which might because of less features in this dataset. However, the performance of linear discriminant analysis really surprises me since the performance of LDA from the paper mentioned later seems not so bad as my results. I believe that it is likely that there are so many outlier in my own splitting training dataset that LDA performs not well.

	Accuracy	Precision	Recall	F1-score	AUROC
kNN	90.83	90.93	90.83	90.84	98.84
SVM	96.94	96.97	96.94	96.95	99.88
Random forest	88.61	88.84	88.61	88.56	98.05
LDA	79.94	80.12	79.94	79.96	95.76

comparison with different classifier

Comparison with different size of training dataset

In this part, SVM and LDA perform better when the dataset is smaller. It is possible that the abandoned data might exactly the ones that point at the wrong side of the linear boundary for SVM with linear kernel. As for LDA, the most of the abandoned data might be the outlier, which will impose great threat to LDA classifier.

	Accuracy with original size	Accuracy with smaller size	F1-score with original size	F1-score with smaller size
kNN	90.83	89.44	90.84	89.47
SVM	96.94	97.01	96.95	97.01
Random forest	88.61	81.32	88.56	80.21
LDA	79.94	80.76	79.96	79.61

comparison with different training dataset size

Compare the results with and without dimensionality reduction (PCA)

As for dimensionality reduction, I try to do PCA on the dataset to get higher performance, while the performance turns out to become rather bad. It reveals that this dataset is low-dimensional data.

	Accuracy without PCA	Accuracy with PCA
kNN	90.83	78.00

comparison with and without PCA

Comparison with results of paper

I compare with the paper "Predicting the Price Range of Mobile Phones using Machine Learning Techniques" and "Prediction of Mobile Phone Price Class using Supervised Machine Learning Techniques". It is reasonable that since I split the training dataset to become new training data and testing data, the training dataset is obviously smaller than the one used by papers.

	Accuracy from paper	Accuracy by my own
SVM	96.94	95.00
Random forest	88.61	83.00
LDA	95.00	79.94

comparison with paper

A self-made dataset: BBC News article

In terms of self-made dataset, I use Selenium and BeautifulSoup to collect the article from BBC News. The criteria for data inclusion is the articles from the 4 categories on BBC News, which are business, technology, science environment, and entertainment arts respectively. Moreover, each category has 50 articles from BBC News. As for method of labeling data, I will assign the same category's article into the same file and label each data when I read the articles from the files and give the them with the same label if they are in same file. Besides, I collect this dataset with my classmate 楊竺耘. Since we do not specify splitting, I will do cross-validation to our dataset. I aim to use 3 classifiers to predict the category of the certain article from this self-made dataset.

Comparison with different hyperparameter

For our own dataset, kNN classifier with k being 9 have the best performance among other classifier. The reason for this results might be the same with the last dataset that other model might be too complex.

	Accuracy	Precision	Recall	F1-score	AUROC
k=1	88.50	89.70	88.50	88.51	92.79
k=3	88.50	89.48	88.50	88.38	97.33
k=5	86.50	88.33	86.50	86.49	97.14
k=7	86.50	88.42	86.50	86.45	97.22
k=9	88.50	90.35	88.50	88.67	97.32

kNN comparison with different k

As for SVM, SVM with linear kernel outperforms other models. It still surprises me that models with Gaussian kernel do not perform the best. I think that it is likely that the I do not find out the best hyperparameter of models with rbf kernel and the dataset is linearly separable.

	Accuracy	Precision	Recall	F1-score	AUROC
Linear	92.00	93.07	92.00	91.97	98.62
polynomial	60.50	85.24	60.50	60.50	97.89
RBF	86.00	89.69	86.00	85.84	98.45

SVM comparison with different kernel

For random forest, random forest with n_estimator being 1000 outperforms other models. The larger the amount of the tress is, the better the classifier performs. As a result, the classifier with most trees has the best performance.

	Accuracy	Precision	Recall	F1-score	AUROC
n_estimator=100	84.00	87.66	84.00	83.97	96.37
n_estimator=300	87.00	89.81	87.00	86.98	97.65
n_estimator=1000	89.50	91.09	89.50	89.45	97.87

Random forest comparison with different n_estimator

Comparison with different classifier

It is no surprise that SVM outperform others from the results presented below. SVM can handle more complex dataset than kNN, so it performs better in out self-made dataset which needs predict the topic of the article.

	Accuracy	Precision	Recall	F1-score	AUROC
kNN	88.50	90.35	88.50	88.67	97.32
SVM	92.00	93.07	92.00	91.97	98.62
Random forest	89.50	91.09	89.50	89.45	97.87

comparison with different classifier

Comparison with different training dataset size

In this part, all the classifiers perform better when the training dataset is larger. It makes sense that when the training dataset is larger, they have more learning opportunity, thus performing better.

	Accuracy with original size	Accuracy with smaller size	F1-score with original size	F1-score with smaller size
kNN	88.50	84.38	88.67	84.98
SVM	92.00	89.38	91.97	89.44
Random forest	89.50	76.88	89.45	77.66

comparison with different training dataset size

Discussion

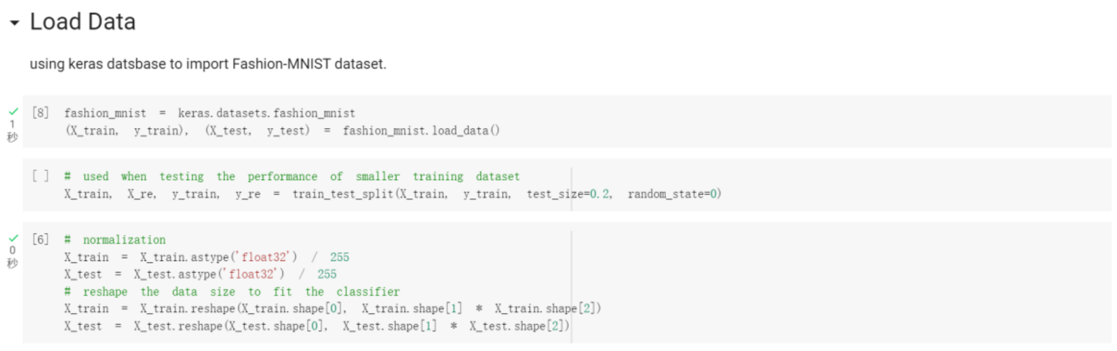
1. **Based on your experiments, are the results and observed behaviors what you expect?**
   Most of the results actually meet my expectation, while some results surprise me a lot and cannot really figure out the reason. For example, SVM with linear kernel outperforms the one with rbf kernel when the datasets are non-image dataset and self-made one. It contradicts what I learn from the Internet. As a result, I suggest that there are two reasons. On the one hand, the hyperparameters of the model with rbf kernel are not the best one. On the other hand, the both dataset is linearly separable which is suitable to use model with linear one.
2. **Discuss factors that affect the performance, including dataset characteristics.**
   To begin with, classifier plays important part. From the presented above results, we can find that classifier will perform better when predicting some datasets, while performing worse when predicting others. Secondly, hyperparameter also has a great effect on the performance. For example, when predicting Fashion-MNIST, the performance of SVM with rbf kernel is 88.29%, while the performance of SVM with polynomial kernel is 77.13%. Obviously, hyperparameter can affect the performance a lot. In addition, the size of training dataset will affect the performance as well. In some cases, the smaller size of training dataset will lower the performance because the model does not have enough data. In other cases, the smaller one will perform better since some noises are eliminated. Last but not least, the characteristic of dataset is essential. For example, Fashion-MNIST is high-dimensional data, so it is useful to do PCA on the dataset so that models can have better performance. By contrast, it will let models perform worse when doing PCA on mobile price range dataset because the number of features in this dataset is not as much as Fashion-MNIST. Therefore, characteristic of dataset also has great effect on the performance.
3. **Describe experiments that you would do if there were more time available.**
   There are two things I will do if I have more time available. On the one hand, I will

continue to fine-tune the hyperparameters of SVM with rbf kernel to get the better performance when predicting the second and third datasets. Since I really want to know whether the knowledge that linear kernel will not perform better than rbf kernel is correct, or SVM with rbf kernel will be so complex for some dataset that it is overfitting. On the other hand, I want to design a 1D CNN for the second dataset if I have more time available. Hence, I can have more precise predicting results. Hopefully, I can have spare time to finish these two things.

4. **Indicate what you have learned from the experiments and remaining questions.**
   In this project, I do learn a lot. First of all, I learn the importance of fine-tuning hyperparameters. In the past, I pay more attention to design classifier by my own. Performance is not the topic issue I was concerned with. Furthermore, I have deeper understanding of the classifier I used in this project. Since I need to have explanation to the results, I had to search the answer on the Internet, thus having deeper understanding to the classifier. Moreover, I learn to infer the characteristic of dataset by analyzing the results. Through the performance of different classifiers and different hyperparameters, I can try to have analysis on the dataset. As for remaining questions, it must be the reasons why SVM with linear kernel will outperform the one with rbf kernel.

Appendix

1. Fashion-MNIST



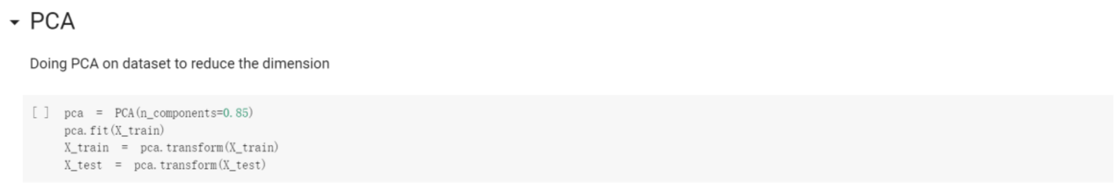
Load Data

using keras datsbase to import Fashion-MNIST dataset.

```
[8] fashion_mnist = keras.datasets.fashion_mnist
    (X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

```
[ ] # used when testing the performance of smaller training dataset
    X_train, X_re, y_train, y_re = train_test_split(X_train, y_train, test_size=0.2, random_state=0)
```

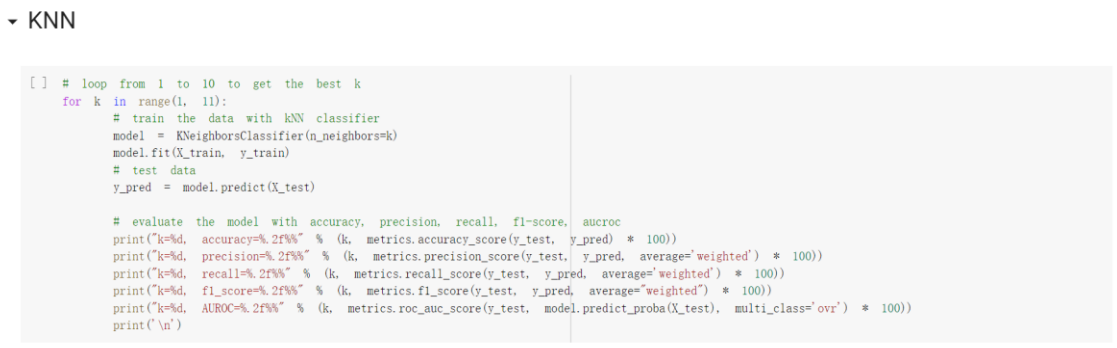
```
[6] # normalization
    X_train = X_train.astype('float32') / 255
    X_test = X_test.astype('float32') / 255
    # reshape the data size to fit the classifier
    X_train = X_train.reshape(X_train.shape[0], X_train.shape[1] * X_train.shape[2])
    X_test = X_test.reshape(X_test.shape[0], X_test.shape[1] * X_test.shape[2])
```



PCA

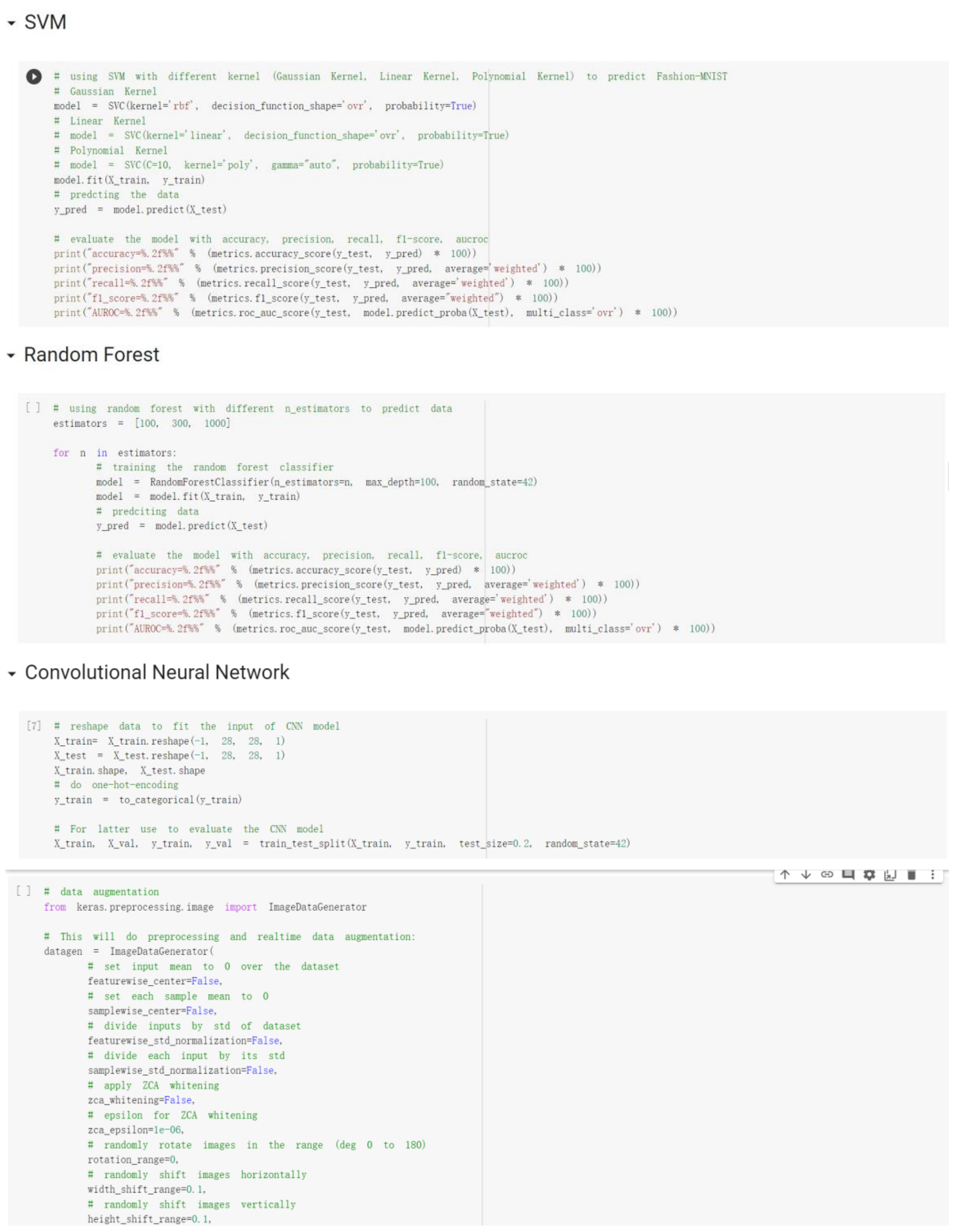
Doing PCA on dataset to reduce the dimension

```
[ ] pca = PCA(n_components=0.85)
    pca.fit(X_train)
    X_train = pca.transform(X_train)
    X_test = pca.transform(X_test)
```



KNN

```
[ ] # loop from 1 to 10 to get the best k
    for k in range(1, 11):
        # train the data with kNN classifier
        model = KNeighborsClassifier(n_neighbors=k)
        model.fit(X_train, y_train)
        # test data
        y_pred = model.predict(X_test)

        # evaluate the model with accuracy, precision, recall, f1-score, aucroc
        print("k=%d, accuracy=%.2f%%" % (k, metrics.accuracy_score(y_test, y_pred) * 100))
        print("k=%d, precision=%.2f%%" % (k, metrics.precision_score(y_test, y_pred, average='weighted') * 100))
        print("k=%d, recall=%.2f%%" % (k, metrics.recall_score(y_test, y_pred, average='weighted') * 100))
        print("k=%d, f1_score=%.2f%%" % (k, metrics.f1_score(y_test, y_pred, average="weighted") * 100))
        print("k=%d, AUROC=%.2f%%" % (k, metrics.roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr') * 100))
        print('\n')
```



SVM

```
# using SVM with different kernel (Gaussian Kernel, Linear Kernel, Polynomial Kernel) to predict Fashion-MNIST
# Gaussian Kernel
model = SVC(kernel='rbf', decision_function_shape='ovr', probability=True)
# Linear Kernel
# model = SVC(kernel='linear', decision_function_shape='ovr', probability=True)
# Polynomial Kernel
# model = SVC(C=10, kernel='poly', gamma="auto", probability=True)
model.fit(X_train, y_train)
# predcting the data
y_pred = model.predict(X_test)

# evaluate the model with accuracy, precision, recall, f1-score, aucroc
print("accuracy=%.2f%%" % (metrics.accuracy_score(y_test, y_pred) * 100))
print("precision=%.2f%%" % (metrics.precision_score(y_test, y_pred, average='weighted') * 100))
print("recall=%.2f%%" % (metrics.recall_score(y_test, y_pred, average='weighted') * 100))
print("f1_score=%.2f%%" % (metrics.f1_score(y_test, y_pred, average="weighted") * 100))
print("AUROC=%.2f%%" % (metrics.roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr') * 100))
```

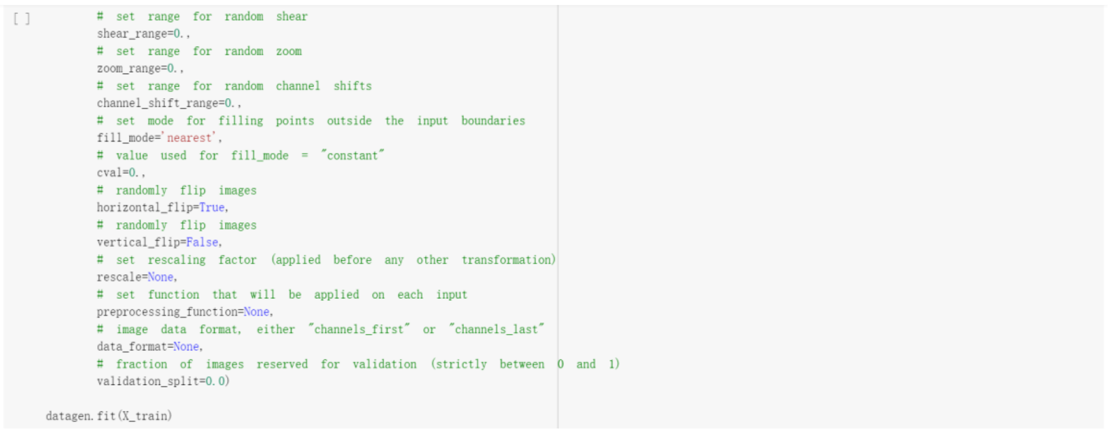
Random Forest

```
[ ] # using random forest with different n_estimators to predict data
    estimators = [100, 300, 1000]

    for n in estimators:
        # training the random forest classifier
        model = RandomForestClassifier(n_estimators=n, max_depth=100, random_state=42)
        model = model.fit(X_train, y_train)
        # predciting data
        y_pred = model.predict(X_test)

        # evaluate the model with accuracy, precision, recall, f1-score, aucroc
        print("accuracy=%.2f%%" % (metrics.accuracy_score(y_test, y_pred) * 100))
        print("precision=%.2f%%" % (metrics.precision_score(y_test, y_pred, average='weighted') * 100))
        print("recall=%.2f%%" % (metrics.recall_score(y_test, y_pred, average='weighted') * 100))
        print("f1_score=%.2f%%" % (metrics.f1_score(y_test, y_pred, average="weighted") * 100))
        print("AUROC=%.2f%%" % (metrics.roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr') * 100))
```

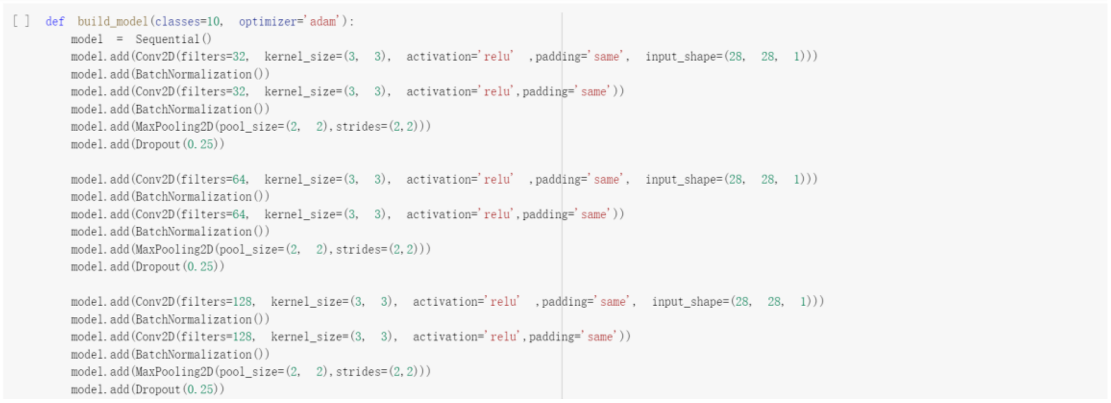
Convolutional Neural Network

```
[7] # reshape data to fit the input of CNN model
    X_train= X_train.reshape(-1, 28, 28, 1)
    X_test = X_test.reshape(-1, 28, 28, 1)
    X_train.shape, X_test.shape
    # do one-hot-encoding
    y_train = to_categorical(y_train)

    # For latter use to evaluate the CNN model
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

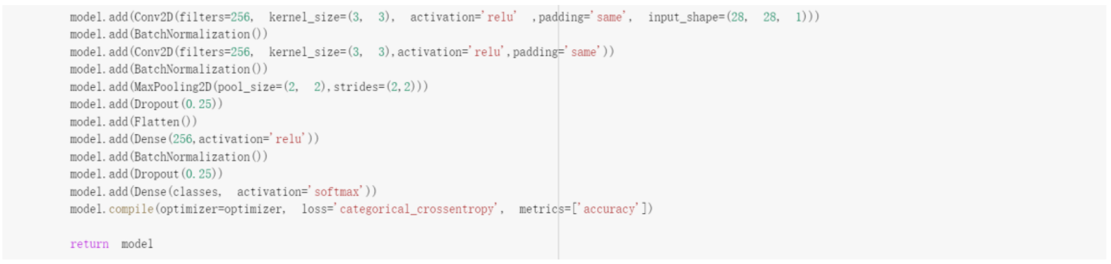
```
[ ] # data augmentation
    from keras.preprocessing.image import ImageDataGenerator

    # This will do preprocessing and realtime data augmentation:
    datagen = ImageDataGenerator(
            # set input mean to 0 over the dataset
            featurewise_center=False,
            # set each sample mean to 0
            samplewise_center=False,
            # divide inputs by std of dataset
            featurewise_std_normalization=False,
            # divide each input by its std
            samplewise_std_normalization=False,
            # apply ZCA whitening
            zca_whitening=False,
            # epsilon for ZCA whitening
            zca_epsilon=1e-06,
            # randomly rotate images in the range (deg 0 to 180)
            rotation_range=0,
            # randomly shift images horizontally
            width_shift_range=0.1,
            # randomly shift images vertically
            height_shift_range=0.1,
```



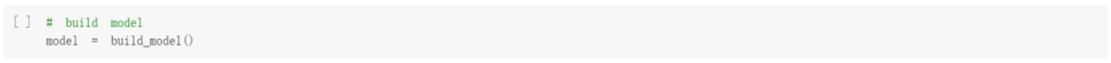
```
[ ]         # set range for random shear
            shear_range=0.,
            # set range for random zoom
            zoom_range=0.,
            # set range for random channel shifts
            channel_shift_range=0.,
            # set mode for filling points outside the input boundaries
            fill_mode='nearest',
            # value used for fill_mode = "constant"
            cval=0.,
            # randomly flip images
            horizontal_flip=True,
            # randomly flip images
            vertical_flip=False,
            # set rescaling factor (applied before any other transformation)
            rescale=None,
            # set function that will be applied on each input
            preprocessing_function=None,
            # image data format, either "channels_first" or "channels_last"
            data_format=None,
            # fraction of images reserved for validation (strictly between 0 and 1)
            validation_split=0.0)

    datagen.fit(X_train)
```



```
[ ] def build_model(classes=10, optimizer='adam'):
        model = Sequential()
        model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu' ,padding='same', input_shape=(28, 28, 1)))
        model.add(BatchNormalization())
        model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu',padding='same'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu' ,padding='same', input_shape=(28, 28, 1)))
        model.add(BatchNormalization())
        model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu',padding='same'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu' ,padding='same', input_shape=(28, 28, 1)))
        model.add(BatchNormalization())
        model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu',padding='same'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
        model.add(Dropout(0.25))
```

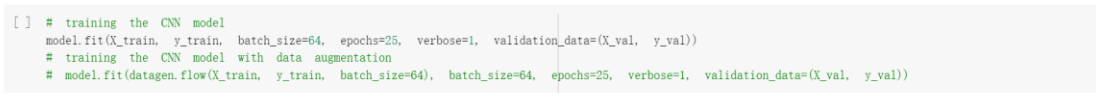


```
        model.add(Conv2D(filters=256, kernel_size=(3, 3), activation='relu' ,padding='same', input_shape=(28, 28, 1)))
        model.add(BatchNormalization())
        model.add(Conv2D(filters=256, kernel_size=(3, 3),activation='relu',padding='same'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2, 2),strides=(2,2)))
        model.add(Dropout(0.25))
        model.add(Flatten())
        model.add(Dense(256,activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(0.25))
        model.add(Dense(classes, activation='softmax'))
        model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

        return model
```

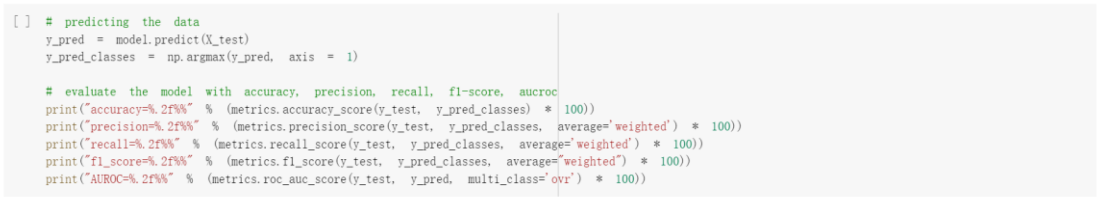


```
[ ] # build model
    model = build_model()
```

▾ Training / Testing



```
[ ] # training the CNN model
    model.fit(X_train, y_train, batch_size=64, epochs=25, verbose=1, validation_data=(X_val, y_val))
    # training the CNN model with data augmentation
    # model.fit(datagen.flow(X_train, y_train, batch_size=64), batch_size=64, epochs=25, verbose=1, validation_data=(X_val, y_val))
```



```
[ ] # predicting the data
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis = 1)

    # evaluate the model with accuracy, precision, recall, f1-score, aucroc
    print("accuracy=%.2f%%" % (metrics.accuracy_score(y_test, y_pred_classes) * 100))
    print("precision=%.2f%%" % (metrics.precision_score(y_test, y_pred_classes, average='weighted') * 100))
    print("recall=%.2f%%" % (metrics.recall_score(y_test, y_pred_classes, average='weighted') * 100))
    print("f1_score=%.2f%%" % (metrics.f1_score(y_test, y_pred_classes, average="weighted") * 100))
    print("AUROC=%.2f%%" % (metrics.roc_auc_score(y_test, y_pred, multi_class='ovr') * 100))
```

2. Mobile Price Range Prediction

Load Data

```
# load data from .csv file
train_df = pd.read_csv('train_data.csv')
train_df.head()

# collect features without price range
features = list(train_df.columns)[0:-1]
X, y = train_df[features], train_df['price_range']
# put data into numpy array
X, y = X.values, y.values

# used when testing the performance of smaller training dataset
# X, X_re, y, y_re = train_test_split(X, y, test_size=0.2, random_state=0)
```

PCA

Doing PCA on dataset to reduce the dimension

```
pca = PCA(n_components=0.85)
pca.fit(X)
X = pca.transform(X)
```

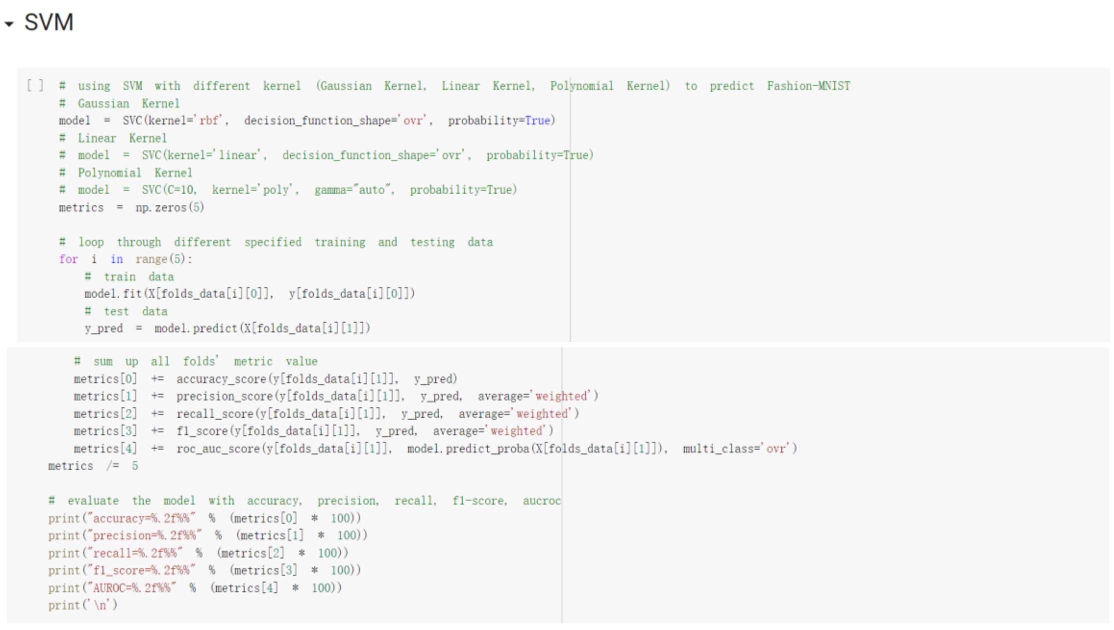
Cross-validation

```
def cross_validation(x_train, y_train, k=5):
    returnList = list()
    folds = list()
    # generate numpy array filled with value from 0 to length of x_train for latter use as index
    random_idx = np.arange(len(x_train))
    seed = 120
    np.random.seed(seed)
    # shuffle the index
    np.random.shuffle(random_idx)
    # get the size of each fold
    n_split = len(x_train) // k

    # separate the index into training part and testing part
    keep = 0
    for i in range(k):
        if i < len(x_train) % k: # used when mode != 0
            folds.append(random_idx[keep : keep + n_split + 1])
            keep += (n_split + 1)
        else:
            folds.append(random_idx[keep : keep + n_split])
            keep += n_split

    for i in range(k):
        returnList.append([np.setdiff1d(random_idx, folds[i]), folds[i]])

    return returnList
```

```
folds_data = cross_validation(X, y, k=5)
```

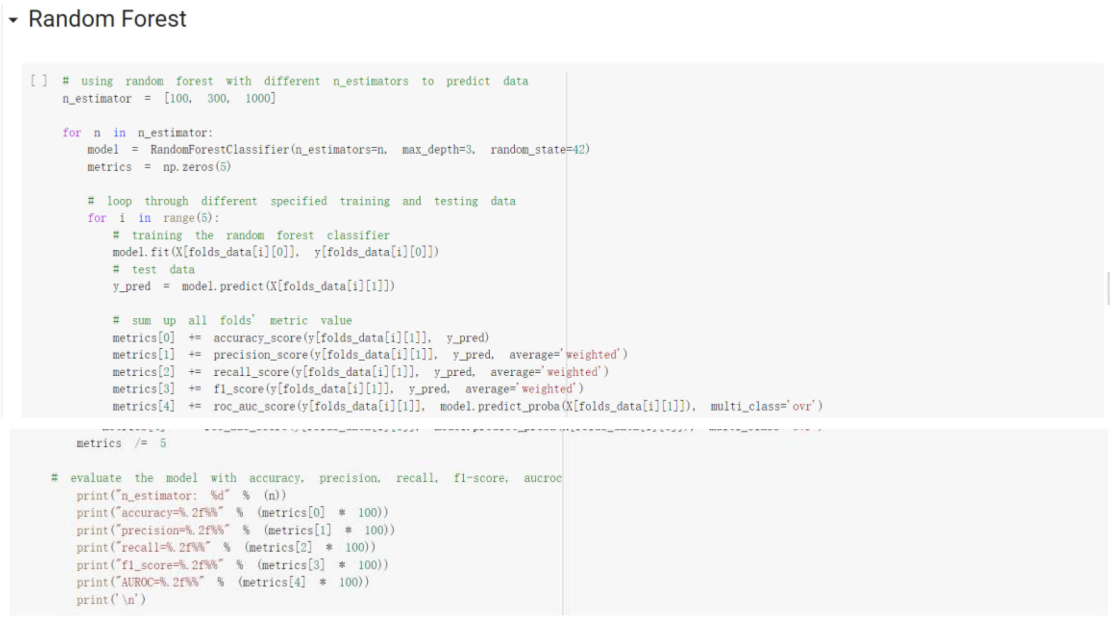
kNN

```
# loop from 1 to 10 to get the best k
for k in range(1, 11):
    model = KNeighborsClassifier(n_neighbors=k)
    metric1, metric2, metric3, metric4, metric5 = 0, 0, 0, 0, 0

    # loop through different specified training and testing data
    for i in range(5):
        # train the data with kNN classifier
        model.fit(X[folds_data[i][0]], y[folds_data[i][0]])
        # test data
        y_pred = model.predict(X[folds_data[i][1]])

        # sum up all folds' metric value
        metric1 += accuracy_score(y[folds_data[i][1]], y_pred)
        metric2 += precision_score(y[folds_data[i][1]], y_pred, average='weighted')
        metric3 += recall_score(y[folds_data[i][1]], y_pred, average='weighted')
        metric4 += f1_score(y[folds_data[i][1]], y_pred, average='weighted')
        metric5 += roc_auc_score(y[folds_data[i][1]], model.predict_proba(X[folds_data[i][1]]), multi_class='ovr')

    # evaluate the model with accuracy, precision, recall, f1-score, aucroc
    print("k=%d, accuracy=%.2f%%" % (k, metric1 * 20)) # (100% / 5 folds = 20)
    print("k=%d, precision=%.2f%%" % (k, metric2 * 20))
    print("k=%d, recall=%.2f%%" % (k, metric3 * 20))
    print("k=%d, f1_score=%.2f%%" % (k, metric4 * 20))
    print("k=%d, AUROC=%.2f%%" % (k, metric5 * 20))
    print('\n')
```



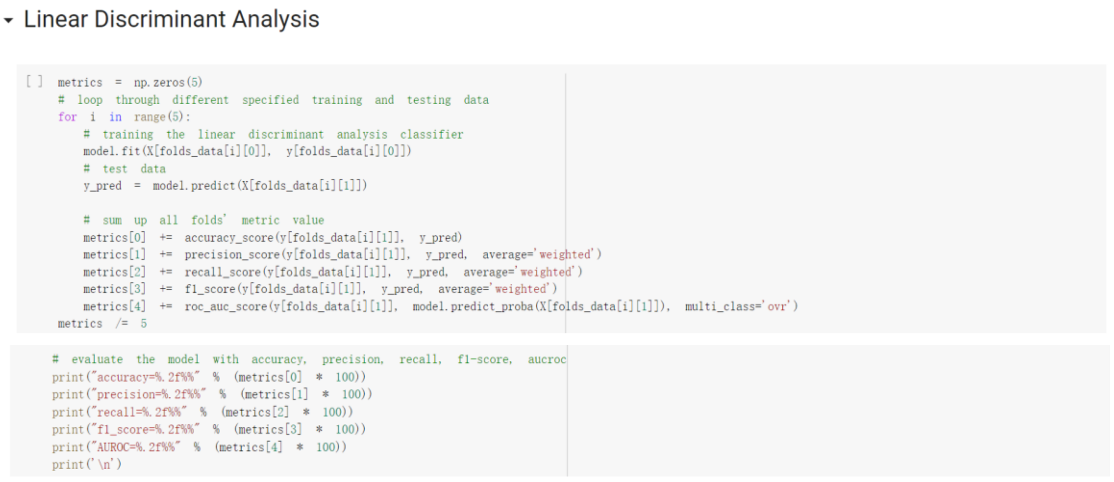
SVM

```
[ ] # using SVM with different kernel (Gaussian Kernel, Linear Kernel, Polynomial Kernel) to predict Fashion-MNIST
    # Gaussian Kernel
    model = SVC(kernel='rbf', decision_function_shape='ovr', probability=True)
    # Linear Kernel
    # model = SVC(kernel='linear', decision_function_shape='ovr', probability=True)
    # Polynomial Kernel
    # model = SVC(C=10, kernel='poly', gamma="auto", probability=True)
    metrics = np.zeros(5)

    # loop through different specified training and testing data
    for i in range(5):
        # train data
        model.fit(X[folds_data[i][0]], y[folds_data[i][0]])
        # test data
        y_pred = model.predict(X[folds_data[i][1]])

        # sum up all folds' metric value
        metrics[0] += accuracy_score(y[folds_data[i][1]], y_pred)
        metrics[1] += precision_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[2] += recall_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[3] += f1_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[4] += roc_auc_score(y[folds_data[i][1]], model.predict_proba(X[folds_data[i][1]]), multi_class='ovr')
    metrics /= 5

    # evaluate the model with accuracy, precision, recall, f1-score, aucroc
    print("accuracy=%.2f%%" % (metrics[0] * 100))
    print("precision=%.2f%%" % (metrics[1] * 100))
    print("recall=%.2f%%" % (metrics[2] * 100))
    print("f1_score=%.2f%%" % (metrics[3] * 100))
    print("AUROC=%.2f%%" % (metrics[4] * 100))
    print('\n')
```



Random Forest

```
[ ] # using random forest with different n_estimators to predict data
    n_estimator = [100, 300, 1000]

    for n in n_estimator:
        model = RandomForestClassifier(n_estimators=n, max_depth=3, random_state=42)
        metrics = np.zeros(5)

        # loop through different specified training and testing data
        for i in range(5):
            # training the random forest classifier
            model.fit(X[folds_data[i][0]], y[folds_data[i][0]])
            # test data
            y_pred = model.predict(X[folds_data[i][1]])

            # sum up all folds' metric value
            metrics[0] += accuracy_score(y[folds_data[i][1]], y_pred)
            metrics[1] += precision_score(y[folds_data[i][1]], y_pred, average='weighted')
            metrics[2] += recall_score(y[folds_data[i][1]], y_pred, average='weighted')
            metrics[3] += f1_score(y[folds_data[i][1]], y_pred, average='weighted')
            metrics[4] += roc_auc_score(y[folds_data[i][1]], model.predict_proba(X[folds_data[i][1]]), multi_class='ovr')
        metrics /= 5

    # evaluate the model with accuracy, precision, recall, f1-score, aucroc
        print("n_estimator: %d" % (n))
        print("accuracy=%.2f%%" % (metrics[0] * 100))
        print("precision=%.2f%%" % (metrics[1] * 100))
        print("recall=%.2f%%" % (metrics[2] * 100))
        print("f1_score=%.2f%%" % (metrics[3] * 100))
        print("AUROC=%.2f%%" % (metrics[4] * 100))
        print('\n')
```



Linear Discriminant Analysis

```
[ ] metrics = np.zeros(5)
    # loop through different specified training and testing data
    for i in range(5):
        # training the linear discriminant analysis classifier
        model.fit(X[folds_data[i][0]], y[folds_data[i][0]])
        # test data
        y_pred = model.predict(X[folds_data[i][1]])

        # sum up all folds' metric value
        metrics[0] += accuracy_score(y[folds_data[i][1]], y_pred)
        metrics[1] += precision_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[2] += recall_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[3] += f1_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[4] += roc_auc_score(y[folds_data[i][1]], model.predict_proba(X[folds_data[i][1]]), multi_class='ovr')
    metrics /= 5

    # evaluate the model with accuracy, precision, recall, f1-score, aucroc
    print("accuracy=%.2f%%" % (metrics[0] * 100))
    print("precision=%.2f%%" % (metrics[1] * 100))
    print("recall=%.2f%%" % (metrics[2] * 100))
    print("f1_score=%.2f%%" % (metrics[3] * 100))
    print("AUROC=%.2f%%" % (metrics[4] * 100))
    print('\n')
```

3. BBC News Article

Load Data

```
[ ] # different categories collected from BBC News
    cats = ['business', 'technology', 'science-environment', 'entertainment-arts']
    X = []
    y = []

    # Convert a collection of raw documents to a matrix of TF-IDF features.
    vectorizer = TfidfVectorizer(stop_words='english')
    # load data from files
    for idx, cat in enumerate(cats):
        cat_dir = f'{cat}'
        for file_name in os.listdir(cat_dir):
            with open(f'{cat_dir}/{file_name}') as file:
                article = file.read()
                X.append(article)
                y.append(idx)

    X = vectorizer.fit_transform(X).toarray()
    y = np.array(y)
    # used when testing the performance of smaller training dataset
    X, X_re, y, y_re = train_test_split(X, y, test_size=0.2, random_state=0)
```

Cross-validation

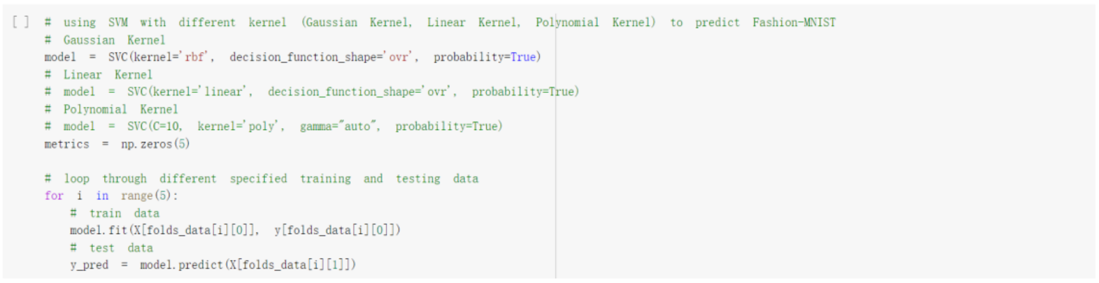
```
[ ] def cross_validation(x_train, y_train, k=5):
        returnList = list()
        folds = list()
        # generate numpy array filled with value from 0 to length of x_train for latter use as index
        random_idx = np.arange(len(x_train))
        seed = 120
        np.random.seed(seed)
        # shuffle the index
        np.random.shuffle(random_idx)
        # get the size of each fold
        n_split = len(x_train) // k

        # separate the index into training part and testing part
        keep = 0
        for i in range(k):
            if i < len(x_train) % k: # used when mode != 0
                folds.append(random_idx[keep : keep + n_split + 1])
                keep += (n_split + 1)
            else:
                folds.append(random_idx[keep : keep + n_split])
                keep += n_split

        for i in range(k):
            returnList.append([np.setdiff1d(random_idx, folds[i]), folds[i]])

        return returnList
```

```
folds_data = cross_validation(X, y, k=5)
```

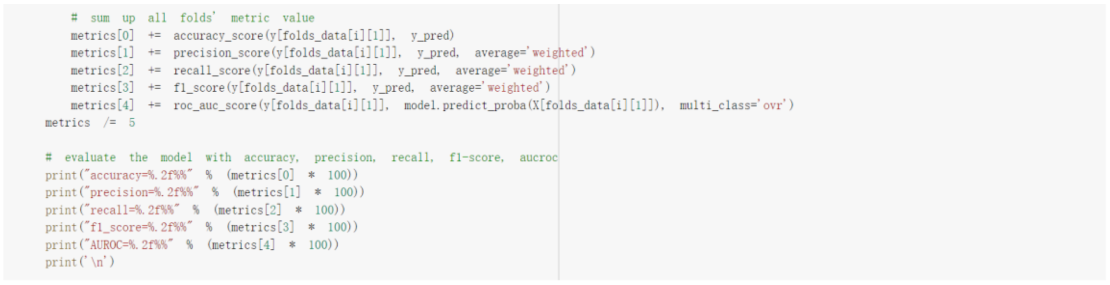
kNN

```
[ ] # loop from 1 to 10 to get the best k
    for k in range(1, 11):
        model = KNeighborsClassifier(n_neighbors=k)
        metric1, metric2, metric3, metric4, metric5 = 0, 0, 0, 0, 0

        # loop through different specified training and testing data
        for i in range(5):
            # train the data with kNN classifier
            model.fit(X[folds_data[i][0]], y[folds_data[i][0]])
            # test data
            y_pred = model.predict(X[folds_data[i][1]])

            # sum up all folds' metric value
            metric1 += accuracy_score(y[folds_data[i][1]], y_pred)
            metric2 += precision_score(y[folds_data[i][1]], y_pred, average='weighted')
            metric3 += recall_score(y[folds_data[i][1]], y_pred, average='weighted')
            metric4 += f1_score(y[folds_data[i][1]], y_pred, average='weighted')
            metric5 += roc_auc_score(y[folds_data[i][1]], model.predict_proba(X[folds_data[i][1]]), multi_class='ovr')

        # evaluate the model with accuracy, precision, recall, f1-score, aucroc
        print("k=%d, accuracy=%.2f%%" % (k, metric1 * 20)) # (100% / 5 folds = 20)
        print("k=%d, precision=%.2f%%" % (k, metric2 * 20))
        print("k=%d, recall=%.2f%%" % (k, metric3 * 20))
        print("k=%d, f1_score=%.2f%%" % (k, metric4 * 20))
        print("k=%d, AUROC=%.2f%%" % (k, metric5 * 20))
        print('\n')
```

SVM

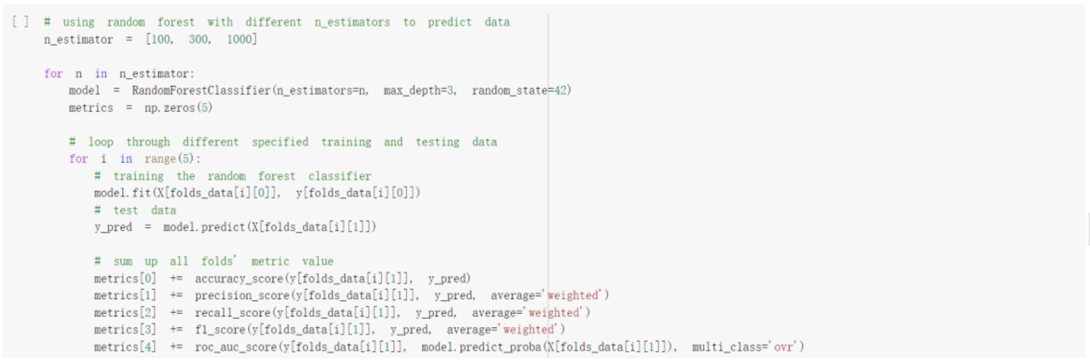


```
[ ] # using SVM with different kernel (Gaussian Kernel, Linear Kernel, Polynomial Kernel) to predict Fashion-MNIST
    # Gaussian Kernel
    model = SVC(kernel='rbf', decision_function_shape='ovr', probability=True)
    # Linear Kernel
    # model = SVC(kernel='linear', decision_function_shape='ovr', probability=True)
    # Polynomial Kernel
    # model = SVC(C=10, kernel='poly', gamma="auto", probability=True)
    metrics = np.zeros(5)

    # loop through different specified training and testing data
    for i in range(5):
        # train data
        model.fit(X[folds_data[i][0]], y[folds_data[i][0]])
        # test data
        y_pred = model.predict(X[folds_data[i][1]])
```

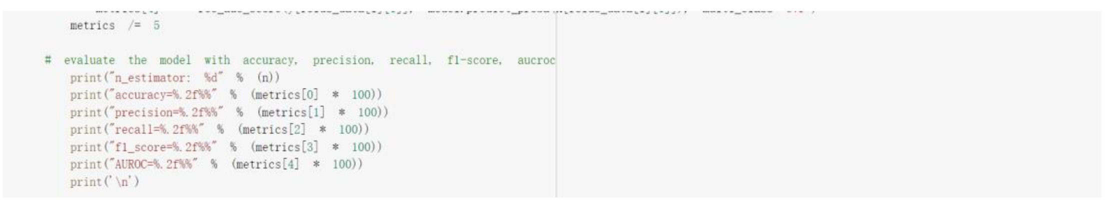


```
        # sum up all folds' metric value
        metrics[0] += accuracy_score(y[folds_data[i][1]], y_pred)
        metrics[1] += precision_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[2] += recall_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[3] += f1_score(y[folds_data[i][1]], y_pred, average='weighted')
        metrics[4] += roc_auc_score(y[folds_data[i][1]], model.predict_proba(X[folds_data[i][1]]), multi_class='ovr')
    metrics /= 5

    # evaluate the model with accuracy, precision, recall, f1-score, aucroc
    print("accuracy=%.2f%%" % (metrics[0] * 100))
    print("precision=%.2f%%" % (metrics[1] * 100))
    print("recall=%.2f%%" % (metrics[2] * 100))
    print("f1_score=%.2f%%" % (metrics[3] * 100))
    print("AUROC=%.2f%%" % (metrics[4] * 100))
    print('\n')
```

Random Forest



```
[ ] # using random forest with different n_estimators to predict data
    n_estimator = [100, 300, 1000]

    for n in n_estimator:
        model = RandomForestClassifier(n_estimators=n, max_depth=3, random_state=42)
        metrics = np.zeros(5)

        # loop through different specified training and testing data
        for i in range(5):
            # training the random forest classifier
            model.fit(X[folds_data[i][0]], y[folds_data[i][0]])
            # test data
            y_pred = model.predict(X[folds_data[i][1]])

            # sum up all folds' metric value
            metrics[0] += accuracy_score(y[folds_data[i][1]], y_pred)
            metrics[1] += precision_score(y[folds_data[i][1]], y_pred, average='weighted')
            metrics[2] += recall_score(y[folds_data[i][1]], y_pred, average='weighted')
            metrics[3] += f1_score(y[folds_data[i][1]], y_pred, average='weighted')
            metrics[4] += roc_auc_score(y[folds_data[i][1]], model.predict_proba(X[folds_data[i][1]]), multi_class='ovr')
```



```
        metrics /= 5

    # evaluate the model with accuracy, precision, recall, f1-score, aucroc
        print("n_estimator: %d" % (n))
        print("accuracy=%.2f%%" % (metrics[0] * 100))
        print("precision=%.2f%%" % (metrics[1] * 100))
        print("recall=%.2f%%" % (metrics[2] * 100))
        print("f1_score=%.2f%%" % (metrics[3] * 100))
        print("AUROC=%.2f%%" % (metrics[4] * 100))
        print('\n')
```