MachineLearning Bonus Assignment

Group 66

Names：Yongcheng Huang

Xingyu Han

Student Number：5343755（Xingyu Han），5560950（Yongcheng Huang）

2.1

GaussianNB

Pro: Not sensitive to the problem of missing some data.

Con: Based on the independence of sample, will perform terrible when there's internal relationship.

DecisionTreeClassifier

Pro: DTC can deal with data that not be preprocessed, while other methods need to be preprocessed.

Con: When there's too many different classes, the error rate will increase fast.

KNeighborsClassifier

Pro: The time complexity of training a KNN model is lower than other models like SVM.

Con: Need lots of calculation when there's too much features.

SVC

Pro: The change of not support vectors will not change the model.

Con: Can only deal with classification when the result only has two classes.

SGDClassifier

Pro: The result given is a possibility that between 0 and 1.

Con: Can't deal with not linear problem since the hyperplane is linear like.

2.2

(1) **GaussianNB** : there is no hyper parameter in GaussianNB.

(2) **DecisionTreeClassifier**:

For max-depth, it controls the maximum depth of the tree, if it is not specified, the tree will keeps expanding until the nodes are pure or less than min_samples_split samples according to another parameter which may take longer time and calculation.

For min-sample-leaf: it controls the minimum number of samples required to be at a leaf node and it may give smoothing effect to the model according to the value.

For random_state: it controls the randomness of the estimator which can fix a deterministic behavior during fitting.

(3) **KNeighborsClassifier**:

For n_neighbors, it controls the number of neighbors to use, the higher number, the more near samples need to be considered.

For weights, it controls which weight function to use in prediction which means different weight function will assign the near samples with different weights, thus can change the prediction of the model.

(4) **SVC**:

For C, it controls the regularization as the strength of the regularization is inversely proportional to C. The value should be positive.

For kernel, it specifies the kernel type to be used and different kernel calculation will give different results.

For degree, it controls the degree of the polynomial kernel function and is ignored by all other kernels. Thus, this parameter depends on the kernel parameter.

For gamma, it is the kernel coefficient for kernel type 'rbf', 'poly' and 'sigmoid'. Different kernel type should have different gamma calculation method. Thus, it also depends on the kernel parameter.

For random_state, it controls the pseudo random number generation for shuffling the data for probability estimates which is helpful when generate reproducible outputs.

(5) **SGDClassifier**:

For loss, it controls which loss function to be used. Different loss function will estimate and update the model in a different way.

For alpha, it's the constant that multiplies the regularization term. Higher value will bring stronger regularization. It can also be used to calculate the learning rate. Thus, it partly depends on the learning_rate parameter.

For learning_rate, it controls the learning rate schedule and it decides how to use parameter eta0. Thus, it depends on the eta0 parameter.

For eta0, it controls the initial learning rate for the schedule.

For penalty, it controls the penalty to be used and may introduce sparsity to the model.

For random_state, it is used for shuffling the data based on the state of 'shuffle' parameter.

3 US Census

3.1 DATA EXPLORTATION

3.1.1

From the plot below, the dataset contains 11 features and most of them are String values. The target variable is binary values with 0s and 1s. As we want to estimate whether the income is above $50000 based on these features, we can take the accuracy as the performance metric. Using accuracy can give us a direct and clear understanding of the result and estimation of the model. We can easily adjust the model based on the accuracy.

```
RangeIndex: 16280 entries, 0 to 16279
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             16280 non-null  int64
 1   education-num   16040 non-null  float64
 2   hours-per-week  16280 non-null  int64
 3   workclass       15344 non-null  object
 4   education       16280 non-null  object
 5   marital-status  16280 non-null  object
 6   occupation      15099 non-null  object
 7   relationship    16280 non-null  object
 8   race            16280 non-null  object
 9   sex             16280 non-null  object
 10  native-country  15980 non-null  object
 11  salary          16280 non-null  int64
```

3.1.2

As the goal is to predict whether the income is above 50000, the accuracy would be an important baseline. Thus, I set my simplest baseline as the accuracy of 80%.

3.1.3

The race and sex features should be considered in our model. These features reflect the

influence of the real society and construct implicit structures in the dataset. If we throw these features, the implicit structures can not be correctly recognized by our model and will cause great bias on the result.

3.2 DATA PREPARATION

3.2.1

From the plot in 3.1.1, we can see that the entries of each features have different numbers which means the feature contains missing values if its number is below 16280 according to the target variable 'salary'. As our model can not handle missing values, I throw all entries that contains missing values in order to avoid their negative effect on the model training. After clearing, the dataset now is shown below:

```
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             14707 non-null  int64
 1   educationnum    14707 non-null  float64
 2   hoursperweek    14707 non-null  int64
 3   workclass       14707 non-null  object
 4   education       14707 non-null  object
 5   maritalstatus   14707 non-null  object
 6   occupation      14707 non-null  object
 7   relationship    14707 non-null  object
 8   race            14707 non-null  object
 9   sex             14707 non-null  object
 10  nativecountry   14707 non-null  object
 11  salary          14707 non-null  int64
```

3.2.2

From the plot in 3.2.1, we can see that the types of most features are 'object' or 'String'. In order to transform them into numerical format, we first collect the unique values of each feature and assign them a unique numerical value according to the number of the unique values. Then we can make sure that each unique value won't affect the others and it will be much easier to handle the data.

3.2.3

For GaussianNB, the scale is important as it will be greatly affected by the mean and variance of the dataset.
For DecisionTree, the scale is not so important as the model can easily handle discrete data and won't cause too much bias.
For K-NN, the scale is important as the distribution of samples will have great influence on the predictions.
For SVM, the scale is important as the choice of support vectors is influenced and thus the model will be changed.
For SGD, the scale also makes sense as the weight coefficients 'W' will be influenced by the scale and distribution.
We introduce 'Scalar' method to fit and scale the dataset to get better performance.

3.2.4

(1) For PCA, I use 'GridSearchCV' to find the suitable parameters. As the plot below shows, the best parameter is about 10 which means taking 10 features from 11 features. Thus, I think PCA won't have much influence on the dataset.

```
best paramters:
- pca__n_components 10
[0.20306913 0.13871658 0.11142936 0.10083003 0.08580487 0.08113893
 0.07734232 0.06764841 0.0534682  0.05077385 0.02977831]
```

(2) Due to the dataset was collected in the United States in 1994, the dataset naturally is influenced by the time and society. For example, the dataset contains small amount of samples about people with different races and native countries.

3.3 EXPERIMENTS

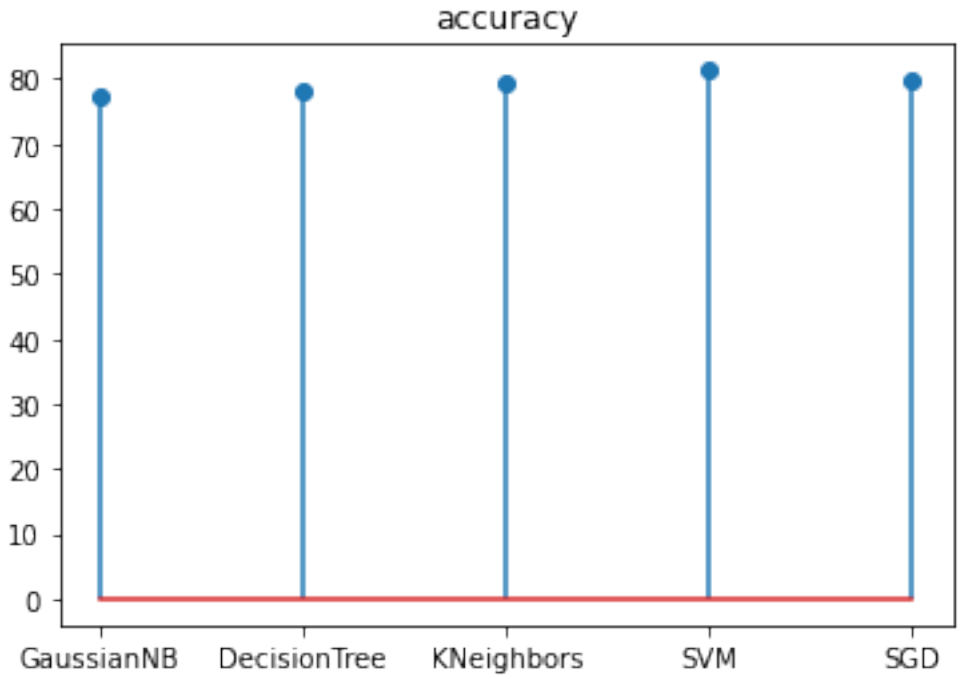
3.3.1

In order to ensure valid performance evaluation, we first apply k-fold fit and evaluation process. We divide the dataset into different train sets and validation sets and we replace them each time according to the k-fold processing. During the k-fold, we extract each model that predefined with fixed parameters in the model sets to train and evaluate the performance of each model. For hyper-parameter tuning, we predefined the parameters that need to be tuned and we introduced 'GridSearchCV' to estimate every model with different parameters to find the best one.

3.3.2

From the plot below, we can see that the accuracy performances of each model are around

78% and they are very close to each other. The SVM algorithm gives the highest accuracy about 80% while GaussianNB shows the lowest about 77%. As the dataset after transformation is mainly discrete values and the variance of different features is not very consistent, we can see the GaussianNB didn't show good performance which is the same as our expectation cause it's good at handling regular features. The other models show a better performance as their decisions are mainly dependent on some specific feature samples, thus they suffer less from the format.



3.3.3

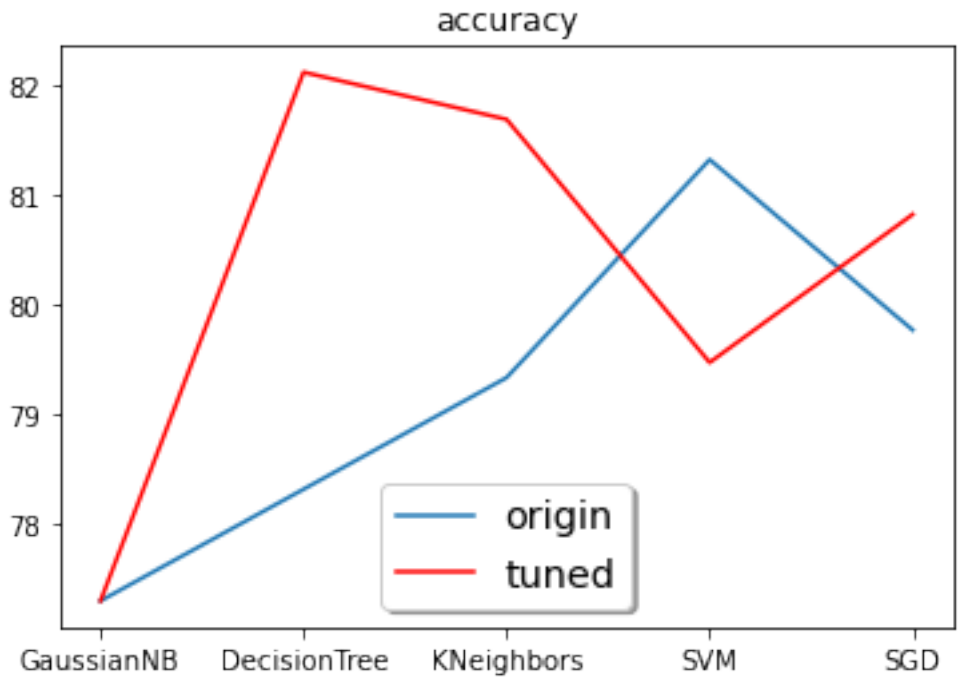
As concluded from 3.3.2, the format of the dataset has a great impact on the training and prediction as the dataset contains many discrete values. Since the values of age or education years do have larger values based on the nature characteristics compared with other features, the variance of different features will also influence the performance. Thus, models that do not heavily rely on the data characteristics will give better performance for example SVM gives the best while models like GaussianNB that good at regular features shows the worst.

3.3.4

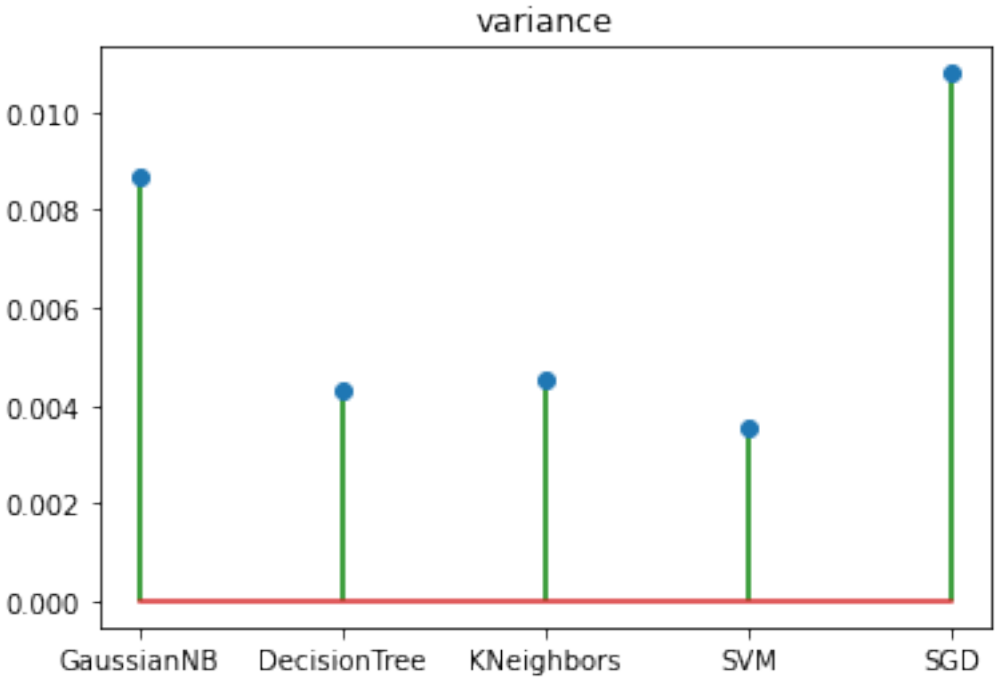
Since the tuning of hyper-parameter is very time-consuming especially when we have many potential parameters. Thus, we aim to introduce 'HalvingGridSearchCV' to approach the result. 'HalvingGridSearchCV' will first train a random sample of data to the combinations of parameters and the parameter with best performance will be selected to train larger samples. The process is repeated until we find the best set. Thus, 'HalvingGridSearchCV' can greatly reduce the time of searching and maintains similar performance compared to 'GridSearchCV'. However, we should notice that models with greedy characteristics may have priority in selection as they will show better performance at the beginning. Thus, multiple searching and

larger range of hyper-parameter may be necessary, and it is acceptable as the training time is reduced.

3.3.5



From the plot above, we can see most models take a better performance after tuning while SVM decreases a little and GaussianNB keeps the same. The Decision Tree and K-NN are enhanced the most. As the tuning process won't change the structure and format of the dataset, the performance of GaussianNB and SGD is as expected. As the parameter has impact on the choice of support vectors, the SVM is affected due to the change of support vectors. Due to deeper tree depth, more leaves and larger neighbours, the performance of Decision Tree and K-NN all exceed 80%. Also, the two models have relatively low variance which means they will have more stable performance according to the plot below.



3.3.6

We final choose Decision Tree to make our predictions, the parameter is below:

```
DecisionTreeClassifier
- best_score = 0.8212019461099704
best paramters:
- max_depth 30
- min_samples_leaf 32
```

Random_state = 42

MNIST Question

4.1.1 The 28x28 pictures are more recognizable than 8x8 ones, thus they are expected to perform better. Since the resolution is higher in the 28x28 dataset pictures, which means that one number is consisted of more pixels. Because it will have more clue about which those pixels are representing, it will perform better in classification which number it is actually.

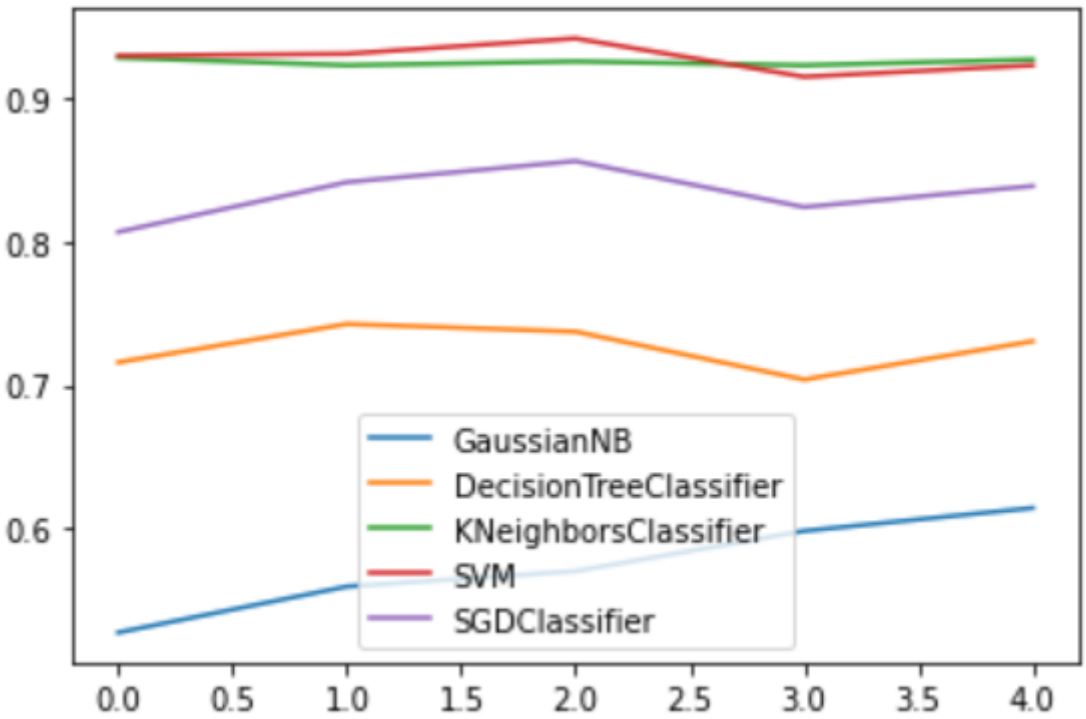
4.1.2 I choose to use Accuracy. Since the MNIST dataset is consist of handwriting numbers and our goal is to classify all these numbers correctly, choosing accuracy as the performance metrics is ideal. The baseline model is the most frequent number(0). Because I use the method value count in pandas to count the existence of each number in *train_labels*, and 0 has the highest number of existence.

4.2.1 Normally, we have two ways of preprocessing: Centering and Scaling. But as we can find in the MNIST dataset, it is already centered(cut to be 28x28) and scaled. However, we still have to reshape the shape of the train_data. This is because when fitting the data to the model, we have to make the dataset in dim 2. For this question, it should be in (3750, 784), 784 = 28*28.

4.3.1 I write the k-fold validation method as what we have done in the scikit learn tutorial, which takes the scoring_method(accuracy_score) to evaluate the score of the given model with k-folded dataset. Also, I have set all models with the hyper parameters and import all the methods we have to use from specific libraries.

4.3.2 As what we can find in the picture below:

```
mean                        std
0.5744  0.030280466751136225
0.7261333333333334  0.014226110579572435
0.9250666666666666  0.0021333333333333564
0.9277333333333333  0.008860398787112634
0.8333333333333333  0.016759740119968725
```



Different model have different accuracy which means different performances. GasussianNB is the worst while DecisionTreeClassifier and SGDClassifier are much better. The performance of KNN and SVM are similar, but by calculating the std of the accuracy we can find that though the mean accuracy of SVM is higher, it's more unstable than KNN. Thus, KNN has the best performance with the given hyper-parameters now.

4.3.3 The worst performance model is GaussianNB and the one with best performance is KNN. GaussianNB performs the worst because of two reasons. Firstly, GaussianNB bases on the hypothesis: every feature is independent, which is not 100% true in this model. Secondly, GaussianNB is based on the prior possibility which is not the same for all classes in the actual model(can be seen from the value_count result). On the other hand, the dataset of MNIST are well-processed image of handwriting numbers showing in gray scale numbers, which means that they are very suitable for KNN to classify. KNN uses the distance between different data points to classify different classes, this is now applied to the gray scales. Since every number will have similar distribution of gray scale, it is easy for KNN to do classification.

4.3.4 To tune the hyper-parameters, I use the `GridSearchCV` from scikit-learn library. First I build a dictionary, which enables a grid search for the models which have hyper-parameters. Then I build a scorer which gives the score of different hyper-parameter. As we have said before, I choose `accuracy_score` from `sklearn`. That's all preparation we need for using `GridSearchCV`! Finally, we can go over each model in the dictionary and use `kfold` method to evaluate the effect

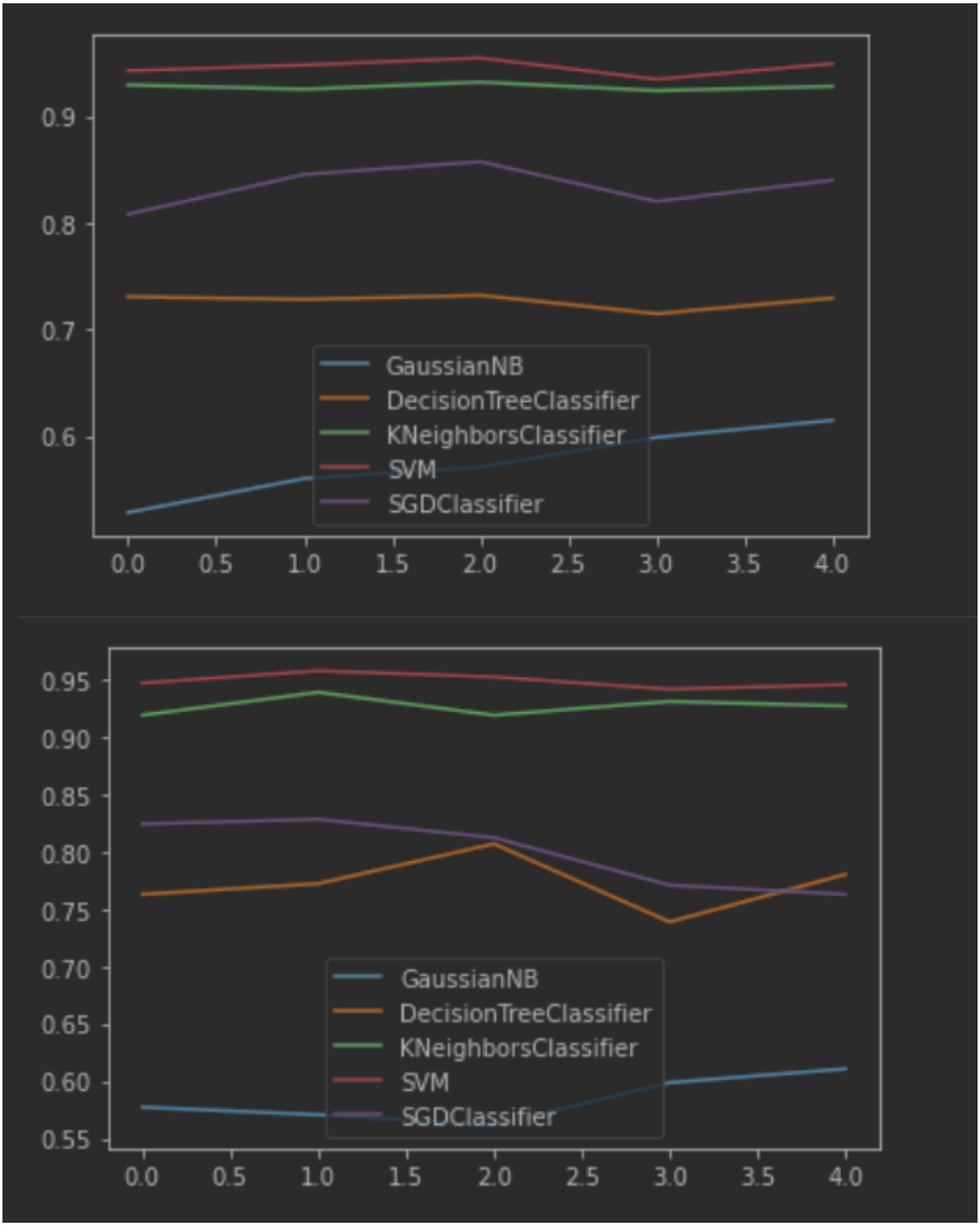
of different hyper-parameters. By doing so, we can get the final result, which contains the best result of different models and the best hyper-parameters chosen.

4.3.5

Classifier	Origin Hyper-parameter	Tuned Hyper-paramerter	Origin Acc	Tunded Acc
GaussianNB	/	/	0.5837333333333333	0.5837333333333333
DecisionTreeClassifier	max_depth=None, min_samples_leaf=2	max_depth=None, min_samples_leaf=3	0.7645333333333333	0.772
KNeighborsClassifier	n_neighbors: 5	n_neighbors: 4	0.9239999999999998	0.9266666666666665
SVM	C = 10, kernel = 'poly', degree = 3, gamma = 'scale'	C=15, degree=1, gamma=scale, kernel=rbf	0.9328000000000001	0.9485333333333333
SGDClassifier	alpha=10, eta0=0.1	alpha=0.1, eta0=0.2	0.7925333333333333	0.7994666666666667

As we can see in the table above, GaussinNB not change since there is no hyper-parameter to change. DecisionTreeClassifier changes since the minimun sample leaf number changes. KNeighborsClassifier is improved since we change the neighbor number used from 5 to 4. SVM changes since the number of C is changed from 10 to 15. Also SGDClassifier changes, because of the number of alpha and eta0.

4.3.6 The extra features seems to have negative result to the result of classification. It is opposite to the expectation before. What I think cause this effect is that there are more useless features in the 28x28 picture, which means that there are lots of 0 in it. The existence of those 0 might shrink the effect of the useful gray scales which represents the numbers but not the background. Also the size of the picture will strongly influence the speed of calculation.



[the first picture is for 28x28, second is for 8x8]

4.3.7 SVM, 8x8 feature

5

For **GaussianNB**, it perfroms quite stable in both two assignments. The accuracy is acceptable but can't be better since it based on the hypothesis that all features are independent. For **DecisionTreeClassifier**, it's good at dealing with discrete features. Thus it is chosen to be the final model of the US CENSUS problem. However, it's disadvantage in dealing problems with many different classes(like MNIST). For **KNeighborsClassifier**, it has a quite fast training speed, comparing with **SVM** and **SGDClassifier**. But the accuracy for MNIST isn't high enough since the pixels of numbers don't have an obvious relationship in space distance. For **SVC**, it is chosen to be the final model of MNIST for it's property of support vector. Since we can make the conclusion with the support vectors while changing the others, it is very suitable for MNIST problem. But in US CENSUS, it is influenced by the data structure a lot. For **SGDClassifier**, it will give quite a useful result for it's property of giving out result in possibility. But the speed of tuning the hyper-parameters is too slow to be used.