An Investigation of Classification Methods of Fashion-MNIST

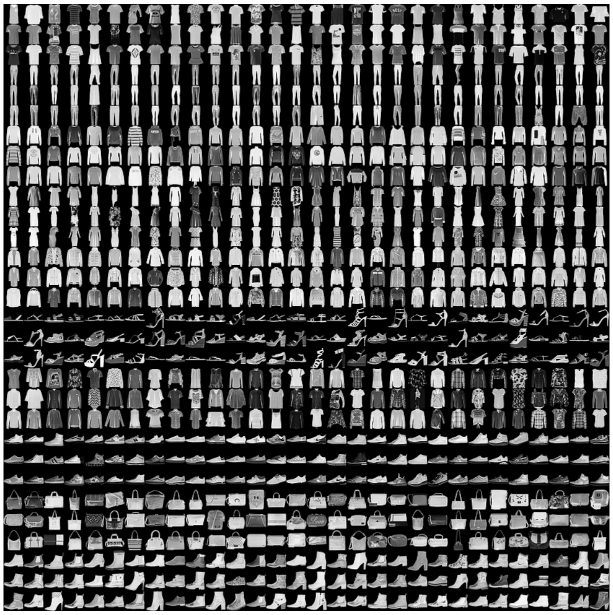
Toby Qin 14554445
Jiachen Pan 19663960
Jingshan Shi 59182834

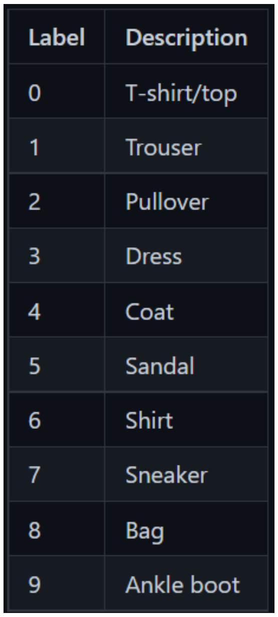
1. Summary

In this project, a dataset called Fashion-MNIST will be analyzed by 4 different classifiers, including k-NN, logistic, feedforward neural networks, and decision trees. 3 analysis ideas, including Confusion Matrices, Learning Curve, and Complexity-error Tradeoffs, are also implemented to analyze the prediction process and results. In conclusion, most of the classifiers used in Fashion-MNIST made prediction mistakes that are hard to identify, even in real life they are easily mistaken by humans. The learning curve varies as the cross-validation generator varies and the learning rate of MLPClassifier is essential to determine after several tests.

2. Data Description

Fashion-MNIST is a dataset of outfit images (see picture on the left), including a training set of 60000 samples and a testing set of 10000 samples. There are 10 labels (see table on the right) and each label corresponds to a kind of outfit. In the article of "AutoML-Zero: Evolving Machine Learning Algorithms From Scratch" (written by Esteban Real, Chen Liang, David R. So, Quoc V. Le), Fashion-MNIST was also used when analyzing a new Machine Learning algorithm in order to test the algorithm in many different datasets other than the most popular one, CIFAR-10, to get a more detailed evaluation.





Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

3. Classifiers

There are 4 classifiers used in this analysis: KNeighborsClassifier, LogisticRegression, MLPClassifier(corresponds to Feed Forward Neural Network), and DecisionTreeClassifier. They are all from the scikit-learn library. Their default settings used in this analysis are as follows unless noted separately for different analysis. All other settings are defaulted by the classifiers.

Classifiers	Setting
KNeighborsClassifier	n_neighbors = 10
LogisticRegression	penalty='l1', solver='liblinear', fit_intercept=True, C=1, random_state=seed
MLPClassifier	hidden_layer_sizes=(64,), activation='relu', solver='adam', n_iter_no_change=100, max_iter=100, learning_rate_init = 0.001, batch_size=110, random_state=seed
DecisionTreeClassifier	max_depth = 10, random_state=seed

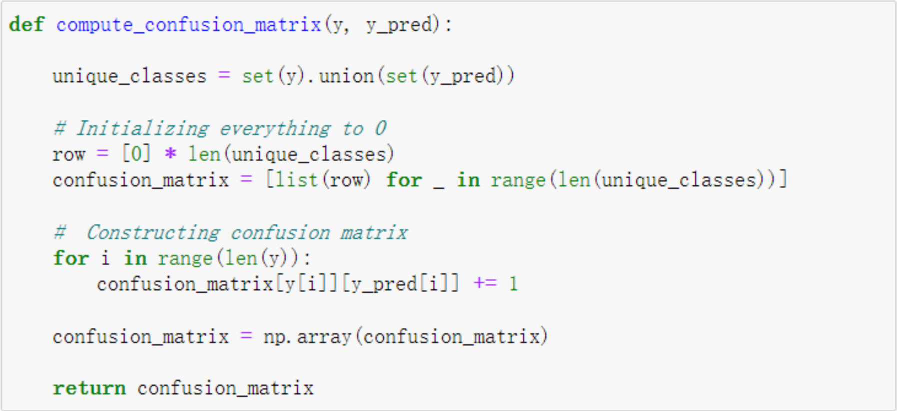
The overall descriptions of each classifier are as follows:
KNeighborsClassifier: The k-Nearest Neighbors algorithm is a nonparametric supervised learning classifier that uses proximity to classify or predict groups of individual data points.
Logistic Regression: Logistic regression is a machine learning algorithm that allows us to create classification models. The algorithm analyzes one or more independent variables and a dependent variable to predict an output.
MLPClassifier (Neural Network): MLPClassifier means Multilayer Perceptron Classifier. MLPClassifier relies on the underlying neural network to perform the classification task.
DecisionTreeClassifier: A decision tree is a nonparametric supervised learning method for classification and regression. It is used to create a model that predicts the value of a target variable by learning decision rules inferred from the characteristics of the data.

4. Experimental Setup

In this experiment, although the training set and testing set have sizes of 60000 and 10000 respectively, only the first 3000 data was used for both training set and testing set, in order to reduce the amount of running time. For the classifiers that generate different results at each running time, a random seed in the size of 1234 is also provided in numpy.random.seed(size).

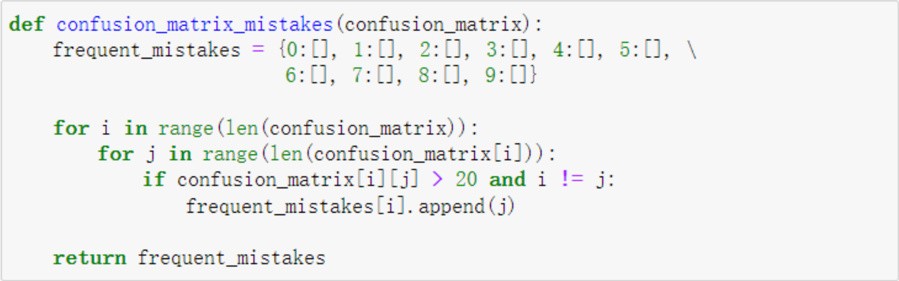
4.1 Confusion matrices

Confusion Matrix is a two-dimensional matrix that the sizes of rows and columns are all C, where C is the number of labels (in this case C = 10). The rows in the matrix are the true label values and the columns are the predicted label values. The following code is the function to compute such a confusion matrix, implemented in Python 2D-list.



```python
def compute_confusion_matrix(y, y_pred):

    unique_classes = set(y).union(set(y_pred))

    # Initializing everything to 0
    row = [0] * len(unique_classes)
    confusion_matrix = [list(row) for _ in range(len(unique_classes))]

    #  Constructing confusion matrix
    for i in range(len(y)):
        confusion_matrix[y[i]][y_pred[i]] += 1

    confusion_matrix = np.array(confusion_matrix)

    return confusion_matrix
```

For each confusion matrix, to find where the classifier made the most of the mistake, another algorithm is implemented to find the mistakes made that are higher than a certain frequency for each true label, expressed in the Python dictionary. In this analysis, the frequency is set to 50.

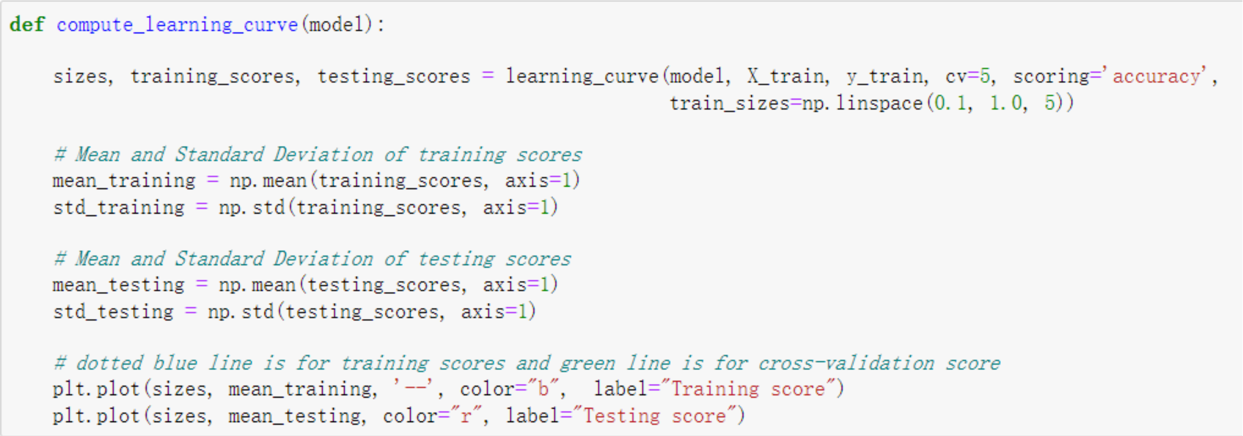


```python
def confusion_matrix_mistakes(confusion_matrix):
    frequent_mistakes = {0:[], 1:[], 2:[], 3:[], 4:[], 5:[], \
                         6:[], 7:[], 8:[], 9:[]}

    for i in range(len(confusion_matrix)):
        for j in range(len(confusion_matrix[i])):
            if confusion_matrix[i][j] > 20 and i != j:
                frequent_mistakes[i].append(j)

    return frequent_mistakes
```

In addition, different versions of the confusion matrix analysis are also set up with different amounts of training data.

4.2 Learning curve

The learning curve represents the number of iterations (exploration times) on the horizontal axis. The vertical axis represents the curves of the learning process of various learning tests. As a learning test, the negative accelerated descent curve with the number of errors, time, reaction latency. If the number of positive responses or the rate of positive responses is used on the ordinate, then it is an S-shaped or negative acceleration rising curve. The following code is implemented to visualize learning curves with different classifiers by using python.



```python
def compute_learning_curve(model):

    sizes, training_scores, testing_scores = learning_curve(model, X_train, y_train, cv=5, scoring='accuracy',
                                                            train_sizes=np.linspace(0.1, 1.0, 5))

    # Mean and Standard Deviation of training scores
    mean_training = np.mean(training_scores, axis=1)
    std_training = np.std(training_scores, axis=1)

    # Mean and Standard Deviation of testing scores
    mean_testing = np.mean(testing_scores, axis=1)
    std_testing = np.std(testing_scores, axis=1)

    # dotted blue line is for training scores and green line is for cross-validation score
    plt.plot(sizes, mean_training, '--', color="b",  label="Training score")
    plt.plot(sizes, mean_testing, color="r", label="Testing score")
```

The learning curve is computed by mean value and standard deviation value from both training data and testing data. For each classifier, I set x-axis as training size and y-axis as accuracy.

4.3 Complexity-error tradeoffs

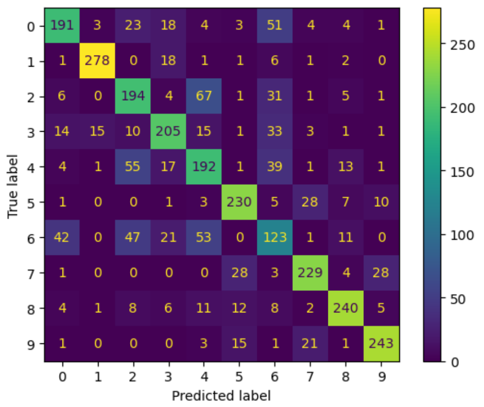
For the DecisionTreeClassifier, in order to find out the influence of the different depth, we use the for loop to set up different depths from 2 to 10 and keep the other variable the same to see how error rate is related to the depth in DecisionTreeClassifier.

For the MLPClassifier, in order to find out the influence of the different hidden_layer_sizes, we use the for loop to set up different hidden_layer_sizes from 2 to 10 and keep the other variable the same to see how error rate is related to the hidden_layer_sizes in MLPClassifier.

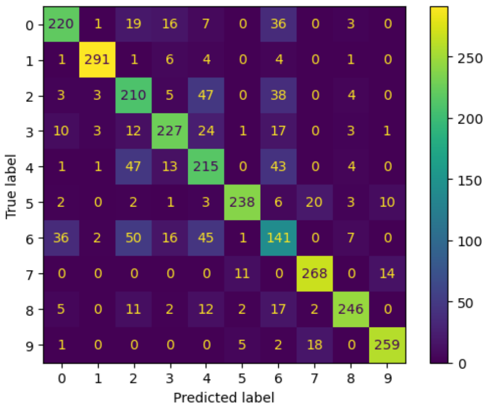
5. Experimental Results

5.1 Confusion matrices

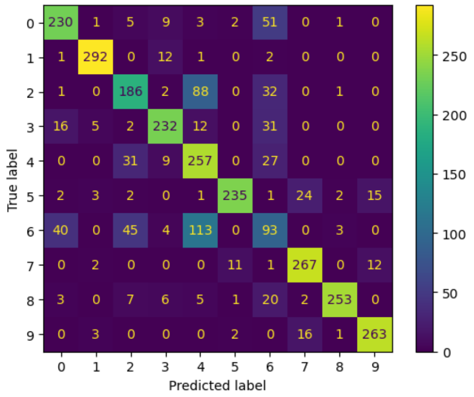
The confusion matrix plotted in a two-dimensional table is as follows. In general, for all the classifiers, each predicted label's maximum frequencies are at the same true label value. And they are all on the diagonal line, depicting the situation where the predictions are correct.

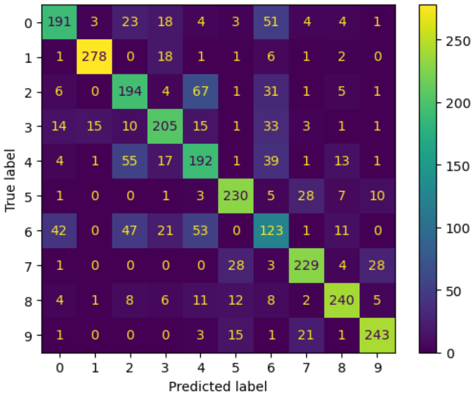


KNeighborsClassifier



LogisticRegression

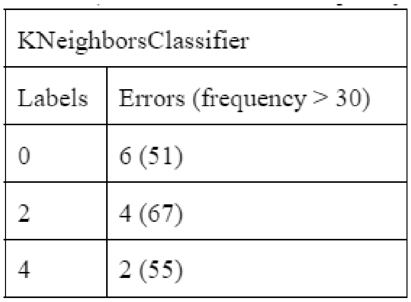




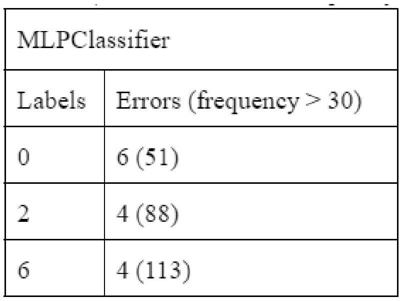
MLPClassifier

DecisionTreeClassifier

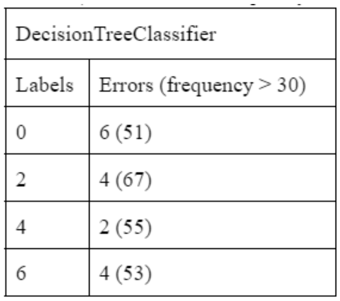
When looking at the errors made at a frequency higher than 50, the mistakes for each classifier are as follows (with errors at a frequency less than 50 omitted):



KNeighborsClassifier	
Labels	Errors (frequency > 30)
0	6 (51)
2	4 (67)
4	2 (55)



MLPClassifier	
Labels	Errors (frequency > 30)
0	6 (51)
2	4 (88)
6	4 (113)

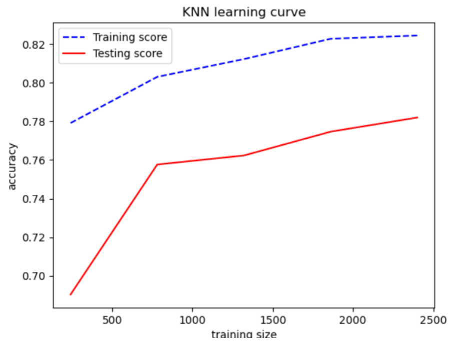


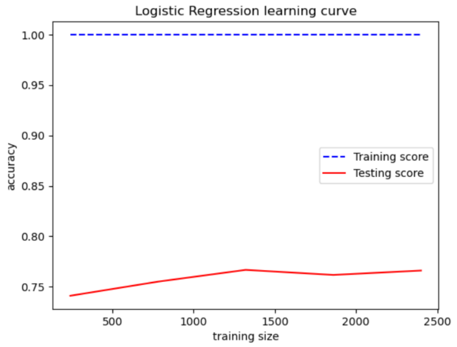
DecisionTreeClassifier	
Labels	Errors (frequency > 30)
0	6 (51)
2	4 (67)
4	2 (55)
6	4 (53)

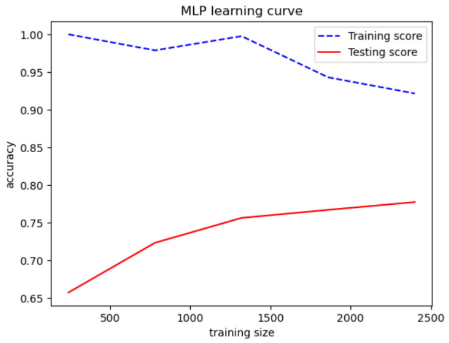
It is true that LogisticRegression made no errors at a frequency higher than 50 in this case. From the tables above, it is clear to see that some popular errors are between label 0(T-shirt/top) & 6(Shirt), label 2(Pullover) & 4(Coat), and label 6(Shirt) & 4(Coat).

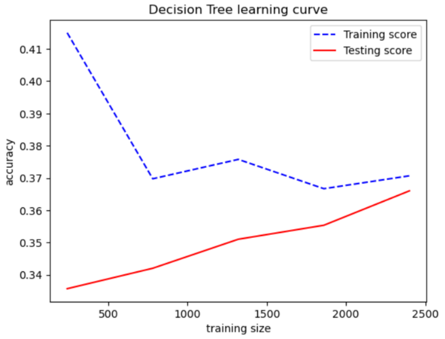
5.2 Learning curve

The learning curve graph is based on the training size and accuracy.





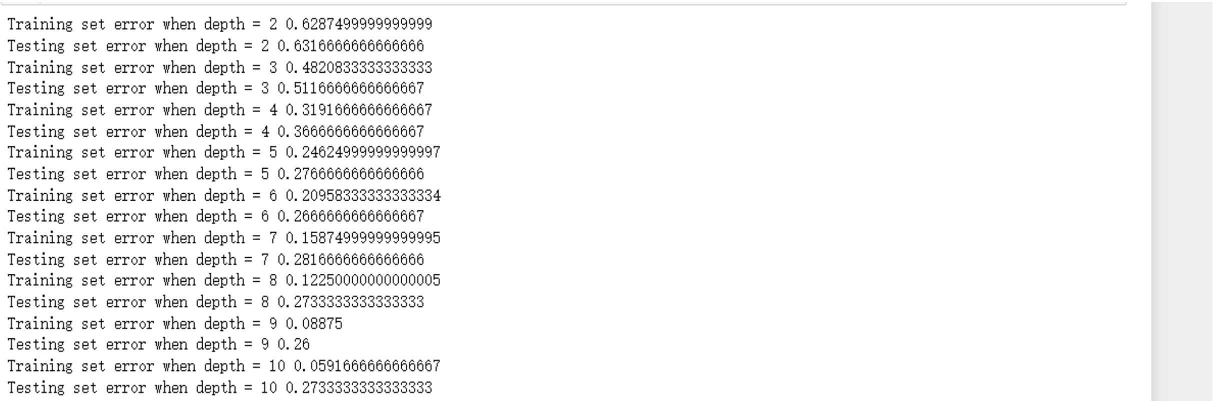




For logistic regression classifiers, the training accuracy is always 1 compared to others. Besides logistic regression, the learning curve is always rapid at the beginning and goes more smoothly after a certain point. The result for the decision tree is not as good as the other 3 classifiers.

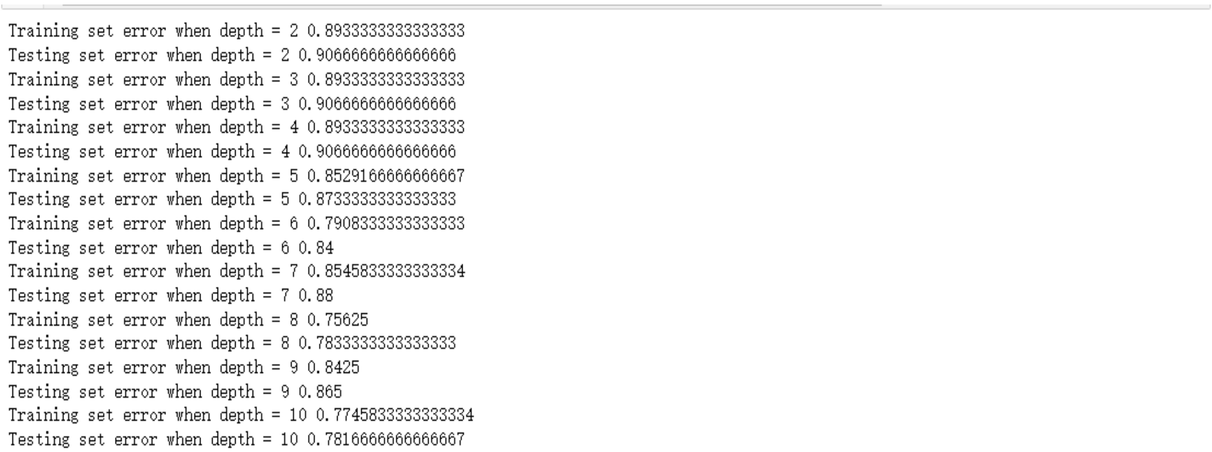
5.3 Complexity-error tradeoffs

For the DecisionTreeClassifier, the error rate is big when the depth=2, and when we increase the depth one by one, the error rate becomes smaller and smaller.



```
Training set error when depth = 2 0.6287499999999999
Testing set error when depth = 2 0.6316666666666666
Training set error when depth = 3 0.4820833333333333
Testing set error when depth = 3 0.5116666666666667
Training set error when depth = 4 0.3191666666666667
Testing set error when depth = 4 0.3666666666666667
Training set error when depth = 5 0.24624999999999997
Testing set error when depth = 5 0.2766666666666666
Training set error when depth = 6 0.20958333333333334
Testing set error when depth = 6 0.2666666666666667
Training set error when depth = 7 0.15874999999999995
Testing set error when depth = 7 0.2816666666666666
Training set error when depth = 8 0.12250000000000005
Testing set error when depth = 8 0.2733333333333333
Training set error when depth = 9 0.08875
Testing set error when depth = 9 0.26
Training set error when depth = 10 0.0591666666666667
Testing set error when depth = 10 0.2733333333333333
```

However, for the MLPClassifier, the error rate is very big and has little effect by the different hidden_layer_sizes we used from 2 to 10.



```
Training set error when depth = 2 0.8933333333333333
Testing set error when depth = 2 0.9066666666666666
Training set error when depth = 3 0.8933333333333333
Testing set error when depth = 3 0.9066666666666666
Training set error when depth = 4 0.8933333333333333
Testing set error when depth = 4 0.9066666666666666
Training set error when depth = 5 0.8529166666666667
Testing set error when depth = 5 0.8733333333333333
Training set error when depth = 6 0.7908333333333333
Testing set error when depth = 6 0.84
Training set error when depth = 7 0.8545833333333334
Testing set error when depth = 7 0.88
Training set error when depth = 8 0.75625
Testing set error when depth = 8 0.7833333333333333
Training set error when depth = 9 0.8425
Testing set error when depth = 9 0.865
Training set error when depth = 10 0.7745833333333334
Testing set error when depth = 10 0.7816666666666667
```

6. Insights

As for the result of the confusion matrix analysis, it is not surprising to see that the classifiers made errors between errors are between label 0(T-shirt/top) & 6(Shirt), label 2(Pullover) & 4(Coat), and label 6(Shirt) & 4(Coat). In real life, both of them are even sometimes hard to identify, such as T-shirt versus Shirt. But one surprising thing is that the LogisticRegression didn't make errors at a frequency higher than 50.

The result of the learning curve as I vary the cross-validation generator. When applying the learning curve to the MLPClassifier, I tried to use different learning rates to see the difference. The result is different on the accuracy shown on the graph, but the slope of the curve is similar. In conclusion, the learning rates in MLPClassifier should not be too low or too high. To find a proper learning rate will take many times to adjust by changing the value.

When it comes to the MLPClassifier, I was shocked by the high error rate at the beginning settings. After several adjustments, we finally get a low error rate MLPClassifier. The hidden_layer_sizes from 2 to 10 has no big effect on the accuracy unless it goes to around 30. The max_depth of DecisionTreeClassifier has a significant effect on the accuracy of data.

7. Contributions

Toby Qin: Generated project ideas, implemented KNeighborsClassifier, implemented Confusion Matrices analysis, wrote sections (1. Summary, 2. Data Description and first half of 3. Classifiers)
Jiachen Pan: I mainly focused on the DecisionTreeClassifier and the MLPClassifier. I did the Complexity-error tradeoffs of these two methods.
Jingshan Shi: I implemented learning curves on all classifiers to determine the cross-validated training and test scores and see the results by changing either the value of training set sizes or hyperparameters of classifiers.