CS-452 Assigment-3

Fall 2020

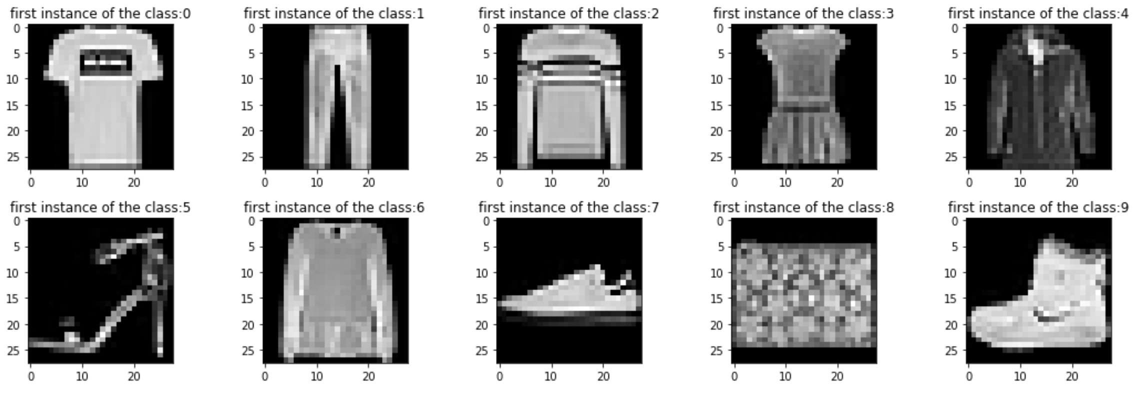
Report

Mustafa Alper Sayan

S051674

1. Introduction

In this assignment we are expected to implement 5 different well known machine learning classification algorithms which are K-neighbor's classification, perceptron, Support Vector Machine (SVM), Decision trees and random forest. We are expected to preprocess the dataset. Fit the data to the relevant classification model, predict the class of each test instance, and lastly report the performance (accuracy, confusion matrix) of our classification models. Furthermore, we should apply one of the well-known dimensionally reduction algorithm Principal component analysis (PCA), to the data with different number of principal components, repeat the training and test process with this reduced data, and compare the effect of reducing the dimensionality of the data on accuracy. The fashion MNIST dataset consists of 70.000 unique grayscale clothing images with size (28*28) and associated with a label from 10 classes. Below are some instances from the dataset:


first instance of the class:0
first instance of the class:1
first instance of the class:2
first instance of the class:3
first instance of the class:4
first instance of the class:5
first instance of the class:6
first instance of the class:7
first instance of the class:8
first instance of the class:9

2. Methodology

2.1. K-neighbor`s Algorithm

K-NN is a non-parametric instance based method used for classification. K-NN algorithm can be especially effected by noise, this can be combated with higher number of k values or by using regularization. K-NN classifies inputs based on k closest samples to the inputted sample. The output is determined by majority vote. If k is number is 1 then the algorithm is also called nearest neighbor algorithm.

2.1.1. K-NN Hyper Parameters

- Neighbor number: Number of neighbors to use, usually an odd number
- Uniform weight: All points in each neighborhood are weighted equally
- Distance weight: Weight points by inverse of their distance. Closer points will have higher influence
- Algorithm ball tree: The ball tree algorithm is like a metric tree. Metric trees organize and structure data points considering the metric space in which the points are located. The algorithm divides the data into two clusters. Each cluster represents a node of the tree. The children are chosen to have maximum distance between them. The process of dividing the data points into 2 clusters is repeated until a specified depth is reached.
- KD tree Algorithm: The data points are split at each node into two sets. KD tree is a binary search algorithm. The split criterion is often chosen as median of the data points. This splitting process is continued until the desired depth is reached.
- Algorithm Brute Force: The highest accuracy reaching algorithm with the most time and memory consuming option. Good for low dimensional or low number of samples.
- Leaf size: Used with KD tree or ball tree, this number is the desired depth
- P value 1: Equivalent to Manhattan distance which is absolute difference between two data points
- P value 2: Equivalent to Euclidian distance.

2.1.2. K-NN steps:

- For each test instance in data
- From each training sample calculate the distance from test instance
- Find k smallest distances
- Find k smallest distances labels
- From majority vote of k smallest labels determine the class of the test instance

2.2. Perceptron Algorithm

The perceptron is a type of supervised learning. It decides whether an input belongs to the class by using weight coefficients. It is a parametric method. We can then define an activation function ϕ(z) that takes a linear combination of certain input values x and a corresponding weight vector w, where z is the so-called net input (z = w1 x +...+ wmx)

Now, if the activation of a particular sample $x^{(i)}$, that is, the output of $\phi(z)$, is greater than a defined threshold θ, we predict class 1 and class -1, otherwise, in the perceptron algorithm, the activation function $\phi(\cdot)$ is a simple *unit step function*, which is sometimes also called the *Heaviside step function*:

$$\phi(z) = \begin{cases} 1 & if\ z \geq \theta \\ -1 & otherwise \end{cases}$$

- **Perceptron Pseudocode**

```
For $i = 1, \ldots, K$, For $j = 0, \ldots, d$, $w_{ij} \leftarrow$ rand$(-0.01, 0.01)$
Repeat
    For $i = 1, \ldots, K$, For $j = 0, \ldots, d$, $\Delta w_{ij} \leftarrow 0$
    For $t = 1, \ldots, N$
        For $i = 1, \ldots, K$
            $o_i \leftarrow 0$
            For $j = 0, \ldots, d$
                $o_i \leftarrow o_i + w_{ij} x_j^t$
        For $i = 1, \ldots, K$
            $y_i \leftarrow \exp(o_i) / \sum_k \exp(o_k)$
        For $i = 1, \ldots, K$
            For $j = 0, \ldots, d$
                $\Delta w_{ij} \leftarrow \Delta w_{ij} + (r_i^t - y_i) x_j^t$
    For $i = 1, \ldots, K$
        For $j = 0, \ldots, d$
            $w_{ij} \leftarrow w_{ij} + \eta \Delta w_{ij}$
Until convergence
```

Where green part is forward propagation and red part is backpropagation. Pseudocode shows the number of classes > 2. Y in the pseudocode equals to sigmoid function. K equals to number of classes, j equals to number of dimensions, N equals to number of samples, w equals to weight coefficients.

- Perceptron Hyper Parameters

    - Penalty L2: L2 regularization is called ridge regression. Formula is as below:

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2$$

Where y equals to actual output, M equals to number of samples, P equals to number of dimensions, w equals to weight coefficients, x equals to input.

- Penalty L1: L1 regularization is called lasso regularization. Formula is below:

$$\sum_{i=1}^{M}(y_i-\hat{y}_i)^2=\sum_{i=1}^{M}\left(y_i-\sum_{j=0}^{p}w_j\times x_{ij}\right)^2+\lambda\sum_{j=0}^{p}|w_j|$$

Where y equals to actual output, M equals to number of samples, P equals to number of dimensions, w equals to weight coefficients, x equals to input, lambda equals to regularization term.

- Penalty elastic net: used for mixing l1 and l2 regularization with different importance
- Penalty None: No regularization should be used
- Alpha: Regularization term. It can be seen in the example formula L1 above.
- L1 ratio: Determines the importance of l1, only used when regularization term is elastic net
- Fit intercept: Used to estimate the intercept
- Max iter: Determines the number of epochs
- Tol: The spotting criterion

2.3. Support Vector Machine Algorithm

Support vector machines are supervised models that analyze classification and regression. Given a set of training samples, each marked as belonging to one of two categories. SVM aims to maximize the distance between two classes. If linear classification is not performed, SVM maps the inputs into high dimensional spaces.

For computing SVM classifier we need to minimize the loss function:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0,1-y_i(\mathbf{w}^T\mathbf{x}_i-b)\right)\right]+\lambda\|\mathbf{w}\|^2$$

Where n equals to number of samples, y equals to output, w equals weight, b equals intercept, lambda equals alpha value. For each i ϵ {1, 2, 3 , …, n}, we introduce a new variable $c_i = \max(0, 1 - y_i(w^T x_i - b))$. Then The equation above can be written as:

$$\text{minimize } \frac{1}{n}\sum_{i=1}^{n}\zeta_i + \lambda\|\mathbf{w}\|^2$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i - b) \geq 1 - \zeta_i \text{ and } \zeta_i \geq 0, \text{ for all } i.$$

Solving equation above gives the figure below:

$$\text{maximize } f(c_1 \dots c_n) = \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (\mathbf{x}_i^T \mathbf{x}_j) y_j c_j,$$
$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

Now we need to learn a classification rule for the transformed data points $\varphi(\vec{x_i})$. Furthermore, we should satisfy the following kernel function $k(x_i,\ x_j) = \varphi(x_i) * \varphi(x_j)$. W in the transformed space satisfies

$$\mathbf{w} = \sum_{i=1}^{n} c_i y_i \varphi(\mathbf{x}_i)$$

Then we must solve the following optimization problems:

$$\begin{aligned}\text{maximize } f(c_1 \dots c_n) &= \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (\varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)) y_j c_j \\ &= \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i k(\mathbf{x}_i, \mathbf{x}_j) y_j c_j\end{aligned}$$
$$\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i.$$

$$b = \mathbf{w}^T \varphi(\mathbf{x}_i) - y_i = \left[\sum_{j=1}^{n} c_j y_j \varphi(\mathbf{x}_j) \cdot \varphi(\mathbf{x}_i)\right] - y_i$$
$$= \left[\sum_{j=1}^{n} c_j y_j k(\mathbf{x}_j, \mathbf{x}_i)\right] - y_i.$$

$$\mathbf{z} \mapsto \operatorname{sgn}(\mathbf{w}^T \varphi(\mathbf{z}) - b) = \operatorname{sgn}\left(\left[\sum_{i=1}^{n} c_i y_i k(\mathbf{x}_i, \mathbf{z})\right] - b\right).$$

- SVM hyper parameters
    - C: Regularization parameter. Regularization is Inversely proportional with c value
    - Kernel linear: degree of the kernel function is 1
    - Kernel poly: Kernel is a polynomial function
    - Kernel sigmoid: Linear classifier
    - Kernel rbf: Radial basis function kernel
    - Degree: Degree of the polynomial function
    - Gamma scale: 1 / (n_features * X.var()) for gamma value
    - Gamma auto: 1 / n_features

2.4. Decision Tree algorithm

Decision tree algorithm is a predictive model used in machine learning. It uses a decision tree to go from observations about a sample to predict a samples output value. There are two main type of decision trees namely decision tree classifier and decision tree regression. In this assignment we used decision tree classifier. This type of tree is split using the following method training examples are broken down into smaller subsets and using this method the tree is built. The aim of this method is to partition the data into sparse or dense clusters. Decision trees are built using divide and conquer algorithm to be more specific it is called recursive partitioning.

- Steps of decision tree algorithm
    - Select a test for the root node. Create branch for each possible outcome of the test
    - Split instances into subsets
    - Repeat recursively for each branch
    - Stop when all the leaf nodes are pure. Or specified depth is reached

- Advantages
    - Time complexity is O(Nkd) which is better than KNN and svm also random forests where N is number of training samples, k is number of features, d is the depth of the decision tree
    - Interpretability
    - Excludes unimportant features
- Disadvantages
    - If max depth is not specified or not set appropriately the tree might over fit training instances.
    - They are biased to split on features having large levels
    - Small changes in dataset can result in different trees.
    - Large trees are not easy to interpret
- Decision tree parameters
    - Criterion Gini: Gini is the measure of a randomly chosen element from the samples would be labeled incorrectly according to the distribution of labels. It is computed by summing the probability p_i of a sample with label i being chosen times $\sum_{k \neq i} p_k$. At minimum it can be 0. The Gini impurity is computed for K classes from i ϵ {1, 2, …, K} let p_i be the samples labeled with class i.

$$\mathrm{I}_G(p) = \sum_{i=1}^{J} \left(p_i \sum_{k \neq i} p_k \right) = \sum_{i=1}^{J} p_i (1 - p_i) = \sum_{i=1}^{J} (p_i - {p_i}^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} {p_i}^2 = 1 - \sum_{i=1}^{J} {p_i}^2$$

    - Criterion Entropy: Prioritizes information gain. Entropy is defined as below:

$$\mathrm{H}(T) = \mathrm{I}_E(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

Where $p_1,\ p_2,\ \ldots,\ p_i$ are the samples that add up to 1 and represent the percentage of each class in the child node that results from splitting the tree.

$$\overbrace{IG(T,a)}^{\text{Information Gain}} = \overbrace{\mathrm{H}(T)}^{\text{Entropy (parent)}} - \overbrace{\mathrm{H}(T|a)}^{\text{Sum of Entropy (Children)}}$$

$$= -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{i=1}^{J} -\Pr(i|a) \log_2 \Pr(i|a)$$

Averaging over all possible values of A

$$\overbrace{E_A\,(IG(T,a))}^{\text{Expected Information Gain}} = \overbrace{I(T;A)}^{\text{Mutual Information between T and A}} = \overbrace{\mathrm{H}(T)}^{\text{Entropy (parent)}} - \overbrace{\mathrm{H}(T|A)}^{\text{Weighted Sum of Entropy (Children)}}$$

$$= -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{a} p(a) \sum_{i=1}^{J} -\Pr(i|a) \log_2 \Pr(i|a)$$

    - Splitter best: features have the highest importance
    - Splitter random: Takes features randomly but with using weighted distribution

    - Max depth: Decides the max depth of the tree, the tree will not split after this part
    - Min samples split: The minimum number of samples required to split node.
    - Min samples leaf: The minimum number of samples at the end of each path
    - Max features auto: square root of the number of features
    - Max features log2: Log2 of the number of features

2.5. Random Forest algorithm

Random forests are an ensemble method for classification, they are constructed using multiple decision trees to combat overfitting. To put more simply they are collections of decision trees with different methods to determine the correct class, one of them being major vote.

- Random forest steps
    - Draw a random bootstrap sample of size n (randomly choose n samples from the training set with replacement).

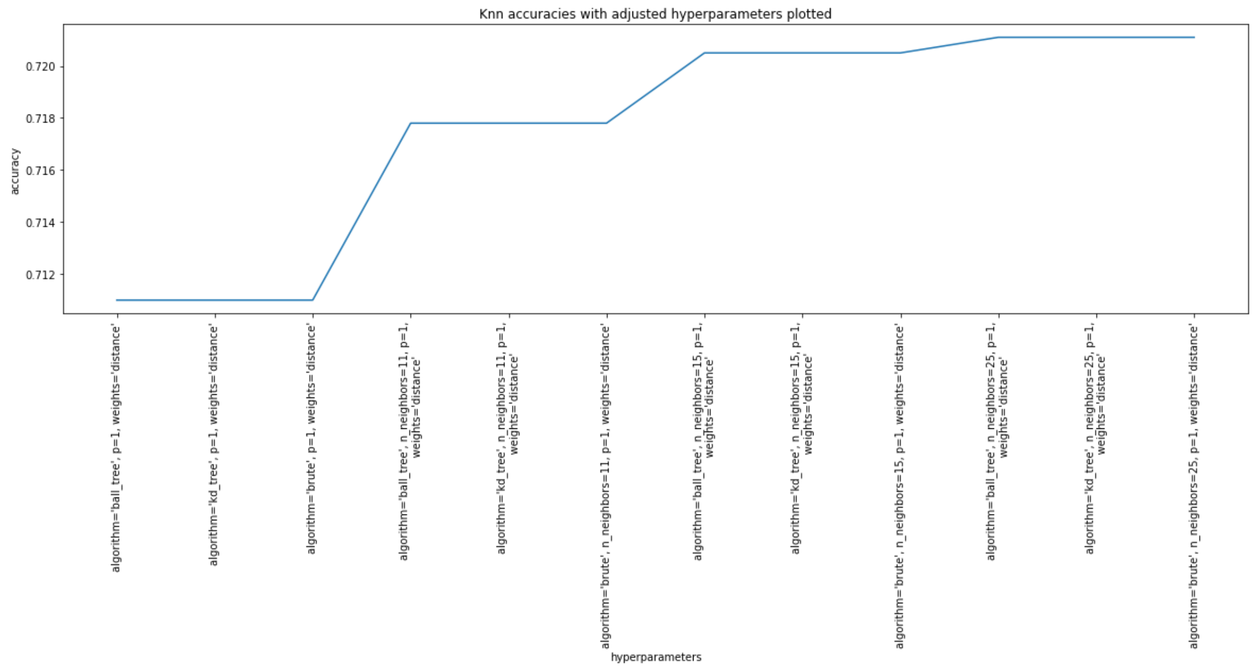
    - Grow a decision tree from the bootstrap sample
        - At each node:
        - Randomly select d features without replacement.
        - Split the node using the feature that provides the best split
        - according to the objective function, for instance, by maximizing the information gain.
    - Repeat the steps 1 to 2 k times
    - Aggregate the prediction by each tree to assign the class label by majority vote.
- Random forest pros
    - Robust to outliers
    - Efficient on large datasets
- Random forest cons
    - Biased towards categorical variables
    - Slow training
    - Not suitable for linear methods
- Random forest parameters (Almost same as decision trees therefore I omitted most of them):
    - N_estimators: Number of trees in the forest

3. Implementation Details

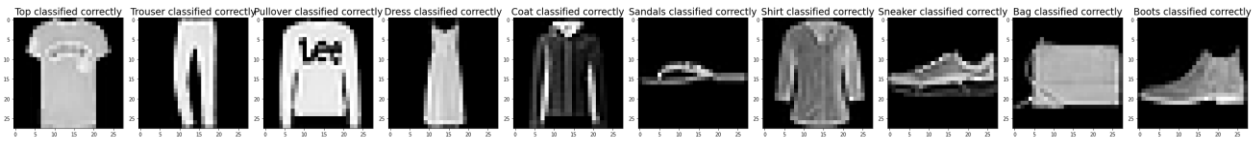
Implementation steps are as follows

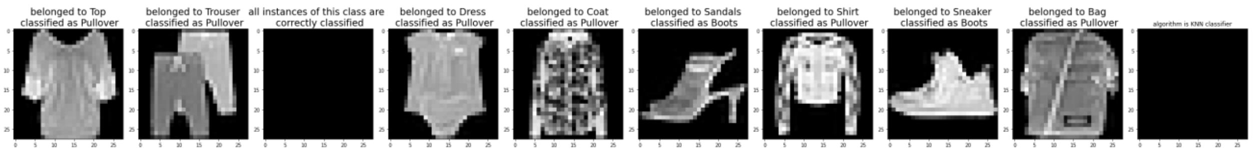
3.1. Load fashion MNIST dataset
3.2. Scale the dataset between range (0, 1)
3.3. Find the optimal hyper parameters for each objects
3.4. Fit and then predict for each train instances
3.5. Plot their accuracy
3.6. Initialize PCA with desired number of components
3.7. Fit and transform the scaled train and test instances
3.8. Find the number of components necessary of each expected number of expected variances
3.9. Initialize the algorithms with the optimal hyper parameters
3.10. Fit the dimensionally reduced objects
3.11. Predict the values of each algorithm
3.12. Plot the accuracies
3.13. Plot the confusion matrixes
3.14. Print the data frame

4. Results

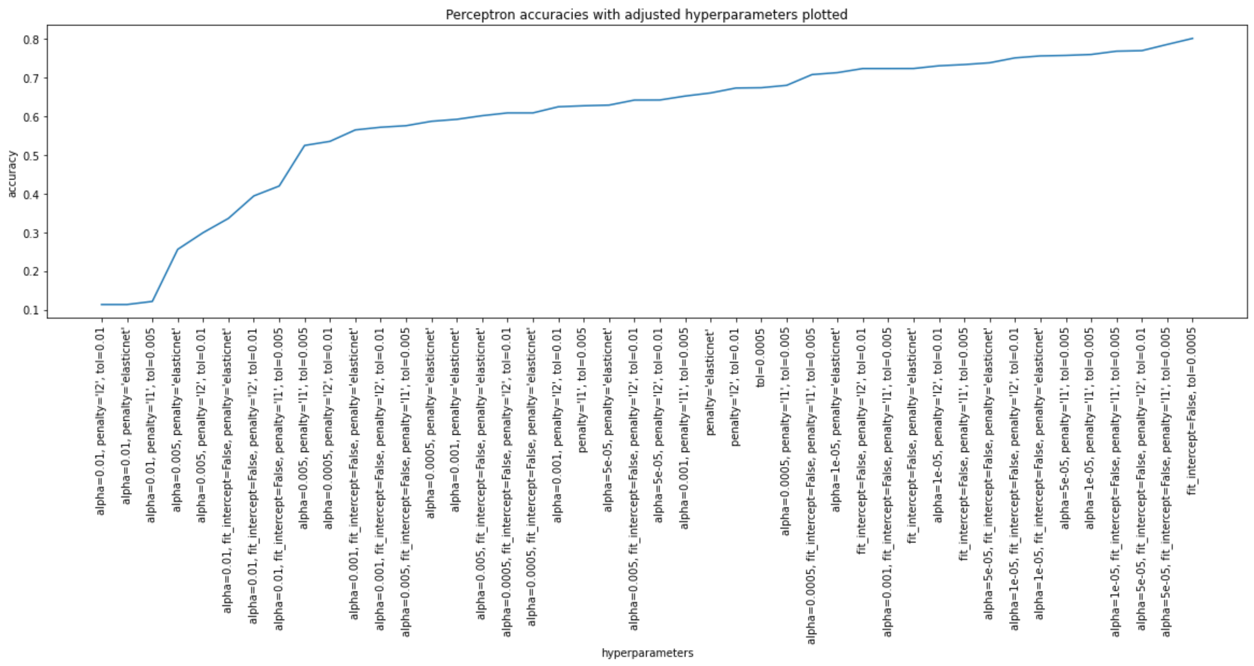


4.1.

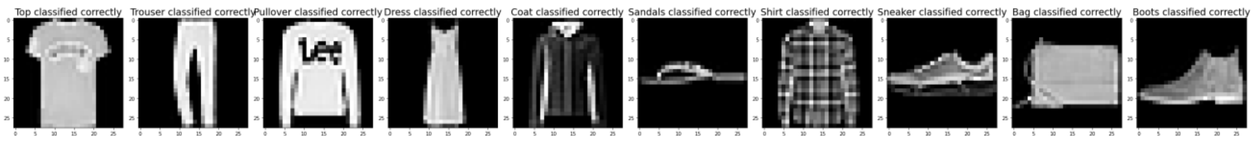

Top classified correctly
Trouser classified correctly
Pullover classified correctly
Dress classified correctly
Coat classified correctly
Sandals classified correctly
Shirt classified correctly
Sneaker classified correctly
Bag classified correctly
Boots classified correctly


belonged to Top classified as Pullover
belonged to Trouser classified as Pullover
all instances of this class are correctly classified
belonged to Dress classified as Pullover
belonged to Coat classified as Pullover
belonged to Sandals classified as Boots
belonged to Shirt classified as Pullover
belonged to Sneaker classified as Boots
belonged to Bag classified as Pullover
algorithm is KNN classifier

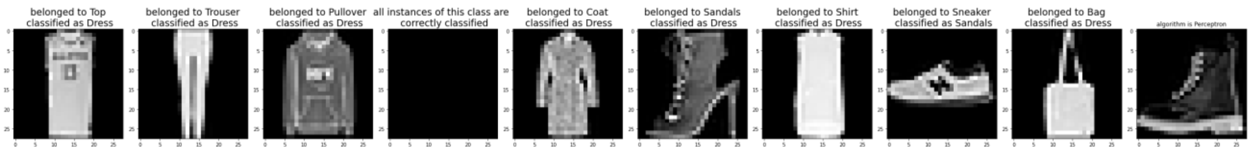
4.2.


Perceptron accuracies with adjusted hyperparameters plotted
accuracy
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
hyperparameters
alpha=0.01, penalty='l2', tol=0.01
alpha=0.01, penalty='elasticnet'
alpha=0.01, penalty='l1', tol=0.005
alpha=0.005, penalty='elasticnet'
alpha=0.005, penalty='l2', tol=0.01
alpha=0.01, fit_intercept=False, penalty='elasticnet'
alpha=0.01, fit_intercept=False, penalty='l2', tol=0.01
alpha=0.01, fit_intercept=False, penalty='l1', tol=0.005
alpha=0.005, penalty='l1', tol=0.005
alpha=0.0005, penalty='l2', tol=0.01
alpha=0.001, fit_intercept=False, penalty='elasticnet'
alpha=0.001, fit_intercept=False, penalty='l2', tol=0.01
alpha=0.005, fit_intercept=False, penalty='l1', tol=0.005
alpha=0.0005, penalty='elasticnet'
alpha=0.001, penalty='elasticnet'
alpha=0.005, fit_intercept=False, penalty='elasticnet'
alpha=0.0005, fit_intercept=False, penalty='l2', tol=0.01
alpha=0.0005, fit_intercept=False, penalty='elasticnet'
alpha=0.001, penalty='l2', tol=0.01
penalty='l1', tol=0.005
alpha=5e-05, penalty='elasticnet'
alpha=0.005, fit_intercept=False, penalty='l2', tol=0.01
alpha=5e-05, penalty='l2', tol=0.01
alpha=0.001, penalty='l1', tol=0.005
penalty='elasticnet'
penalty='l2', tol=0.01
tol=0.0005
alpha=0.0005, penalty='l1', tol=0.005
alpha=0.0005, fit_intercept=False, penalty='l1', tol=0.005
alpha=1e-05, penalty='elasticnet'
fit_intercept=False, penalty='l2', tol=0.01
alpha=0.001, fit_intercept=False, penalty='l1', tol=0.005
fit_intercept=False, penalty='elasticnet'
alpha=1e-05, penalty='l2', tol=0.01
fit_intercept=False, penalty='l1', tol=0.005
alpha=5e-05, fit_intercept=False, penalty='elasticnet'
alpha=1e-05, fit_intercept=False, penalty='l2', tol=0.01
alpha=1e-05, fit_intercept=False, penalty='elasticnet'
alpha=5e-05, penalty='l1', tol=0.005
alpha=1e-05, penalty='l1', tol=0.005
alpha=1e-05, fit_intercept=False, penalty='l1', tol=0.005
alpha=5e-05, fit_intercept=False, penalty='l2', tol=0.01
alpha=5e-05, fit_intercept=False, penalty='l1', tol=0.005
fit_intercept=False, tol=0.0005

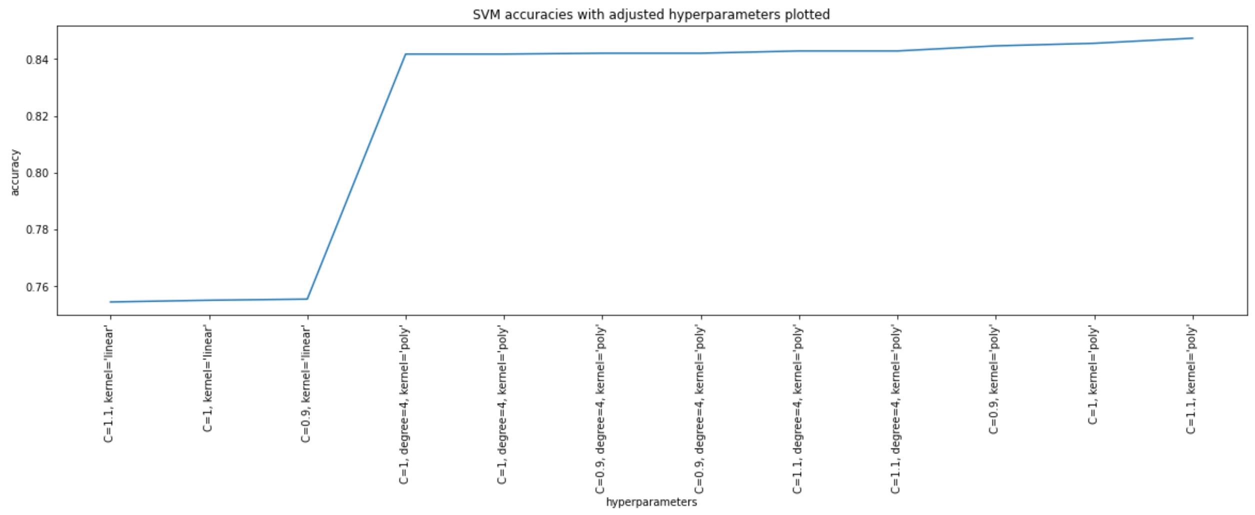
4.3.


Top classified correctly
Trouser classified correctly
Pullover classified correctly
Dress classified correctly
Coat classified correctly
Sandals classified correctly
Shirt classified correctly
Sneaker classified correctly
Bag classified correctly
Boots classified correctly

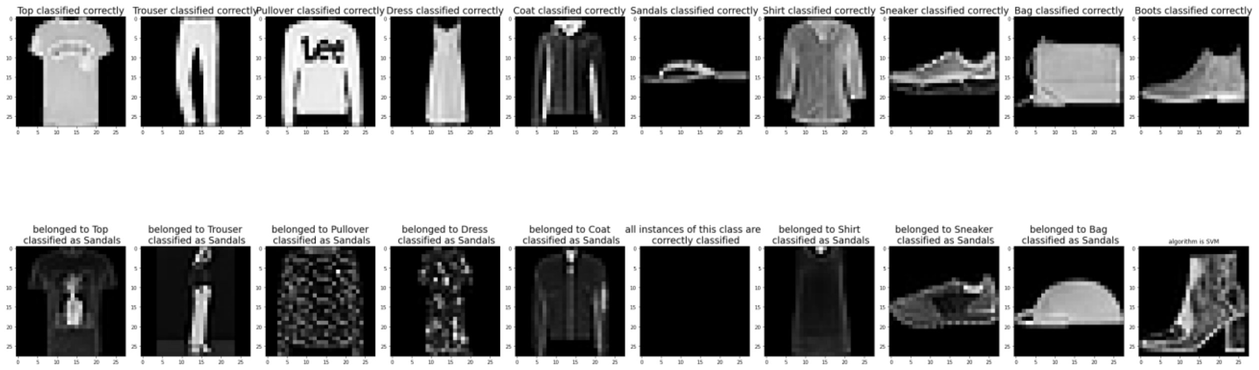
4.4.

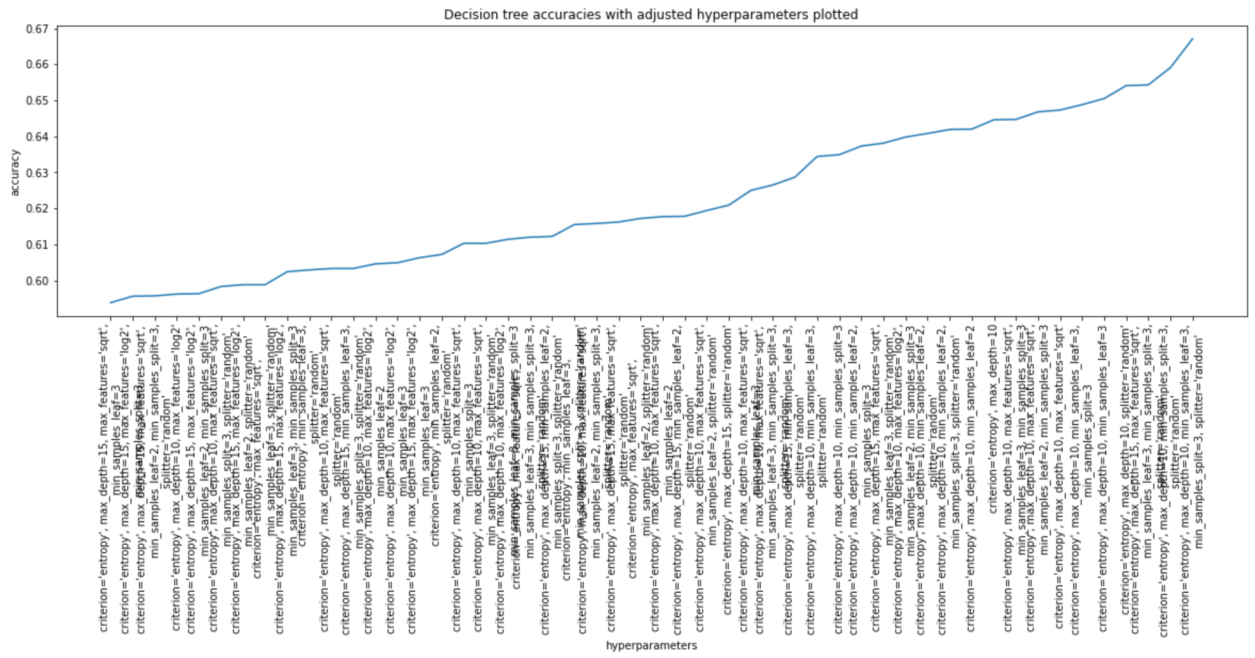

belonged to Top classified as Dress
belonged to Trouser classified as Dress
belonged to Pullover classified as Dress
all instances of this class are correctly classified
belonged to Coat classified as Dress
belonged to Sandals classified as Dress
belonged to Shirt classified as Dress
belonged to Sneaker classified as Sandals
belonged to Bag classified as Dress
algorithm is Perceptron

4.5.

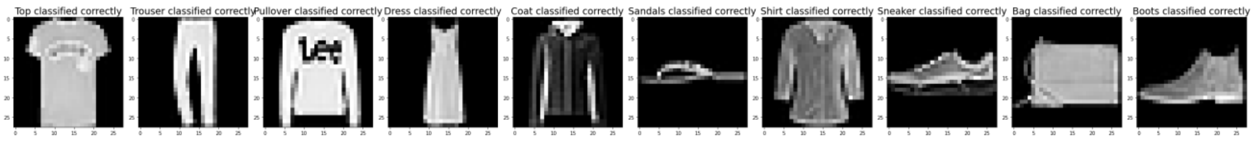

SVM accuracies with adjusted hyperparameters plotted
accuracy
0.84
0.82
0.80
0.78
0.76
hyperparameters
C=1.1, kernel='linear'
C=1, kernel='linear'
C=0.9, kernel='linear'
C=1, degree=4, kernel='poly'
C=1, degree=4, kernel='poly'
C=0.9, degree=4, kernel='poly'
C=0.9, degree=4, kernel='poly'
C=1.1, degree=4, kernel='poly'
C=1.1, degree=4, kernel='poly'
C=0.9, kernel='poly'
C=1, kernel='poly'
C=1.1, kernel='poly'

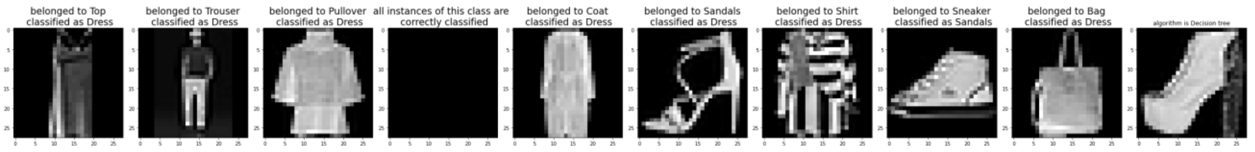
4.6.


Top classified correctly
Trouser classified correctly
Pullover classified correctly
Dress classified correctly
Coat classified correctly
Sandals classified correctly
Shirt classified correctly
Sneaker classified correctly
Bag classified correctly
Boots classified correctly
belonged to Top classified as Sandals
belonged to Trouser classified as Sandals
belonged to Pullover classified as Sandals
belonged to Dress classified as Sandals
belonged to Coat classified as Sandals
all instances of this class are correctly classified
belonged to Shirt classified as Sandals
belonged to Sneaker classified as Sandals
belonged to Bag classified as Sandals
algorithm is SVM

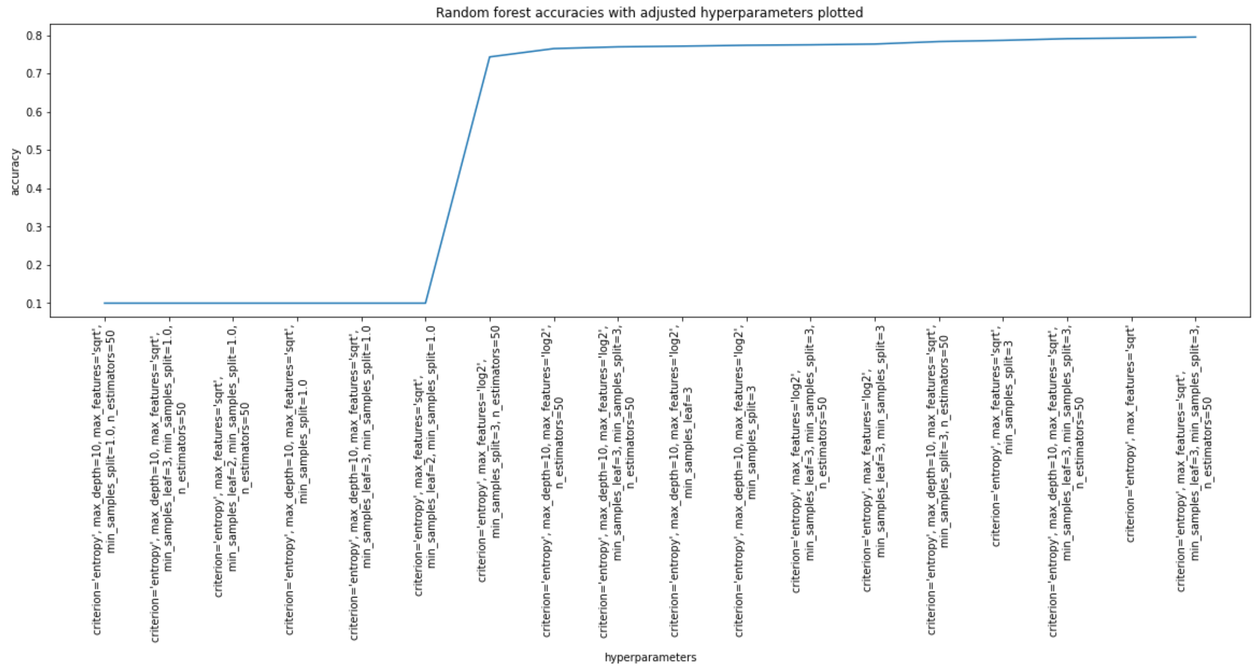

Decision tree accuracies with adjusted hyperparameters plotted
accuracy
hyperparameters
0.67
0.66
0.65
0.64
0.63
0.62
0.61
0.60
criterion='entropy', max_depth=15, max_features='sqrt', min_samples_leaf=3
criterion='entropy', max_depth=10, min_samples_split=3, splitter='random'

4.7.

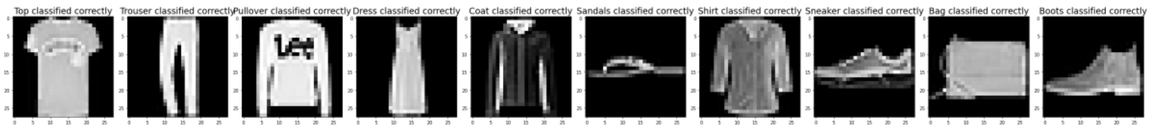

Top classified correctly
Trouser classified correctly
Pullover classified correctly
Dress classified correctly
Coat classified correctly
Sandals classified correctly
Shirt classified correctly
Sneaker classified correctly
Bag classified correctly
Boots classified correctly

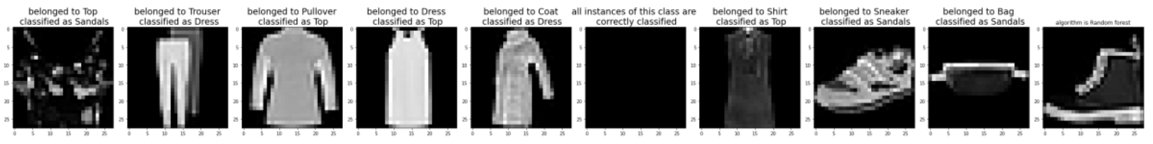

belonged to Top classified as Dress
belonged to Trouser classified as Dress
belonged to Pullover classified as Dress
all instances of this class are correctly classified
belonged to Coat classified as Dress
belonged to Sandals classified as Dress
belonged to Shirt classified as Dress
belonged to Sneaker classified as Sandals
belonged to Bag classified as Dress
algorithm is Decision tree

4.8.


Random forest accuracies with adjusted hyperparameters plotted
accuracy
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
criterion='entropy', max_depth=10, max_features='sqrt', min_samples_split=1.0, n_estimators=50
criterion='entropy', max_depth=10, max_features='sqrt', min_samples_leaf=3, min_samples_split=1.0, n_estimators=50
criterion='entropy', max_features='sqrt', min_samples_leaf=2, min_samples_split=1.0, n_estimators=50
criterion='entropy', max_depth=10, max_features='sqrt', min_samples_split=1.0
criterion='entropy', max_depth=10, max_features='sqrt', min_samples_leaf=3, min_samples_split=1.0
criterion='entropy', max_features='sqrt', min_samples_leaf=2, min_samples_split=1.0
criterion='entropy', max_features='log2', min_samples_split=3, n_estimators=50
criterion='entropy', max_depth=10, max_features='log2', n_estimators=50
criterion='entropy', max_depth=10, max_features='log2', min_samples_leaf=3, min_samples_split=3, n_estimators=50
criterion='entropy', max_depth=10, max_features='log2', min_samples_leaf=3
criterion='entropy', max_depth=10, max_features='log2', min_samples_split=3
criterion='entropy', max_features='log2', min_samples_leaf=3, min_samples_split=3, n_estimators=50
criterion='entropy', max_features='log2', min_samples_leaf=3, min_samples_split=3
criterion='entropy', max_depth=10, max_features='sqrt', min_samples_split=3, n_estimators=50
criterion='entropy', max_features='sqrt', min_samples_split=3
criterion='entropy', max_depth=10, max_features='sqrt', min_samples_leaf=3, min_samples_split=3, n_estimators=50
criterion='entropy', max_features='sqrt'
criterion='entropy', max_features='sqrt', min_samples_leaf=3, min_samples_split=3, n_estimators=50
hyperparameters

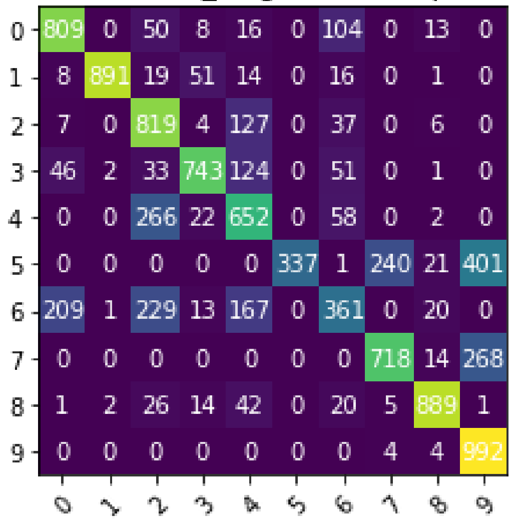
4.9.


Top classified correctly
Trouser classified correctly
Pullover classified correctly
Dress classified correctly
Coat classified correctly
Sandals classified correctly
Shirt classified correctly
Sneaker classified correctly
Bag classified correctly
Boots classified correctly


belonged to Top classified as Sandals
belonged to Trouser classified as Dress
belonged to Pullover classified as Top
belonged to Dress classified as Top
belonged to Coat classified as Dress
all instances of this class are correctly classified
belonged to Shirt classified as Top
belonged to Sneaker classified as Sandals
belonged to Bag classified as Sandals
algorithm is Random forest

4.10.

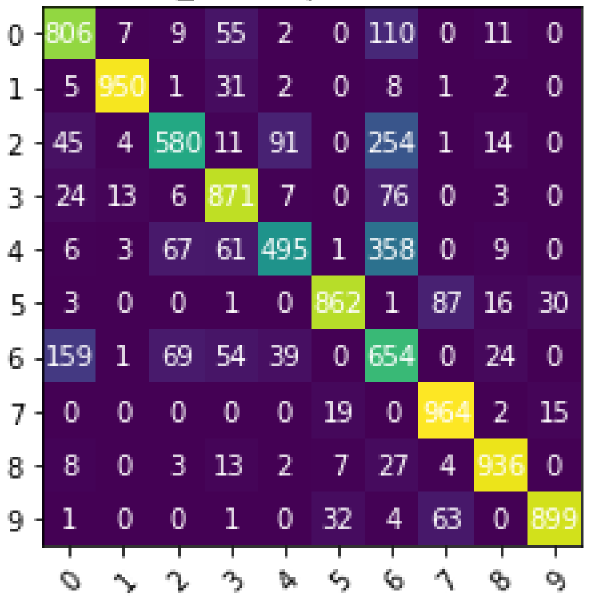
Confusion matrix of the knn classifier
accuracy= %72.11
params= algorithm='brute', n_neighbors=25, p=1, weights='distance'



	0	1	2	3	4	5	6	7	8	9
0	809	0	50	8	16	0	104	0	13	0
1	8	891	19	51	14	0	16	0	1	0
2	7	0	819	4	127	0	37	0	6	0
3	46	2	33	743	124	0	51	0	1	0
4	0	0	266	22	652	0	58	0	2	0
5	0	0	0	0	0	337	1	240	21	401
6	209	1	229	13	167	0	361	0	20	0
7	0	0	0	0	0	0	0	718	14	268
8	1	2	26	14	42	0	20	5	889	1
9	0	0	0	0	0	0	0	4	4	992

4.11.

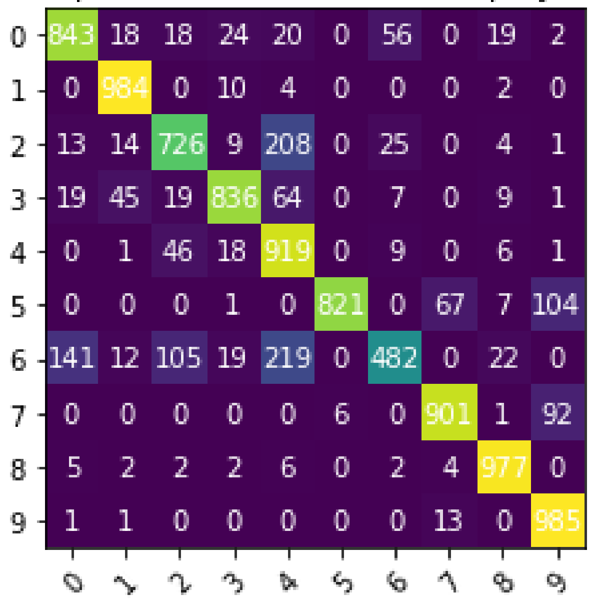
Confusion matrix of the perceptron algorithm
accuracy= %80.17
params= fit_intercept=False, tol=0.0005



	0	1	2	3	4	5	6	7	8	9
0	806	7	9	55	2	0	110	0	11	0
1	5	950	1	31	2	0	8	1	2	0
2	45	4	580	11	91	0	254	1	14	0
3	24	13	6	871	7	0	76	0	3	0
4	6	3	67	61	495	1	358	0	9	0
5	3	0	0	1	0	862	1	87	16	30
6	159	1	69	54	39	0	654	0	24	0
7	0	0	0	0	0	19	0	964	2	15
8	8	0	3	13	2	7	27	4	936	0
9	1	0	0	1	0	32	4	63	0	899

4.12.

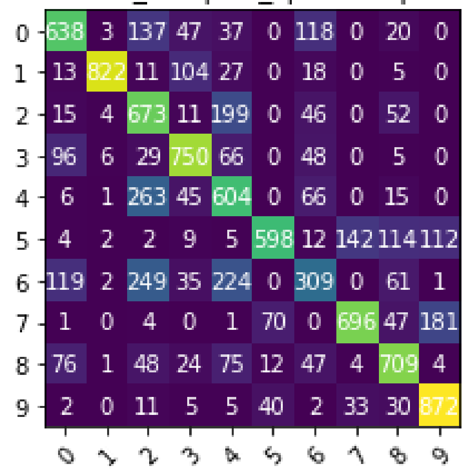
Confusion matrix of the svm algorithm
accuracy= %84.74000000000001
params= C=1.1, kernel='poly'



	0	1	2	3	4	5	6	7	8	9
0	843	18	18	24	20	0	56	0	19	2
1	0	984	0	10	4	0	0	0	2	0
2	13	14	726	9	208	0	25	0	4	1
3	19	45	19	836	64	0	7	0	9	1
4	0	1	46	18	919	0	9	0	6	1
5	0	0	0	1	0	821	0	67	7	104
6	141	12	105	19	219	0	482	0	22	0
7	0	0	0	0	0	6	0	901	1	92
8	5	2	2	2	6	0	2	4	977	0
9	1	1	0	0	0	0	0	13	0	985

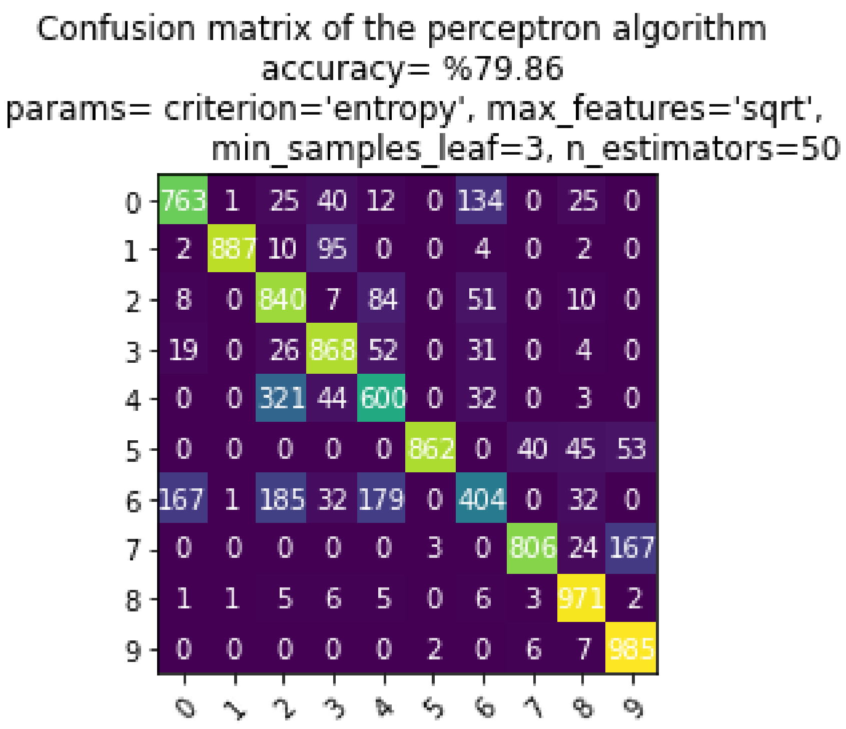
4.13.

Confusion matrix of the perceptron algorithm
accuracy= %66.71000000000001
params= criterion='entropy', max_depth=10, min_samples_leaf=3, min_samples_split=3, splitter='random'



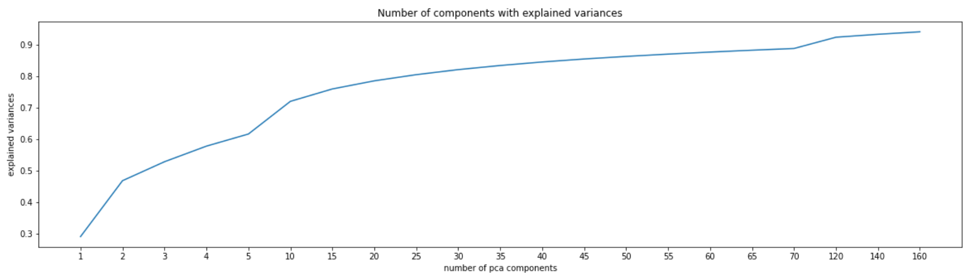
	0	1	2	3	4	5	6	7	8	9
0	638	3	137	47	37	0	118	0	20	0
1	13	822	11	104	27	0	18	0	5	0
2	15	4	673	11	199	0	46	0	52	0
3	96	6	29	750	66	0	48	0	5	0
4	6	1	263	45	604	0	66	0	15	0
5	4	2	2	9	5	598	12	142	114	112
6	119	2	249	35	224	0	309	0	61	1
7	1	0	4	0	1	70	0	696	47	181
8	76	1	48	24	75	12	47	4	709	4
9	2	0	11	5	5	40	2	33	30	872

4.14.

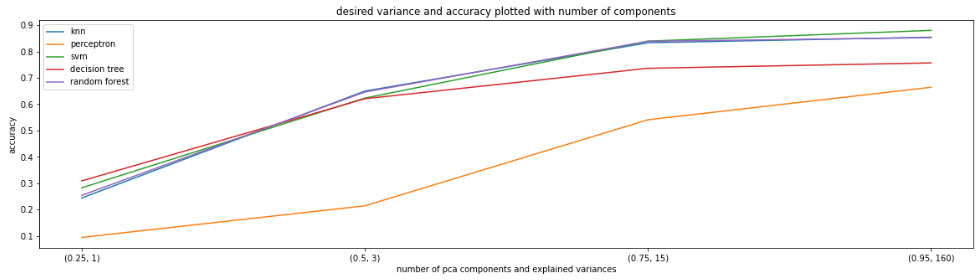


	0	1	2	3	4	5	6	7	8	9
0	763	1	25	40	12	0	134	0	25	0
1	2	887	10	95	0	0	4	0	2	0
2	8	0	840	7	84	0	51	0	10	0
3	19	0	26	868	52	0	31	0	4	0
4	0	0	321	44	600	0	32	0	3	0
5	0	0	0	0	0	862	0	40	45	53
6	167	1	185	32	179	0	404	0	32	0
7	0	0	0	0	0	3	0	806	24	167
8	1	1	5	6	5	0	6	3	971	2
9	0	0	0	0	0	2	0	6	7	985

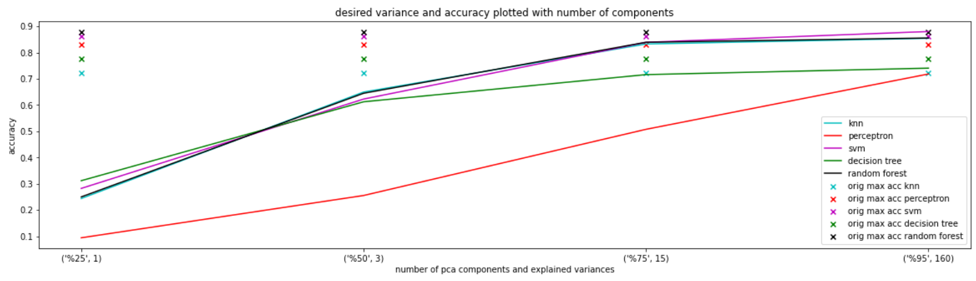
4.15.



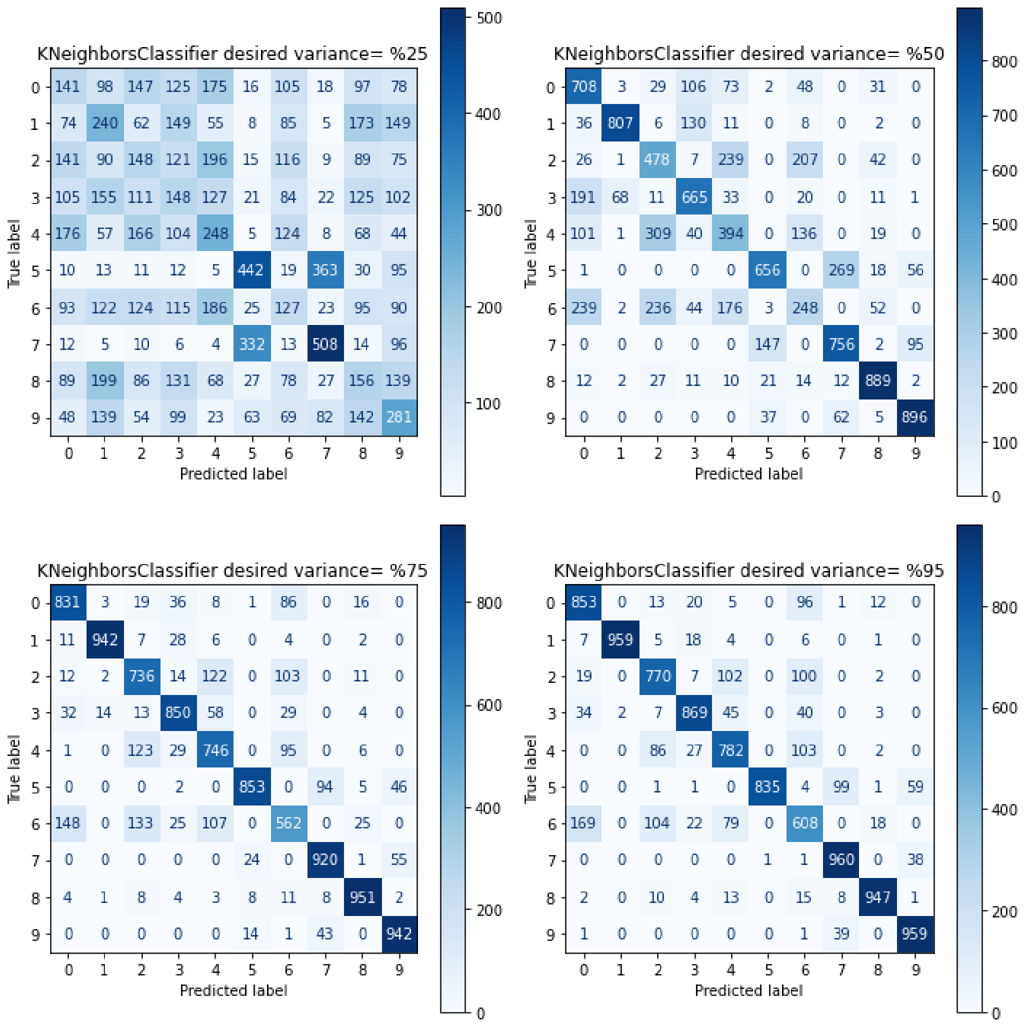
4.16.



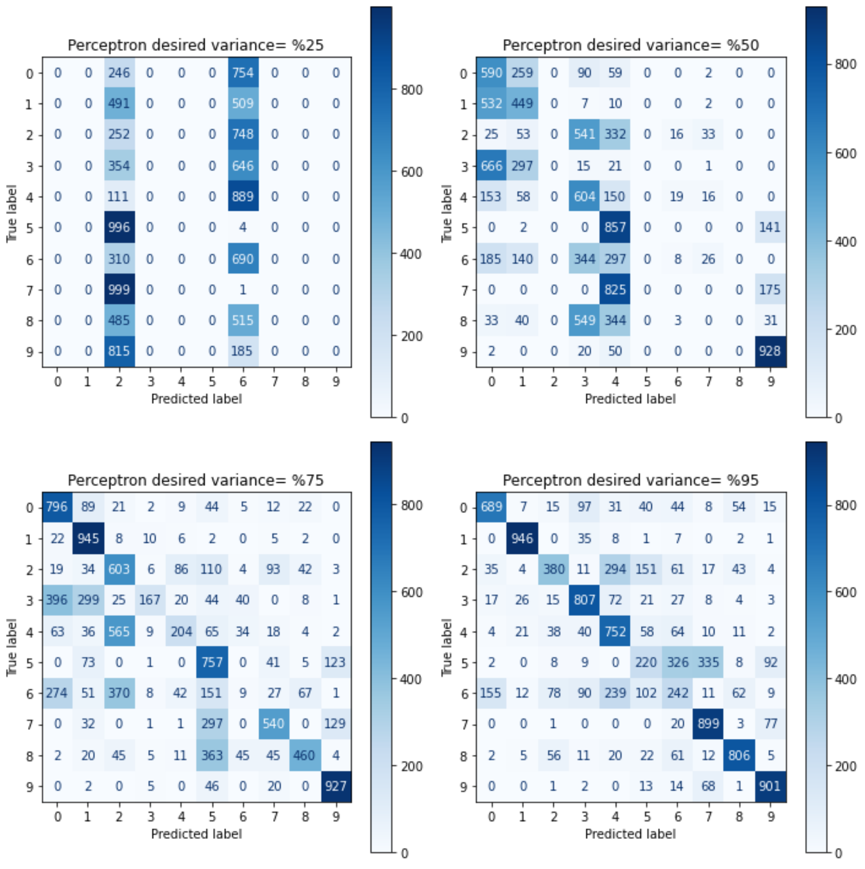
4.17.



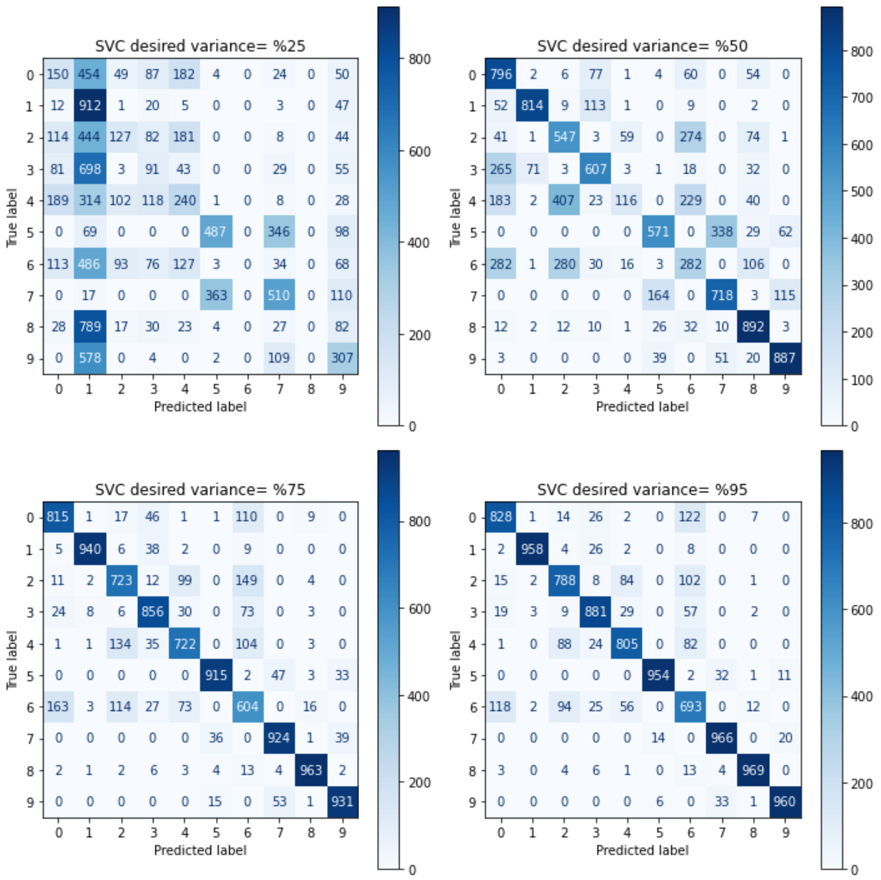
4.18.


KNeighborsClassifier desired variance= %25
KNeighborsClassifier desired variance= %50
KNeighborsClassifier desired variance= %75
KNeighborsClassifier desired variance= %95
True label
Predicted label

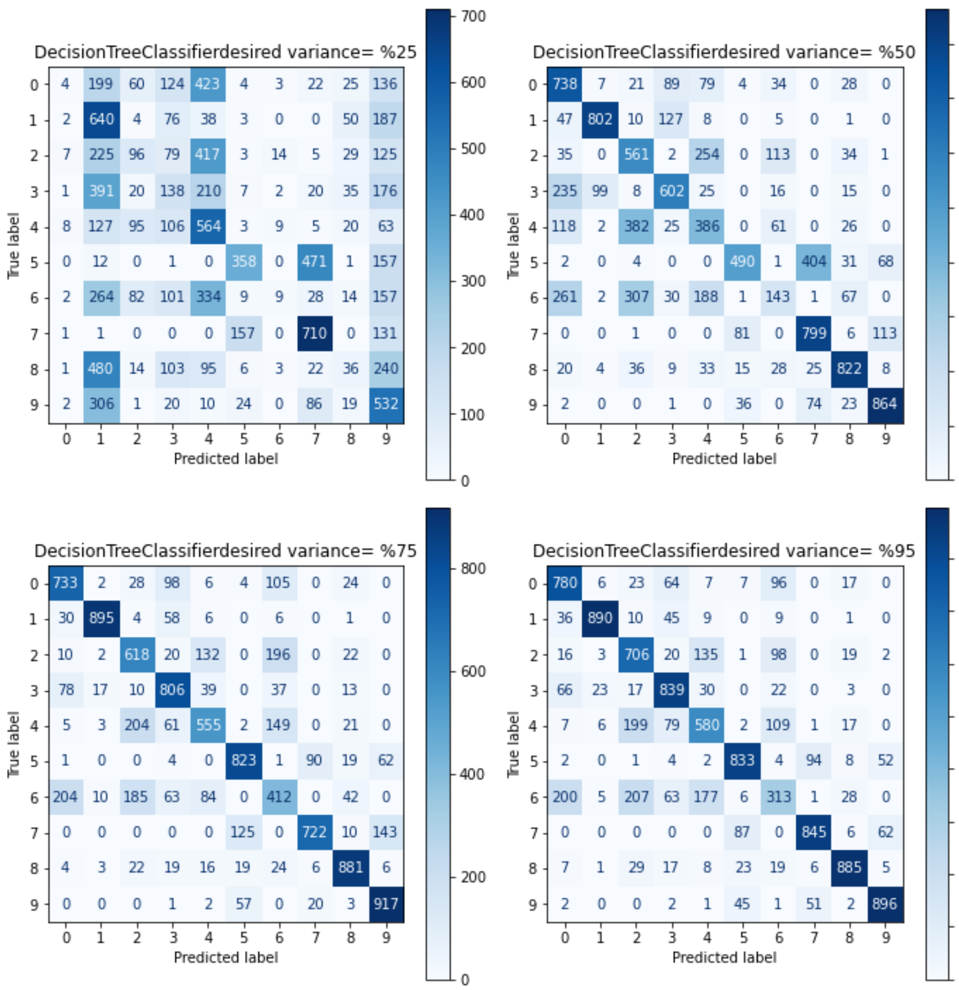
4.19.


Perceptron desired variance= %25
Perceptron desired variance= %50
Perceptron desired variance= %75
Perceptron desired variance= %95
True label
Predicted label

4.20.


SVC desired variance= %25
0: 150 454 49 87 182 4 0 24 0 50
1: 12 912 1 20 5 0 0 3 0 47
2: 114 444 127 82 181 0 0 8 0 44
3: 81 698 3 91 43 0 0 29 0 55
4: 189 314 102 118 240 1 0 8 0 28
5: 0 69 0 0 0 487 0 346 0 98
6: 113 486 93 76 127 3 0 34 0 68
7: 0 17 0 0 0 363 0 510 0 110
8: 28 789 17 30 23 4 0 27 0 82
9: 0 578 0 4 0 2 0 109 0 307
SVC desired variance= %50
0: 796 2 6 77 1 4 60 0 54 0
1: 52 814 9 113 1 0 9 0 2 0
2: 41 1 547 3 59 0 274 0 74 1
3: 265 71 3 607 3 1 18 0 32 0
4: 183 2 407 23 116 0 229 0 40 0
5: 0 0 0 0 0 571 0 338 29 62
6: 282 1 280 30 16 3 282 0 106 0
7: 0 0 0 0 0 164 0 718 3 115
8: 12 2 12 10 1 26 32 10 892 3
9: 3 0 0 0 0 39 0 51 20 887
SVC desired variance= %75
0: 815 1 17 46 1 1 110 0 9 0
1: 5 940 6 38 2 0 9 0 0 0
2: 11 2 723 12 99 0 149 0 4 0
3: 24 8 6 856 30 0 73 0 3 0
4: 1 1 134 35 722 0 104 0 3 0
5: 0 0 0 0 0 915 2 47 3 33
6: 163 3 114 27 73 0 604 0 16 0
7: 0 0 0 0 0 36 0 924 1 39
8: 2 1 2 6 3 4 13 4 963 2
9: 0 0 0 0 0 15 0 53 1 931
SVC desired variance= %95
0: 828 1 14 26 2 0 122 0 7 0
1: 2 958 4 26 2 0 8 0 0 0
2: 15 2 788 8 84 0 102 0 1 0
3: 19 3 9 881 29 0 57 0 2 0
4: 1 0 88 24 805 0 82 0 0 0
5: 0 0 0 0 0 954 2 32 1 11
6: 118 2 94 25 56 0 693 0 12 0
7: 0 0 0 0 0 14 0 966 0 20
8: 3 0 4 6 1 0 13 4 969 0
9: 0 0 0 0 0 6 0 33 1 960
True label
Predicted label

4.21.



DecisionTreeClassifierdesired variance= %25

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	4	199	60	124	423	4	3	22	25	136
1	2	640	4	76	38	3	0	0	50	187
2	7	225	96	79	417	3	14	5	29	125
3	1	391	20	138	210	7	2	20	35	176
4	8	127	95	106	564	3	9	5	20	63
5	0	12	0	1	0	358	0	471	1	157
6	2	264	82	101	334	9	9	28	14	157
7	1	1	0	0	0	157	0	710	0	131
8	1	480	14	103	95	6	3	22	36	240
9	2	306	1	20	10	24	0	86	19	532

DecisionTreeClassifierdesired variance= %50

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	738	7	21	89	79	4	34	0	28	0
1	47	802	10	127	8	0	5	0	1	0
2	35	0	561	2	254	0	113	0	34	1
3	235	99	8	602	25	0	16	0	15	0
4	118	2	382	25	386	0	61	0	26	0
5	2	0	4	0	0	490	1	404	31	68
6	261	2	307	30	188	1	143	1	67	0
7	0	0	1	0	0	81	0	799	6	113
8	20	4	36	9	33	15	28	25	822	8
9	2	0	0	1	0	36	0	74	23	864

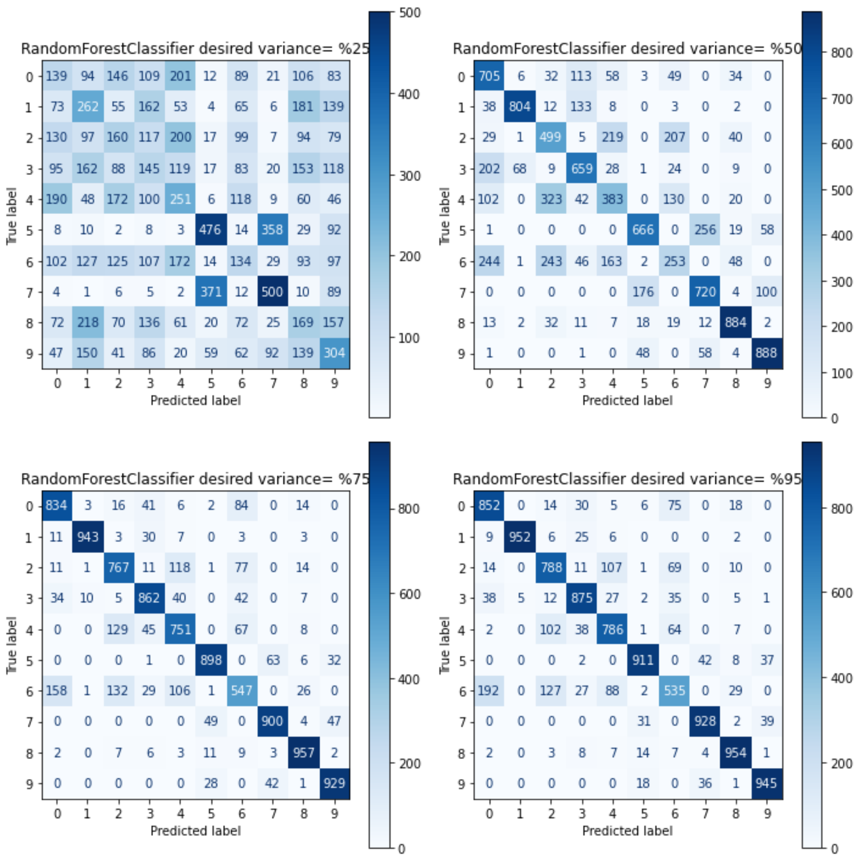
DecisionTreeClassifierdesired variance= %75

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	733	2	28	98	6	4	105	0	24	0
1	30	895	4	58	6	0	6	0	1	0
2	10	2	618	20	132	0	196	0	22	0
3	78	17	10	806	39	0	37	0	13	0
4	5	3	204	61	555	2	149	0	21	0
5	1	0	0	4	0	823	1	90	19	62
6	204	10	185	63	84	0	412	0	42	0
7	0	0	0	0	0	125	0	722	10	143
8	4	3	22	19	16	19	24	6	881	6
9	0	0	0	1	2	57	0	20	3	917

DecisionTreeClassifierdesired variance= %95

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	780	6	23	64	7	7	96	0	17	0
1	36	890	10	45	9	0	9	0	1	0
2	16	3	706	20	135	1	98	0	19	2
3	66	23	17	839	30	0	22	0	3	0
4	7	6	199	79	580	2	109	1	17	0
5	2	0	1	4	2	833	4	94	8	52
6	200	5	207	63	177	6	313	1	28	0
7	0	0	0	0	0	87	0	845	6	62
8	7	1	29	17	8	23	19	6	885	5
9	2	0	0	2	1	45	1	51	2	896

4.22.



RandomForestClassifier desired variance= %25

	0	1	2	3	4	5	6	7	8	9
0	139	94	146	109	201	12	89	21	106	83
1	73	262	55	162	53	4	65	6	181	139
2	130	97	160	117	200	17	99	7	94	79
3	95	162	88	145	119	17	83	20	153	118
4	190	48	172	100	251	6	118	9	60	46
5	8	10	2	8	3	476	14	358	29	92
6	102	127	125	107	172	14	134	29	93	97
7	4	1	6	5	2	371	12	500	10	89
8	72	218	70	136	61	20	72	25	169	157
9	47	150	41	86	20	59	62	92	139	304

RandomForestClassifier desired variance= %50

	0	1	2	3	4	5	6	7	8	9
0	705	6	32	113	58	3	49	0	34	0
1	38	804	12	133	8	0	3	0	2	0
2	29	1	499	5	219	0	207	0	40	0
3	202	68	9	659	28	1	24	0	9	0
4	102	0	323	42	383	0	130	0	20	0
5	1	0	0	0	0	666	0	256	19	58
6	244	1	243	46	163	2	253	0	48	0
7	0	0	0	0	0	176	0	720	4	100
8	13	2	32	11	7	18	19	12	884	2
9	1	0	0	1	0	48	0	58	4	888

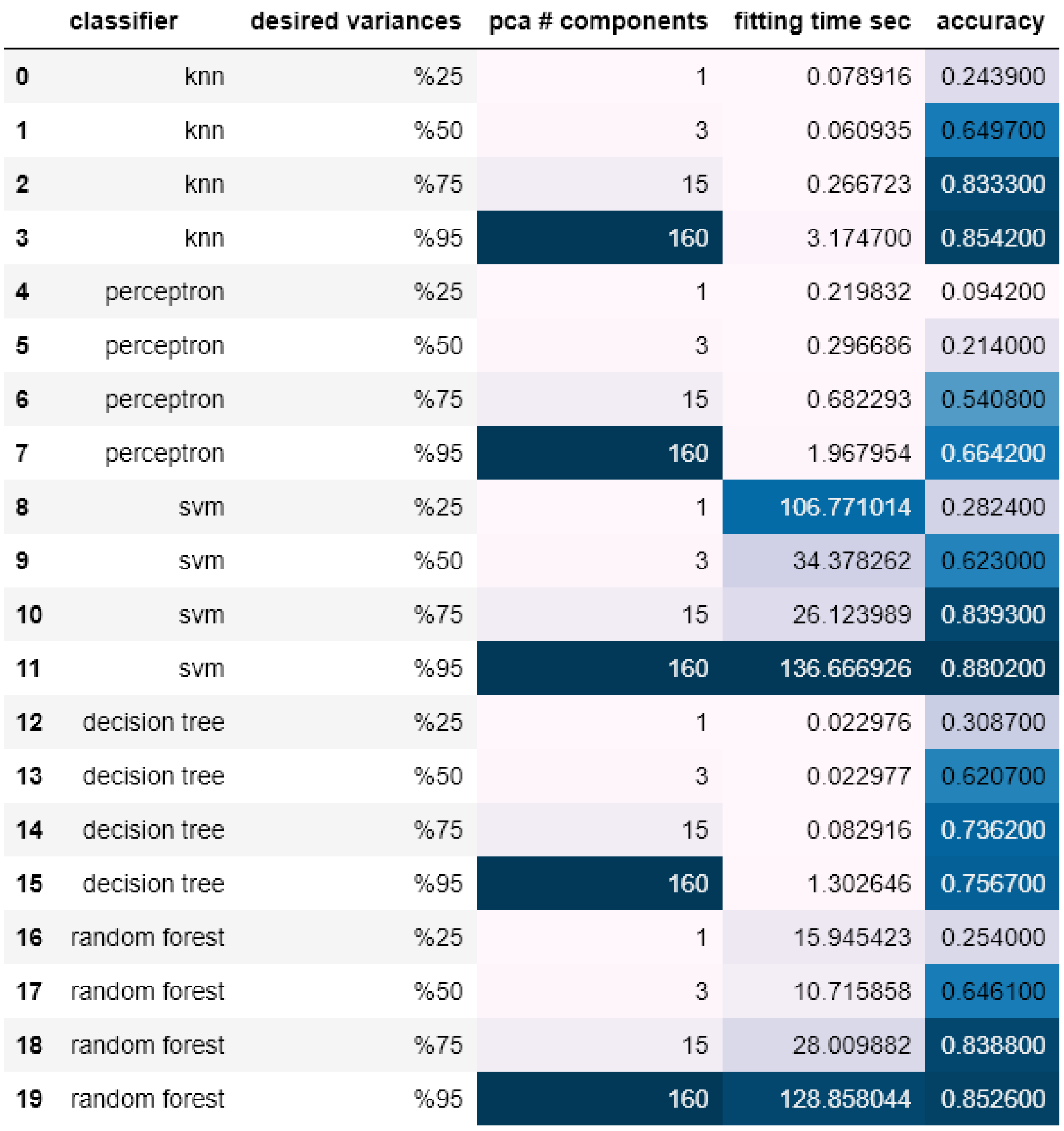
RandomForestClassifier desired variance= %75

	0	1	2	3	4	5	6	7	8	9
0	834	3	16	41	6	2	84	0	14	0
1	11	943	3	30	7	0	3	0	3	0
2	11	1	767	11	118	1	77	0	14	0
3	34	10	5	862	40	0	42	0	7	0
4	0	0	129	45	751	0	67	0	8	0
5	0	0	0	1	0	898	0	63	6	32
6	158	1	132	29	106	1	547	0	26	0
7	0	0	0	0	0	49	0	900	4	47
8	2	0	7	6	3	11	9	3	957	2
9	0	0	0	0	0	28	0	42	1	929

RandomForestClassifier desired variance= %95

	0	1	2	3	4	5	6	7	8	9
0	852	0	14	30	5	6	75	0	18	0
1	9	952	6	25	6	0	0	0	2	0
2	14	0	788	11	107	1	69	0	10	0
3	38	5	12	875	27	2	35	0	5	1
4	2	0	102	38	786	1	64	0	7	0
5	0	0	0	2	0	911	0	42	8	37
6	192	0	127	27	88	2	535	0	29	0
7	0	0	0	0	0	31	0	928	2	39
8	2	0	3	8	7	14	7	4	954	1
9	0	0	0	0	0	18	0	36	1	945

4.23.



	classifier	desired variances	pca # components	fitting time sec	accuracy
0	knn	%25	1	0.078916	0.243900
1	knn	%50	3	0.060935	0.649700
2	knn	%75	15	0.266723	0.833300
3	knn	%95	160	3.174700	0.854200
4	perceptron	%25	1	0.219832	0.094200
5	perceptron	%50	3	0.296686	0.214000
6	perceptron	%75	15	0.682293	0.540800
7	perceptron	%95	160	1.967954	0.664200
8	svm	%25	1	106.771014	0.282400
9	svm	%50	3	34.378262	0.623000
10	svm	%75	15	26.123989	0.839300
11	svm	%95	160	136.666926	0.880200
12	decision tree	%25	1	0.022976	0.308700
13	decision tree	%50	3	0.022977	0.620700
14	decision tree	%75	15	0.082916	0.736200
15	decision tree	%95	160	1.302646	0.756700
16	random forest	%25	1	15.945423	0.254000
17	random forest	%50	3	10.715858	0.646100
18	random forest	%75	15	28.009882	0.838800
19	random forest	%95	160	128.858044	0.852600

4.24.

5. Conclusion

In this assignment we used 5 different algorithms and tuned their parameters on the fashion MNIST dataset to find the most optimal algorithm. From the plot 4.1 which plots the accuracies reached with different parameters for KNN we see that we just gain %1 accuracy by tuning all the parameters other than number of neighbors from this we can conclude that the most important parameter to tune for KNN is number of neighbors. From the plot 4.3 we can see that parameter tuning is very important for perceptron algorithm as can be seen on the plot the accuracy for the worst performing algorithm is %10 but with optimal parameters this number jumps up to %80. From the same plot we can conclude that the most important hyper parameter to tune is alpha which determines the regularization coefficient. For the worst performing perceptron algorithm at %10 accuracy this means we smoothed the model weights so much that we have caused under fitting. If we were to tune this parameter to much lower numbers we would have seen overfitting effects although this was not captured on the plot. From the plot 4.5 we can conclude that after we determine the kernel dimensionality the accuracy stays relatively same which means a function with degree 1 is not enough to separate the values efficiently, because of training time of each SVM algorithm at my machine was 10 minutes I could not tuned the C parameter optimally to draw a conclusion. From the plot 4.9 we can conclude that the most important feature for the hyper parameters tuned is min samples leaf, at min samples leaf 1 we are overfitting our model, from the same plot we can conclude that min samples leaf 3 is the sweet spot for this parameter. After examining confusion matrices at 4.11 – 4.15 the most wrongfully classified class is shirt with it classified as a top pull-over coat. All the algorithms tested had the lowest accuracy on this class. And conclusion is simple after checking figures 4.2, 4.4, 4.6 and 4.8 we can see that these classes are closely related to each other and even for a human I is difficult to differentiate. We used PCA to reduce the dimensionality of the dataset. Checking plot 4.15 we can see that with just 15 PCA components we can determine that %75 explained variance is reached and from the table above at 4.23 we can conclude that on average we are losing %2 of accuracy but we are gaining 6 times faster algorithms at both training and prediction times. If time was a concern, we would prefer to use 15 PCA components. As can be seen on the plot at 4.17 we can conclude that at %95 explained variance the accuracy of each algorithm is almost same. With reduced time on both training and predicting since we reduced the number of features from 784 to 160. From the table above the highest accuracy we have reached after applying PCA is %88 with SVM. But after examining the fitting time we can see that it also has the highest training time. If we were only concerned in accuracy, it would be appropriate to use this algorithm. But if training and predictions or interpretability of the algorithm was important we would have preferred the decision tree algorithm since drop in accuracy is close to %3 with fitting time being 100 times faster.