# Algorithm Selection

I have chosen to implement K-nearest neighbours (K-nn), random forests, and support vector machines (SVM) upon the fashion-mnist dataset using Python with scikit-learn.

These algorithms have different approaches for classification problems. K-nn calculates the distance of a test case with its k nearest neighbours and does a majority vote to classify it. Random Forests is an ensemble method based on decision trees. Support vector machines are maximum margin classifiers, so they create a decision boundary between different classes.

For K-nearest neighbours, I will explore the number of neighbours (k) used in the algorithm. This is because the algorithm will find the k nearest training samples to the test case and predict the label from these based upon a majority vote. In random forests, I will experiment with the number of trees (n\_estimators) as the algorithm uses averaging of the decision tree classifiers to improve the accuracy and reduce over-fitting. As for support vector machines, I will investigate the use of different kernels which reduces the complexity of finding the mapping function in the algorithm. After finding the kernel with the most optimum results, I will further experiment on c, the regularisation parameter.

**Methodology**

To train the data I imported the mnist-reader from the fashion-mnist git to load the training sets. Then I used scikit-learn’s python library to import the algorithm models and fit them to the dataset. To reduce the run-time of the algorithms (especially for SVM), I implemented PCA to the dataset which reduces the dimensionality of the dataset. I also chose the minimum number of principle components such that 95% of the variance is retained, this is so that accuracy is not significantly affected. To apply PCA, I also normalised the data, so that all the independent variables are between 0 and 1 which is important for SVM and K-nn. PCA significantly reduced runtime in SVM.

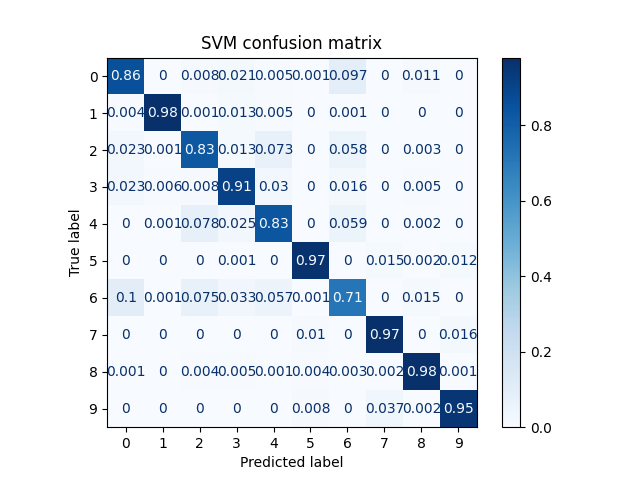
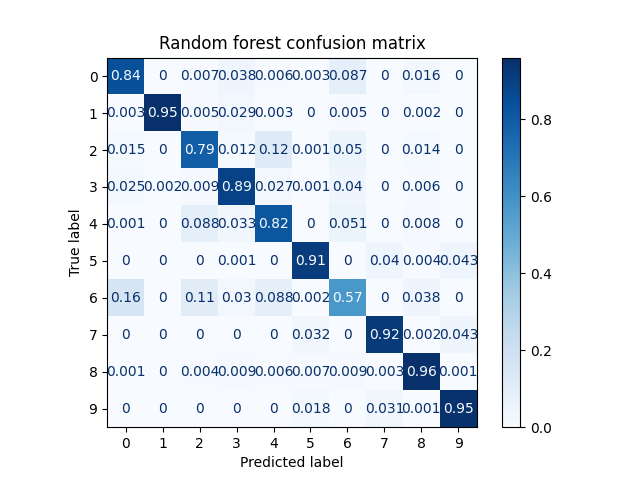
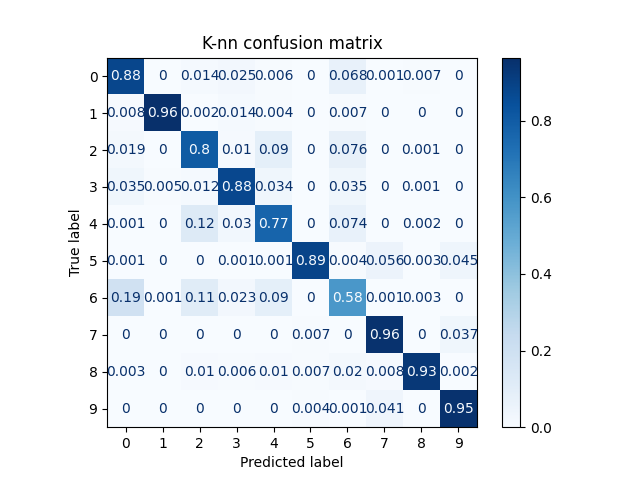
For all tests, I continue testing than initially planned until classification accuracy does not improve, I also repeat my test 3 times and calculate an average so that errors in my test results are minimised. For k-nn, I started from k=1 to 10 in increments of 1. For random forests, I started from n\_estimators= 50 to 150 in increments of 10. I then tested values outside of that scope from 1 to 1000 but in increments of 1000. SVM has 4 different usable pre-set kernels which I have tested. I then test c from 1 to 15 on the kernel with the best initial accuracy.

I imported a module from the scikit-learn library that measures the overall classification accuracy of the algorithms. To see the classification accuracy of each class, I used the confusion matrix classification metric in scikit-learn, which also shows which class is confused for other classes.

**Results**

The best classification accuracies for each algorithm:

* K-nn (k=6): 0.8611
* Random forest (n\_estimators=500): 0.8614
* Support vector machines (kernel= “rbf”, c=13): 0.9

Below are graphs of the confusion matrix of each algorithm with their optimised hyperparameters:

All the algorithms struggled with classifying a shirt, having the worst classification accuracy, and commonly mistaking it for a T-shirt or pullover. This is understandable because they have the same shape and are different in materials which the algorithms cannot identify properly. SVM did better at classifying shirt by at least 13 percentage points compared to the other methods. All of them also confuse pullover and coat with each other and consequently they have the 2nd and 3rd worst classification accuracy. The classes with the best accuracy are trouser, sneaker, bag, ankle boot and sandal which surprises me because I thought the algorithms would get confused between the footwears. SVM has better classification accuracy for every class compared to the other methods except for ankle boot where they all have the same accuracy and T-shirt. SVM can identify 5 classes within a 5% margin of error whereas K-nn and Random forests can identify 3 classes within a 5% margin of error and has a wider range of classification accuracy compared to SVM.

**Conclusion**

For k-nn, accuracy from k=1 to 6 increased by 1.44 percentage points meaning that the algorithm is insensitive to the hyperparameter. From k=6 to 8, accuracy stays the same. After that accuracy slightly decreases or stagnates, this could be because having a higher k means that the boundaries of different classes are less distinct. Having a larger k also means that noise in the data affects the classification less as well as over-generalising whereas having a small k leads to over-fitting. K-nn had a slow runtime because it does not actually train the data and so completing predictions takes a lot of time, it would likely be worse for datasets of higher dimensionality and large datasets as well as requiring a lot of memory.

As for random forests, at a certain point of number of trees, the classification accuracy stays the same regardless of more trees. After 120 trees, accuracy stayed roughly the same, the best accuracy I found was at 500 trees, but it was minute compared to the accuracy at 120 trees. Hence, random forests are very insensitive to the number of trees after a certain point. Increasing the number of trees reduces the variance and over-fitting as random forests take the majority vote of the predictions. Random forests had the shortest runtime possibly due to running the trees concurrently.

For SVM, the kernel “rbf” was the best choice because it gave the highest classification accuracy. The second-best kernel was “poly”, “rbf” was more accurate by the slightest amount, followed by “linear”, “sigmoid” was the worst out of all of them. This means that SVM is sensitive to the kernels. This is because the kernel function is used for the calculating in higher dimensions of space without explicitly doing so. Choosing the right kernel with the right parameters that fits the data helps reduce over-fitting. The regularisation parameter c had the best result at 13. The parameter c, common to all SVM kernels, trades off misclassification of training examples against simplicity of the decision surface. A high C aims at classifying all training examples correctly. This I might be why at c= 13 it gave me my highest classification accuracy. The accuracy after c=13 decreases slightly or stagnates, runtime rapidly increases. Accuracy from c=1 to 13 increases by 1.59 percentage points. meaning the algorithm is insensitive to the hyperparameter. However, increasing c corresponds to less regularisation. Even with PCA, SVM had a slow runtime since it takes a lot of time training the data possibly due to forming boundaries and support vectors from the data set.

Support vector machines is my best preforming algorithm. The algorithm is effective in high dimensional space because it uses the kernel trick. The kernel trick is a way to map the non-linear dataset into a higher dimensional space where we can find a hyperplane that can separate the data. This trick is often computationally cheaper than computing the coordinates explicitly. This works well for our dataset as it is a high number of dimensions because they are pixels of an image.

However, without PCA, I would reconsider SVM as the best performing algorithm because it is computationally expensive, and it takes a long time compared to the other algorithms for 4 percentage points better classification accuracy. Random forests are not computationally costly and is insensitive to the hyperparameter I have tested if I did not implement PCA, I would use this algorithm.