COMP5318 Assignment 1

Tutors: Baosheng Yu, Xiyu Yu

Group members: Xiaowei Wang (xwan6155) - 450095203
Ereina Camille Gomez (egom8541) - 410025257
Azadeh Arnaz (anas6977) - 440583529

Fashion-MNIST Image Classification with KNN and Random Forest

Abstract - The k-Nearest Neighbours (KNN) is a simple yet effective method for classification. A major drawback with respect to KNN is the time cost for classifying new samples. kd-Tree KNN, is a modification from brute force KNN where training samples are partitioned and organised in a tree-like structure. Preliminary experiments revealed brute force KNN had a faster running time compared to kd-Tree KNN at dimensions of 39 and above, therefore further optimisation was done on kd-Tree KNN efficiency. Random Forest is an ensemble learning method where a stronger learner is created by employing several weaker decision tree learners. Experiments were carried out on the Fashion-MNIST dataset and KNN classifier's performance was compared against *sklearn*'s Random Forest. Experimental results showed that KNN outperformed Random Forest in terms of classification accuracy after PCA and k-neighbours hyper-parameter tuning. Furthermore, the kd-Tree optimised KNN had the fastest running time over all classifiers.

1 Introduction

Supervised image classification remains an important problem in Computer Vision. Using machine learning methods for automating classification is motivated by the ubiquity of large amounts of data and their applications[1]. The Fashion-MNIST dataset was introduced as a more challenging dataset to its predecessor MNIST, with the aim of functioning as a benchmark for various machine learning methods[2]. The dataset consists of 28x28 greyscale images of clothing articles from 10 different classes. As classifiers on MNIST have been shown to achieve accuracies above 99.7% [3], evaluating performance accuracy using Fashion-MNIST allows for granularity in performance tuning and comparisons across classifiers. A more challenging dataset such as the Fashion-MNIST is important for evaluating how classifiers perform with highly variational image data.
The aim of our study is to assess the performance accuracy of a KNN classifier on the Fashion-MNIST and compare its performance accuracy to that of the Random Forest classifier. We explore Principle Component Analysis (PCA) for feature dimensionality reduction, tuning hyper-parameter k, the time cost of brute and kd-Tree variants of KNN and compare KNN efficiency and classification accuracy to Random Forest.

2 Method

2.1 Preprocessing

The Fashion-MNIST dataset represents each image as their 28x28 pixel values, resulting in 784 dimensions or a vector of length 784 in the data matrix. The motivation for dimensionality reduction for this task can be summarised by two benefits: (1) minimise overfitting from noisy features and redundancy from highly correlated features (2) to reduce the computational time required during classification while still preserving useful information from the original dataset[4].

2.1.1 Data Centralization by Mean

Data centralization is not a requirement for principal component analysis, however it can simplify notation and computation. To be more specific, the definition of correlation variance for a random variable X is:

$$C = \sum_{i=0}^{n} (x_i - \mu_x)(x_i - \mu_x)^T$$

If we centralize data by subtracting mean μ_x, the equation above can be simplified as :

$$C = \sum_{i=0}^{n} x_i x_i^T$$

Furthermore, in terms of our case, the grayscale value of each pixel is in the range between 0 and 255, hence there is no need to scale the data.

2.1.2 Principal Components Analysis (PCA)

PCA is an effective dimension reduction algorithm that could be applied in fields such as face recognition and image compression and is a common technique for finding patterns in data of high dimension. In our case, the dataset has 784 dimensions resulting in lots of redundant data. Therefore, PCA could improve model training efficiency significantly.

Method Description:

1. Subtract the Mean

As mentioned in 2.21, we have to subtract the mean from each of the data dimensions. The mean subtracted is the average across each dimension.

$$\mu = \frac{1}{n} \times \sum_{i}^{n} x_i$$

2. Find covariance matrix

$$S = \frac{1}{n}(x_i - \mu)(x_i - \mu)^T$$

3. Calculate the eigenvectors and eigenvalues of the covariance matrix

Since the covariance matrix is square, we can calculate the eigenvectors and eigenvalues for this matrix by eigen decomposition.

$$Sv_i = \lambda_i v_i, i = 1, 2, 3...., n$$

4. Select n components based on explained variance ratio and forming a feature vector

We could sort the eigenvectors as descending order by their eigenvalue and choose top k eigenvectors as feature vectors

5. Apply PCA algorithm to new Data

At first, we need to subtract the mean.

$$\Gamma = \gamma - \mu$$

Then, project this normalized probe onto the the collection of Eigenvectors by matrix multiplication

$$FinalData = u^T \Gamma$$

It will give us the original data solely in terms of the k-vectors we chose. Therefore, the dimension of original data （*n_samples* ✕784） will be reduced to （*n_samples* ✕ *k*）.

Choose number(k) of Components:

The total variance is the sum of variances of all individual principal components and the fraction of variance explained by a principal component is the ratio between the variance of that principal component and the total variance.

Although PCA maximizes the variance of the first k components, and minimize the variance of the last p−k components, There is a trade-off, we want to choose k big enough to make the lost information whereas we also want to keep k relatively small to compress images.

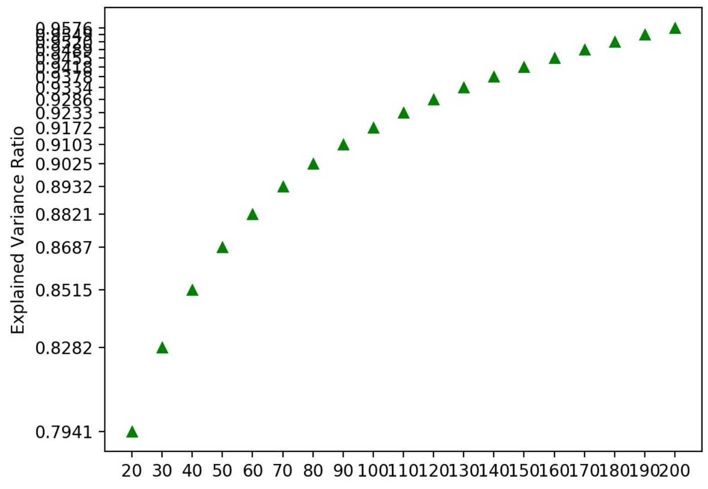


Figure 1(x-axis is number of components; y-axis is explained variance ratio)

As shown in ***Figure 1***, the first 20 PCs explain 79.41% of variance. If we choose first 20 PCs to fit KNN classifier, the accuracy is 82.95%. However, if we chose first 95 PCs to fit KNN classifier, the accuracy is 85.95%, which implicates that data that have low dimensions may have negative impact on classifier.

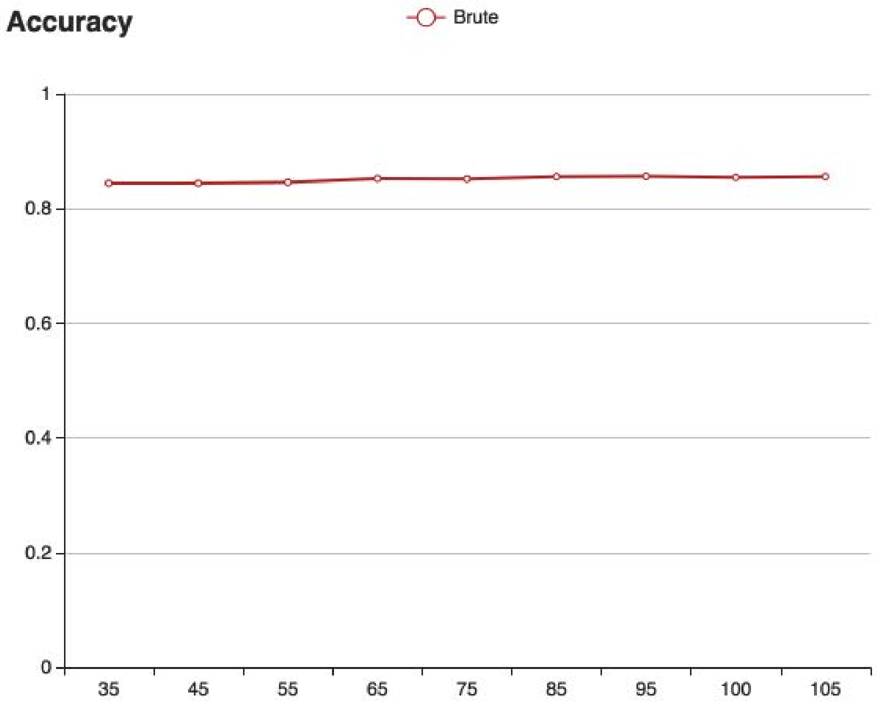


Figure 2 (x-axis is number of components; y-axis is accuracy score of knn classifier)

Figure 2 demonstrates that if the number of PCs is greater than 35, it will have minor impact on accuracy of classifier. However, the dimension could have impact on efficiency of classifier. This issue will be discussed in ***Section 3.2***.

Visualizing PCA:

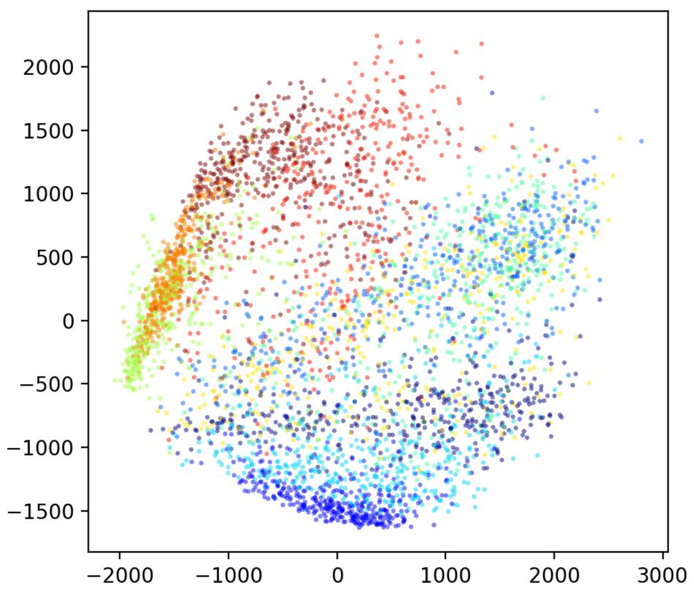


Figure 3(a visualization of PCA with 2 PCs. Different colors indicates different categories)

As shown in ***Figure 3***, with 2PCs (which explains 46.81% of variance), we can still find some interesting patterns. For instance, the dark blue points (label 1 - trousers) are plotted at bottom of the figure.

2.2 Classifier

2.2.1 KNN Classifier

The KNN algorithm is a non-parametric method used for classification and regression problems. For a classification problem, it works by first calculating the distance between a test data point to all data points in the training dataset for which the classes are already known. In our implementation, similarity of neighbours is measured by the Manhattan distance metric where *n* denotes total number of dimensions.

$$D_{manhattan} = \sum_{i=1}^{n} |x_i - y_i|$$

The *k* closest training examples are then identified and it classifies the test sample with the most popular label of these *k* neighbours. We denote uniform weights across all neighbours, meaning that each neighbour's label equally contributes to reaching consensus regardless of the specific value of its distance (i.e for *k* > 1, some neighbours may be closer than other neighbours but all are equally important in determining the label of the test set).

Implementation of KNN:

1. For every picture in our database, compute its associated vector.
2. When the k-Nearest Neighbours for a picture are requested, compute its similarity to every other picture in the dataset.
3. Sort the pictures by ascending similarity.
4. Return the last k elements.

Advantages

- Simple to implement
- Lazy learning equates to fast training stage
- Effective with a large enough training dataset
- Classes don't have to be linearly separable

Disadvantages

- Time to compute distances during test stage can be very slow with large number of training samples
- Can be sensitive to class imbalances and noisy features[5]

With respect to the advantages and disadvantages of KNN, we hypothesise given the dataset is balanced we suspect KNN with PCA transformed matrix (reducing noisy features) will perform better than no-PCA. Furthermore, we anticipate a larger training time for no-PCA with KNN due to the large dimensionality. With regards to *k* , we suspect a *k* > 1 will produce better accuracy than when *k* = 1 due to more local knowledge within the search space making it more robust to noise. In the experiment section we test different values for hyper-parameter *k*.

2.2.2 KNN optimisation with kd-Tree

A kd-Tree is similar to a decision tree except that we split using the median value along the dimension having the highest variance. Every internal node stores one data point, and the leaves are empty.

Implementation of kd-Tree k nearest search:

1. Construct A kd-Tree

- If there is just one node, set that node as a leaf node.
- Otherwise, divide the nodes in half by a line perpendicular to one of the axes(dimension).
- Recursively construct k-d trees for the two sets of points.

2. Perform K Nearest neighbour Search on kd-Tree

- Traverse the whole tree from the root,prune to search space when:
    1. Keep variable of closest point C found so far. Prune subtrees once their bounding boxes say that they can't contain any point closer than C (we could maintain a heap queue to simplify the implementation)
    2. Search the subtrees in order that maximizes the chance for pruning

Computation Complexity of simple KNN and kd-tree Knn:

As mentioned by Section 2.2.1 , complexity of a simple knn search with brute-force is :

$$O(nd) + O(nd) + O(nlogn) + O(k)$$

Where n is the number of images and d is the dimension in our case. k is very small compared to n and d, so we can neglect $O(k)$.
In terms of kd_Tree KNN, the nearest neighbour search is similar to binary search, which complexity is $O(logn)$, hence the complexity of k nearest search is $O(klogn)$. Constructing a kd-Tree has $O(nlogn)$ complexity.

Drawbacks of kd-tree:

Kd-tree provides a more efficient way to find k nearest neighbour. However, due to the Curse of Dimensionality, when k-d trees are used with high-dimensional data, most of the points in the tree will be evaluated and the efficiency is no better than exhaustive search[5]. In ***section 3.2***, we will discuss how dimension has influence on efficiency.

2.2.3 Random Forest Classifier

To deepen the analysis of our KNN classifier's performance on the dataset, we will compare it to *sklearn*'s implementation of Random Forest. We chose Random Forest for comparison as it has shown good performance accuracy across a range of image classification problems[6].
Random Forest is a non-parametric ensemble learning method that makes use of several weak (decision tree) learners to form a stronger learner through majority vote[7].

Implementation of Random Forest:

```
1. for $i$ to $T$ number of trees:
        B = Get $n$ number of samples bootstrapped with replacement from original dataset X
        Build decision tree using B
```

- at each node, randomly select *m* number of features
- split by finding best split using gini criteria across *m* features
- terminate if gini = 0, max_depth is reached, or size of partition is less than the minimum sample split threshold

2. Predict test data by obtaining the majority prediction of all T trees

Finding the best split is done through gini impurity measure. If a dataset T contains examples from n classes:

$$Gini(T) = 1 - \sum_{j=1}^{n} (p_j)^2$$

Where p_j is the relative frequency of class *j* in T. A Gini coefficient of 0 expresses perfect equality, where all the samples in T are of the same class.

Advantages

- Fast test classification after already defined rules built during training
- Easy to incorporate both categorical and numeric features
- Bootstrapping helps with smaller datasets
- Feature sampling reduces overfitting and highly correlated trees

Disadvantages

- Training phase can be slow with large amount of samples and can be memory intensive
- Splits are made using local knowledge at every node which does not necessarily result in global optima
- If just using majority votes as the prediction rule, imbalanced classes will affect predictions

Cross-validation was done on the hyper-parameter number of trees (see Appendix). It was observed that number of trees set to 100 resulted in the best accuracy, which is the setting that will be used for comparison experiments with our KNN classifier.

3 Experiments and Results

3.1 Hyper-parameter *k* selection

We used 10-fold cross validation and obtained the averaged accuracy of the 10 folds for 8 different choices of k values without PCA. As seen in Figure 4, the highest mean overall accuracy of 84.55% is when k = 5, so we set k to 5 for future experiments. Based on this we suspect this is the optimal value that balances the bias-variance trade-off.

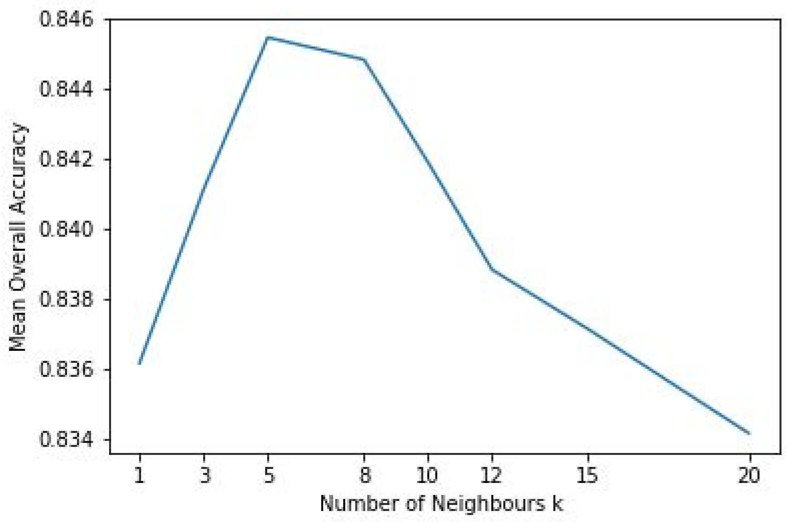


Figure 4(Cross validation accuracy by number of neighbours k)

3.2 PCA versus no PCA

As discussed in ***section 2.1.2***, experimentation with different k components ranging from 35 to 105 showed no significant difference in accuracy scores for brute-force KNN. Therefore, we choose the number of k components based on a trade-off between accuracy and computation time.

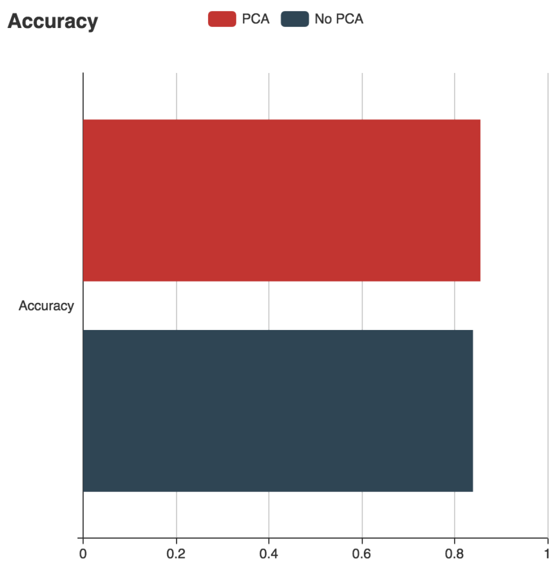


Figure 5.1(with 75 PCs, accuracy score of knn classifier,

PCA 85.85%; No Pca 85.25%)

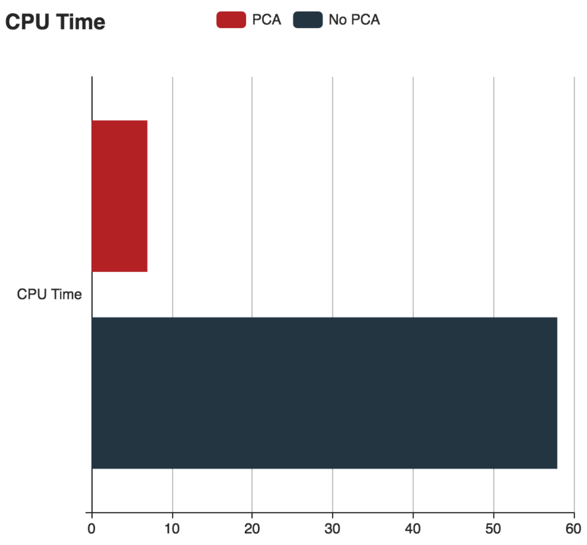


Figure 5.2(x-axis is the CPU time, pca - 6.885 seconds
No pca - 60.805 seconds)

Figure 5.1 and ***5.2*** show that arbitrarily choosing 75 components resulted in a performance accuracy on-par with that of no-pca(PCA = 85.85%, No-Pca = 85.25%). Furthermore, pca resulted in more significant gains in terms of CPU time, 75 components had an execution time of 6.885 seconds compared to 60.805 seconds without pca.
As we had seen no significant accuracy gains using k components in the range of 35 to 105, we seeked to exploit the component(s) with the least observed execution times. In **Figure 6.1** we observe that CPU time generally increases with the number of components . We decided to set number of components = 60, which roughly accounts to 88% of explained variance of the original matrix with an execution time of around 5 seconds.

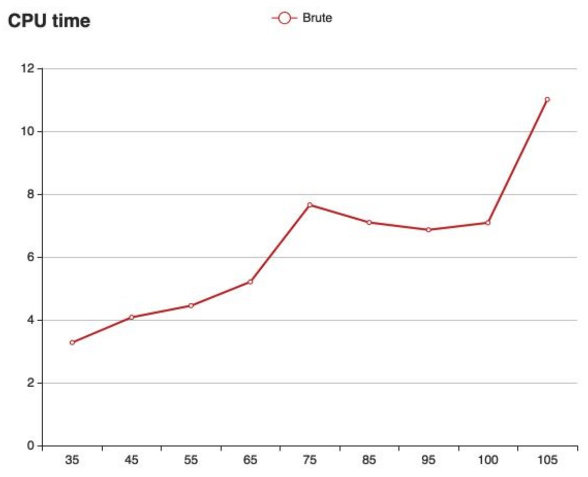


Figure 6.1(x-axis: num of components, y axis : cpu time)

3.3 Time complexity analysis

Preliminary experiments

To evaluate our optimisation of brute KNN, we further analysed CPU time as a function of number of components. Shown in **Figure 6.2**, we found that at PC = 39, execution time of kd-tree surpasses brute KNN. This suggests that the run-time of kd-tree increasing with number of dimensions. Based on these observations, we decided to re-work the implementation to speed up computation.

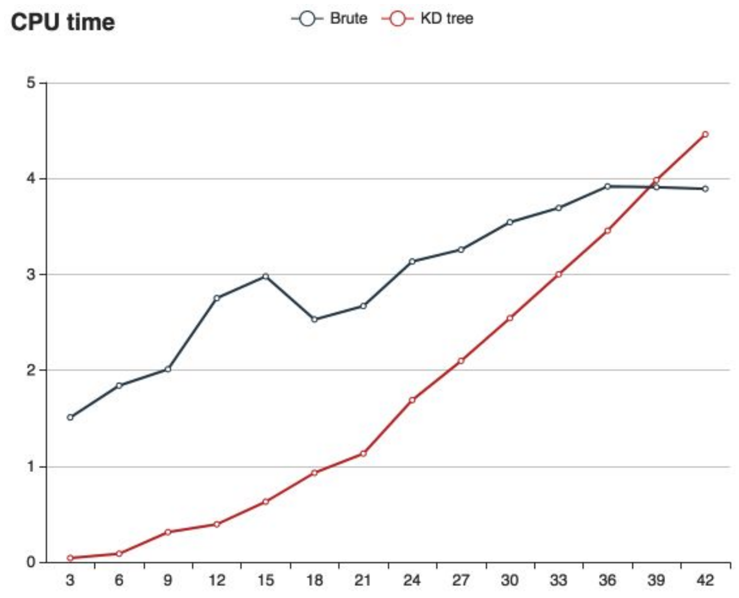


Figure 6.2 (x-axis = num of components, y-axis = cpu time in seconds)

kd-Tree Optimisation

Through code inspection, a bottleneck was found in the tree recursion function resulting in very large recursion depths and this was fixed. kd-Tree was re-tested and resulted in an execution time of 5.77 seconds with 60 PC components.

3.4 Overall Accuracy

Experiments were conducted on 5000 unseen test samples, from which the 2000 are evaluated for classification accuracy. Overall accuracy was defined as the total number of correct predictions over the total number of samples. Run time reflects the total time to train the full training set and classifying 5000 samples of the test set.

Table 1: Accuracy and macro-averaged Precision,Recall and F-scores

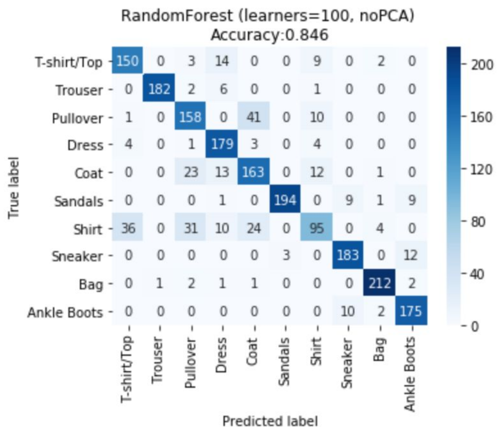
	Kd-tree KNN(k = 5, PCA = 60)	brute KNN (k = 5, PCA = 60)	RandomForest (learners=100, noPCA)	RandomForest (learners=100, PCA=60)
Accuracy	85.25%	85.25%	84.55%	81.55%
Precision	85.36%	85.36%	84.64%	81.64%
Recall	85.36%	85.36%	84.54%	81.40%
F-score	85.13%	85.13%	84.22%	81.24%
Time (sec)	5.80s	432.3	25.8	19.9

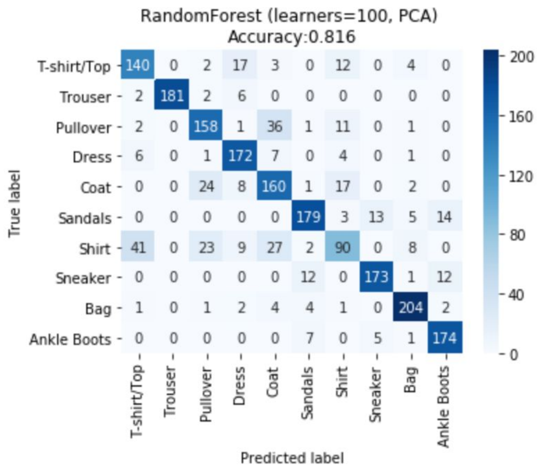
As shown in Table 1, the brute KNN classifier achieved the highest overall accuracy (85.25%) and out-performed Random Forest, exhibiting a higher Precision, Recall and F-score.
In addition, the overall accuracy of kd-tree and brute are identical, because kd-tree had no impact on the prediction. However kd-tree can search top k nearest image way more efficiently when dataset has low dimensions. As shown in the last row of Table 1, kd-tree KNN outperformed brute-Knn and was 75x faster to classify the images. Random Forest with noPCA performed achieved a higher accuracy (84.55%) than Random Forest PCA (81.55%), however PCA was faster than Random Forest noPCA and brute KNN.

3.4 Extensive Analysis

Confusion matrices were created to display predicted labels against the true labels for each classifier. Correct predictions lie on the diagonal of the matrix.
As shown in **Figure** 7, the class *shirt* was the most difficult to predict for Random Forest classifiers. There is an apparent trend that true class *shirt* was often confused with *T-shirt/Top*, *Pullover* and *Coat* classes. Random Forest without PCA performed worse and this is reflected with generally higher values outside of the diagonal. One confusion matrix is created for KNN as both kd-tree and brute implementations resulted in the same predictions. performed slightly better in predicting the *shirt* and *pullover* classes (**Figure 8**), yet still exhibited similar patterns of misclassification such as the overlapping of *T-shirt/Top*, *Pullover* and *Coat* classes and weakest class-wise performance in classifying *shirt*.

Figure 7 Confusion Matrices for Random Forest





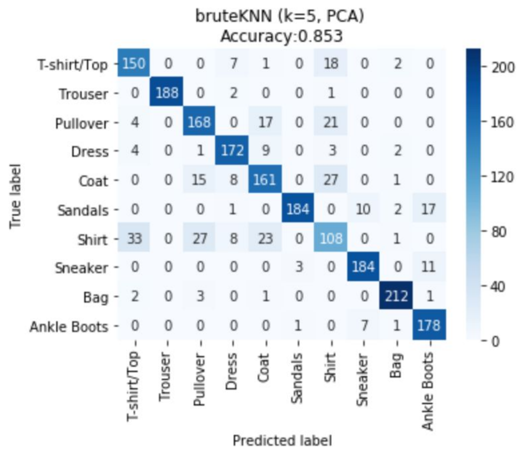


Figure 8 Confusion Matrix for brute KNN

4 Discussion

KNN

The longer running time for brute KNN is suspected to be due to large dimensionality and the large training sample size that needs to be accessed for every test prediction. As a result, kd-Tree outperforms brute KNN in terms of overall performance, as kd-Tree had considerable gains in classification time. We suspect kd-tree enhances the learning speed by reducing the search space when possible. Importantly, PCA reduction contributed to the faster running times. Improvements could still be made to the efficiency of kd-Tree. One suggestion could be to incorporate a KNN model-based approach, where the value of number of neighbours *k* is learned, and varies for each prediction[8]. This may further reduce the search space but may potentially negatively affect accuracy. Overall, KNN was a simple yet effective method in classifying the majority of classes and was able to achieve good performance with 85.25% accuracy and our kd-Tree optimised method achieved the fastest running time of 5.8 seconds.

Random Forest
PCA reduction did not reap comparable or better accuracy for the Random Forest classifier. We suspect this is due to the algorithm's split rule where only an individual feature is used as opposed to a set of features at each node. The results suggest some loss of information with PCA dimensionality reduction using this approach. To further improve on Random Forest accuracy, a different approach for feature reduction may be required with this method such as Recursive Feature Elimination with Out-Of-Bag Error minimisation to find the limited set of variables from the original dataset that leads to a good prediction model. We suggest parameter tuning such as max depth of trees and higher values of PCA for accuracy experimentation. Its advantage was relatively faster running time to brute KNN. Although Random Forest can be memory and computationally intensive with a large number of features and learners, *sklearn*'s implementation seemed to make use of parallelised construction of trees to speed up time cost.

Inter-classifier discussion
Both KNN classifiers achieved higher overall accuracy for this task. Moreover, PCA was more effective pre-processing technique for KNN as it resulted in good accuracy at a lower dimension for the search space. It is observed that both classifiers exhibited similar difficulties in predicting the *shirt* class, suggesting that feature representation with and without PCA still resulted in ambiguities across classes due to overlapping features.

Feature Representation
This observation stresses the importance of the feature representation used for our classifiers. Using PCA restricted us to linear mappings of features and although majority of classes reported good F-scores, with respect to the overlapping classes it was a simplification of more distinct, fine-grained qualities (i.e. locally correlated pixels, generalisable shapes and patterns). Given this, we suggest exploration with non-linear mappings of features through the use of autoencoders that may be able to capture important features that were overlooked using our approach[9]. We suspect that better feature representation would improve the accuracy of our existing model.

5 Conclusion and Future Work

In summary, KNN was the most effective method for this image classification as it achieved a higher accuracy compared to Random Forest. KNN was able to discriminate the majority of classes, however few overlapping classes (shirt, pullover, coat) were more difficult. This general phenomena was observed even for Random Forest classifiers. As a result, the feature representation of the data was evaluated and future work would be to explore non-linear features of the dataset to improve discriminative power of features and to exploit the performance of our current algorithm. The use of PCA reduction and kd-Tree optimised KNN method resulted in a more efficient classifier.

References

1. Wang C, Blei D, Fei-Fei L. Simultaneous Image Classification and Annotation. CVPR. 2009.
2. Xiao H, Rasul K, Vollgraf R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv preprint arXiv:1708.07747. 2017.
3. Wan M, Zeiler M, Zhang M, Cun YL, Fergus R. Regularization of neural networks using dropconnect. Proceedings of the 30th international conference on machine learning (ICML- 13). 2013;1058–1066.
4. Manning-Dahan T. PCA and Autoencoders. http://tylermd.com/pdf/pca_ae.pdf. 2017.
5. Bhatia N, Vandana. Survey on nearest neighbor techniques. IJCSIS. 2010; 80; 2.
6. Kulkarni VY, Sinha PK. Random forest classifiers: a survey and future research directions. International Journal of Advanced Computing. 2013; 36; 1; 1144-1153.
7. Breiman L. Random forests. Machine Learning 2001; 45; 1; 5–32.
8. Guo G, Bell DA, Wang H. KNN Model-based Approach in Classification. https://www.researchgate.net/publication/2948052. 2004.
9. Charte D, Charte F, Garcia S, del Jesus MJ, Herrera, F. A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines. Information Fusion. 2017; 78-96.

Appendix

Instructions for running the code:

Running code by terminal:

1. Cd into ./algorithm
2. Put .h5 files in ./data
3. The python script knn_naive.py need 4 arguments

Arg1 is the path of training data; Arg2 is the path of training label; Arg3 is the path of testing label. Arg4 is the algorithm used for classifier(choose from "kd-tree" and "brute-force")

For example, the command line should look like :

```
python3 knn_naive.py ../data/images_training.h5 ../data/labels_training.h5
../data/images_testing.h5 kd-tree
```

Running code by Pycharm:

File->Open->select the folder

Select knn_naive.py and run.

Testing Environment:

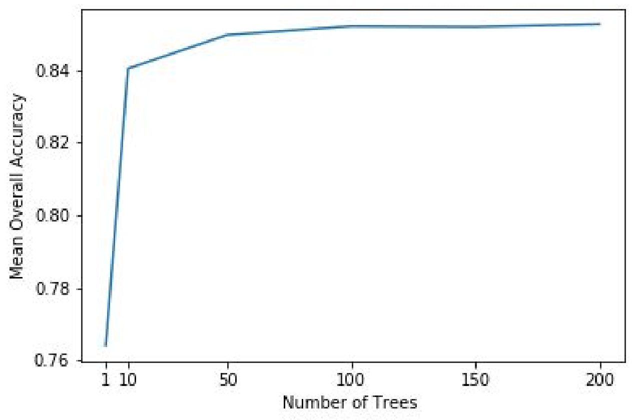
OS X Yosemite Version 10.10.5

MacBook Pro

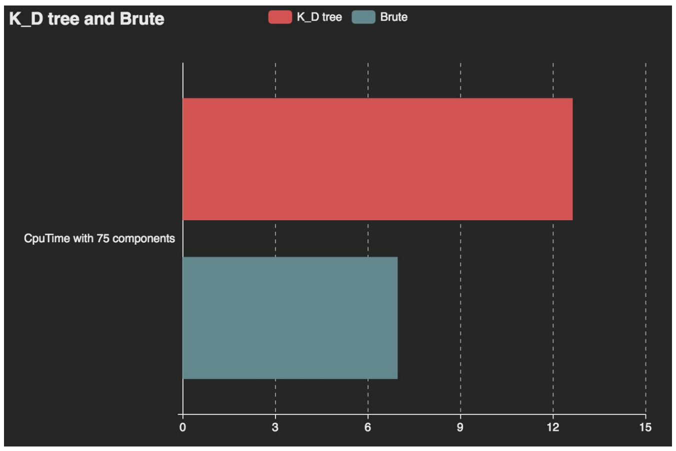
Processor 2.7 GHz Intel Core i5

Memory 8 GB 1867 MHz DDR3

Graph A: Number of trees tuning for Random Forest using 10-fold Cross-validation



Graph B:CPU time comparison of kd-Tree and Brute KNN methods with 75 PCA components


K_D tree and Brute
K_D tree
Brute
CpuTime with 75 components