Machine Learning

BSMALEA1KU

Bence Zoltan Balazs (beba@itu.dk)
Kryštof Hrubý (krhr@itu.dk)
Sebestyen Nagy (sena@itu.dk)

word count: 2737 (Overleaf word counter)

1 Introduction

This machine learning project was conducted on a subset of the Fashion-MNIST [5] data. The main goal was to compare different multi-class classification algorithms on 5 types of clothes. The data were 28x28 grayscale images, they were labelled in *t-shirt/top*, *trousers*, *pullover*, *dress* and *shirt* categories. The grayscale value of each pixel was originally provided between 0 and 255. These images were divided into a training set of 10,000 images and a test set of 5,000 images. The training set was used to train models and the test set was used to check how accurate the models are on data that they have not seen before.

2 Dimensionality Reduction

Dimensionality reduction is the transformation of high-dimensional data into a meaningful representation of reduced dimensionality. Ideally, the reduced representation should have a dimensionality that corresponds to the intrinsic dimensionality of the data. The intrinsic dimensionality of data is the minimum number of parameters needed to account for the observed properties of the data [2]. Dimensionality reduction facilitates, among others, classification, visualization, and compression of high-dimensional data. [4]

2.1 Principal component analysis

Principal component analysis is an unsupervised dimensionality reduction technique, which aims to maximize the variance. This is achieved by projecting the data onto some eigenvectors of the covariance matrix, such that the points are most spread out, making their differences most apparent.
Since the inputs of this project are images, it is expected that nearby pixel values are highly correlated, and can therefore be disregarded. To find the optimal number of dimensions, the eigenvectors (and values) of the covariance matrix are calculated and sorted. Eigenvectors with high eigenvalues explain more of the variance, so a suitable dimension can be chosen, by setting the amount of variance wanted explained - see Figure 3.
Our implementation of PCA can be found in the notebook titled `pca.ipynb`, based on the course textbook [1]. It is implemented as a class, with each method corresponding to a step in the process.

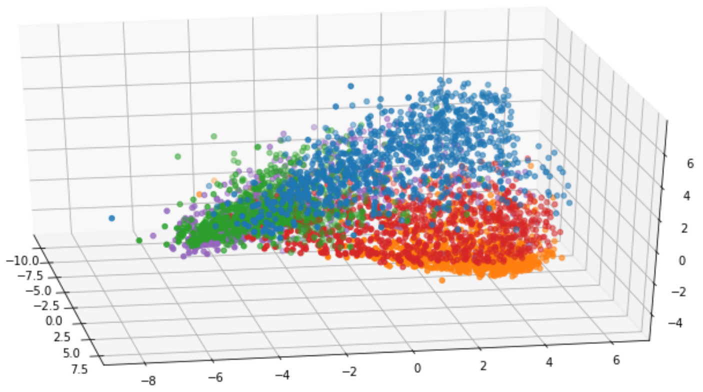


Figure 1: The train data reduced to 3 dimensions using PCA

1. Points are stored in a matrix X

$$X \in \mathbb{R}^{NxD}$$

2. The mean vector of points is calculated (`calc_mean()`).

$$M = (m_1, m_2, ..., m_D)^\top$$

3. The data are centered by subtracting mean vector (`sub_mean()`).

$$X_c = X - M$$

4. The covariance matrix is calculated (`cov_matrix()`).

$$\hat{\Sigma} = \frac{(X_c)^\top (X_c)}{\#points}$$

5. The eigenvalues and eigenvectors of the $\hat{\Sigma}$ are calculated, and sorted in descending order into matrix W using corresponding eigenvalues (`eigen_vecs()`).

$$W = (e_1, e_2, ..., e_D)$$

6. Choose K leading eigenvectors. K is the dimension of the space that points are transformed to (`calc_suitable_output_dim()`).

$$W_K = (e_1, e_2, ..., e_K)$$

7. Points are tranformed to a new k-dimensional space (`transform_data()`).

$$z^{(i)} = W_K^\top (x_c^{(i)})$$

The only parameter of the model to tune, is the number of output dimensions. To make an informed decision, three helper functions were defined:

- plotting the eigenvalues - see Figure 2 (`plot_eigenvalues()`)
- plotting the variance explained by the first K eigenvectors - see Figure 3 (`plot_variance_proportion()`)
- a function which returns the number of eigenvalues, which are higher than the mean of the eigenvalues (`calc_suitable_output_dim()`)

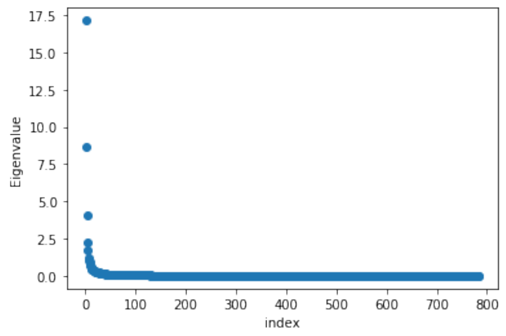


Figure 2: A plot of eigenvalues in descending order

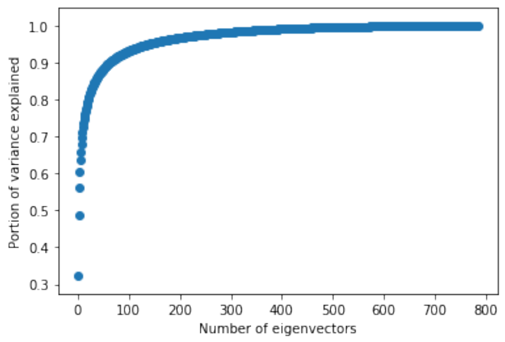


Figure 3: The proportion of variance explained by the largest k eigenvectors

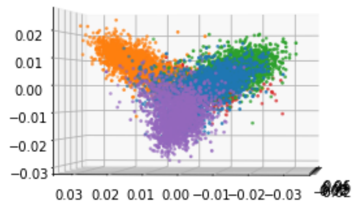


Figure 4: LDA reduced train data

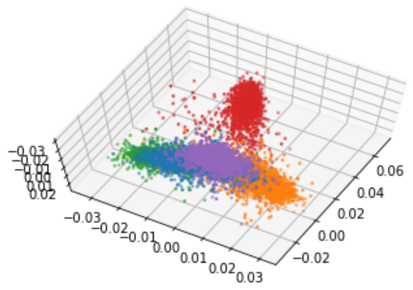


Figure 5: LDA reduced train data

2.2 Linear discriminant analysis

Linear discriminant analysis is a supervised dimensionality reduction technique, where we aim to find a hyperplane, onto which the datapoints can be projected, such that the distance between class means are maximized, and the distance between datapoints and their class means minimized.

Our implementation is the function `general_LDA()` in the notebook `LDA.ipynb`, based on the course textbook [1]. Dataset is $X = \{(\boldsymbol{x_i}, \boldsymbol{y_i})\}$ where $y_{ij} = 1$ if $\boldsymbol{x_j} \in C_i$ and 0 otherwise. K is the number of classes.

1. We want to find matrix $\boldsymbol{W}$ such that
$$\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{x}$$
2. The mean ($\boldsymbol{m_i}$) for each dimension for each class is calculated (`class_means`).
3. The scatter matrices for each of the classes are calculated (`scatter_mats`).
$$\boldsymbol{S_i} = \sum_j y_{ij}(\boldsymbol{x_j} - \boldsymbol{m_i})(\boldsymbol{x_j} - \boldsymbol{m_i})^\top$$
4. The total within-class scatter matrix is calculated by summing the scatter matrices of the individual classes (`S_W`).
$$\boldsymbol{S_W} = \sum_{i=1}^{K} \boldsymbol{S_i}$$
5. The scatter of means is calculated, which is how much the means are scattered around the overall mean (`scatter_mean`).
$$\boldsymbol{m} = \frac{\sum_{i=1}^{K} \boldsymbol{m_i}}{K}$$
6. To calculate the between-class scatter matrix, the scatter for each class is calculated (`temp_mats`), and then these scatters are summed (`S_B`).
$$\boldsymbol{S_B} = \sum_{i=1}^{K} N_i(\boldsymbol{m_i} - \boldsymbol{m})(\boldsymbol{m_i} - \boldsymbol{m})^\top;\ N_i = \sum_j y_{ij}$$
7. To find a hyperplane $\boldsymbol{W}$ suitable for the projection, we want want to maximize function $J(\boldsymbol{W})$. After the projection the between-class scatter matrix is $\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W}$, which we want large, and the within-class scatter matrix is $\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W}$, which we want small. The solution is the largest eigenvectors of $\boldsymbol{S}_W^{-1} \boldsymbol{S}_B$.
$$J(\boldsymbol{W}) = \frac{|\boldsymbol{W}^\top \boldsymbol{S_B} \boldsymbol{W}|}{|\boldsymbol{W}^\top \boldsymbol{S_W} \boldsymbol{W}|}$$

The output dimension is the only parameter of the model which is the number of eigenvectors chosen in the last step.

3 Classification

3.1 Multivariate Gaussian distribution modeling

Multivariate Gaussian distribution modelling is a simple supervised probabilistic model. This method only works under the assumption that each class is in one cluster and have a Gaussian like distribution. Initially it calculates the the mean($\boldsymbol{\mu}$), the covariance matrix($\boldsymbol{\Sigma}$) and the prior probability ($P(C_k)$) for each class.

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\} \tag{1}$$

Once it has the Gaussian parameters and the priors it computes the class conditional densities based on Equation (1). When it has the density functions, it evaluates the equations for each input data point. The one with highest output value is selected to be the predicted class for the data point.

$$P\left(C_k|\boldsymbol{x}\right) = p\left(\boldsymbol{x}|C_k\right) P\left(C_k\right)$$

3.2 Multilayer Perceptron

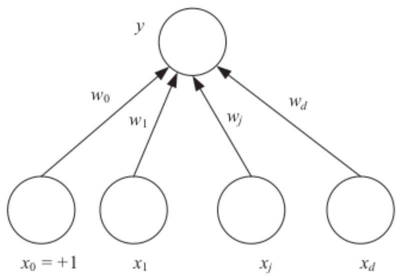


Figure 6: A simple perceptron

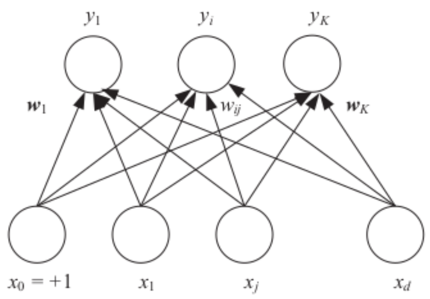


Figure 7: Parallel perceptrons

The basic processing element of a MLP is the perceptron, which is just a linear combination of the input. For each input x_i, a connection weight w_i is calculated, which optimally leads to an output associated with the correct class label. The weights are the parameters of the model to be learned, see Figure 6.
For a K-class classification problem, K perceptrons are used (Figure 7), and the class with the highest output(*max)* is chosen as the predicted label. Alternatively, one can use *softmax*, which instead of the winning class label, returns the posterior probabilities for each class.
A multilayer perceptron (see Figure 8) is a network of perceptrons, which due to its architecture is often called a neural network. Compared to a simple perceptron, in a MLP each node is a perceptron itself, which can be connected in multiple layers, such that the input on the next layer, is the output of perceptrons on the previous layer. Multilayer perceptrons are also also able implement nonlinear discriminants, which is done by the use of activation functions. Activation functions apply

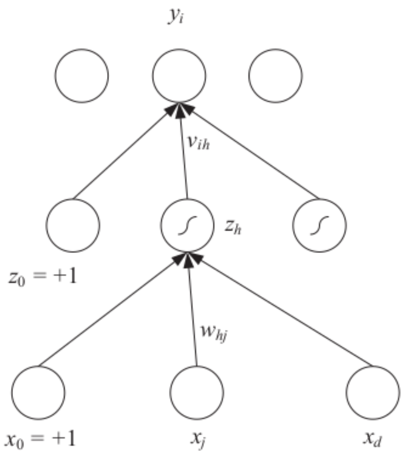


Figure 8: A multilayer perceptron

some non-linear tranformation to the output of a perceptron in a hidden layer, thus making the MLP solve non-linear problems. Otherwise the output of the MLP would just be a linear combination of the input, so the multiple layers be reduced to a single layer.
Our implementation follows a template from tensorflow's github repo[1]. The `mlp()` function builds the model based on the number of hidden layers, number of nodes in each layer, and activation function in each layer. The calculated weights and biases are stored in a dictionary, and the output layer is returned. The `neural_net()` function is a wrapper, which initializes the placeholders, and parameters. It calls `mlp()` to create the model, then runs a tensorflow session using batch learning to train the model. The wrapper prints the loss for every 100 training epoch, and reports the accuracy over the training and the validation set. To be able to report other accuracy measures, the function returns the predictions made for the training and evaluation set.

3.3 Support Vector Machines

Support Vector Machines (SVM) are one of the most powerful supervised learning algorithms. They can take the form of classifiers (SVCs) or regressors (SVRs).
Given two classes, a support vector classifier tries find a decision hyperplane that separates the two classes with the largest margin. If the two classes are not linearly separable, a soft margin definition is introduced that allows for misclassified instances as well as instances within the margin.
Let us have two classes and labels $-1, +1$. The sample is $X = \{(\boldsymbol{x_i}, y_i)\}$ where $y_i = +1$ if $\boldsymbol{x_i} \in C_1$ else $y_i = -1$. The discriminant is $g(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ where $\boldsymbol{w}$ is the normal vector to the hyperplane, $g(\boldsymbol{x}) > 0$ means that $\boldsymbol{x} \in C_1$ and $g(\boldsymbol{x}) \leq 0$ means that $\boldsymbol{x} \in C_2$. The distance of $\boldsymbol{x_i}$ to the hyperplane is

$$d_i = \frac{|\boldsymbol{w}^\top \boldsymbol{x_i} + b|}{\|\boldsymbol{w}\|} = \frac{y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b)}{\|\boldsymbol{w}\|}.$$

Since SVM tries to find a hyperplane with the largest margin between two classes, constraints need to be introduced to ensure the optimal solution. The soft margin constraints are

$$y_i(\boldsymbol{w}^\top \boldsymbol{x} + b) \geq 1 - \xi_i, \forall i \tag{2}$$

where $\xi_i = 0$ if $\boldsymbol{x_i}$ is correctly classified and not in the margin, $\xi_i \in (0, 1)$ if x_i is in the margin but correctly classified and $\xi_i >= 1$ if $\boldsymbol{x_i}$ is misclassified. We want to minimize the length of $\boldsymbol{w}$ (in order to maximize distances of all points to the hyperplane) as well as the deviation error $\sum_i \xi_i$. The whole problem can be defined as a standard quadratic optimization problem:

$$min\ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i \tag{3}$$

$$subject\ to\ y_i(\boldsymbol{w}^\top \boldsymbol{x_i} + b) \geq 1 - \xi_i,\ \forall i \tag{4}$$

where C is the penalty parameter. Because this is a convex quadratic optimization problem and the linear constraints are also convex, the dual problem can be solved to get the same solution. The dual is

$$max\ \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \langle \boldsymbol{x_i}/\boldsymbol{x_j}\rangle \tag{5}$$

$$subject\ to\ \sum_i \alpha_i y_i = 0 \tag{6}$$

$$0 \leq \alpha_i \leq C, \forall i. \tag{7}$$

The advantage of solving the dual problem is that it can be solved without using quadratic programming techniques. An efficient algorithm that is widely used to solve the dual of SVM is called Sequential minimal optimization.

[1]https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/multilayer_perceptron.py

Algorithm 1 Sequential minimal optimization

```
1:  procedure EXAMINE_EXAMPLE($\alpha_1$)
2:      $E_1 \leftarrow g(x_i) - y_1$                                    ▷ If $\alpha_2$ unbound error cache is used
3:      if $\alpha_1$ satisfies KKT conditions within tolerance then
4:          return 0                                                 ▷ No optimization done
5:      else
6:          for $\alpha_2 \leftarrow$ get next $\alpha$ $-$ Heuristic 2 do
7:              result $\leftarrow$ take_step($\alpha_1, \alpha_2, E_1$)
8:              if result == 1 then
9:                  return 1                                         ▷ $\alpha_1, \alpha_2$ optimized
10:     return 0                                                     ▷ No optimization done
11: procedure RUN(max_iter)
12:     curr_iter $\leftarrow$ 0
13:     while Alg. didn't converge & curr_iter < max_iter do
14:         num_changed_alphas $\leftarrow$ 0               ▷ Important for Heuristic 1 and convergence check
15:         set $\leftarrow$ choose what sequence of $\alpha$s iterate over $-$ Heuristic 1
16:         for $\alpha_1 \leftarrow$ set do
17:             num_changed_alphas $\leftarrow$ examine_example($\alpha_1$)
18:             curr_iter $\leftarrow$ curr_iter + 1
19:     return SMO.alphas, SMO.b
20: procedure TAKE_STEP($\alpha_1, \alpha_2, E_1$)
21:     Compute how much $\alpha_2$ can change $-$ lower and higher bound
22:     $E_2 \leftarrow g(x_2) - y_2$                                    ▷ If $\alpha_2$ unbound error cache is used
23:     if $\alpha_1, \alpha_2$ cannot be optimized then
24:         return 0
25:     Update $\alpha_2$
26:     Check that $\alpha_2$ is within bounds
27:     Update $\alpha_1$ using new $\alpha_2$
28:     Update b threshold
29:     Update caches
30:     return 1
```

Note: An α has a high optimization potential if its value is $\in (0, C)$. These αs are called unbound. `Heuristic 1` chooses what set of αs iterate over. When there are no unbound αs or the unbound αs cannot be optimized the set consists of all αs. If there are unbound αs then the set consists of unbound αs. `Heuristic 2` first chooses unbound α_2 that with maximum $|E_1 - E_2|$ $(E_1 = g(\boldsymbol{x_1}) - y_1,\ E_2 = g(\boldsymbol{x_2}) - y_2)$, then loops over unbound αs and finally loops over all αs. However, when α_1, α_2 were optimized, no other α is chosen and algorithm returns to function `run()`.

3.3.1 Sequential minimal optimization

The Sequential minimal optimization algorithm (SMO) is an optimization algorithm, that can solve the final SVM quadratic problem quite fast and efficiently. The idea behing SMO is to keep choosing two αs (α_1, α_2) and optimizing the objective function with respect to chosen αs while other αs are fixed until all αs are optimal. The SMO was implemented in a jupyter notebook titled `svm.ipynb` based on [3]. `Algorithm 1` shows a simplified pseudocode of the implemented SMO algorithm. The algorithm has 3 important parts:

- Main routine responsible for choosing α_1s and determining whether the algorithm converged (`run()`).
- Check that α_1 breaks KKT conditions, then choose suitable α_2 (`examine_example()`).
- Try to optimize the objective function by optimizing α_1, α_2 with respect to the dual constraints while other αs are fixed (`take_step()`).

The implemented SMO algorithm uses many caches and hacks to make it faster. Some were taken from [3], while others were thought of.

3.3.2 Kernel trick

The classification discriminant in SVM is $g(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b = \sum_i \alpha_i y_i \langle \boldsymbol{x_i}/\boldsymbol{x} \rangle + b$. The input points are present only in scalar products both in optimization and classification equations. Therefore, the scalar product can be swapped for a kernel function $K(\boldsymbol{x_i}, \boldsymbol{x_j})$. Linear ($K(x_1, x_2) = x_1^\top x_2$), polynomial ($K(x_1, x_2) = (x_1^\top x_2 + c)^d$) and gaussian ($K(x_1, x_2) = exp(-\gamma \|x_1 - x_2\|)$) kernels were implemented and used for training and predictions.

3.3.3 K-class SVM

K-class SVM consists of multiple two-class SVMs. There are two approaches. Either each SVM can be used to separate one pair of classes which results in $K(K-1)$ models. Or one-vs-all approach where a SVM can be trained to separate one class from the rest which results in K models.
One-vs-all approach was used in our implementation. The disadvantage of this approach is imbalanced datasets, which means that one class representing the rest of the classes has much more instances than the one. Thus, the larger class dataset was reduced. The training and reduction are done in one function (`multiclass_svm_train()`). Prediction for a point $\boldsymbol{x}$ is done by selecting the class whose one-vs-all SVM model has the highest discriminant value for $\boldsymbol{x}$ (`multiclass_svm_predict()`).

4 Results & Interpretation

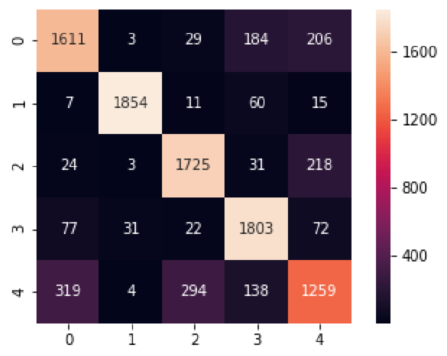
Confusion matrices were computed for each model for both the train and the test set (Gaussian - Figures 9, 10, MLP - Figures 11, 12, SVM - Figures 13, 14). As seen in these figures, the classifiers could identify classes 1 and 3 the best, which are *trousers* and *dresses* respectively. This is probably due to the unique shape of these items, since the most confused classes are in fact very similar - classes 0 and 4, *t-shirt/top* and *shirt* respectively. Furthermore, low input resolution of pictures could negatively influence classifying between these two classes.
The performance of trained models was measured by calculating average accuracy and macro F1 (see Table 1). Average accuracies are fairly high for all three models. These accuracies also seem somewhat similar which might indicate that the given classification task is not difficult. However, model parameters of MLP and SVM could be further optimized to yield better results while the Gaussian model has none in this case.

	Models	Avg accuracy	Avg precision	Avg recall	Macro-F1
Train set (4-dim LDA)	Gaussian	0.9300	0.8262	0.8245	0.8243
	MLP	0.9495	0.8745	0.8739	0.8728
	SVM	0.9592	0.8987	0.8985	0.8982
Test set (4-dim LDA)	Gaussian	0.9176	0.7942	0.7918	0.7828
	MLP	0.9266	0.8164	0.8138	0.8138
	SVM	0.9253	0.8132	0.8114	0.8118

Table 1: Models' performance

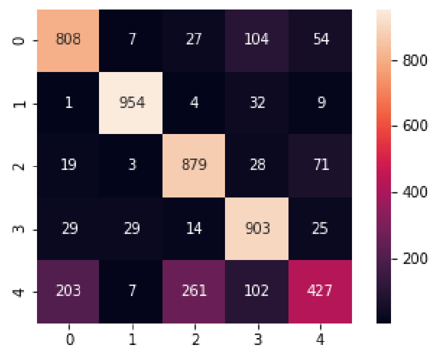
5 Discussion

For all the models the following pre-processing was done: the train and test sets were normalized to $(0, 1)$ interval. Other transformations of the original data, such as binary thresholding didn't yield significantly better results. Furthermore, the data was transformed to 4-dimensional space using LDA transformation. Transforming the data to 4 dimensions, still explains all of the variance, and low dimensionality significantly speeds up the learning. Moreover, input data transformed by PCA or a combination of LDA and PCA did not yield better train and validation accuracy.
The test set was merely used to report the models' performance on the given task. After normalization of the sets, LDA transformation weights for training set were used to transform the test set into 4 dimensions. The test set was then only fed to the final three models to evaluate the models' performance.



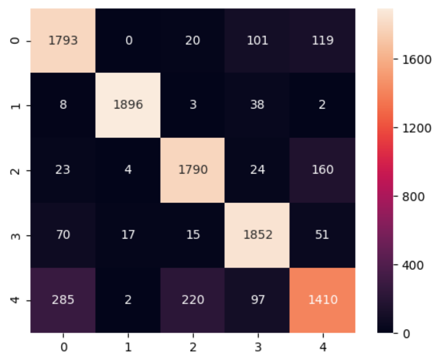
	0	1	2	3	4
0	1611	3	29	184	206
1	7	1854	11	60	15
2	24	3	1725	31	218
3	77	31	22	1803	72
4	319	4	294	138	1259

Figure 9: Gaussian model train set confusion matrix



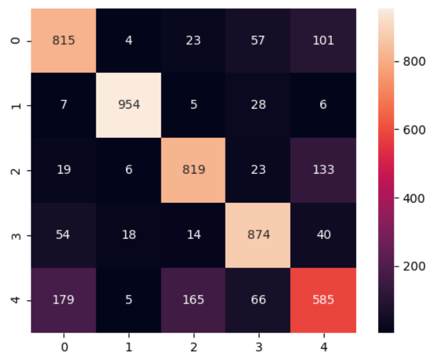
	0	1	2	3	4
0	808	7	27	104	54
1	1	954	4	32	9
2	19	3	879	28	71
3	29	29	14	903	25
4	203	7	261	102	427

Figure 10: Gaussian model test set confusion matrix



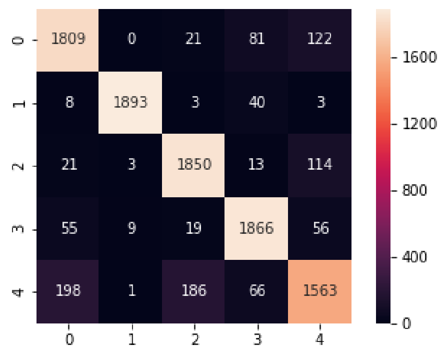
	0	1	2	3	4
0	1793	0	20	101	119
1	8	1896	3	38	2
2	23	4	1790	24	160
3	70	17	15	1852	51
4	285	2	220	97	1410

Figure 11: MLP train set confusion matrix



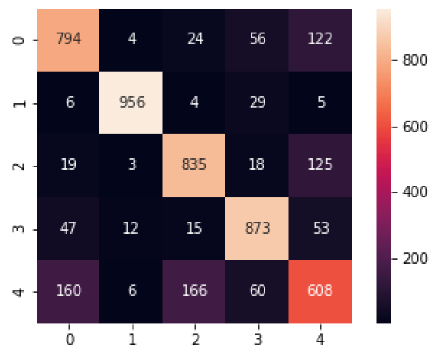
	0	1	2	3	4
0	815	4	23	57	101
1	7	954	5	28	6
2	19	6	819	23	133
3	54	18	14	874	40
4	179	5	165	66	585

Figure 12: MLP test set confusion matrix



	0	1	2	3	4
0	1809	0	21	81	122
1	8	1893	3	40	3
2	21	3	1850	13	114
3	55	9	19	1866	56
4	198	1	186	66	1563

Figure 13: SVM train set confusion matrix



	0	1	2	3	4
0	794	4	24	56	122
1	6	956	4	29	5
2	19	3	835	18	125
3	47	12	15	873	53
4	160	6	166	60	608

Figure 14: SVM test set confusion matrix

5.1 Preproccessing

Some more pre-processing on the images could help finding more distinctive features. Blur filters and binary thresholding were implemented, but they did not suffice to improve the model. It was deemed that diving deeper into image processing was out of scope for this project.

5.2 Gaussian model

According to visual observation and models' results, each class's distribution seemed quite similar to Gaussian distribution. Therefore, there was no parameter tuning to be done since Gaussian model hyperparameters are mixtures. 10-fold algorithm was used to determine how consistently this model predicts on the dataset. The difference among validation accuracies was negligible.

5.3 MLP

10-fold cross-validation was used to tune the parameters of the model and to find the model which would be generalized. The model was not overfit because validation accuracies in 10-fold were fairly similar for the LDA transformed data.
Due to lack of time and computing power, we only tuned the number of layers, the number of nodes per layer, and the input data together, and found out that 4 layers, with 50 nodes each gave the highest validation accuracies. The rest of the parameters were tuned in isolation. We tried a couple of different values for epochs, and found that 1000 epochs made model already overfit and achieved 99.9% training accuracy. Therefore, we settled on 500 epochs, as the training and validation accuracies were fairly similar.
Batch learning was used, with batch sizes of 1000, and random sampling with replacement. For activation functions, we settled on using sigmoids, as they performed better than the rectified linears. Running different optimizers with different learning rates did not significantly alter the results, therefore learning rate was set to 0.001 and the Adam optimizer was used.
The model could be further improved, by more exhaustive joint parameter tuning, instead of selecting parameters separately. A convolutional neural net could also be implemented, instead of a fully connected one.

5.4 SVM

Since K-class SVM model consists of multiple SVM models, each one-vs-all model was trained and evaluated separately. Multiple K-class SVMs were then created from the best performing one-vs-all models of each class. Simple cross-validation was performed on the K-class models and the one with highest validation accuracy was selected as the final K-class model (see Table 2).
Since the training of each model took long time, no specialised algorithm was used to find the optimal parameters. Instead, models were trained for each of the kernels with different kernel parameters. The maximum number of iteration was also taken into account and cross-validation was done for each 10000th iteration. A linear kernel was used for *trouser* separating SVM model, because all of the models had very similar train and validation accuracies. Although train and validation accuracies were fairly similar for models separating *shirt/top*, *pullover* and *dress* classes, gaussian kernels performed slightly better and thus were chosen. The *shirt* class proved difficult to separate from others and only gaussian kernels seemed to perform reasonably well. Greater γ was chosen to accomodate for the task difficulty. Even greater γ values seemed to cause the model to overfit.
Different values of C seemed to have no significant influence on the validation accuracies. Due to the long

Class (label)	Kernel	Kernel param.	C	Max iter.	Accuracy	
					train set	validation set
T-shirt/top (0)	rbf	$\gamma = 0.5$	1.0	50000	0.9439	0.9362
Trouser (1)	lin		1.0	50000	0.9903	0.9911
Pullover (2)	rbf	$\gamma = 1.0$	1.0	50000	0.9630	0.9526
Dress (3)	rbf	$\gamma = 0.5$	1.0	50000	0.9650	0.9540
Shirt (4)	rbf	$\gamma = 2.0$	1.0	50000	0.9294	0.8944

Table 2: Multiclass SVM model

training time, only few experiments were done in that area. It might be possible that for some one-vs-all models, a different C value could have a huge impact. Overall, the SVM model could be further improved by implementing an optimal parameter search algorithm, as well as using more image processing techniques. The structural SVM and all-vs-all approach should also be implemented and considered in future work.

6 Conclusion

To conclude, all our models performed similarly on the test set, particularly struggling with the *shirt* class. Since the measures of performance are so similar, no conclusion can made on which model is the best for the task of image classification.

References

[1] Ethem Alpaydin. *Introduction to machine learning.* MIT press, 2009.

[2] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition (2Nd Ed.)* San Diego, CA, USA: Academic Press Professional, Inc., 1990. ISBN: 0-12-269851-7.

[3] John C Platt. “Fast training of support vector machines using sequential minimal optimization, advances in kernel methods”. In: *Support Vector Learning* (1999), pp. 185–208.

[4] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. “Dimensionality reduction: a comparative”. In: *J Mach Learn Res* 10.66-71 (2009), p. 13.

[5] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *CoRR* abs/1708.07747 (2017). arXiv: `1708.07747`. URL: `http://arxiv.org/abs/1708.07747`.