Vietnam National University HCMC
University Of Information Technology

FINAL PROJECT

CS116.M11.KHCL

SUPPORT VECTOR MACHINE

Members
19520197 Lê Đoàn Thiện Nhân
19521281 Trương Minh Châu

Lecturer
Nguyễn Vinh Tiệp

Contents

Abstract

Support Vector Machine (SVMs) is a well-known Machine Learning model because of its powerful, versatile, capable of performing linear and non-linear classification and regression. In this project, we will find out what is SVM and its use in order to gain knowledge and to know how to apply this Machine Learning model in reality. The dataset that we are using to experience is Fashion-MNIST, which is very popular in image classification. Before putting it into the SVMs model, our team will do some important steps to preprocess each image in the dataset by using the Alexnet model to extract all the features appearing in each image. Then, to split the dataset into 3 parts (Data for training, data for finding hyperparameters, and data for testing), We will have 54000 images for training, 6000 images for finding hyperparameters, and 10000 images for testing. To find hyperparameters for our targeted model, we use Grid Search in the Sci-kit Learn library to run all listed parameters that we have added. In addition, we also use a confusion matrix and classification report which is belonged to the aforementioned library to visualize our result after doing all experiments. The report below will depict our process of researching, applying, and evaluating the SVMs model.

1 Introduction

The support vector machines (SVMs), first developed by Vapnik and Chervonenkis in 1995, are a type of linear machine learning just like the well-known perceptron algorithm and thus has the function of classifying input patterns by first being trained on labelled data sets (supervised learning). However, SVMs represent a significant functional enhancement over perceptrons. The strength of SVMs lies within the undeniable fact that they use non-linear kernel functions to implicitly map the input to a high-dimensional "feature spaces". In the feature spaces, linear classifications may be possible, which become non-linear in the transformation back to the original input space. Thus, even though SVMs are linear machine learning with regard to feature spaces, they are essentially non-linear classifiers.

1.1 Definition

In the machine learning field, SVMs are one of the classical techniques used in supervised learning models with associated learning algorithms that can help solve or analyse big data for both classification and regression problems. Notwithstanding, primarily, it is used for classification problems in machine learning.

This algorithm implements the classification as well as regression based on the principle of risk minimization with the structural risk minimization. Not only that, but SVMs model is also based on theoretical statistics, so there is a mathematical background to ensure that the search results are correct. However, SVMs algorithm is mathematically complex and computationally expensive.

The main objective of this algorithm is to search out a hyperplane called for the best line or decision boundary segregating N-dimensional space to distinctly classify the data points, with N being the number of features.

1.2 Type

There are two types of SVM which are:

- **Linear SVMs** is used for linearly separable data, i.e., given a dataset being classified into two categories by utilizing a single straight line. Then such data points are termed as linearly separable data, and the used classifier is seen as a Linear SVM classifier.

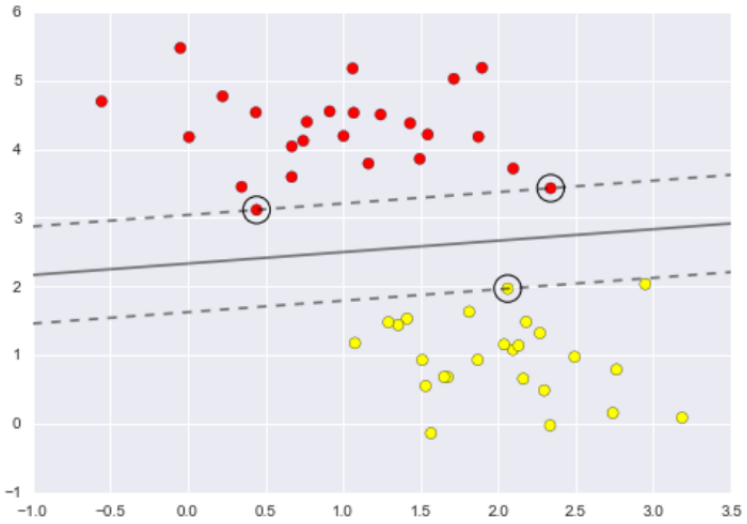

Figure 1: Linear SVMs model

- **Non-linear SVMs** is used for data that is non-linearly separable data, i.e., it is impossible to use a straight line to categorize the dataset. Instead, we will use a kernel trick sets the data points during a higher dimension being decomposed using planes or other mathematical functions. Such data points are termed as non-linear data, and also the classifier used is termed as a Non-linear SVM classifier.

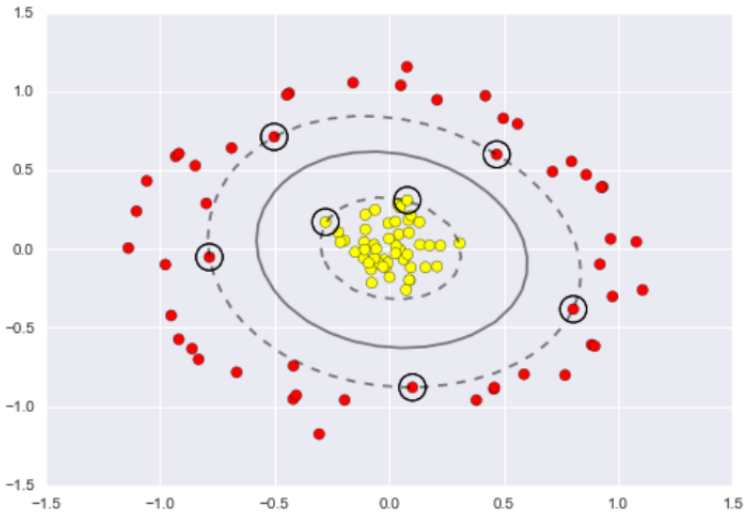

Figure 2: Non-Linear SVMs model

1.3 Main idea

Given a training dataset, with being a representation as points in vector space, then the SVMs algorithm finds a decision hyperplane that can best divide the points on this space into correspondingly distinct classes. The quality of this hyperplane is determined by the distance (called the boundary) of each layer's nearest data point to this plane. Then, the larger the boundary distance, the better the decision plane, as well the more accurate the classification.

2 Pros and Cons

The advantages of SVMs are:

- Works well with high dimensional spaces as well as in cases where a number of dimensions are greater than that of samples.
- The algorithm consumes efficient memory because of only using the subset of the points in the support vectors to predict in the decision function.
- We can create many decision functions from different kernel functions. Common kernels are provided, but it is also possible to specify custom kernels. Even using the right kernel can greatly improve the algorithm.

The disadvantages of SVMs include:

- If the amount of features is far greater than that of samples, avoiding over-fitting in choosing kernel functions and therefore the regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an upscale five-fold cross-validation.
- In many cases, it is not possible to linearly divide the data classes in an original space used to describe a problem. Thus, it is often necessary to map data points in the original space into a new space having more dimensions, so that their separation becomes easier.

3 Implementation steps of SVMs algorithm

The SVMs algorithm requires expressing the data as vectors of float numbers. So if the input is not numbers, we need to find a way to convert it to the numerical form of SVMs.

- **Pre-processing data** is the process of performing the appropriate data transformation for the calculating process to avoid the large numbers describing attributes, with the recommendation for scaling data into $[-1, 1]$ or $[0, 1]$.
- **Choosing kernel function** means selecting the suitable kernel function corresponding to each specific problem to achieve high accuracy in the classification process.
- **Cross validation** is taken into account as an awfully useful technique for assessing the effectiveness of a model, particularly in cases would like to mitigate overfitting. This technique is also of using in determining the hyperparameters, within the sense, electing parameters that result in the lowest test error. This contributes decisively to the high accuracy of the classification process.
- **Training with the sample set** by applying the found parameters above. During the training process, using the algorithm to optimize the distance between hyperplanes in the classification process at the same time define the classifier function in the feature space due to map data points into the feature space by choosing kernels, which solve for both cases are separable data and non-linearly separable data in the feature space.
- **Testing the test set** to check the accuracy of the model.

4 Applications of SVMs algorithm

The purpose of using SVMs is to correctly categorize unseen data or, in short, to classify data. And before introducing some applications of SVMs in real world, we firstly figure out briefly information about what the data classification problem is and why SVMs algorithm is chosen to be used for that problem.

4.1 Data classification

This is one of the most widely used and studied data mining techniques today. With its aim being to predict classification labels for datasets or new samples.

- **Input** is a set of training data samples, with a classification label for each data sample.
- **Output** is a classifier based on training set or classification labels

Classifying data based on the training set and the values in a classifier attribute, then using it to determine the class for the new data. The data classification technique is carried out in two steps:

- **Step 1**: Building the model from the training set.
  - Each sample/dataset is assigned to a predefined class.
  - The class of a sample/dataset is determined by the class-label attribute.
  - The training set is used to build the model.
  - Models are represented by classification rules, decision trees or mathematical formulas.
- **Step 2**: Using the model to check the correctness of the model and use it to classify new data.
  - Categorizing new or unclassified objects.
  - Evaluating the accuracy of the model by
    - The known class of a sample/dataset under test is compared with the results obtained from the model.
    - Ratio of accuracy as a percentage of samples/datasets correctly classified by the model among tests.

4.2 The reasons why using SVMs

There are 4 reasons respectively

- It is effective to solve the large dimensional data problems.
- SVMs algorithm is good at solving the overfitting issue with data having noise and group separation or small training data size.
- It is a fast classification method.
- This algorithm has good aggregate and high computing one.

4.3 Applications in real world

Some common applications using SVMs in several fields below as follows

Face detection - SVMs classify features in the image as faces or non-faces to draw a boundary box round what are faces.

Text and hypertext categorization - Both inductive and transductive models use training data to classify texts and hypertexts into contrasting classes first, then SVMs based on the basis of the generated score to categorize and compares with the threshold value.

Classification of images - Using the SVMs provides a better searching accuracy for image classification compared to the traditional query-based searching techniques.

Bioinformatics - This field comprises protein classification as well as cancer one, and the SVMs model is responsible for determining the genetic structure of patients classification based on the basis of genes and other biological problems.

Protein fold and remote homology detection - Applying SVMs algorithm for protein remote homology detection.

Handwriting recognition - The use of SVMs is to recognize widely used handwritten characters.

Generalized predictive control (GPC) - Using SVMs based on GPC for controlling chaotic systems with useful parameters.

5 Parameters

The selection of the optimal parameters is even more complicated because the generalization performance of the SVMs model (also known as its complexity) depends on the well-established all three parameters as follows

5.1 Kernel parameters

kernel: {'linear', 'poly', 'rbf', 'sigmoid'}, default='rbf'

The kernel within the SVMs is accountable for transforming the input data into the required format. A number of the kernels employed in the model are linear, polynomial, radial basis function (RBF) (also called Gaussian kernel), and sigmoid. While the opposite for creating a non-linear hyperplane is to use RBF as well as the polynomial kernel. The more advanced kernels are recommended for separating nonlinear nature classes in complex applications. We can build more accurate classifiers with this transformation. The foremost used sort of kernel function is RBF because of a localized and finite response along the entire x-axis.

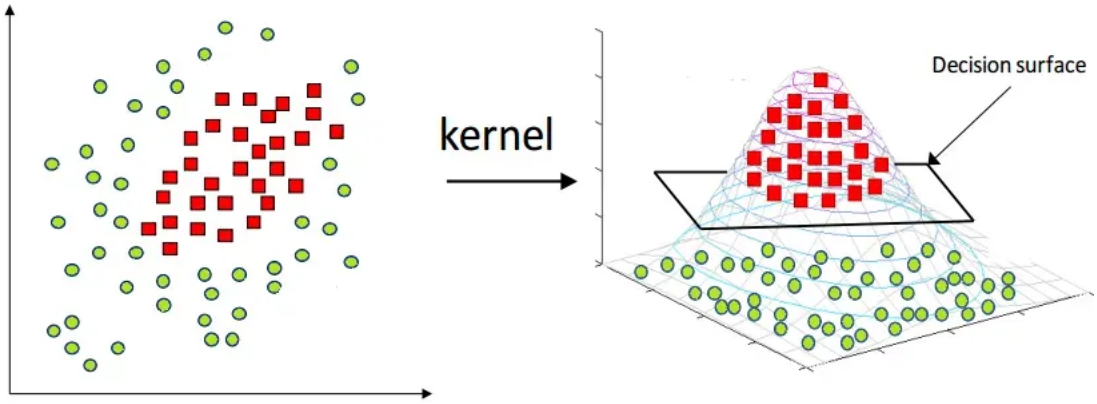

Figure 3: Kernel

5.2 Regularization parameters

C: float, default=1.0

The regularization parameter is considered as a penalty hyperparameter (denoted C) used to set the tolerance of the model allowing the misclassification of data points to be able to achieve a lower generalization error, where this

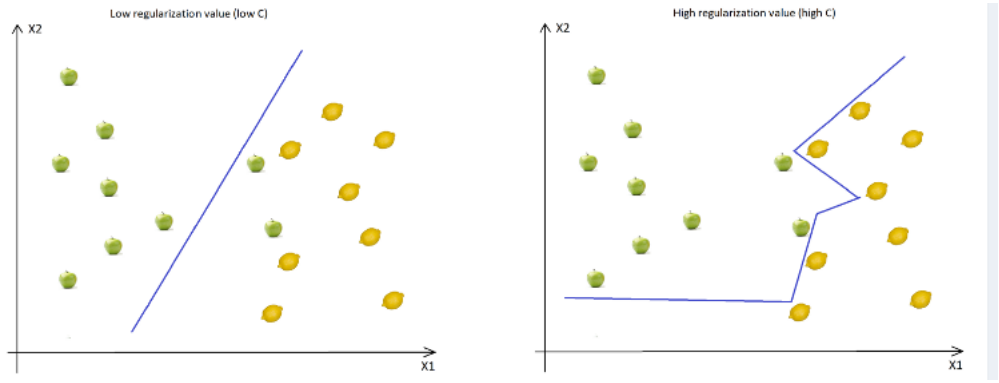

Figure 4: Regularization parameters

misclassification represents how much of the error is actually bearable. Through this, the compensation between the misclassified term and the decision boundary can be nullified.
What is trained depends on the value of the parameter C and the tolerance. While with the higher value of C and the smaller tolerance, the hard-margin classifier gives a better generalization view, the opposite is true for soft-margin classifier with inversion in the value pair of C and tolerance from the former. At the same time, large or small value of parameter C results in the corresponding penalty for misclassification, i.e., the large value of C lead to high penalty and vice versa. Besides that, a smaller margin will be accepted with the larger C if the decision function is better at classifying all training points correctly, but be aware that the model may overfit the training dataset. For the lower C would encourage a larger margin, so a simpler decision function, at the cost of training accuracy.

5.3 Gamma parameters

gamma: {'scale', 'auto'} or float, default='scale'

The gamma parameter determines how fit the training dataset is to the model, with the low value meaning a loose fit and the high value experiences an opposite trend, getting the fit for the model more appropriately. This ratio is also true for the pair of gamma values and the accuracy, which means the gamma value is proportional to accuracy. Not only that but for parameter gamma with low value only providing consideration for nearby points calculates the plane separately, while high value considers all data points to calculate the final separation line.

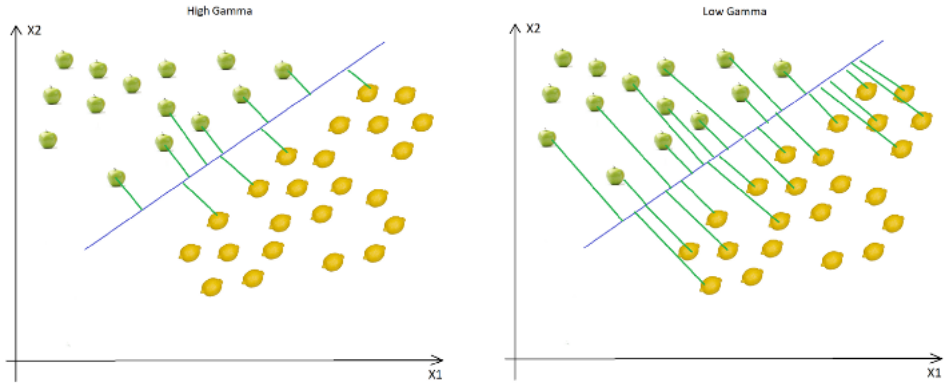

Figure 5: Gamma parameters

In short, gamma are intermediate values helping the model find good decision boundaries.

6 Hyperparameter optimization algorithms

In machine learning, hyperparameter tuning or optimization refers to the problem of automatically choosing optimal hyperparameters for a learning algorithm. Within the same machine learning model require different constraints, weights, or learning rates, collectively called hyperparameters, to generalize different data patterns. These measurement parameters have to be tuned ready to optimally solve the problem for its model. Hyperparameter optimization finds a tuple of meta-parameters that yields an optimal model which minimizes a predefined loss function on given independent data. Here are some popular algorithms

6.1 Grid search

The traditional technique of optimizing hyperparameters has been grid search, or a parameter sweep, which is solely an exhaustive searching through a manually specified subset of the meta-parameter space in a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set. Due to the parameter search space being discretized including real-valued or unbounded value spaces for certain parameters, that manually set bounds and discretization as the Cartesian products of them before applying grid search. The main reason why the grid is an embarrassingly parallel problem though suffers from the curse of dimensionality is to evaluate the hyperparameter settings independently of each other.

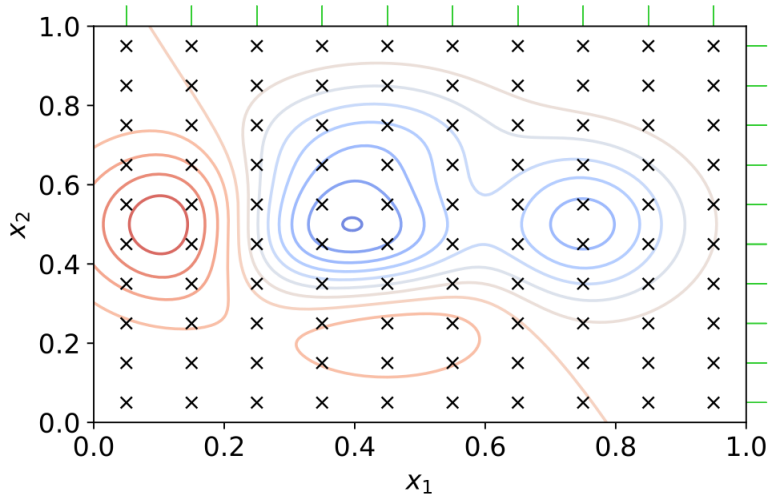

Figure 6: Blue contours indicate regions with strong results, whereas red ones show regions with poor results.

6.2 Random search

This is a variation that solves the limitations of grid search by finding the parameters in the grid at random instead of listing all the possible combinations. Random search only goes across a fixed number of settings hyperparameters that can be applied simply to discrete settings, as well as can be generalized to continuous and mixed spaces. This approach reduces unnecessary computations but still suffers the dimensional curse to achieve a preset fixed sampling density.

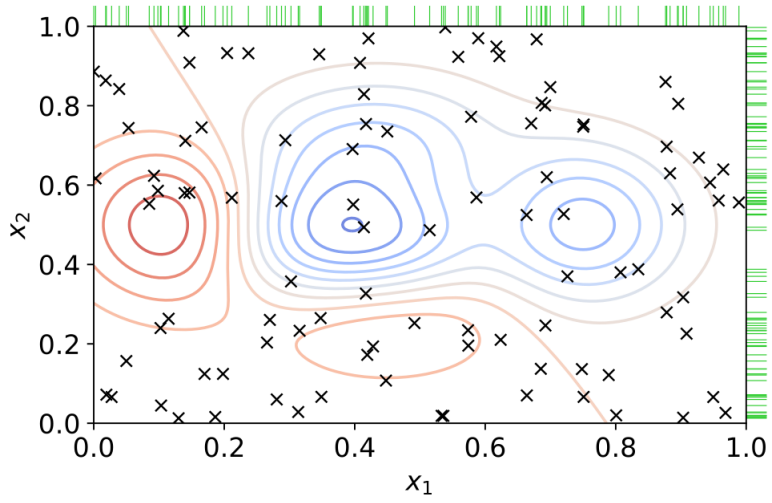

Figure 7: The green bars show that more individual values for each hyperparameter are considered compared to grid search.

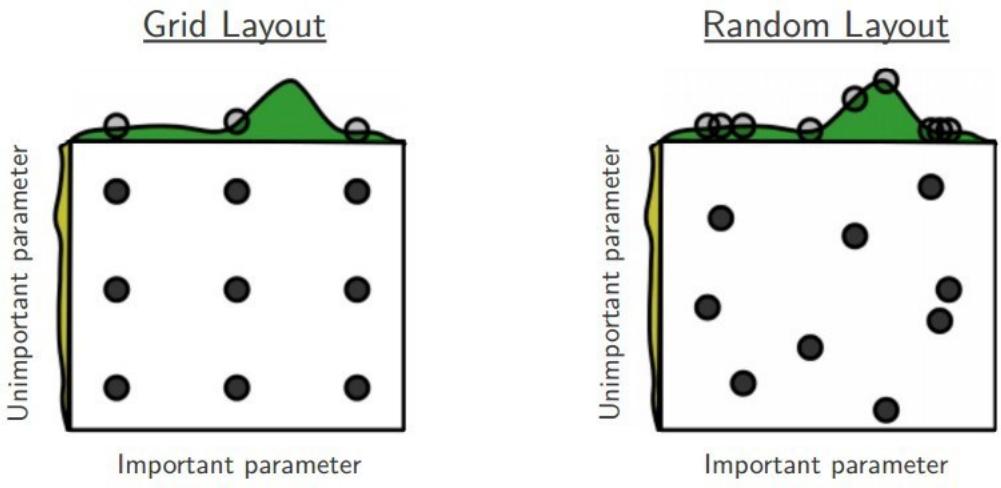

Figure 8: Benefits of using random search against grid search

6.3 Bayesian optimization

Bayesian optimization is one amongst a form of global optimization method for noisy black-box functions through building a probabilistic model of function mapping from hyperparameter values to the target evaluated on an validation set. By iteratively evaluating a promising hyperparameter configuration against the current model, then updating helps to gather observations that reveal the maximum amount as possible about this function and particularly its location of optimization. This is the method of trying to strike a balance between the hyperparameter for that the result is most uncertain and also the expected hyperparameter close to the optimal. In fact, Bayesian optimization is highly regarded for convalescing results than the above two methods because of its ability to reason about the quality of the tests before they are run.

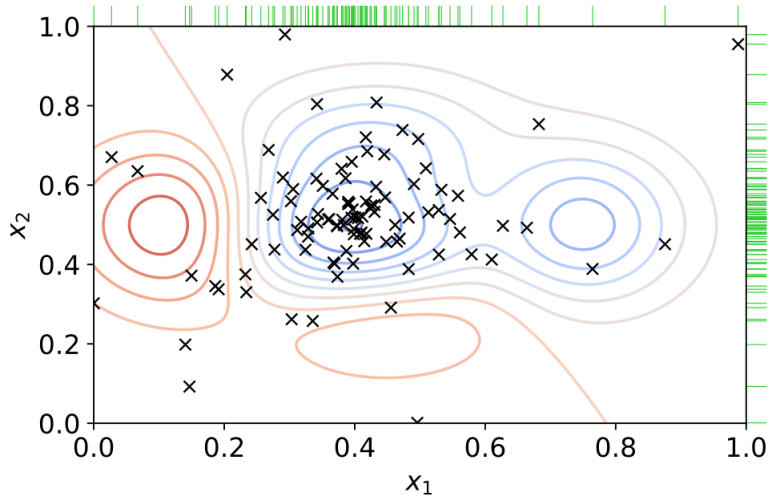

Figure 9: Methods like Bayesian optimization smartly explore the space of potential choices of hyperparameters by deciding which combination to explore next based on previous observations.

7 Experiment

Before embarking on an experiment, it is essential to understand each step of the setting and the meaning of the parameters because the way of fine-tuning those parameters affects the generality of the model as well as how to achieve the best results. After understanding the essence of what the SVMs algorithm is, the general advantages and disadvantages of applying that algorithm to a real model, we summarize that this is a highly recommended algorithm for analysis problems. Therefore, we decided to build a model using SVMs to classify images into the corresponding label.

7.1 Dataset

We first give a brief introduction to the dataset used to train the model called **Fashion-MNIST**. This is a dataset of Zalando's article images consisting of 60,000 samples in a training set and 10,000 samples in a test set. Each sample related to a label from 10 classes is a 28×28 greyscale image. Fashion-MNIST intended to serve as a direct drop-in replacement for the original **MNIST** dataset for benchmarking machine learning algorithms, with sharing the identical image size and structure of training and testing splits. Per the creators of Fashion-MNIST, here are some good reasons to stand in lieu of MNIST dataset

- MNIST is too easy.
- MNIST is overused.
- MNIST can not represent modern CV tasks as noted by Keras author François Chollet "Many good ideas will not work well on MNIST (e.g. batch norm). Inversely many bad ideas may work on MNIST and no transfer to real CV."

7.2 Visualization

Fashion-MNIST dataset is classified into 10 classes with balance data between classes. Each training and test sample is assigned to one of the following classes (called labels)
Data visualization is not only useful for cleaning or exploring data structures, detecting outliers and groups of

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

anomalies, identifying trends and clusters, and detecting local patterns, but also is considered an important step to consider whether the dataset is suitable for our model or not and to evaluate the output of the model. Therefore, the creators used the following two methods for the analysis of two datasets are MNIST and Fashion-MNIST to check data quality and demonstrate the limitations of the former compared to the latter.

7.2.1 UMAP

Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualisation in particular, as well as for general non-linear dimension reduction. The algorithm is founded on three assumptions about the data:

- Uniform distribution on a Riemannian manifold.
- The Riemannian metric is locally constant (or can be approximated as such).
- The manifold is locally connected.

From these assumptions, it is possible to model the manifold with a fuzzy topological structure. The embedding is found by looking for a low dimensional projection of the data that has the closest possible equivalent fuzzy topological structure.

UMAP is extremely efficient at embedding large and high dimensional datasets. It particularly scales well with both input dimension and embedding dimension. For the most effective possible performance, we recommend installing the closest neighbour computation library **pynndescent**. UMAP will work without it, but if installed it will run faster, particularly on multicore machines.

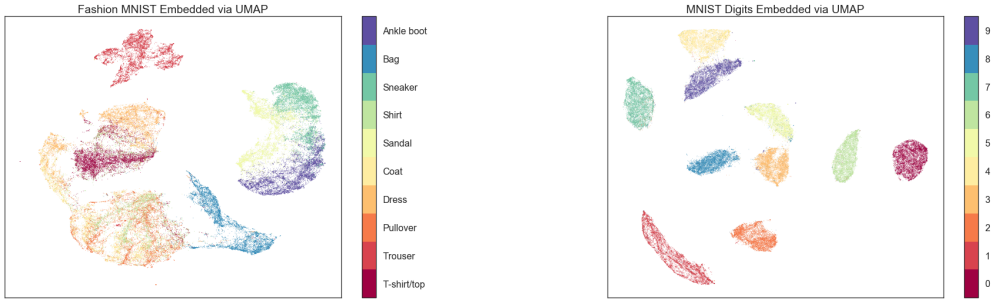

Figure 10: UMAP on Fashion-MNIST (left) and original MNIST (right)

From the figure 10, we can see that when applying the UMAP algorithm to survey the complexity and feasibility of 2 datasets, MNIST is much fairly straightforward than Fashion-MNIST.

7.2.2 PyMDE

PyMDE is a Python library for calculating vector embeddings for finite sets of elements such as images, nodes in a network, or any other abstract object. What sets PyMDE apart from other embedding libraries is that it provides a simple yet popular embedding framework called Minimum Allocation Embedding (MDE), which makes it easy to recreate famous embeddings and generate custom-builds adapted to a specific application. This library is used to visualize data sets, large or small; generate feature vectors for supervised learning; high-dimensional vector data compression; and more.

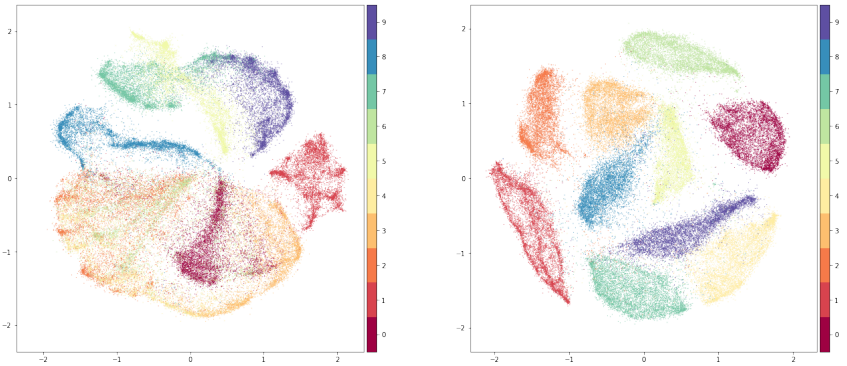

Figure 11: PyMDE on Fashion-MNIST (left) and original MNIST (right)

7.3 Implement a classification model to classify images

Before starting to implement any model, we need to import some of the following libraries (some other libraries listed in more detail in the code file, these are just a few necessary ones)

```
import os
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
```

```
import torch
import torch.nn as nn
from sklearn.svm import SVC
from torchvision import models
from torch.autograd import Variable
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
import torchvision.transforms as transforms
from sklearn.metrics import confusion_matrix, classification_report
```

The input to the SVMs model is a pair (x, y) where x is the prediction data matrix where each row is an observation and each column is a prediction and y is an array of labels with each row corresponding to the value of the

corresponding row in x. Here the y value can be a categorical array, a character or a string, a logical or numeric vector, or a cell array containing a character vector. Recall the Fashion-MNIST dataset consisting of an image dataset and a corresponding label set as input pairs x and y of the model, so we have to preprocess the data from the image to the feature matrix by applying pretrained AlexNet model to extract features of those images.

```
model_alexnet = models.alexnet(pretrained=True)
model_alexnet.classifier = model_alexnet.classifier[0:len(model_alexnet.classifier)-1]
```

Because the AlexNet model processes the input image as a colour image, but with FashionMNIST, the data only has one channel, so we have to convert it to correspond to the default input of the model to avoid error. Here we do that on command line 4.

```
def processing(input_img):
    img_ = np.reshape(input_img, (28,28))
    img_pil_ = Image.fromarray(img_)
    img_pil = Image.merge("RGB",(img_pil_, img_pil_, img_pil_))
    to_tensor = transform(img_pil)
    to_tensor.unsqueeze_(0)
    to_var = Variable(to_tensor)
    to_feature = model_alexnet(to_var)
    to_out = np.squeeze(to_feature.data)
    return to_out
```

In the process of extracting features from the input images, we will classify those features into a training set or a test set corresponding to the original position of the image as the following commands.

```
train = []
test = []
for i in range(60000):
    if i < 10000:
        img_test = test_images[i]
        test_feature = processing(img_test)
        test.append(test_feature)
    img_train = train_images[i]
    train_feature = processing(img_train)
    train.append(train_feature)
print(len(train), len(test))
```

After finishing processing the data into the desired format, we start tuning the hyperparameters, kernel, gamma and C respectively by GridSearch algorithm to find the best hyperparameters so that the SVMs model achieves the desired results with the highest possible accuracy. This whole process we only perform on the training set.

```
# Set up parameters
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', 'sigmoid', 'linear']}
```

```
# Split training set and finding para set
x_train, para_x, y_train, para_y = train_test_split(train_, labels, test_size=0.1, random_state=19520197)
para_x_train, para_x_test, para_y_train, para_y_test = train_test_split(para_x, para_y, test_size=0.2,
                                                                        random_state=19521281)
```

```
# Find super para
grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
grid.fit(para_x_train, para_y_train)
```

We use the best hyperparameter we just found to calculate the accuracy of the SVMs model on both the train and test sets, and then plot the confusion matrix to see the number of correctly predicted images corresponding to that label. Do the same for the SVMs model with default parameters and the model using the linear kernel.

```
# Use the newly found parameters training data set
modified_clf = SVC(C=best_param['C'], kernel=best_param['kernel'], gamma=best_param['gamma'])
modified_clf.fit(x_train, y_train)
print("---Result of Modified SVC with hyperparameters---")
print("Train score: ", modified_clf.score(x_train, y_train))
print("Test score: ", modified_clf.score(test_, test_labels))
```

```
# Predict Modified SVM
y_pred = modified_clf.predict(test_)
cnf_matrix_nb = confusion_matrix(test_labels, y_pred, labels=sample)
np.set_printoptions(precision=2)
plt.figure()
plot_confusion_matrix(cnf_matrix_nb, classes=sample,
                      title=f'CM, without normalization, MODEL Modified SVM')
```

7.4 Experimental results

7.4.1 Tuning hyperparameters

Applying grid search algorithm to refine three types of common parameters in SVMs model, we get the following three statistical tables corresponding to three kernels respectively RBF, Sigmoid and Linear. For each table will show the score that the kernel achieved corresponding to the pair of parameters C and gamma as well as the execution time of that kernel.

Looking at the statistics table of RBF we can see that this kernel gives the best results although in the first steps the results seem low, and we predict that it is the linear that gives the optimal results. But the results proved that our prediction was wrong. RBF returns best results up to 84.9% when C value is 10 and gamma is 0.001.

[C] / [gamma]	0.1	1	10	100	1000
1	0.106 (2.0 min)	0.106 (1.4 min)	0.106 (1.5 min)	0.106 (1.5 min)	0.106 (1.5 min)
0.1	0.106 (1.8 min)	0.138 (1.4 min)	0.144 (1.5 min)	0.144 (1.5 min)	0.144 (1.5 min)
0.01	0.376 (1.7 min)	0.732 (1.3 min)	0.743 (1.3 min)	0.743 (1.3 min)	0.743 (1.3 min)
0.001	0.788 (1.2 min)	0.83 (31.8s)	0.849 (30.3s)	0.848 (31.2s)	0.848 (31.1s)
0.0001	0.702 (1.6 min)	0.795 (40.2s)	0.835 (27.2s)	0.845 (24.5s)	0.837 (24.7s)

Table 1: Statistics table of RBF kernel

Although it returns the best value, RBF is not a hyperparameter that affects the execution speed of the model, but rather a kernel Linear. When looking at the statistics table, we can see that the linear always gives high results close to the score of the kernel RBF when the execution time is only about 15 seconds for the first results. That's why it made us confused when we just looked at the first lines of results.

[gamma] \ [C]	0.1	1	10	100	1000
1	0.834 (11.9s)	0.831 (9.2s)	0.831 (9.6s)	0.831 (9.6s)	0.831 (9.6s)
0.1	0.834 (11.5s)	0.831 (9.0s)	0.831 (9.6s)	0.831 (9.7s)	0.831 (9.6s)
0.01	0.834 (14.1s)	0.831 (9.6s)	0.831 (9.6s)	0.831 (9.5s)	0.831 (9.7s)
0.001	0.834 (14.0s)	0.831 (9.6s)	0.831 (9.7s)	0.831 (9.6s)	0.831 (9.6s)
0.0001	0.834 (9.1s)	0.831 (9.6s)	0.831 (9.7s)	0.831 (9.4s)	0.831 (9.5s)

Table 2: Statistics table of Linear kernel

To be more general and intuitive, we will look for the value of the C and gamma pair of the kernel RBF in the kernel Linear to see the result and its speed.

```
C=10 gamma=0.001 kernel=rbf; ave_score=0.849 ave_time=30.3s

C=10 gamma=0.001 kernel=sigmoid; ave_score=0.556 ave_time=13.3s

C=10 gamma=0.001 kernel=linear; ave_score=0.831 ave_time=9.7s
```

Figure 12: The average score between 2 kernels RBF and Linear

From the figure 12 we can conclude that although RBF gives the best results, Linear gives relatively a good score as well as running time only one third compared to RBF. So if we need a model that executes fast and returns relatively good results, then we can consider using kernel Linear for any value of hyperparameter C with gamma because of their score always in the top.

[gamma] \ [C]	0.1	1	10	100	1000
1	0.106 (1.2 min)	0.106 (51.5s)	0.106 (54.8s)	0.106 (54.7s)	0.106 (54.7s)
0.1	0.106 (1.2 min)	0.106 (50.7s)	0.106 (55.0s)	0.106 (54.8s)	0.106 (54.8s)
0.01	0.106 (1.4 min)	0.106 (55.9s)	0.106 (55.9s)	0.106 (55.6s)	0.106 (56.1s)
0.001	0.534 (1.0 min)	0.654 (23.9s)	0.556 (13.3s)	0.584 (9.7s)	0.58 (9.6s)
0.0001	0.602 (35.5s)	0.777 (21.4s)	0.826 (12.3s)	0.845 (9.6s)	0.832 (9.6s)

Table 3: Statistics table of Sigmoid kernel

Finally, based on the above table, prove that this is the kernel that gives the worst results in both the execution time and the model's score when gamma has values of 1, 0.1, 0.01 respectively for any value of C. The reason we think such an outcome occurs is that when the output of the Sigmoid activation function is very close to 0 or 1, the gradient will be almost zero, thus making it impossible for the back-propagation to continue updating some model parameters. Therefore, if the model parameters are not initialized properly, the Sigmoid function may have a gradient close to zero in the positive interval, resulting in the model not being trained efficiently.

7.4.2 Modified SVMs model

After finding a combination of hyperparameters that helps the model achieve optimal results, it is put into the SVMs model to evaluate the training data set to check how much the optimal level is.

Train score	Test score
0.966	0.889

Table 4: **Result of Modified SVC with hyperparameters**

After training the model based on the training dataset, we continue to take the test set to re-evaluate whether the model is really optimal or not through the model's prediction results.

	precision	recall	f1-score	support
T-shirt/top	0.96	0.95	0.96	1000
Trouser	0.97	0.98	0.97	1000
Pullover	0.80	0.84	0.82	1000
Dress	0.88	0.89	0.89	1000
Coat	0.83	0.83	0.83	1000
Sandal	0.97	0.96	0.97	1000
Shirt	0.71	0.68	0.69	1000
Sneaker	0.93	0.96	0.95	1000
Bag	0.83	0.83	0.83	1000
Ankle boot	0.99	0.97	0.98	1000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

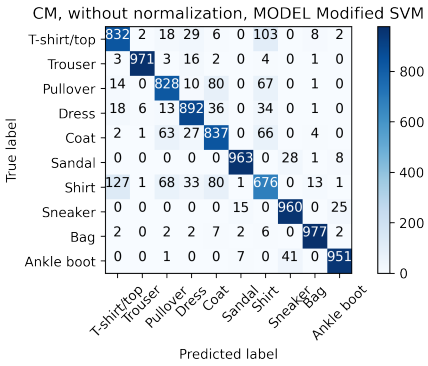

Figure 13: Modified SVMs model prediction

7.4.3 Default SVMs model

We wondered if the SVMs model was run with default hyperparameter values, what would be the result and whether it would be better than the above method or not.

Train score	Test score
0.885	0.867

Table 5: **Result of Default SVC model**

The table above shows that the score when training the model on the train set and the test set of the training dataset is about 0.081 lower for the train set and 0.022 for the test set compared to the modified SVMs model.

	precision	recall	f1-score	support
T-shirt/top	0.95	0.95	0.95	1000
Trouser	0.97	0.97	0.97	1000
Pullover	0.77	0.78	0.78	1000
Dress	0.85	0.88	0.86	1000
Coat	0.80	0.78	0.79	1000
Sandal	0.96	0.94	0.95	1000
Shirt	0.66	0.62	0.64	1000
Sneaker	0.92	0.95	0.94	1000
Bag	0.81	0.83	0.82	1000
Ankle boot	0.99	0.96	0.98	1000
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

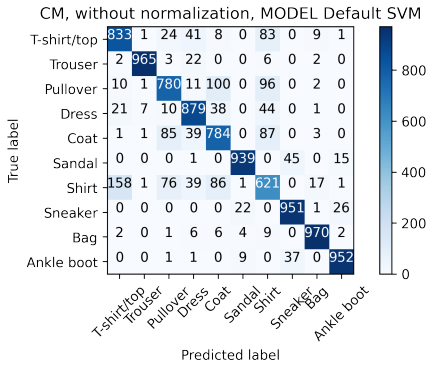

Figure 14: Default SVMs model prediction

7.4.4 Linear SVMs model

In the process of finding the best hyperparameter above, we found that the model using the Linear kernel returns good scores for any value of the hyperparameters C and gamma. So in this part, we just change the kernel value without affecting the other two hyperparameters to see how the training score is.

Train score	Test score
0.968	0.863

Table 6: **Result of Default SVC model**

From the above scoreboard, we conclude that the linear SVMs model only returns high scores for the train set, and when tested on the test set, it will predict less accurate than our targeted model with hyperparameters (about 0.863 for 'linear' compared to 0.889 for model with specified parameters)

	precision	recall	f1-score	support
T-shirt/top	0.95	0.95	0.95	1000
Trouser	0.96	0.98	0.97	1000
Pullover	0.75	0.79	0.77	1000
Dress	0.87	0.86	0.86	1000
Coat	0.78	0.79	0.78	1000
Sandal	0.96	0.95	0.96	1000
Shirt	0.65	0.63	0.64	1000
Sneaker	0.93	0.94	0.94	1000
Bag	0.81	0.78	0.79	1000
Ankle boot	0.98	0.98	0.98	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

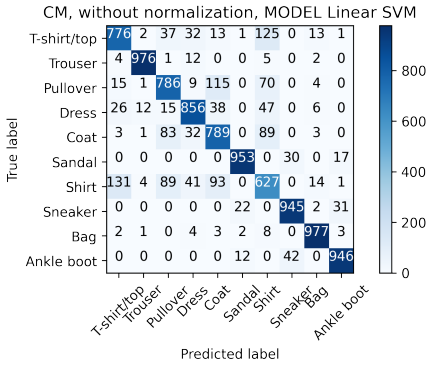

Figure 15: Default SVMs model prediction

8 Conclusion

SVMs are the advanced Machine Learning model that can achieve high scores in classification. Although this model is efficient, it still has slow performance in classifying process due to the large number of vectors being put into the model, what kernel is the best fit, etc. Furthermore, it requires big memory spaces in the training process, which can lead to an overflow issue during storing data process. The hypertuning parameters step plays an important part in a whole experiments in order to have better result. Based on the results from the above experimental part, the accuracy is not less than 85% and the confusion matrix shows that the SVMs model predicts quite well for the classes. If the better parameter is found, the prediction rate will also be more accurate, only a few classes with similar features are skewed as in the t-shirt/top and shirt classes. By far, we believe that SVMs can be the best model to do classify if we can solve the opened problems such as finding good parameters, preprocessing datasets, etc.

9 Appendix

9.1 AlexNet

Alexnet is the model that won the Imagenet large-scale visual recognition challenge in 2012. The model was first proposed in 2012 in the paper research named "Image Classification with Deep Convolution Neural Network" by Alex Krizhevsky and his colleagues. This model has 8 layers with learnable parameters, consists of five layers with

a combination of max pooling followed by 3 fully connected layers, and they use ReLU activation in each of these layers except the output layer. Additionally, this model is trained on the Imagenet dataset.

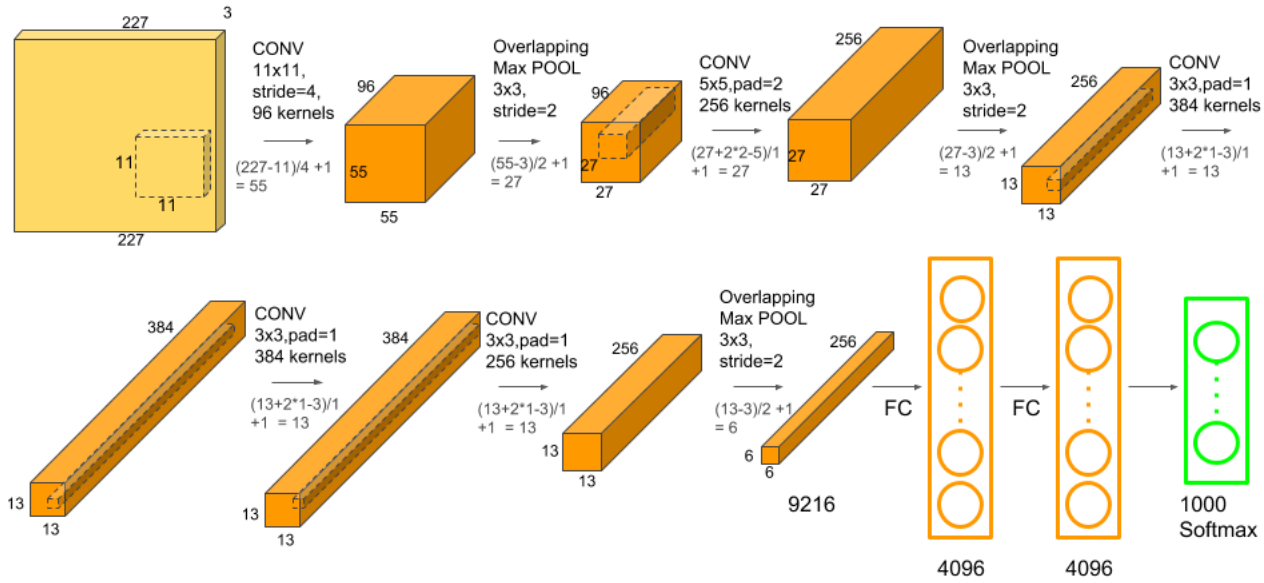

Figure 16: AlexNet Architecture

9.2 Performance measures

Performance measurement is the process of collecting, analysing and/or reporting information regarding the performance of an individual, group, organization, system or component.

9.2.1 Accuracy

Accuracy is a ratio of true predicted observation to the total observations, and it is very intuitive. If our model has high accuracy, and we ensure that there is no overfitting issue appearing in training and testing process, then, our model may have more accurate in predicting and can be trusted.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

9.2.2 Precision

Precision is the rate of true predicted positive observations to the total predicted positive observations. If precision gets high score, the false positive rate will at low.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

9.2.3 Recall

Recall is the proportion of true predicted positive observations to all observations in actual class.

9.2.4 F1 score

F1 score is the weighted average of Precision and Recall, and this score takes both false positives and false negatives into account.F1 is usually more useful than accuracy in case there is no different in cost for both false positives and false negative in Accuracy, Therefore, if there is different in cost of false positives and false negatives, we should look at both Precision and Recall.

$$F1Score = \frac{2 * (Recall * Precision)}{(Recall + Precision)} \tag{3}$$

9.2.5 Confusion metrics

A confusion matrix is a table that is used to define the performance of a classification algorithm. It also visualizes and summarizes the performance of a classification algorithm.

References

[1] Rohith Gandhi, Support Vector Machine - Introduction to Machine Learning Algorithms, on Towards Data Science website

[2] Jake VanderPlas, "In-Depth: Support Vector Machines", in Python Data Science Handbook

[3] Support Vector Machines Tutorial - Learn to implement SVM in Python, on DataFlair website

[4] Kernel Functions - Introduction to SVM Kernel & Examples, on DataFlair website

[5] Support Vector Machines, on scikit-learn website

[6] Ajitesh Kumar, SVM RBF Kernel Parameters with Code Examples, on vitalflux website

[7] Renuka Joshi (Intern at Exsilio), "Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures"

[8] Li Yang, Abdallah Shami, On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice, arXiv:2007.15745 [cs.LG]

[9] LiYangHart, Hyperparameter-Optimization-of-Machine-Learning-Algorithms, https://github.com/LiYangHart/Hyperparameter-Optimization-of-Machine-Learning-Algorithms

[10] Zalando Research, Fashion-MNIST, https://github.com/zalandoresearch/fashion-mnist