Fashion MNIST

May 5, 2022

Anish Kasi, anishk4
Harris Nisar, nisar2 (Team lead)
Saba Paya, snpaya2

Contents

1 Introduction

For this project we are given the Fashion MNIST data to predict the class label in the testing data [3]. The dataset we are working with contains 70,000 28x28 images of 10 different categories of clothing where 60,000 are placed in the training dataset and the remaining 10,000 are used for the testing data. An example of the dataset can be seen in Figure 1.

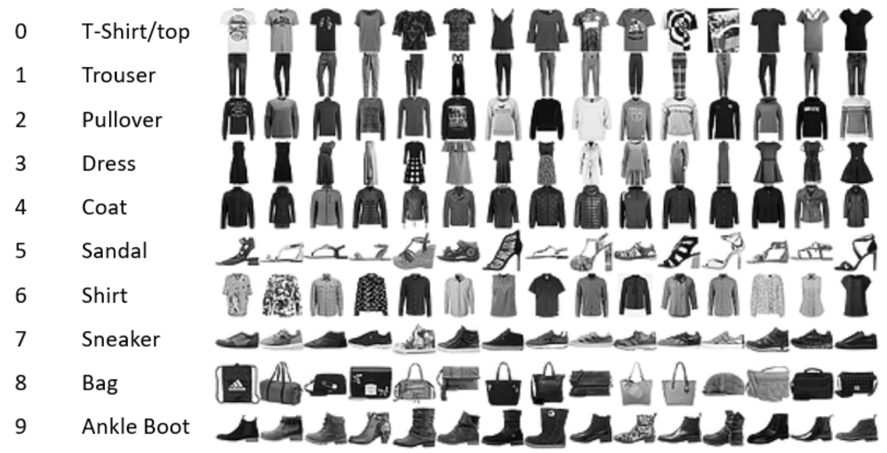


Figure 1: Sample of Fashion MNIST

To begin our analysis, we first performed exploratory data analysis to determine the class distribution of the training and testing datasets and found the class distribution to be balanced. We also clustered the dataset using k-means and found that the dataset was best clustered when using 10 clusters, matching the number of classes in the dataset. From there, we pre-processed the data using principal component analysis (PCA) and found that in order to explain 90 percent of the variance in the data, we need 137 principal components. To classify the dataset using the 137 principal components, we trained a k-means model, a light gradient boosting machine (LGBM), and a multi-class logistic regression model. All models returned a testing accuracy of at least 0.85. Finally, we developed an ensemble model that combines outputs from several base models as features in new datasets. We then used these datasets to train various meta classifiers.

2 Literature Review

The Fashion-MNIST dataset is popularly used for machine learning models. As a result, many efforts exist to predict the class labels of the different clothing items. Convolutional neural networks (CNN) are the current state of the art for computer vision tasks. In [2], the authors focus on four different CNN architectures and compare them to traditional non-convolutive machine learning algorithms to see which ones perform better. CNNs take into account the spatial information of the region (i.e. image) and detect shapes and edges of the region to detect images. Since CNN's tend to overfit, the authors chose to further implement a dropout method which consists of temporarily removing neurons from a neural network. Their implementation consisted of the following four models: CNN with a dropout of 1, CNN with a dropout of 2, CNN with a dropout of 3, and simple CNN model with fewer layers. The CNN dropout of 3 had the highest test accuracy at 99.1% and all 4 of the CNN models outperformed traditional non-convolutive machine learning algorithms [citation needed]. Through our literature review, we also found this model to have the best accuracy on this dataset.

Another common approach is to use CNNs as a feature extractor. These features are then used to train another type of model. Usually CNNs are not trained from scratch, but rather use a transfer learning framework. Transfer learning involves training a model to perform a specific task but leveraging layers trained on a different task. This second task often has has a large dataset so the network can learn important features that are more general to the task. Cayir et al. [4] use convolutional networks and transfer learning methodologies to extract features from an image using a pre-trained convolutional network. They use these features to fit classical machine learning methods, including SVM, random forest and gradient boosting machines. They test this method using the fashion MNIST dataset and find that CNN in combination with random forest did the best (0.9084 test accuracy). It is not surprising that the first approach outperformed this one. Because the first model was trained end to end, the convolutional layers could learn specific features for the fashion MNIST dataset. In the second approach, only the final classifiers were trained. Moreover, in the first approach, the final prediction was based on some non-linear combination of the features extracted from the convolutional layers. In the second approach, these features were not combined, but rather flattened and fed into the final classical models.

3 Exploratory Data Analysis

3.1 Summary Statistics

In order to begin our analysis, we had to first explore the dataset. The data contained 10 labels ranging from 0 to 9 as the response variable. The training data for each of the 10 labels contained 6,000 observations while the testing data labels contained 1,000 observations (Table 1).

Table 1: Train and Test Class Label Frequency

Class Label	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Train	6,000	6,000	6,000	6,000	6,000	6,000	6,000	6,000	6,000	6,000
Test	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000

3.2 Unsupervised Learning

Unsupervised learning is a technique used to analyze and cluster unlabeled datasets by looking for patterns. Clustering and dimensionality reduction are examples of two types of algorithms that exist to help identify these hidden patterns. When performing unsupervised learning algorithms, the goal is to get insights from large volumes of new data rather than trying to predict the outcome variable. Since our data contains a large number of image data, performing unsupervised learning techniques will allow us to identify observations that are similar and categorize them.

Variable clustering is a mathematical way to group similar variables together into subsets or cluster groups to understand the data. We then choose the most predictive variables of the cluster groups held by similar information. Since we don't choose every variable from each cluster, this is another way of reducing the predictor variable pool and can be further used for feature selection techniques. With clustering we are essentially splitting up the data into groups and seeing how well the variables in a group relate to each other and how different they are from the variables placed in the other clusters. When performing clustering techniques, measuring the distance between points is important as these distance metrics (i.e. Euclidean, Manhattan, Hamming, etc) determine how points should be clustered by where they are in proximity to each other while looking to minimize the distance within cluster groups and maximize the distance between clusters.

3.2.1 K-Means Clustering

One of the most common clustering methods is k-means clustering. Once the number of clusters is chosen, the algorithm randomly assigns each observation to an initial cluster. For each of the clusters, the cluster mean vector is computed and each observation gets assigned to the closest current cluster mean. The algorithm keeps iterating until there are no more changes to the cluster assignments.

To perform k-means clustering, we first, scaled each variable in the dataset to have a mean of 0 and a standard deviation of 1 so all the variables have the same unit. Since we don't know the optimal number of clusters, we, then, tried a range of 2-18 clusters and compared their with-in cluster sum of squares. We saw that the sum of squares began leveling off around 10 and thus chose that

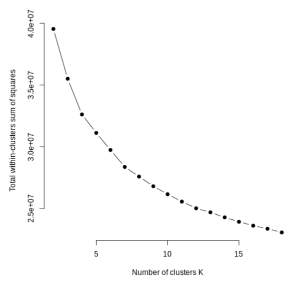


Figure 2: K-Mean Clusters

Table 2: Dominating Label for Each Cluster Group

Cluster	Label
1	8
2	6
3	7
4	0
5	1
6	9
7	5
8	9
9	8
10	4

as our optimal number of clusters (Figure 2). Using 10 as our optimal k value, we ran our k-means algorithm and found that the clusters do a pretty decent job of separating the labels. Table 2 shows us that 8 of the 10 labels were the most common label of their cluster. Only two labels, 8 and 9, were the most common label for more than one cluster – both having two clusters where they were the most common.

3.2.2 Principal Component Analysis (PCA)

PCA is a dimension reduction technique that derives a low-dimensional set of features from a high-dimensional list of variables that still contain most of the information. It uses a mathematical technique that transforms potentially correlated variables into a smaller number of uncorrelated variables, principal components. The first principal component captures the most variance in the data through making linear combinations of the original variables that give the most variance. Each principal component after accounts for as much of the remaining variability as possible. The second principal component is the linear combination of variables that takes into account the remaining variance as much as they can and so on.

Centering and scaling variables to be on the same unit range is crucial for PCA as well. This allows the variables to be comparable. However, scaling is not always necessary and rather dependent on the context of the data. Variables that are extremely disproportionate should be scaled to prevent some variables from overpowering the PC loadings because of their larger scales. When variables are more similar, as with image pixels in our case, it is better to keep the data unscaled so as to not lose signal information that variables with larger variations contain.

Once we centered the data, we plotted the principal components against the variances to see how influential each principal component is. Figure 3 shows

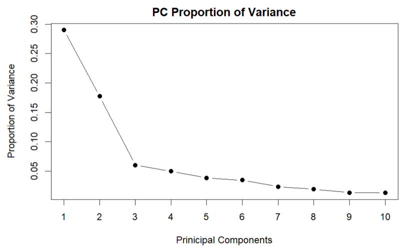


Figure 3: PC Proportion of Variance

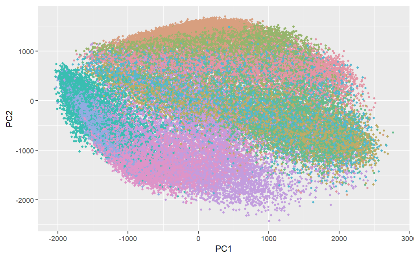


Figure 4: PCA for 10 Labels

that the first two principal components provide the most influence. The first principal component captures about 29% of the variance and the second principal component captures 18% of the variance. Figure 4 further depicts a visual representation of the 10 class labels. There is visible separation between the classes but also some overlap.

4 Multi-class Classification Model

After exploratory data analysis, we explored building classification models using this data. The classification problem entails a training dataset of examples, $D_n : \{x_i, y_i\}_{i=1}^{n}$ (n is number of samples), where each $x_i \in R^p$ and represents the features (p number of features) and each $y_i \in \{0, .., C-1\}$ is a class label, where C is the number of classes. The goal is to find some function f that maps the p-dimensional feature vectors to a class label:

$$f : R^p \to \{0, .., C\}$$

Before training, we processed the data using PCA and kept enough components to explain 90 percent of the variance in the dataset. This processing led to us using 137 features (p=137) instead of the original 784 features.

4.1 Methods

We trained a K-nearest neighbor (KNN) model, a gradient boosting model (LightGBM), and a multi-class logistic regression model to solve the classification problem. In order to tune the hyperparameters of each model, we used a grid search cross validation strategy using 5-fold cross validation. Grid search allowed us to try different combinations of hyperparameters. After fitting each model, we analyzed their performances and compared them to determine the best model for the classification task. Here we present each model in more detail.

4.1.1 K-Nearest Neighbor (KNN)

KNN is a non-parametric supervised learning method used for classification and regression tasks. In the classification setting, the output is a class membership. An object is classified by a vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. Formally, given a dataset of training examples, our prediction ($\hat{y_o}$) on a new point x_o, becomes:

$$\hat{y_o} = argmax_{c \in C} \sum_{x_i \in N_k(x_o)} 1\{y_i = c\}$$

where $x_i \in N_k(x_o)$ are the k nearest neighbors to the new point x_o.

For example, if k = 1, then the object is simply assigned to the class of that single nearest neighbor. Various metrics can be used to assess the closeness or similarity between two point, such as Minkowski distance, Manhattan distance, and Euclidean distance. Here we chose to use the Euclidean distance.

We tuned the number of neighbors using a 5-fold cross validation approach and found that 7 neighbors gave the best results. We then used the testing data to evaluate the model.

4.1.2 Gradient Boosting (LightGBM)

Like other boosting methods, gradient boosting combines weak learners into a single strong learner in an iterative fashion. Using this method, we learn an $F_T(x)$ such that:

$$F_T(x) = \sum_{i=1}^{n} \alpha_t f(x; \theta_t)$$

We fit this model by minimizing a loss function:

$$min_{\{\alpha_t, \theta_t\}_{t=1}^{T}} L(y_i, F_T(x_i))$$

where L is chosen based on the problem. Since this problem is challenging to solve over the entire parameter space, it is optimized in a stage wise fashion. Here we choose LightGBM, a gradient boosting framework developed by Microsoft [1] to fit our gradient boosting machine. This framework uses trees as a weak learner. There are several hyperparamters to tune in this model. We chose to tune the number of learners and the maximum number of leaves for each learner using 5-fold grid search cross validation. We found that 1000 learners and 40 maximum number of leaves gave the best results.

4.1.3 Logistic Regression

Logistic regression is commonly used for binary classification tasks. In this case, we take the output of a weighted sum of the features and transform that to a probability distribution using the sigmoid function. Here, we extend this approach to the multi-class setting to model the following probability distribution:

$$p(y_i = k | x_i) \forall K$$

In order to approximate this probability distribution, we utilize the more general softmax function. This function maps a R^p input to a R^p output, where the output can be interpreted as a probability distribution (all positive values that sum to 1). Each element of the output is calculated as follows:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

In the case of logistic regression, z is the output of the weighted sum of our features. In order to fit this model, we calculate the current models prediction ($\hat{y}$) and compare that to our training data labels using a cost function to optimize. A common cost function is the negative log likelihood loss:

$$NLL(\hat{y}, y) = -\sum_{k=1}^{K} y_k log(\hat{y}_k)$$

Combining this cost function with the softmax function, we get the final optimization problem:

$$argmax_w - \sum_{k=1}^{K} y_k log(softmax(w^T x))$$

We can also add a regularization term to prevent overfitting:

$$argmax_w - \sum_{k=1}^{K} y_k log(softmax(w^T x)) + C||w||^2$$

This optimization problem can be solved using gradient descent. Here we used L2 regularization. C represents the strength of regularization. We tuned this parameter using 5-fold cross validation and found C=0.9 to be the best choice for this hyperparameter. We then used the testing data to evaluate the model.

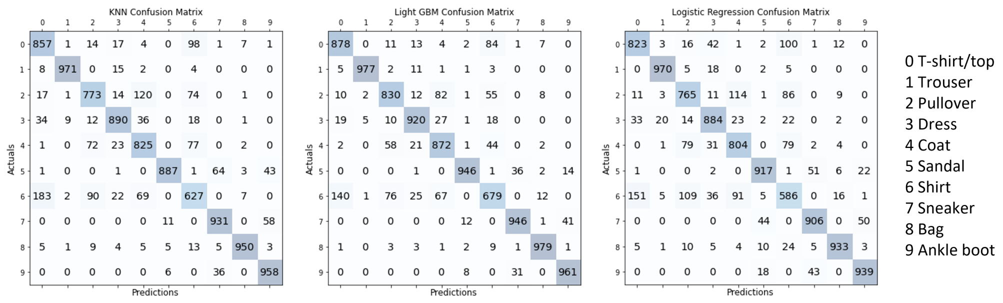
4.2 Results

In order to evaluate our models, we determined the overall accuracy on the testing dataset (Table 3). All of the models performed reasonably well (at least 0.85). Light GBM performed the best (0.9).

Table 3: Overall testing accuracy for each approach

	KNN	LightGBM	Logistic Regression
Overall Accuracy	0.87	0.90	0.85

In order to evaluate the effectiveness of our model at classifying particular types of clothing, we calculated the per class precision, recall, and f1-score (Table 4). Class 6 (shirts) had the worst performance based on these metrics. We also calculated the confusion matrix for each classifier (Figure 5). Based on these matrices, we see that class 6 was often classified as class 0 (t-shirt), class 2 (pullover), or class 4 (coat) for all three classifiers.



KNN Confusion Matrix

Actuals	0	1	2	3	4	5	6	7	8	9
0	857	1	14	17	4	0	98	1	7	1
1	8	971	0	15	2	0	4	0	0	0
2	17	1	773	14	120	0	74	0	1	0
3	34	9	12	890	36	0	18	0	1	0
4	1	0	72	23	825	0	77	0	2	0
5	1	0	0	1	0	887	1	64	3	43
6	183	2	90	22	69	0	627	0	7	0
7	0	0	0	0	0	11	0	931	0	58
8	5	1	9	4	5	5	13	5	950	3
9	0	0	0	0	0	6	0	36	0	958

Predictions

Light GBM Confusion Matrix

Actuals	0	1	2	3	4	5	6	7	8	9
0	878	0	11	13	4	2	84	1	7	0
1	5	977	2	11	1	1	3	0	0	0
2	10	2	830	12	82	1	55	0	8	0
3	19	5	10	920	27	1	18	0	0	0
4	2	0	58	21	872	1	44	0	2	0
5	0	0	0	1	0	946	1	36	2	14
6	140	1	76	25	67	0	679	0	12	0
7	0	0	0	0	0	12	0	946	1	41
8	1	0	3	3	1	2	9	1	979	1
9	0	0	0	0	0	8	0	31	0	961

Predictions

Logistic Regression Confusion Matrix

Actuals	0	1	2	3	4	5	6	7	8	9
0	823	3	16	42	1	2	100	1	12	0
1	0	970	5	18	0	2	5	0	0	0
2	11	3	765	11	114	1	86	0	9	0
3	33	20	14	884	23	2	22	0	2	0
4	0	1	79	31	804	0	79	2	4	0
5	1	0	0	2	0	917	1	51	6	22
6	151	5	109	36	91	5	586	0	16	1
7	0	0	0	0	0	44	0	906	0	50
8	5	1	10	5	4	10	24	5	933	3
9	0	0	0	0	0	18	0	43	0	939

Predictions

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

Figure 5: Confusion matrices for all methods

Table 4: Precision, recall, and f1-score for various techniques per class.

	0	1	2	3	4	5	6	7	8	9
Precision - KNN	0.77	0.99	0.80	0.90	0.78	0.98	0.69	0.90	0.98	0.90
Recall - KNN	0.86	0.97	0.77	0.89	0.82	0.89	0.63	0.93	0.95	0.96
f1 - KNN	0.81	0.98	0.78	0.90	0.80	0.93	0.66	0.91	0.96	0.93
Precision - LightGBM	0.83	0.99	0.84	0.91	0.83	0.97	0.76	0.93	0.97	0.94
Recall - LightGBM	0.88	0.98	0.83	0.92	0.87	0.95	0.68	0.95	0.98	0.96
f1 - LightGBM	0.85	0.98	0.83	0.92	0.85	0.96	0.72	0.94	0.97	0.95
Precision - Logistic Regression	0.80	0.97	0.77	0.86	0.78	0.92	0.65	0.90	0.95	0.93
Recall - Logistic Regression	0.82	0.97	0.77	0.88	0.80	0.92	0.59	0.91	0.93	0.94
f1 - Logistic Regression	0.81	0.97	0.77	0.87	0.79	0.92	0.62	0.90	0.94	0.93

4.3 Discussion

We found reasonable success in classifying clothing with all three methods. KNN worked well, perhaps due to some underlying latent space in the data. Logistic regression was the worst performer, but has advantages in the explainability of the results. All three models struggled with class 6 (shirts). However, it becomes clear why the model struggled when looking at the labels of the classes the models predicted. We find that class 6 was often confused with other types of garments that could be seen as a shirt, such as class 0 (t-shirt), class 2 (pullover), and class 4 (coat).

5 Ensemble Model and Feature Engineering

5.1 Overview

The Ensemble Model we use is a technique known as Stacking. Stacking or Stacked Generalization is an ensemble machine learning algorithm that uses a meta-learning algorithm to learn how to best combine the predictions from two or more base machine learning algorithms.

The benefit of stacking is that it can harness the capabilities of a range of well-performing models on a classification or regression task and make predictions that have better performance than any single model in the ensemble.

The final model in the stacking classifier can either take the probabilities or predictions of the base model as input.

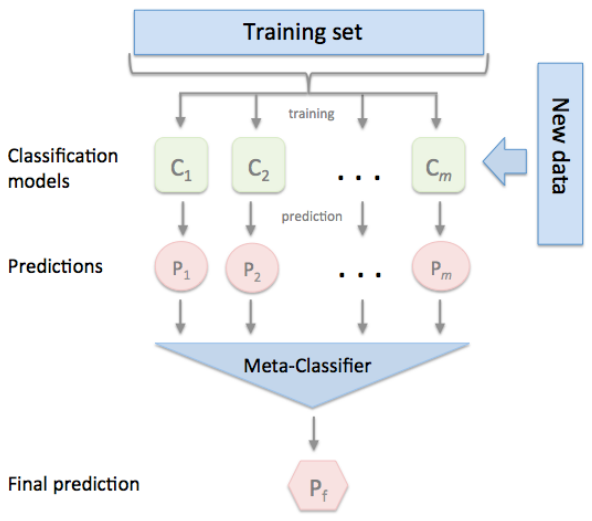


Figure 6: Stacking classification pipeline

5.2 Ensemble Model

The base models that we selected for our ensemble model were the following:

- Logistic Regression - It makes no assumptions about distributions of classes in feature space. It is efficient to train and provides a probabilistic interpretation of the class values.
- Gradient Boosted Decision Trees - This model was the best performing base model.
- Random Forest Model - This model uses bagging to get the best prediction. It does not assume that the model has a linear relationship. It performs feature subsampling and dataset subsampling reducing over-fitting.
- K Nearest Neighbors - This model is a discriminative model that classifies points based on their distance from each other. Since none of our other base models use this heuristic, we decided to add KNN as our final base model.
- Multinomial Naive Bayes - This model is a generative model and thus, tries to learn the joint distribution between x and y rather than finding the difference between classes. We chose this as our final model since it takes as input predictions from a discriminative model and models the joint distribution of these predictions.

The base models were trained using the best hyperparameters selected from the previous section and the predictions were then used as input to the final model.

One of the constraints that were specified is that the final model should have no more than 20 features. This means that we could not concatenate the output probabilities of each model since that would give us a 40-dimensional vector (4 models x 10 classes). We construct four datasets for the final model.

1. Dataset 1 - Pass in the average probability vector along with the individual predictions for each of the models which gives us a 14 dimensional vector (10 classes + 4 models).
2. Dataset 2 - Pass in the output probability vector of the best performing base model (GBDT) along with the predictions of the 4 models. 14-dimensional vector.
3. Dataset 3 - Use only the predictions of the 4 models. 4-dimensional input vector.
4. Dataset 4 - Use predictions along with the first 16 PCA components.

For each dataset, we train a GBDT, SVM and KNN classifier as our meta classifier and evaluate the performance of each model. We used Optuna for hyperparameter tuning of the meta classifiers. Optuna uses a history record of trials to determine which hyperparameter values to try next. Using this data, it estimates a promising area and tries values in that area. Optuna then estimates an even more promising region based on the new result. It repeats this process using the history data of trials completed thus far. Specifically, it employs a Bayesian optimization algorithm called Tree-structured Parzen Estimator. The following parameters were tuned for each model -

- GBDT - number of trees, lambda for L2, number of leaves
- KNN - number of neighbors
- SVM - kernel (linear, poly or rbf), C (tolerance)

We do not tune any hyperparameters for the Multinomial naive bayes model.

5.3 Results

We report the final accuracy of the meta classifier on the different datasets. Each model was tuned separately on each dataset. Each model was tuned for 50 trials (50 different combinations of hyperaparameters) and the model that gave the best accuracy was selected.

Looking at the table (Table 5), we see that the best performance was obtained using Multinomial NB classifier and Dataset 2. The performance of the discriminative stacking classifier is still poorer than the base model which means that the predictions of the different base models seem to be more or less the same and there is no extra information being passed to the model. However, the generative model performs well which means that the generative model is able to learn a good joint distribution. Note that the performance on dataset 3 for

Table 5: Ensemble model results

Model	Dataset 1	Dataset 2	Dataset 3	Dataset 4
GBDT(light GBM)	0.8798	0.8737	0.8699	0.8699
SVM	0.8629	0.8634	0.8621	0.8766
KNN	0.8679	0.8743	0.8606	0.8785
Multinomial NB	0.8892	0.8919	0.2083	NaN

multinomial NB is extremely poor since we just have label information and not the probabilities. Instead, we fit the multinomial NB models with just the prediction probabilities of Dataset 1 and 2 since the predictions seem to be hurting model performance. We found the performance to be boosted to 0.8939 and **0.8983** respectively. This means that using the predicted output probabilities of the GBDT as input to the multinomial NB provides the richest information. The multinomial NB could not be fitted with Dataset 4 as it cannot handle negative feature values. (PCA input featuers).

We analyse the best accuracy a model can achieve (check if the true label is predicted by any base model) on the train set: 100% and the test set: 93.78% This means that the model can achieve a considerable boost over the base models. In addition, we compare the results by performing a voting classification. Here, we simply perform a 'hard' voting by selecting the majority prediction of the different base models as the final prediction. Performing voting gave us a test accuracy of 0.8878 and a train accuracy of 0.9722. We see that voting outperforms the discriminative final models but not the generative model.

References

[1] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[2] Valderi Leithardt. Classifying garments from fashion-mnist dataset through cnns. *Advances in Science, Technology and Engineering Systems Journal*, 6(1):989–994, 2021.

[3] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[4] Aykut Çayir, Işil Yenidoğan, and Hasan Dağ. Feature extraction based on deep learning for some traditional machine learning methods. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, pages 494–497, 2018.