**Predicting Class Labels in Fashion MNIST Data**

**Kamila Makhambetova (kamilam3)**

**Katherine Vande Pol (vandepol)**

**Stephanie Chen (schen176)**

*Project Description and Summary*

This project utilizes the Fashion-MNIST dataset (Xiao, Rasul, and Vollgraf, 2017), which consists of training data with 60,000 observations (rows) and testing data with 10,000 rows. Each row represents a 28x28 pixel grayscale image, associated with a label of the image item, which is an object belonging to one of 10 classes: 0 (T-shirt/top), 1 (Trouser), 2 (Pullover), 3 (Dress), 4 (Coat), 5 (Sandal), 6 (Shirt), 7 (Sneaker), 8 (Bag), and 9 (Ankle boot). Each 28x28 image provides a total of 784 pixels, each having a value between 0 and 255, indicating the grayscale lightness or darkness, with higher numbers indicating a darker color. The training and testing data sets have 785 columns, with the first column consisting of the class labels and the other 784 columns containing the pixel-values of the associated images.

The objective of this project was to utilize the training dataset to develop a model that classified the testing data into the correct label classification based on the 784 pixel values for a majority of the testing data observations. Initial models utilized descriptive statistics and unsupervised learning on the training data to observe the data characteristics, followed by supervised models that utilized the training data to make predictions for the testing data classifications. Supervised models were based on machine-learning methods, with a final ensemble model that utilized the outputs of 3 of the most accurate models from the previous analyses to predict the testing data classifications. Models were evaluated based on testing accuracy, with the accuracies from the individual models being used to weight the models in the ensemble procedure (weighted voting). This ensures that the models with a higher accuracy had more weight in determining the predicted class.

Individual models evaluated included Random Forest (RF), linear discriminant analysis (LDA), logistic regression (LR), Gradient Boosting (GBM), and k-nearest neighbors (kNN). Tuning parameters for the GBM method were selected based on those reported in previous literature. The number of neighbors for kNN was determined using a search from 1 to 11, with the selected number being chosen based on the maximum testing accuracy. The testing accuracy for each model was 88.75% (RF), 82.56% (LDA), 85.36% (LR), 86.74% (GBM), 90.17% (tuned GBM), and 86.18% (kNN). Based on these accuracies, the RF, tuned GBM, and kNN models were selected for use in the final ensemble model, which resulted in a testing accuracy of 89.84%. Each of these models was also tested using principal components from the covariates (with enough principle components to explain 98% of the variation). However, all of the models using the principle components resulted in a lower testing accuracy than those with the original covariates, and were therefore not selected or reported. Of all individual and ensemble models, the model with the highest testing accuracy was the tuned GBM model, with an accuracy of 90.17%.

*Literature Review*

After reviewing the literature of other analyses on the Fashion-MNIST dataset, the analysis with the highest testing accuracy found was 98.11% (Deshpande, 2021). However, this paper utilized deep learning methods (including a convolutional neural network), which is not the topic of the current project. Therefore, papers only utilizing machine learning methods will be further discussed here. Of these that only utilized machine learning, the highest accuracy found was 90.6% (Bhadra, 2021).

Bhadra (2021) compared 5 methods to classify the Fashion-MNIST images. The first step of their analysis was to normalize the pixel values by dividing each by 255, creating a 0-1 scale. Next, they conducted a principal component (PC) analysis with 100 PCs. This is a technique where linearly uncorrelated vectors are selected to explain the maximum variation in the covariate values. These explain the majority of the variation of the covariates, but using 100 variables rather than 784. Next, several methods were tested and compared: LR, support vector machine (SVM), RF, GBM, and XGBoost. Of these, the 2 methods with the highest testing accuracy were the SVM (90.6%) and XGBoost (88.5%). SVM aims to find the best separating hyperplanes between the points with different class labels (such that the distance between class groups and the separating plane is maximized). This was done with a non-linear method, using a radial kernel. This allows for more flexibility and non-linear separation between the points. The basis of the XGBoost method is a decision tree, which uses each covariate and its values to create a set of cascading questions (e.g., does the observation have a PC1 value above or below 0.5?). This then sorts each observation according to its covariates into groups, with each group consisting of points that generally have the same class label. In this analysis, XGBoost used a set of 100 simple decision trees (with just 3 splitting decisions) to predict the 10 classes, using the mean classification error to tune the model to best performance.

The report of Kannavdhawan (2020) compared 11 methods: linear discriminant analysis (LDA), Naive Bayes, LR, perceptron, SVM with linear kernel, kNN, RF, decision trees, multi-layer perceptron, quadratic discriminant analysis (QDA), and SVM with a radial kernel. The covariates were standardized by dividing by 255 to achieve a 0-1 scale, and a PC analysis was conducted with enough PCs to explain 85% of the variation in the original covariates. Similar to Bhadra (2021), the two methods with the highest testing accuracy were SVM using the radial kernel (89.7%), and XGBoost (89.3%). The main differences between these papers’ methods were that Kannavdhawan (2020) used a set of PCs to explain 85% of the variance rather than 100 PCs. In addition, the trees for XGBoost had 5, 6, 8, 10, 20, or 25 splitting decisions, and compared several different error metrics for tuning the model, eventually choosing the model parameters to produce the highest testing accuracy.

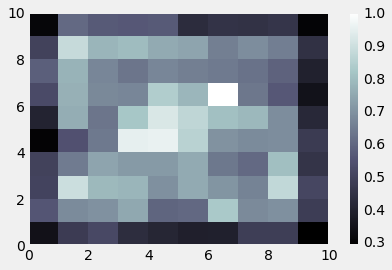
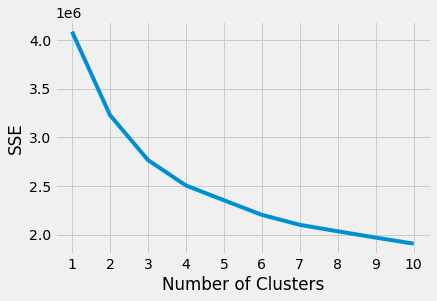
*Summary Statistics, Data Processing, and Unsupervised Learning*

There were a total of 6,000 observations within each of the 10 classes for the training dataset, and a total of 1,000 observations within each class for the testing dataset. The pixel values ranged from 0 (white) to 255 (black). The maximum, minimum, median, and mode values for each pixel column in the training and testing datasets are included in the project code, but were too long to include in this report. Most pixels included values across the range of 0 to 255. However, some pixels had a narrower range (e.g., pixel 1 had a range of 0 to 16). These descriptive statistics were re-calculated after this scaling to verify that the values were as expected, and scaled covariates were then used for all subsequent analyses.

| **Table 1. Number of observations by label.** | | | |
| --- | --- | --- | --- |
| **Training Data** | | **Testing Data** | |
| **Label** | **N Obs** | **Label** | **N Obs** |
| 0 | 6000 | 0 | 1000 |
| 1 | 6000 | 1 | 1000 |
| 2 | 6000 | 2 | 1000 |
| 3 | 6000 | 3 | 1000 |
| 4 | 6000 | 4 | 1000 |
| 5 | 6000 | 5 | 1000 |
| 6 | 6000 | 6 | 1000 |
| 7 | 6000 | 7 | 1000 |
| 8 | 6000 | 8 | 1000 |
| 9 | 6000 | 9 | 1000 |

The first unsupervised learning method was K-means clustering, which utilizes the distance between observations (determined based on the Euclidean distance between covariate vectors) to group observations with similar covariate values into a total K clusters. The optimal K value is determined by prior knowledge of the data, or by using the sum of squared error (SSE). The SSE method is generally more objective, and was utilized in this case. The optimal K is determined by the ‘elbow’ in the plot in Figure 1 below. This is the point at which increasing the number of clusters has a smaller effect on reducing the SSE. In this case, the optimal K was 4.

After the number of clusters were determined, the cluster assignment for each point in the training dataset was determined, and the known class assignments were used to determine the dominating class label for each cluster. For clusters 0, 1, 2, and 3 the dominating labels were 9, 1, 5, and 4 which correspond to images of ankle boots, trousers, sandals, and coats. Considering that there are 10 actual labels assigned to these points, the K-means clustering was not effective at separating these observations into their classes.



**Figure 1.** K-means SSE by number of clusters plot. **Figure 2.** Self-organizing Map average node values.

The second unsupervised method was a self-organizing map (SOM), where a ‘net’ of a given number of points (neurons) is adjusted based on the data, one observation at a time. When an observation is added, the closest neuron in the net moves towards that observation by a small amount, with other neurons moving closer by decreasing amounts as the distance from the observation increases. Then, the next observation is added and the net moves again. When all of the data has been included, the neurons will be located in the centers of clusters with similar covariate values.

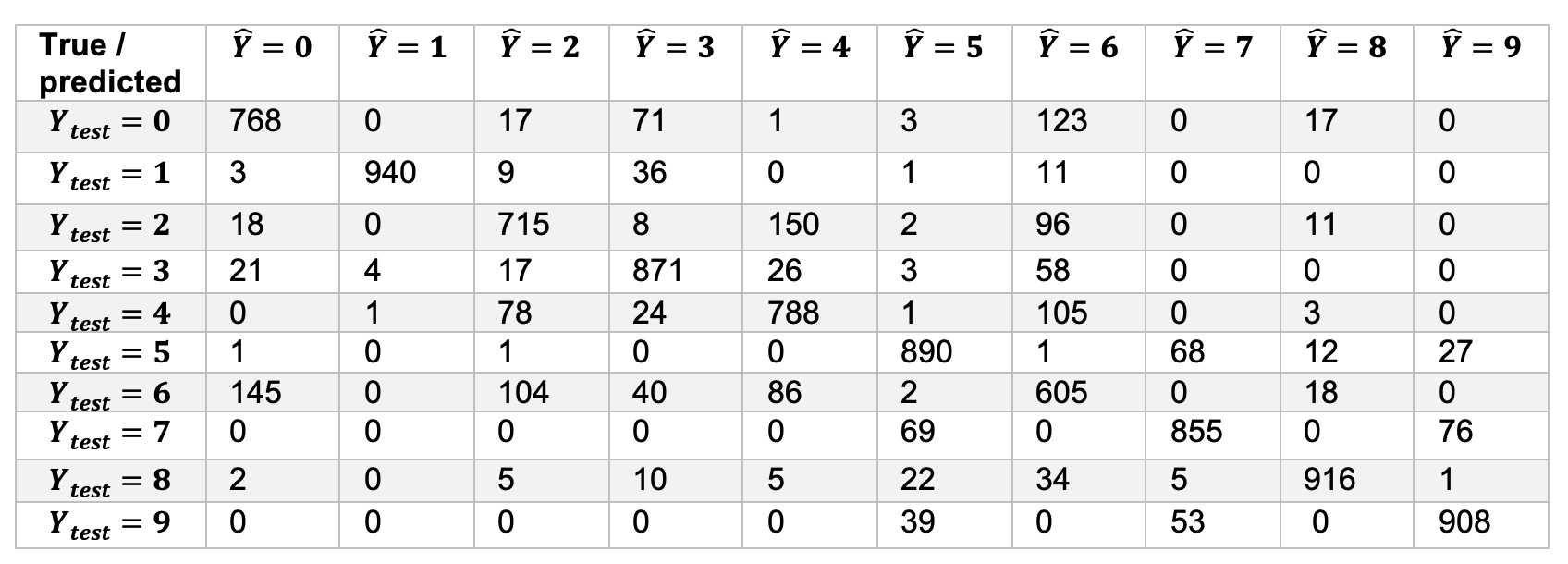
There is no set number of neurons, but general recommendations range from a maximum of 5\*sqrt(number of observations in the data) to a minimum of 10\*the number of expected classes. Given that this dataset had a large number of observations, with only 10 expected classes, a 10x10 net was used to result in a usable number of clusters. The color plot shown in Figure 2 depicts the average value of each node (based on the points assigned to that neuron), with white representing a node where the average covariate value is close to 1, and black representing an average value close to 0.3. This demonstrates that the SOM is able to differentiate points by pixel values, with some nodes having very different average values compared to others.

The number of points in each node was also determined, as well as the most common class label of the points assigned to each node (Appendix Table 1). The number of points within a node ranged from 1 (with a label of 6) to 3,000 (also with a dominating label of 6). This method was able to better separate the classes than K-means, with all of the classes being represented among the final nodes of the self-organizing map. This was accomplished with 100 nodes, which is well above the number of actual class labels, and makes interpretation difficult. However, this does suggest that some more advanced methods may be able to determine the class labels based on the covariate values (there are differences between classes according to the covariates).

*Multi-Class Classification Models*

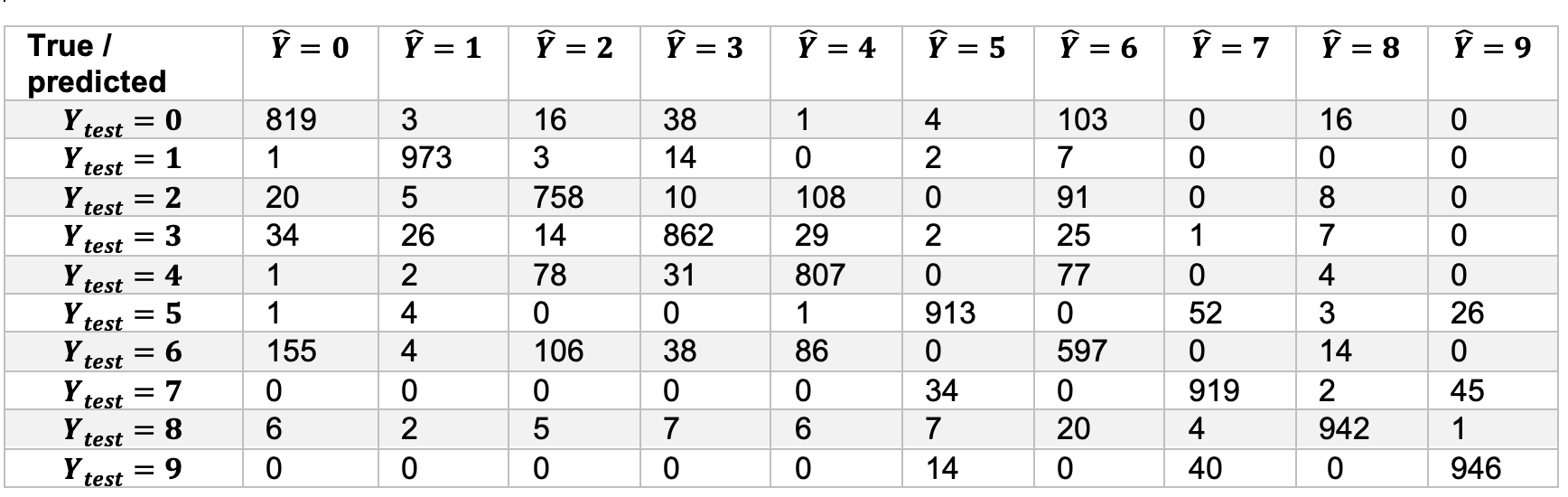
In the multi-class classification section, 3 different supervised learning algorithms were implemented: LR, LDA, and RF. All 3 of these models were trained using the 0-1 normalized training dataset and were validated on the 0-1 normalized testing dataset. All 3 methods were implemented using the existing packages: “sklearn.ensemble”, “sklearn.discriminant\_analysis”, and “sklearn.linear\_model” as the implementation of these models from scratch is very time consuming and computationally expensive. To see what predicted label class contributed to the misclassification rate, a confusion matrix was calculated for each model. The resulting accuracy and misclassification rate are summarized in Table 5 below.

The motivation for implementing LDA was due to the failed attempt to run the QDA analysis. During the attempt to perform QDA, it was discovered that there is collinearity between pixels (covariates), which caused a low accuracy of 56.4%, because the covariance matrix for QDA is not invertible. In addition, QDA with the uncorrelated principal components was tested, since this would address the collinearity issue, however, this method still resulted in a lower testing accuracy than the LDA with the original covariates. The accuracy of LDA was 82.56%, with a misclassification rate of 17.44%. According to the confusion matrix below in Table 2, the primary misclassifications were that 123 observations from class 0 were predicted as class 6, 150 observations from class 2 were predicted as class 4, 96 observations from class 2 were predicted as class 6, 105 observations from class 4 were predicted as class 6, 145 observations from class 6 were predicted as class 0, and 104 observations from class 6 were predicted as class 2.



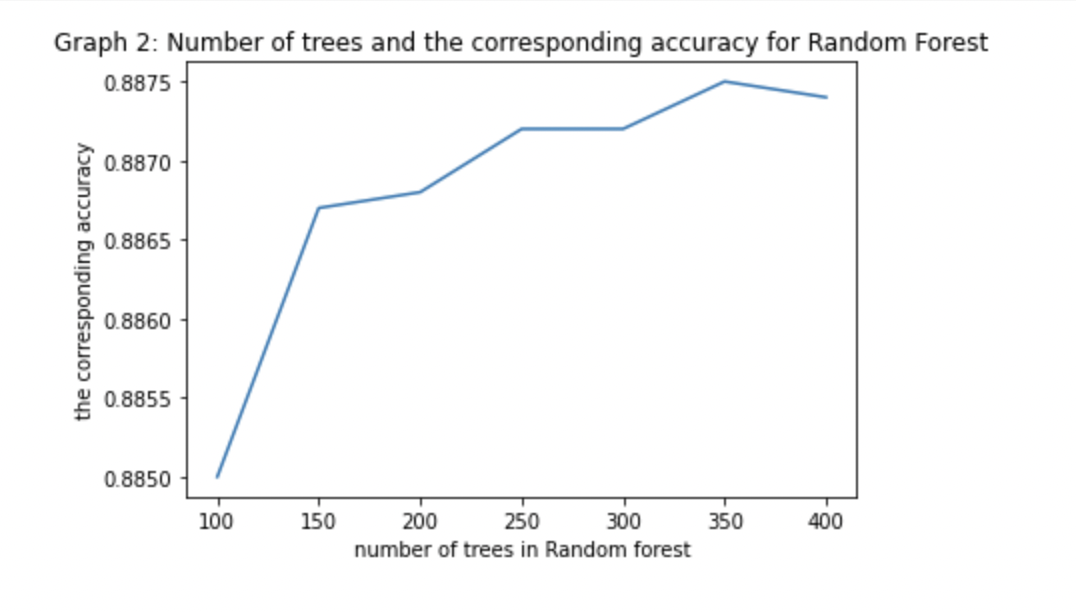
**Table 2.** Confusion matrix for LDA.

As the covariates (pixels) didn’t come from a normal distribution, we decided to perform Logistic Regression, as it should perform better than LDA, which is based on the assumption that covariates are normally distributed. The LR model was tuned by setting multi\_class= ‘multinomial’, as we have more than 2 classes; setting solver= ‘sag’ as the train and test datasets are very large and ‘sag’ solver helps to optimize the loss function faster; setting random\_state=542 to shuffle the data; and setting max\_iter=400 because when we ran the logistic model with 100 iterations it didn’t converge, so we increased it to 400 for the model to converge. The accuracy of LR was 85.36%, with a misclassification rate of 14.64%. According to the confusion matrix in Table 3 below, the primary causes of misclassification were that 103 observations from class 0 were predicted as class 6, 108 observations from class 2 were predicted as class 4, 91 observations from class 2 were predicted as class 6, 155 observations from class 6 were predicted as class 0, and 106 observations from class 6 were predicted as class 2.



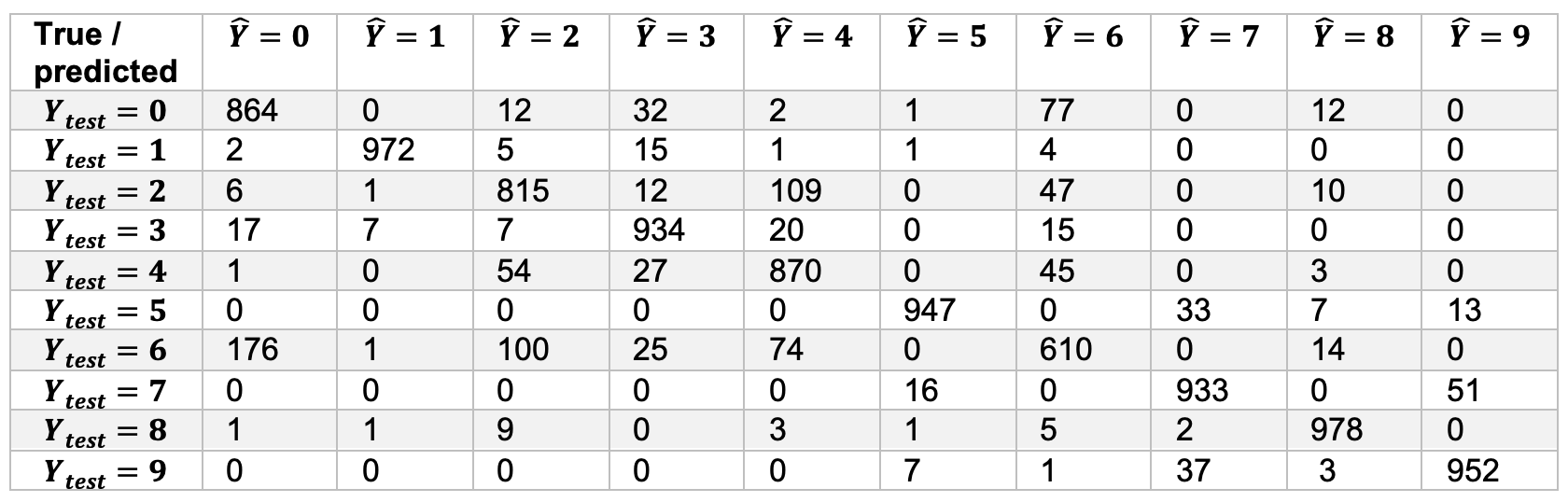
**Table 3.** Confusion matrix for Logistic Regression.

For the next analysis, we decided to extend a Random Forest model for a multi-class problem. RF is an ensemble learning method that constructs decision trees with a random selection of a subset of the covariates. For classification problems, the output of the RF is the class selected by the majority of the trees based on the covariate values. The RF model was tuned by setting criterion= ‘entropy’, as we use it for classification; setting max\_depth=100; and setting random\_state=542 to shuffle the data. To choose the number of trees in the random forest, we trained the model with 7 different numbers of trees in the range from [100,400], validated on the testing dataset, and calculated the accuracy for each model.



**Figure 3.** Number of trees and the corresponding accuracy for Random Forest.

According to Figure 3, the model with 350 trees maximized the test accuracy. Therefore, for the final RF model, we set n\_estimators=350. The accuracy of the resulting model was 88.75%, with a misclassification rate of 11.25%. According to the confusion matrix in Table 4 below, the primary misclassifications were 77 observations from class 0 were predicted as class 6, 109 observations from class 2 were predicted as class 4, 176 observations from class 6 were predicted as class 0, 100 observations from class 6 were predicted as class 2, and 74 observations from class 6 were predicted as class 4.



**Table 4.** Confusion matrix for Random Forest.

According to Table 5, we can see that RF had the highest accuracy (88.75%) because it is an ensemble method. LDA had the lowest accuracy (82.56%) compared to the other 2 models, since the pixels (covariates) are not normally distributed.

| **Model** | **Accuracy** | **Classification Error** |
| --- | --- | --- |
| Random Forest | 88.75% | 11.25% |
| LDA | 82.56% | 17.44% |
| Logistic Regression | 85.36% | 14.64% |

**Table 5.** Accuracies and classification errors for supervised learning models.

*Ensemble Model*

Ensemble modeling is a type of machine learning technique that combines multiple models into a joint model to improve the prediction of an outcome. The idea is that ensemble learning will create a stronger model by using an “ensemble” or group of “weaker” machine learning models. A single machine learning model may not be sufficient to provide accurate predictions on a dataset. Each algorithm can have limitations such as high variance, low accuracy, or feature noise and bias. In order to improve model accuracy, it may be beneficial to create and combine multiple models to reduce overall model error while upholding its generalization. The base models we selected to build our ensemble model are the following:

* Random forest
* k-Nearest Neighbors (kNN)
* Gradient boosting machine (GBM)

The random forest method was chosen as a base model because it performed the best out of all the multi-class classification models in the previous section in terms of having the highest accuracy and lowest classification error. The accuracy of the random forest model was 88.75%.

kNN is a supervised machine learning algorithm that can be used for regression and classification. kNN calculates the distance between each of the testing data points to all of the training data points. The most popular distance is the Euclidean distance, which is what we implemented in our kNN algorithm. Then, the algorithm selects the k number of points that lie closest to the test data point and, for classification, the algorithm calculates the probabilities that the test data point belongs to each of the classes of ‘k’ training data. The class that has the highest probability is selected as the prediction for that specific test data point.

To tune the kNN model, we need to find the optimal k value. The optimal k is determined by fitting multiple kNN models on the training data for k values ranging from 1 through 10. Then, predictions are obtained for each kNN model using the testing data. The kNN model that achieves the highest testing accuracy is the one which has the optimal k. Based on Table 6 below, the optimal k is 4, with an accuracy of the kNN model of 86.18%.

| **k** | **Accuracy** |
| --- | --- |
| 1 | 85.67% |
| 2 | 85.30% |
| 3 | 85.84% |
| 4 | 86.18% |
| 5 | 85.89% |
| 6 | 86.06% |
| 7 | 85.58% |
| 8 | 85.71% |
| 9 | 85.61% |
| 10 | 85.73% |

**Table 6.** kNN model accuracies for k values ranging from 1-10.

GBMs are ensemble machine learning algorithms that can be used for regression and classification. Gradient boosting is an optimization problem with the goal of minimizing the model’s loss by adding one weak learner at a time using a gradient descent algorithm. For classification, a logarithmic loss function, also known as deviance, was used to create the GBM. The weak learners in gradient boosting are decision trees. The decision trees are built such that their split values are determined based on minimizing the loss. The gradient descent algorithm works by first calculating the loss of a decision tree. Then, another tree is added to the model to reduce the loss, which is achieved by altering the parameters of the additional tree. The output of the new tree is then added to the output of the existing trees to improve the final output of the model. This algorithm continues until adding additional trees no longer improves the loss, or a fixed maximum number of trees have been added.

The GBM parameters that were tuned were the number of boosting rounds and the maximum tree depth for weak learners. Tuning the GBM can prevent overfitting. The parameters can be tuned by using a method called “early stopping”. This method works by optimizing the model’s performance on the test data and then stopping the training method once the performance on the test data stops improving after a certain number of iterations. Xiao, Kashif, and Roland (2017) determined that the optimal number of boosting rounds was 100 and the maximum tree depth was 10. Predictions were obtained for the GBM using the testing data, and the accuracy was 90.17%

The predictions from the 3 models mentioned above were used as inputs in the weighted average ensemble model. In the weighted average ensemble model, some input models are assumed to have better accuracy than others, and therefore, more weight is given to these input models when making the final predictions. The weighted average ensemble model is similar to the voting ensemble model except in the voting ensemble model, the predictions are averaged with equal weights. For our weighted average ensemble model, “soft” voting was implemented, which occurs when the model predicts the class label based on the argmax of the sums of the predicted probabilities. The weights for each input model were the model accuracies on the testing data. Predictions were obtained for the weighted average ensemble model using the testing data, and the final accuracy was 89.74%.

A summary of each model’s accuracy and classification error is provided in Table 7 below. The model that produced the lowest accuracy was the kNN model. The model that produced the highest accuracy was the GBM followed closely behind by the ensemble weighted average model. Therefore, out of all the models ran in this analysis, the GBM performed the best in terms of being able to predict the class labels in the Fashion-MNIST dataset.

| **Model** | **Accuracy** | **Classification Error** |
| --- | --- | --- |
| Random Forest | 88.75% | 11.25% |
| GBM | 90.17% | 9.83% |
| kNN | 86.18% | 13.82% |
| Ensemble Weighted Average | 89.74% | 10.26% |

**Table 7.** Summary of model accuracies and classification errors.

*References*

Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." *arXiv preprint arXiv:1708.07747* (2017).

Bhadra, Anushka. “Fashion-mnist-SVM-90.6% accuracy.” https://www.kaggle.com/code/anushkabhadra/fashion-mnist-svm-90-6-accuracy/notebook. (2021).

Deshpande, Rutvik. “Fashion MNIST / CNN Beginner (98%).” https://www.kaggle.com/code/rutvikdeshpande/fashion-mnist-cnn-beginner-98/notebook. (2021).

Kannavdhawan. “Comparing multiple linear and non- linear classifiers on Fashion MNSIT with 5 labels.” https://medium.com/@kannavdhawan1/comparing-multiple-linear-and-non-linear-classifiers-on-fashion-mnsit-with-5-labels-dd4cac6f682f. (2020).

*Appendix Tables*

| **Appendix Table 1. Summary of Self-Organizing Map** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | N Obs | Dominating Label | Node | N Obs | Dominating Label | Node | N Obs | Dominating Label |
| 0 | 566 | 7 | 34 | 118 | 9 | 68 | 994 | 6 |
| 1 | 1403 | 8 | 35 | 133 | 5 | 69 | 471 | 3 |
| 2 | 318 | 2 | 36 | 966 | 2 | 70 | 717 | 0 |
| 3 | 1591 | 9 | 37 | 1556 | 3 | 71 | 2061 | 2 |
| 4 | 839 | 0 | 38 | 242 | 8 | 72 | 148 | 4 |
| 5 | 271 | 1 | 39 | 413 | 3 | 73 | 3000 | 6 |
| 6 | 1258 | 5 | 40 | 346 | 6 | 74 | 58 | 8 |
| 7 | 324 | 7 | 41 | 671 | 3 | 75 | 1998 | 1 |
| 8 | 793 | 1 | 42 | 1 | 6 | 76 | 296 | 2 |
| 9 | 125 | 3 | 43 | 144 | 0 | 77 | 148 | 1 |
| 10 | 707 | 9 | 44 | 156 | 3 | 78 | 47 | 7 |
| 11 | 1067 | 5 | 45 | 504 | 3 | 79 | 157 | 4 |
| 12 | 160 | 2 | 46 | 942 | 8 | 80 | 103 | 8 |
| 13 | 1237 | 8 | 47 | 4 | 7 | 81 | 93 | 7 |
| 14 | 1510 | 8 | 48 | 298 | 8 | 82 | 107 | 0 |
| 15 | 390 | 0 | 49 | 215 | 6 | 83 | 513 | 0 |
| 16 | 263 | 0 | 50 | 104 | 4 | 84 | 963 | 1 |
| 17 | 2274 | 5 | 51 | 205 | 1 | 85 | 1966 | 0 |
| 18 | 289 | 5 | 52 | 8 | 1 | 86 | 3 | 8 |
| 19 | 303 | 4 | 53 | 279 | 1 | 87 | 298 | 4 |
| 20 | 123 | 7 | 54 | 530 | 0 | 88 | 289 | 1 |
| 21 | 166 | 5 | 55 | 162 | 8 | 89 | 283 | 2 |
| 22 | 1123 | 9 | 56 | 4 | 3 | 90 | 279 | 5 |
| 23 | 834 | 0 | 57 | 591 | 4 | 91 | 774 | 3 |
| 24 | 579 | 1 | 58 | 8 | 0 | 92 | 100 | 9 |
| 25 | 1299 | 7 | 59 | 375 | 0 | 93 | 116 | 2 |
| 26 | 1406 | 0 | 60 | 640 | 0 | 94 | 2977 | 3 |
| 27 | 723 | 7 | 61 | 2305 | 3 | 95 | 11 | 3 |
| 28 | 136 | 1 | 62 | 205 | 4 | 96 | 1221 | 3 |
| 29 | 110 | 2 | 63 | 1218 | 8 | 97 | 88 | 2 |
| 30 | 36 | 8 | 64 | 1153 | 1 | 98 | 406 | 5 |
| 31 | 465 | 5 | 65 | 564 | 3 | 99 | 65 | 0 |
| 32 | 5 | 0 | 66 | 123 | 8 |  |  |  |
| 33 | 118 | 8 | 67 | 1256 | 5 |  |  |  |