1. **Exploratory analysis**

The dataset that we decided to use is the users' measurements, clothing fits, reviews, etc. from *RentTheRunway*. This dataset consists of 105508 users, 5850 items, 192544 number of transactions. Among transactions, there are 192462 of the transactions such that they have no null value. Under this circumstance, the dataset is definitely large enough for us to perform predictive analysis with methods from class or methods that suit better for our task. The dataset contains the following properties of a user and the user's comments with respect to a certain clothing item that they interacted with:

Table 1.

Name	**Description**
fit	a clothing item's fit for a user (small, fit, large)
user_id	unique identifier of a certain user
bust size	measurement around the chest over the fullest part of the breasts of a user
item_id	unique identifier of a certain item
weight	weight of the user
rating	user's preference score on a particular item
rented for	reason the user rented the clothing
review_text	an user's comments about a certain item
review_summary	short summary of the comments made by an user
category	clothing type of an item
height	height of the user
size	size of the clothing item
age	age of the user
review_date	the date that the review is received

After looking through the structure of the dataset, we infer that the dataset contains three important and crucial features for doing a predictive task: weight, height, and size. We normalize the weight, height, and size columns in the dataset to gain more insights about the features.

Figure 1.

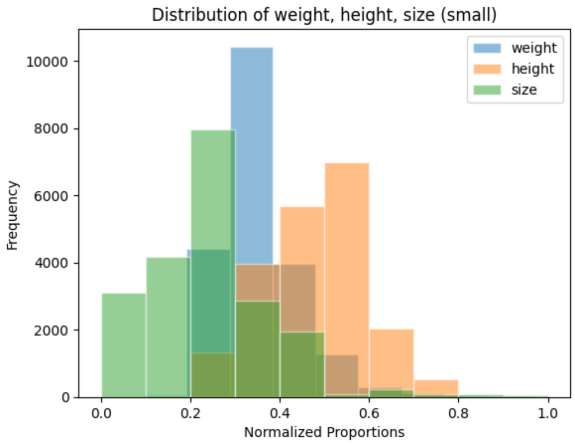


Figure 2.

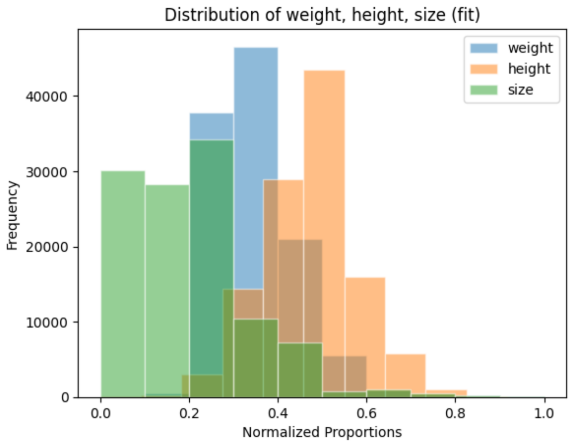
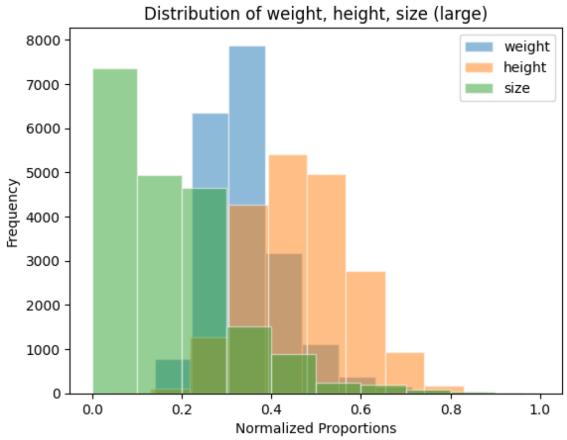


Figure 3.



By plotting the above features, we observe some interesting findings that we did not expect at the beginning. For clothing items in the "large" class (i.e. the size of an item is too large for the user), the distribution of this "size" data is right-skewed. Users in this class tend to buy clothing size that is smaller in general compared to the other two classes ("small" and "fit"), as the other two distributions show a less right-skewed and approximately normal distribution. This is interesting as we initially expect the size distribution in the "large" class to be left-skewed because the items in this class would typically have a larger size. In addition, the shape and spread of the distributions of the "weight" and "height" feature data looks fairly similar.

2. Predictive Task

As we dived deep into the dataset, we thought that the fit categories would be interesting to perform a categorical predicting task. We then checked through the entire dataset and found that "fit" only has three categories: {small, fit, large}. Intuitively, as fit must associate the clothing sizes with the customers' body measurements, we believed that whether an clothing item fits a customer or not is greatly determined by the three essential factors: the item's size, the customer's weight, and the customer's height. As a result, we picked 'weight', 'height' and 'size' from each user to form the feature matrix and feed it to our baseline model. However, some data does not contain the properties we need. We must perform modifications to our dataset before using the dataset to generate features we need.

At first, we make a list of properties we need, and then we iterate through the dataset to grab all the data that contains all the properties in the list. For example, if the list elements are ["weight", "height", "size"], then we only take the data that does not have invalid entry values from the query keys in the list above. We name this filtered dataset new_dataset. During the process of dropping out invalid data, we also noticed that the values of height, weight and size are not integers yet. We have to do some more data cleaning work on the new_dataset so that we can generate our desired feature matrix. In the new_dataset, we need to extract the integer values of the weight of each data from the form of a string to an int. We do so by removing the unit and using the integer conversion function. Next, we convert the height of each data from feet to inches. After that, we convert the size of the clothing items from the form of string to int as well. After all the data cleaning, we get a clean dataset with a size of 153,441, which is still large enough to perform the baseline prediction.

We will build our baseline predictor in the form:
$f(weight,\ height,\ size) \rightarrow fitting$
using baseline features [1, d['weight'], d['height'], d['size']] from each data. Our predicted output given a single feature will be a value within the set {"small", "fit", "large"} of each data. Since there are three categories that the model needs to predict, we need to perform multiclass classification.

Before building our baseline model, we calculated the percentage of each class of {small, fit, large} in the new_dataset.

Table 2.

Class	Proportion
small	0.134019
fit	0.736035
large	0.129946

Table 2 shows that class "fit" accounts for approximately 73.6% of the data, class "small" accounts for

approximately 13.4% of the data, and class "large" accounts for approximately 13% of the data. The new dataset is large and contains imbalanced data. We decided to build two different models, one using an imbalanced dataset and one using a modified dataset that is balanced using some kinds of balancing methods. Given these two kinds of dataset we are using, we decided to use two different performance metrics to evaluate our models.

For the imbalance dataset, we are going to use the F1 score metric to evaluate. The F1 score is the harmonic mean of precision and recall, where a F1 score of 1 is the best and a F1 score of 0 is the worst in our predictive model. We choose the weighted average F1 score that is to take the mean of each class's F1 score with consideration of each class's support.

Let *True Positive* be *TP*. Let *False Positive* be *FP*. Let *True Negative* be *TN*. Let *False Negative* be *FN*. Then, the formula for calculating F1 score for one class is:

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

$$Recall = \frac{TP}{TP+FN}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

The formula for calculating weighted average F1 score for n classes is:

$$F1 = \frac{1}{n}\sum_{i=1}^{n}(F1_i * number\ of\ class\ i)$$

To generate a balanced dataset, we will use the undersampling technique to balance our dataset. Undersampling is the process of balancing an unevenly distributed dataset through keeping all of the data in the minority class and setting a threshold to limit the size of the majority class. We achieve this by taking the average of the sizes of two minority classes, "small" and "large", which turns out to be 20251, and set this average value as the new size for the "fit" class. We then generate a new dataset that contains all the data in the "small" and "large" classes and 20251 data of the "fit" class by randomly sampling the original "fit" class. This new dataset is of size 60754. The proportion of each class in the modified dataset is shown in Table 3.

Table 3.

Class	Proportion
small	0.338480
fit	0.333328
large	0.328192

The dataset is now balanced, so we are going to use accuracy for evaluation, which has the formula:

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

3. Model

A classification model is essential in our predictive task compared to a regression model because classification models work better on predicting classes, while regression models commonly work for predicting numerical values. Given that our predictions are different classes, we need to choose a classifier. Most commonly used classifiers are Logistic Regression, Decision Tree, Stochastic Gradient Descent (SGD), LinearSVC for a large dataset. We are going to fit the baseline features into each of the four models to determine one model for our predictive task.

First, we use the train_test_split function from Sklearn library to split the new_dataset into training, testing and validation sets with a ratio of 0.7:0.15:0.15. As we mentioned in the previous section, the baseline feature is [1, d['weight'], d['height'], d['size']] from each data. Therefore, for our baseline model, we transformed all three sub-datasets into baseline feature representations. Then, we fitted all four classifiers with the training data using default hyperparameters. Also, we calculated the validation and testing weighted average F1 scores of all four models. The data is shown in table 4.

Table 4.

classifier	Validation weighted F1 score	Testing weighted F1 score
Logistic Regression	0.621762	0.629634

Decision Tree	**0.627872**	**0.637713**
SGD	0.621859	0.629015
LinearSVC	0.621158	0.628730

The result demonstrates that the *DecisionTreeClassifier* has the highest weighted F1 score on both validation and testing datasets. Thus, we will use *DecisionTreeClassifier* as our classifier with consideration of different hyperparameters in our predictive task. This default decision tree classifier model gave us a weighted validation F1 score of 0.627872 and a weighted testing F1 score of 0.637713.

We tried to test the model with different function parameters of *DecisionTreeClassifier* to see if there is any improvement on the weighted F1 score. The parameter we are changing is "max_depth", which specifies the maximum depth of each tree. Since the default value for "max_depth" is set to None, the decision tree will stop splitting until all data points in a leaf come from the same class for all leaves. This will memorize the noise of the training data, causing over-fitting and failure to fit unseen data. We are changing the maximum depth of the tree from 2 to 25 to find the model with the highest weighted F1 score as our baseline model. Every time we fit the model with the training data using a different number of "max_depth", we also use the model to predict the validation data and training data. At the same time, we record the weighted F1 score of these predictions. Validation weighted F1 score is shown in table 5.

Table 5.

max depth	weighted F1 score
2	0.621185
3	0.621185
4	0.621185
5	0.621185
6	0.621180
7	0.621230
8	0.623594
9	0.622699
10	0.623586
11	0.624480
12	0.624822
13	0.625513
14	0.625434
15	0.626430
16	0.627312
17	0.627537
18	0.628268
19	0.627984
20	0.627728
21	0.627965
22	0.628039
23	0.627817
24	0.627709

This pipeline gave us 18 as the best value for "max_depth" in this case. The associated weighted F1 score is 0.628268. Using the fitted *DecisionTreeClassifier* model with 18 as the "max_depth", we predicted from the testing data and calculated the weighted F1 score, which is 0.636956. This is even less than the testing weighted F1 score given by the default *DecisionTreeClassifier* model. We need to modify the feature representations of data to improve the model.

In order to optimize the baseline model, we tried to add more features. From all other properties, we think that customers may give feedback on the fitting of the clothes

in “review_text” and “review_summary”. Thus, we use the Bag of Words method to tokenize “review_text” and “review_summary” of training data. Before transforming the text into a vector, we remove all punctuation, convert uppercase letters to lowercase, convert words to words stem and remove all stop words for each text in “review_text” and “review_summary”. After these transformations, we count each word’s frequency and get the most frequent 4000 words as our bag-of-words.

The new feature will be the baseline features appended with the tokenized vector using bag-of-words. We will use the same process as above to find the parameter that produces the highest validation weighted F1 score. All the validation weighted F1 scores are shown in table 6. From the table, we can see the parameter that produces the highest validation weighted F1 score is 13 with a weighted F1 score of 0.766067. This improved model gave us a testing weighted F1 score of 0.770700, which is higher than what the baseline model produced. Thus, we think this new feature representation can help us predict the fitting of products.

Table 6.

max depth	weighted F1 score
2	0.711064
3	0.730774
4	0.720891
5	0.738238
6	0.759126
7	0.758221
8	0.757526
9	0.757022
10	0.761161
11	0.763741
12	0.764466
13	0.766067
14	0.764611
15	0.763968
16	0.762468
17	0.762473
18	0.760951
19	0.759077
20	0.757908
21	0.756730
22	0.757447
23	0.756998
24	0.753500

Because of the imbalanced classes, we also decide to obtain a balanced dataset from the dataset we evaluate with F1 score, as mentioned in the end of section 2, Predictive Task. Using the same train_test_split technique, we obtain new training, testing and validation sets with a ratio of 0.7:0.15:0.15. Then, we use the same features and the same classifier as the last task, the only difference is that we use accuracy to evaluate this new model because the dataset is now balanced. The accuracy for the new validation set at each max tree depth is shown in the following table:

Table 7.

max depth	accuracy
2	0.508065
3	0.550532
4	0.594754
5	0.597717
6	0.627784

7	0.638648
8	0.652584
9	0.650499
10	0.654230
11	0.651706
12	0.650170
13	0.653023
14	0.651267
15	0.644793
16	0.643147
17	0.638318
18	0.640074
19	0.635356
20	0.631186
21	0.630088
22	0.622297
23	0.622517
24	0.615604

4. Literature

We first obtained our dataset from the list of datasets that the Professor frequently uses. In the paper *Decomposing fit semantics for product size recommendation in metric spaces,* each transaction with fit feedback is embedded by its related customer and clothing item, which constructs the latent representations. To address the class imbalance problem, metric learning with prototyping technique is used to re-sample data from different classes to change the data distribution.After that, they use LMNN to bring transactions with the same fit feedback closer in the vector space. This effectively improves the performance in the final KNN classifier, which relies on the distance between the predicted data point and the class that the data point is most close to.

The second paper we looked into is *A Deep Learning System for Predicting Size and Fit in Fashion E-Commerce,* which also used the dataset RentTheRunWay. Besides discussing the common issue - imbalance class - in this dataset, this paper further reveals the difficulties in predicting the fit feedback. In real practice, the size of a clothing item can have a significant effect on the fit feedback by the customer. However, sizes of clothing items differ from regions to regions, brands to brands, or even items to items from the same brand. There is no agreement that can consistently regulate the sizes, and therefore makes it even more difficult to predict the fit feedback given a query. In this work, it uses a deep learning based content-collaborative methodology for fir prediction. Its method uses both the interaction data and customer and article features, whereas, normally, the other approaches only use customer and article features. The neural network(SFNet) is inspired by the Siamese Networks, the difference is that instead of having two identical networks processing the inputs, the SFNet processes customer latent embeddings in one neural network and article latent embeddings in another neural network “in parallel” and joins the outputs from the two distinct networks with more non-linear layers to feed the learned representations to the last softmax layer for the final classification. Our conclusion is quite similar: without prior knowledge about the fit category, we can train a model with embedded features about a user, its related review, and the customer-clothing item interaction to predict whether a clothing item fits with a decent performance.

5. Results and Conclusions

Using different feature representations from the original imbalance dataset to fit our models, we generate two lists of validation and training weighted F1 scores for each model. The specific values are shown in table 5 and table 6. The plot for all weighted F1 scores of validation and training data and the testing F1 score generated by the “best model” is shown in figure 4 for baseline model and figure 6 for improved model.

However, when we try to use a confusion matrix to evaluate the performance of the baseline model using the first strategy, the majority of the predictions are predicted as “fit”. As seen in figure 5, “fit“ class is nearly perfect, however, the model is extremely biased towards “fit” such that 97% of the “large” class and “small” are predicted as “fit”. The true positive labels are very poorly predicted for these two classes. The low performance in dealing with false positives and false negatives results in relatively low F1 scores (Figure 4).

Figure 4.

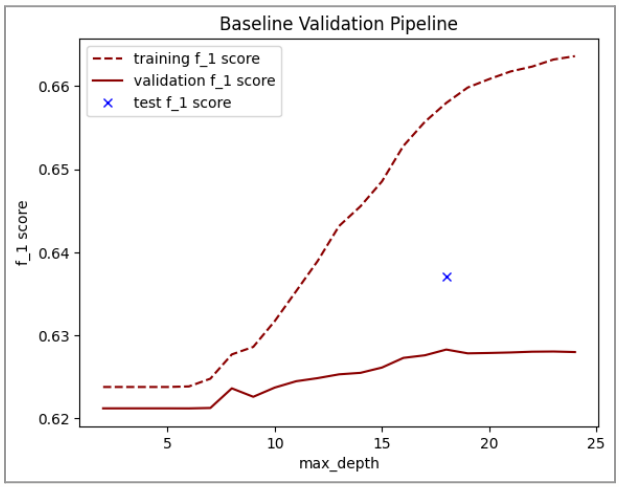
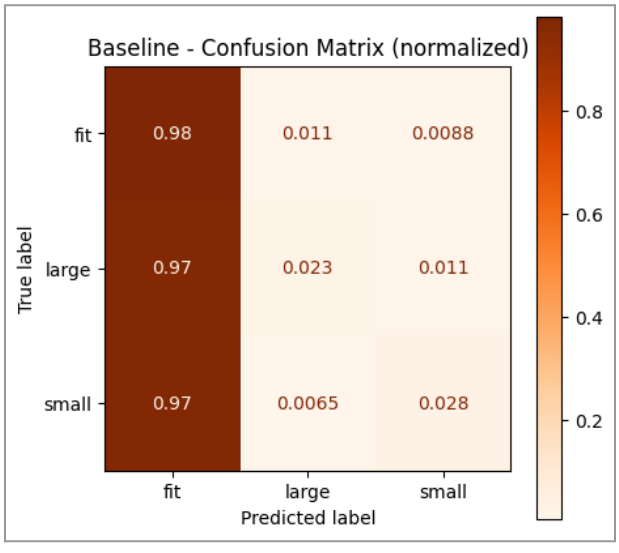


Figure 5.



Same issue happened in the improved model (model 2) as shown in figure 7. Adding the text features extracting from the “review_text” and “review_summary” of each data to the baseline feature did produce a higher weighted average F1 score and more true positive labels compared to the baseline model. We infer that the “review_text” and “review_summary” data do have some importance in predicting the fitting data. However, as we can see, there are still 61% of the “large” class and 61% of the “small” class being predicted as “fit”. This model 2 improved the baseline model but still needs to be optimized further.

Figure 6.

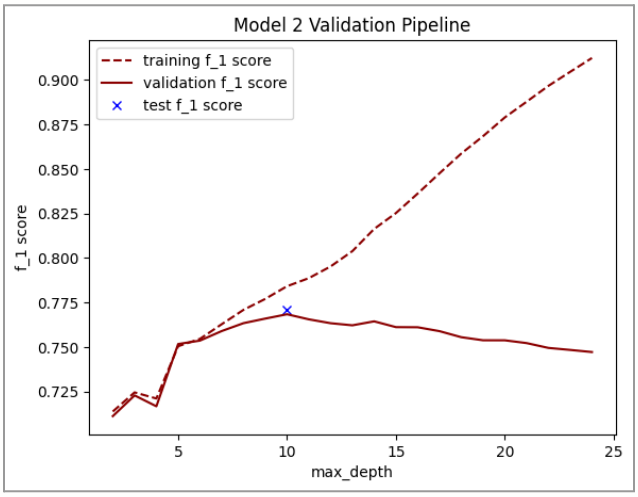


Figure 7.

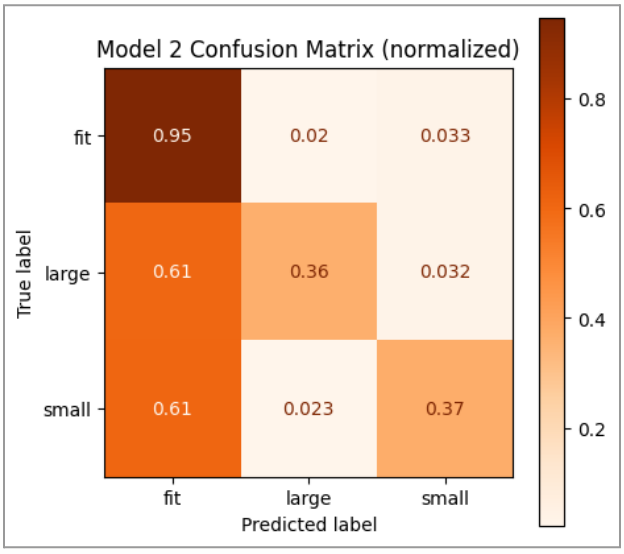


Figure 8 shows the accuracy score for training and validation sets for model 3 when training with different "max_depth". The general testing accuracy score is 0.651195, which is somewhat lower than what we expect. However, in figure 9, the majority of the "small" and "large" class, 58% and 69%, respectively, are predicted correctly in the corresponding class.

Figure 8.

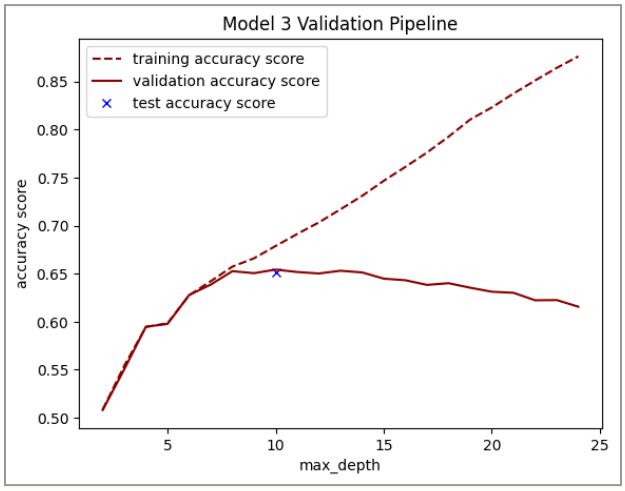
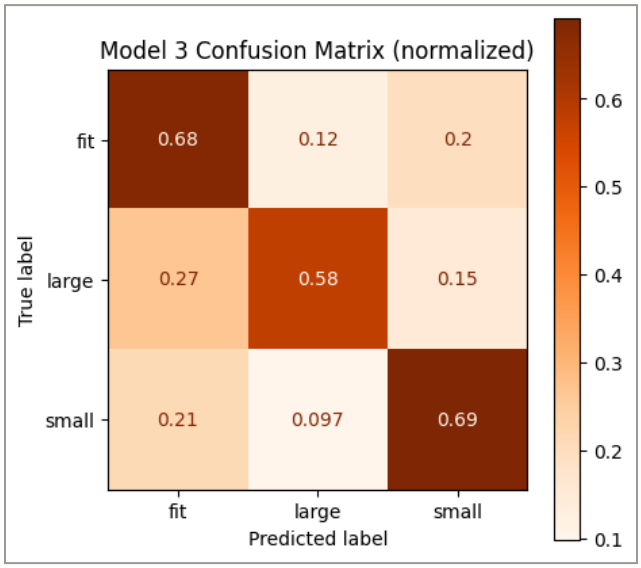


Figure 9.



We tuned one hyper-parameter of our model: max_depth. Intuitively, we expect the performance to be higher with higher max depths because the model can handle more complex "decisions". This idea is correct until max depth is over 10, the performance starts to decrease probably due to overfitting (Figure 8).

With strategy 2, although the accuracy is not as high as expected, we managed to improve the true positive predictions. This is also a significant improvement if we compare it to the previous models with imbalanced class, we do not want the model to be biased that it can only predict an input as "fit", when the input should be in fact "small" or "large".

Reference

Decomposing fit semantics for product size recommendation in metric spaces
Rishabh Misra, Mengting Wan, Julian McAuley
RecSys, 2018

A deep learning system for predicting size and fit in fashion e-commerce
Abdul-Saboor Sheikh, Romain Guigourès, Evgenii Koriagin, Yuen King Ho, Reza Shirvany, Roland Vollgraf, and Urs Bergmann
RecSys, 2019