Multi-Class Fit Prediction for RentTheRunway Dataset

Nidhi Dhamnani (A59012902) , Sri Harsha Pamidi (A59005361)
Srinivas Rao Daru (A59003596), Wonsuk Jang (A15427847)

1 DATASET

We are using the RentTheRunway clothing fit dataset (https://cseweb.ucsd.edu//~jmcauley/datasets.html#clothing_fit) for this assignment used in the paper [1]. The final size of our dataset is **146,380** observations after excluding the undefined entries. We used git for source code management and version control (https://github.com/nidhidhamnani/cse-258-project).

1.1 Data Statistics/Filtering

The original data has total entries of **192,544** observations. However, many entries had null values for different attributes, causing us to drop such entries. Also, after checking the distribution of entries for each attribute, we found very few number of entries for a particular category for some attributes, thus dropping such entries also. After filtering all such entries we got **146368** observations. **From Figure 1(a), 1(b)** we can see that most of the users reviewed around 10 items and each item has been reviewed around 200 times on average, making the data item-biased. In the next section, **section 1.2**, we talk about the distribution of different attributes and their relevance to the task.

1.2 Data Properties and Findings

- **User Id(Integer), Item Id (Integer):** These are the unique identities of item and users. In our case, we just consider the inputs independent of the user and the item, so we are not using these attributes.
- **Fit -Labels (Categorical):** In terms of the labels for fit, the data is highly biased, which can be seen in **Figure 1(c)**. Majority of the item fits (73.6% percent) and only a small fraction of items are either large (13% percent) or small (13.4% percent). This attribute is used as the label for each input attributes .
- **Category (Categorical):** This attribute has uninformative data. In **Figure 1(d)** we can see that one of the value of the category was just dress. Also, other values are subcategories of dress, making the data ambiguous. So this attribute was discarded for our task.
- **Weight (Integer), Height (Integer), Bust Size (String), Body Type (Categorical), Size (Integer):** The important information needed for predicting the fit would be about the measurements of different user. The listed attributes fall under this category. The distribution of these attributes are also close to Gaussian, as can be seen in **Figure 1(e)-1(f),2(a)**, making them an unbiased real world data.
- **Rating (Integer):** The rating data distribution is a little biased towards higher end, as can be seen in **Figure 2(b)**, but the rating captures the user satisfaction of the product in general and would give use some useful information about the user fit information.
- **Age (Integer):** The age attribute becomes useful in relating preferences of people in different age groups, even though having a similar measurements. The data distribution for this attribute can be observed as a approximated Gaussian as in **Figure 2(c)**, so an unbiased attribute.
- **Rented For:** This attribute has a unbiased distribution, which can be seen in **Figure 2(d)** except for one category, which has been dropped to make a better data as explained earlier. This attribute gives us the information about the reason a person is using the dress for and would give us their preferences of the fit for that purpose.
- **Review Text(String), Review Summary(String):** These two attributes gives us a general description of a product in

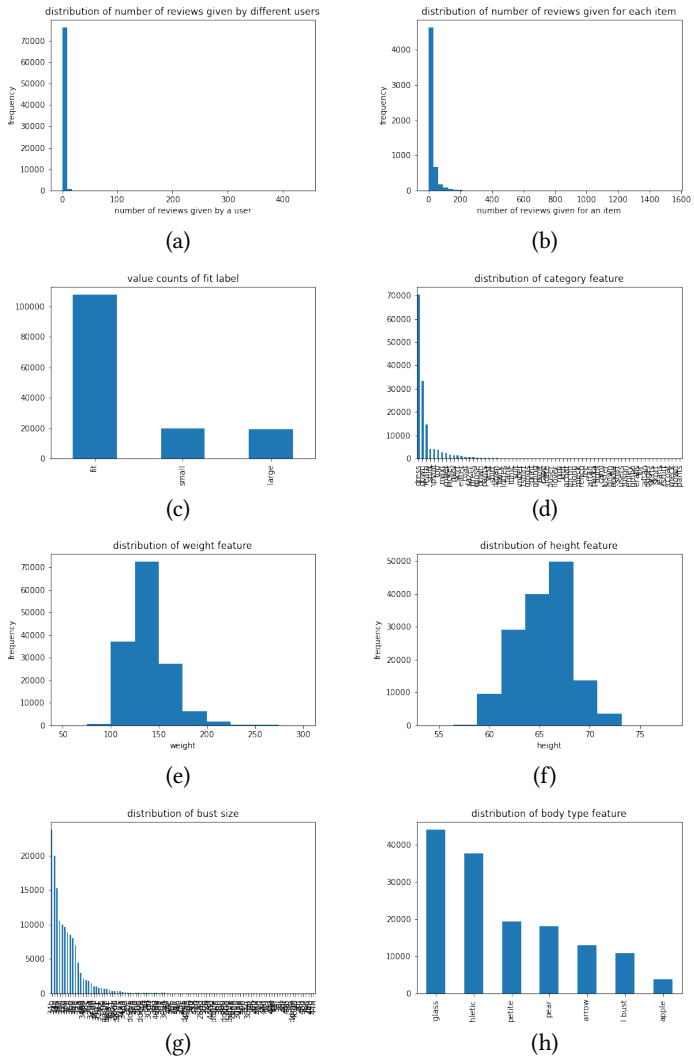

Figure 1: (a)-(h) Distributions of different attributes

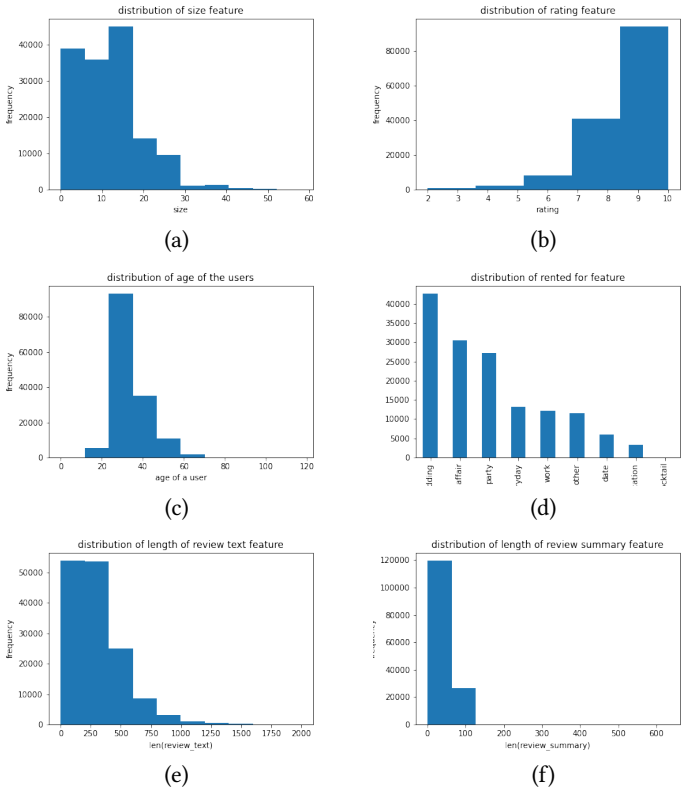

Figure 2: (a)-(h) Distributions of different attributes

addition to the measurement attributes of an item, which would further help in strengthening our decision based on only measurements. Most of the reviews were in length of 100, plotted in **Figure 2(e),2(f)**.

2 PREDICTIVE TASK

The goal of our project is to correctly predict whether an item will be a good fit or will be large or will be small for a customer. This task has been chosen because of the scope of its applications as a pre-task in recommending items to users and also in helping lower the returning rates of the products because of wrong fitting. We consider a general multi-class classification task independent of the user and item. In **section 3**, we discuss models which were trained on data with different attribution values and no specific information about the user or item ids. For this, the given attributes listed in **section 1.2** are used directly to build general model to predict. Later, in **section 6**, we explain how this model itself can be used in recommendation tasks and suggestion about how to use user, item information from existing data for new pairs.

2.1 Evaluation Metrics

We divided our data into two parts - train and test which consisted of 80% and 20% of observations respectively. We trained our model on train data and used test data to evaluate the model performance. We tuned the hyperparameters of our models by using 5-fold cross validation technique which involves dividing the train data into 5 parts and using 4 parts for training and 1 part for validation and repeating the process by having each part as the validation set once.

The evaluate the performance of our trained model, we used the following metrics:

- Accuracy = (TP+TN)/(TP+FP+FN+TN)
- Precision = TP/(TP+FP)
- Recall = TP/(TP+FN)
- F1 Score = 2*(Precision*Recall)/(Precision+Recall)

(TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative)

In our case, the data is unbalanced therefore we did not completely rely only on accuracy to measure the performance. We believe precision (ratio of correctly predicted positive observations to the total predicted positive observations), recall (ratio of correctly predicted positive observations to the all observations in actual class - yes), and F1 score are better evaluation metrics in such cases as we care more about the TP and FP for fit class. Therefore, we picked our best model by comparing the scores of precision, recall, and F1 score.

2.2 Baseline

As we saw in Section 1, the dataset is highly unbalanced and contains majority of the item prediction as 'fit', therefore for baseline we used the model which always predicts the item as 'fit'.

Table 1: Baseline Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.735	0.737
F1 Score	0.623	0.625
Precision	0.541	0.543
Recall	0.735	0.737

2.3 Features

From the given attributes **(Section-1.2)** we used the following attributes as features for our models.

Table 2: Features

Attribute Name	Final Type
Weight	Integer
Height	Integer
Bust Size	Converted to Integer
Rating	Float
Rented For	One-hot encoded
Body Type	One-hot encoded
Age	Integer
Length of Review Text	Integer
Length of Review Summary	Integer
Word embedding of Review Text	float vector
Word embedding of Review Summary	float vector

2.4 Data Preprocessing

We followed the below steps for data preprocessing:

- Dropped null values for data sanity.
- There was only one entry for 'rented for' type as 'party'. Therefore, to remove anomolies in the dataset, we removed the observation.
- Converted weight from string to int by removing the unit of weight (lbs).
- Converted height to cm instead of ft and inch.
- Converted the categorical columns ('rented for' and 'body type') to one-hot encoding.
- Converted the text fields ('review_text' and 'review_summary') to lowercase and removed punctuation. For some models, this text has been used to create a vector by word embedding.

After converting, all the entries to numerical values, we used min-max normalizer to normalize our data.

2.5 Justification for Models

Our problem statement involves correctly predicting the fit of an item. The fit can have 3 possible values, namely, 'fit', 'large', and 'small'. Therefore we used various multi-class classification algorithms to solve the problem. Below is the list of different models we tried and their scores. In some models, we tried ablation experiment to determine the bad features for that model and remove the worst feature to improve scores.

As our task is a multi-class classification problem, we have considered various categories of machine learning models available in sklearn package and others. Models like logistic regression, Naives Bayes, KNN gave similar scores which were bad in comparison because of their sensitiveness to data distributions and relations. Later, we tried some recent state-of the art models like XGBoost, Gradient Boosting and Stochastic Gradient Descent approaches which increased our results a lot. In addition, we tried out Deep Learning models along with word embeddings gave us the best results, given their robustness to data distributions.

3 MODELS

For our models, we have tried Multiclass Logistic Regression, K Nearest Neighbors, Naive Bayes, Stochastic Gradient Descent, Random Forest, Gradient Boosting, XGBoost, and a Deep Learning Model with pre-trained word embeddings. We discuss the pros and cons of each model as well and how we optimized each model when it was possible. After comparing the accuracy, F1 score, precision, and recall for each model, we have found that the Deep Learning Model with pre-trained word embeddings gave us the best results.

3.1 Multiclass Logistic Regression

Table 3: Logistic Regression Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.567	0.564
F1 Score	0.600	0.598
Precision	0.655	0.655
Recall	0.567	0.564

For the logistic regression model, we used sklearn's implementation of linear regression. In order to optimize our model, we used l2 regularization for our loss function. Additionally, we used lbfgs for our implementation to perform our logistic regression. For our loss function, we used cross-entropy loss. The pros of logistic regression for our task is that it is efficient to train. Consequently, our logistic regression model was able to increase precision. However, the cons of logistic regression for our task is that our task may be non-linear, leading to lower accuracy, F1-score, and recall for both train and test data.

3.2 K Nearest Neighbors

Table 4: K Nearest Neighbours Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.789	0.688
F1 Score	0.755	0.639
Precision	0.773	0.612
Recall	0.789	0.688

For our K Nearest Neighbors (KNN) model, we used sklearn's implementation of K Nearest Neighbors. For our model, we decided to have a default value of 5 nearest neighbors and leaf size of 30 as it gave us the best results. Furthermore, for our KNN algorithm, we went with the auto option, which determined the best methods among BallTree, KDTree, and brute-force search that gave us the best results. The pros of our KNN Model is that it is easy to implement and does not make any assumptions about the model. On the other hand, one negative of the KNN model is that it does not perform well with imbalanced data. Additionally, KNN does not have a loss function, not allowing for optimization and leading the model to be simple. Consequently, our model was able to beat the trivial model for the training data, but the performance decreased for the test data. Overall, our model was able to beat the trivial model's F1 score and recall score.

3.3 Naive Bayes

Table 5: Naive Bayes Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.735	0.737
F1 Score	0.623	0.625
Precision	0.541	0.543
Recall	0.735	0.737

For our Naive Bayes Model, we used sklearn's implementation of the Gaussian Naive Bayes Model. In order to optimize the Naive Bayes Model, our model uses the negative joint log-likelihood. Some pros of the Naive Bayes Model is that it is simple and efficient to train. On the other hand, the cons of Naive Bayes Model is that it assumes that all the features are independent, which we

found out not to be the case during our experiments. Furthermore, as our implementation of Naive Bayes Model assumes a normal distribution, the imbalance of data may have led to worse results. Consequently, our model was not able to beat our trivial model of predicting fit all the time and performed similar to it.

3.4 Stochastic Gradient Descent

Table 6: Stochastic Gradient Descent Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.726	0.729
F1 Score	0.660	0.665
Precision	0.642	0.650
Recall	0.726	0.729

For our Stochastic Gradient Descent (SGD) model, we used sklearn's implementation of the SGDClassifier. For our model, we used the default parameters as specified by sklearn. In order to optimize our model, we added a l2 regularization term and used stochastic gradient descent in order to minimize our loss function. The pros of SGD is that unlike gradient descent, it frequently updates the parameters for optimization. On the other hand, the cons of SGD is that the constant updating of the parameters can actually lead to noise and unstability. For our model, although it was not able to beat the trivial model's accuracy and recall, it was able to beat the F1 score and had a much higher precision than the trivial model.

3.5 Random Forest

Table 7: Random Forest Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.999	0.729
F1 Score	0.999	0.646
Precision	0.999	0.634
Recall	0.999	0.729

For our implementation of the Random Forest model, we used sklearn's implementation of the Random Forest. For our parameters, we used 100 trees and did not specify a max depth. In order to optimize our random forest, we used cross-validation implemented in the library. The pros of random forest is that it is non-parametric and generally works well with a large dataset due to its randomness. On the other hand, the cons of random forest is that it is also prone to overfitting and gives a low prediction accuracy compared to other machine learning algorithms. Consequently, although we have almost a perfect score for accuracy, F1 score, precision, and recall for the train data, our performance significantly decreases in the test data, leading us to conclude that it has overfitted the data. However, it was able to beat the F1 score and precision of the trivial model for the test data. In order to decrease the over fitting we tried reducing the tree depth which gave us bad test results, so we have have not changed them.

3.6 Gradient Boosting

Table 8: Gradient Boosting Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.738	0.738
F1 Score	0.643	0.642
Precision	0.673	0.663
Recall	0.738	0.738

For our implementation of Gradient Boosting model, we used sklearn's implementation of gradient boosting. For our number of estimators, we decided to use 100 as the gradient boosting algorithm is robust to over-fitting, giving us an option to include a higher number of estimators. Likewise, we used a depth of 3 in order to add some depth to our model and avoid overfitting. In order to optimize our model, we used the cross-entropy loss. Furthermore, for our learning rate, we used a learning rate of 0.1 as it gave us the best results. Additionally, gradient boosting computes multiple gradients in order to find the function that points towards the negative gradient and thus help optimize the model. The pros of gradient boosting is that it trains faster for large datasets and it provides support for handling multiple categorical features, thus being great for multi-class classification. On the other hand, the cons of gradient boosting is that it is prone to overfitting. Our model was able to beat the trivial model that predicted all fit for both the train data and test data, although for the test data it was only able to significantly improve on precision for test data while improving slightly on the accuracy, F1 score, and recall.

3.7 XGBoost

Table 9: XGBoost Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.749	0.737
F1 Score	0.662	0.643
Precision	0.733	0.654
Recall	0.749	0.737

For our XGBoost model, we used the implementation of XGBoost from the XGBoost package. In order to optimize our model, we used a learning rate of 0.3 and a max depth of 6, allowing our model to have depth for better prediction. Moreover, for our loss function, we used cross-validation, allowing us to optimize our loss at each step. In order to regularize the loss, we also added L2 regularization to penalize our model during each iteration. Some pros of XGBoost is that it allows for cross-validation after each iteration and supports regularization, allowing for better tuning of the hyperparameters in order to optimize the model. On the other hand, some cons of XGBoost is that it is sensitive to outliers as it attempts to fix errors in each iteration. For our model, although it performed much better than the trivial model on the test data, it was only able to beat the F1 score and precision score for the test data.

3.8 Deep Learning With Pre-Trained Word Embeddings (Recommended Model)

Table 10: Deep Learning Model Evaluation

Metric	Train Data	Test Data
Accuracy	0.7560	0.7512
F1 Score	0.7510	0.6786
Precision	0.7785	0.7102
Recall	0.7262	0.7512

We have implemented a multiclass classification model using a sequential model in tensorflow. Our sequential model has 2 dense layers with relu activation and a final dense layer with softmax activation with Total params: 9,523, Trainable params: 9,523, Non-trainable params: 0. Furthermore, we added a dropout layer with a probability of 0.2 in order to account for vanishing gradients and avoid overfitting.

Table 11: Model: "Sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 60)	6960
dense_1 (Dense)	(None, 40)	2440
dropout (Dropout)	(None, 40)	0
dense_2 (Dense)	(None, 3)	123

We are using pre-trained GloVe word embeddings for the review_text and review_summary [2]. These word vectors are trained on corpus - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d vectors) and is made available under the Public Domain Dedication and License v1.0.

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

In order to optimize our model, we are using Adam optimizer with learning rate 0.001 and categorical_crossentropy loss. Since we have unbalanced data, our model is trained on accuracy, f1 score, precision and recall metrics. Some pros of using deep learning for the classification task is that we can add a lot of richness into our model by customizing the layers and have the ability to tune hyperparameters for better optimization. On the other hand, some cons of using deep learning for classification can be that it may be prone to overfitting if the hyperparameters and the model is not well designed and can be computationally intensive. For our final model, we recommend the deep learning model with pre-trained word embeddings as it was able to beat the trivial model convincingly for all of our scores for both the train and test data.

3.9 Model Comparison and Summary

Our deep learning model with pre-trained word embeddings outperforms others since it considers all information rather than just the length of the review text and review summary. We are also capturing user preferences that basic size and fit data cannot, and underlying item correlations were uncovered using text reviews. Furthermore, our deep learning model is flexible in that it allows us to add layers or remove layers depending on the data in addition to tuning the hyperparameters, giving us much more room for improvements than other models which may be less flexible. Although other models were able to improve on precision and F1-score, the other models failed to improve on the accuracy due to the data imbalance. Additionally, the other models failed to effectively deal with the data imbalance compared to our deep learning model with pre-traned word embeddings, which was effective thanks to the pre-processing and the dense layers and the dropout layers that avoided overfitting and vanishing gradients.

4 LITERATURE

Our dataset comes from an online clothing industry called RentTheRunaway which allows female users to rent vintage women's clothes. The dataset specifies various details about the customer's size, clothes' sizes, and fit feedback from the customer ranging from small, large, and fit. It has been used by other researchers including Professor McAuley in order to perform a classification task to determine whether a given clothing fit for the user or not and a recommendation task to recommend the best sizes to fit each customer. However, Professor McAuley went even further to apply an ordinal regression procedure in order to factorize the semantics about customers' feedback on fit and developed different heuristics to deal with the label imbalance due to the dataset being dominated by fit labels. Currently, the field of fashion recommendation systems is relatively new and still yet to be thoroughly explored according to Professor McAuley.

In addition to the RentTheRunaway dataset, different datasets from different clothing brands such as ModCloth has been used in order to work on the problem of fashion recommendation based on fit. Before our project, other researchers have implemented different models to solve this problem. One approach was a Bayesian logit and probit regressions models with ordinal categories to model fit. Another approach uses skip-gram models to learn the latent factors of customers and products to find latent features that might help with fit. However, this approach falls short in that it assumes specific features that may not be available in other platforms. However, for Professor McAuley's work, he decided to use metric learning and collaborative factoring in addition to a Large Margin Nearest Neighbor (LMNN) model to derive prototypes compared to other works in the past that used K-Nearest Neighbor (KNN) classification for prototyping.

In Professor McAuley's work, he uses a latent factor formulation in which he assigns a score to each data based on the fitness of the cloth on the customer. Furthermore, he defines a parent for each product which is based on the type of the cloth in order to maintain fit ordering. For his scoring function, he uses a Hinge loss based of the ordinal regression task and optimizes the loss through projected gradient descent. For prototyping, he re-samples from different classes in order to handle label imbalance and uses LMNN in order to maintain a margin between the three different types of fitness (small, fit, large). For the implementation of his models, he

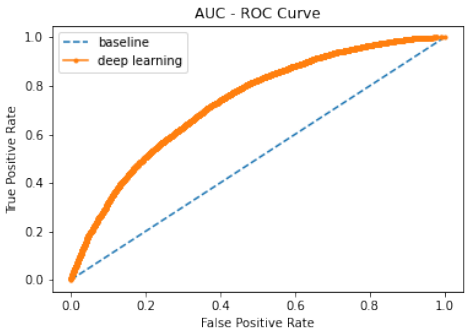

Figure 3: AUC - ROC Curve

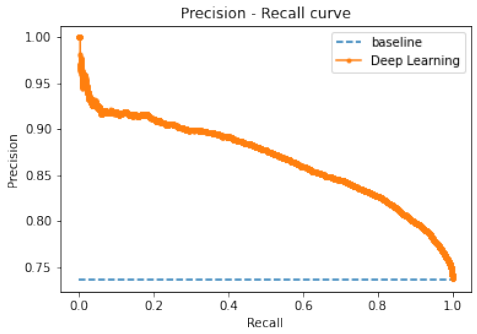

Figure 4: Precision - Recall Curve

uses the PyLMNN package which tunes the hyperparameters using L-BFGS.

For his models, he implements a 1-LV-LR, K-LV-LR, K-LF-LR, K-LV-ML, and K-LF-ML models. For the 1-LV-LR and K-LV-LR models, he uses 1 and K latent variables and one bias terms for each customer and product and uses Logistic Regression for the final classification. For the K-LF-LR model, he builds a similar model as 1-LV-LR and K-LV-LR except he uses a Hinge Loss mentioned above, also using Logistic Regression to make the final prediction. On the other hand, K-LV-ML uses metric learning approach outlined above with K-LV to make the final classification rather than Logistic Regression. Finally, the K-LF-ML model uses metric learning approach with K-LF in order to produce the final classification. To determine the effectiveness of each model, he used AUC.

From the experiment, he found that models with K-dimensional latent variables outperformed the simple models with just one latent variable. Additionally, he states that he recommends the K-LF-ML model as it substantially outperformed other models. Furthermore, he states that the RentTheRunAway dataset gave more improvements than ModCloth dataset maybe due to the fact that the ModCloth dataset may have a cold start problem due to products and customers having few transactions. Additionally, he states that the models suffer a bit from a cold start problem at the beginning, but once given sufficiently many samples, the performance improves significantly.

The results from the professor's work aligns with our work as we also had trouble with data imbalance issues. As the dataset contained a lot of fit labels, we had to focus on balancing the data somehow by unconvering potential latent factors that could improve performance, leading us to explore many different features. In addition, we found out that having more factors in our model was more effective in improving performance as our deep learning model with multiple layers and word embedding have helped in improving our performance from our baseline model. Overall, our findings aligned with the previous works done on fit prediction tasks for the different fashion datasets.

5 RESULTS

Figure-3 illustrates the AUC - ROC curve for the fit class, with TPR on the y-axis and FPR on the x-axis under one-vs-rest setting. The AUC-ROC curve is a performance measurement for classification problems in which ROC is a probability curve and AUC represents the degree or measure of separability. It quantifies the model's ability to differentiate between classes. The higher the AUC, the better the model predicts each class. For our best classification, the ROC AUC score (Area Under the Receiver Operating Characteristic Curve from prediction scores) is **0.73**, showing that our model has performed well.

The Precision-Recall curve is illustrated in **Figure-4**, with Precision on the y-axis and Recall on the x-axis. The Precision-Recall curve represents the trade-off between Precision and Recall at various probability thresholds, that is, the trade-off between false positives and false negatives. If we had a balanced dataset, the baseline would have been 0.5. However, in our case, the baseline is determined by the ratio of positives (P) to positives (P) and negatives (N), which is expressed as P / (P + N). The deep learning model's auc score (Area Under the Curve (AUC) using the trapezoidal method) is **0.868**, and its f1 score is **0.67**, thus showing that our model has indeed performed well.

5.1 Analysis of Different Model Results and Features

We can see that the model used in the start did not work well on the biased label data. Initially, models like multi-class logistic regression, K-Nearest Neighbors, Naive Bayes, Stochastic Gradient Descent, and Random Forest could not catch the better output partitions being a highly imbalanced data. However, later machine learning models such as Gradient Boosting and XGBoost helped because of their good differentiation even in biased data. Ultimately, our deep learning model with pre-trained word embeddings using the GloVe word embeddings worked the best as it was able to do a better job uncovering the latent factors that could help classify the fit with the given features. From this, we conclude that our deep learning model with pre-trained word embeddings worked the best because of its better partitions even in biased data thanks to the carefully designed feature vectors.

When exploring which features worked out well and which features did not work as well, we found the word-embeddings using the reviews from the customers worked the best in helping us with predict fit while using just height by itself did not work as well. We

believe that our word embeddings of the customer reviews greatly helped our deep learning model in capturing user information of the item, allowing our model to distinguish itself from the other models that we have trained. This may be due to the fact that the word embeddings are based on the actual reviews by the customers, which may contain elements that are not reflected by other physical information about the customer based on their height, weight, etc. Additionally, as there are cases where customers may not feel that the clothes fit them even if their size seemingly matches, we believe that it is very important to also consider performing a sentiment analysis in addition to just looking at the physical data to get the entire picture of the customer's feedback. In our experiment, performing sentiment analysis using the word embeddings based on GloVe helped us greatly increase the effectiveness of our model, allowing us to see the importance of sentiment analysis when performing the fit recommendation task. From this, we interpret the word-embeddings as an effective way in converting the reviews of each customer to a feature that can help the model uncover hidden dimensions to effectively perform the fit prediction. On the other hand, using just height by itself did not work well. We believe this may be due to the fact that there are some cases when we should consider more than height for some customers. As some customers may not feel like certain clothes do not fit when the height may seemingly match, it may not be wise to just use height by itself to predict fit. On the other hand, using other physical features in addition to the word-embeddings gave us the best results, leading us to conclude that the word-embeddings from the reviews along with the physical information of the customers worked the best for our fit prediction.

5.2 Future Work

In the previous sections we have discussed prediction task by building models which does not use any user or item id. This same model can be used to predict whether an item fits for a user or not. This information would help us in extending our task to recommend an item for a user. We have divided the attributes as user related and item related. First, we discuss the relation of the attributes to user and item. Attributes related to user's measurements like height, weight, etc. describe user's information, so these features are considered as user related data. Additionally, the word embedding of the review texts give us vectors that represent the user's feedback on the item. As there are many reviews per item than user, taking the average of all those vectors for an item would give us the average information provided by all users to that item and these attributes are considered as item related data. So when a user and item pair is given we can calculate the inputs for the model and predict the fit. For a case where existing user and item pair with review is given, we can directly use the actual attributes. Unfortunately, in the real world, we get many cases where user and item may be new. In the next section, we discuss many such cases.

5.2.1 Handling new User and Item Pairs. In existing user and item id but new pair case, we use averages of the respective related data. In a new user, but existing item case, we use averages across all the users for user related attributes and use that item average for the item related attributes. In new item but existing user case, we use averages across all the items for items related attributes and use that user average of the user related attributes. In both new user and item case, we would just predict fit.

REFERENCES

[1] Rishabh Misra, Mengting Wan, and Julian McAuley. 2018. *Decomposing Fit Semantics for Product Size Recommendation in Metric Spaces.* New York, NY, USA. https://doi.org/10.1145/3240323.3240398

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. *GloVe: Global Vectors for Word Representation.* http://www.aclweb.org/anthology/D14-1162