Classifying Different Types of Fashion Data

Charlotte Stieve, Julia LaRosa, Meroska Gouhar
Northeastern University
Code: https://github.com/jlarosa02/AI25_Fashion

Abstract

With fashion data available in multiple formats such as images, structured attributes, and product descriptions, developing models that can accurately classify clothing items is both a complex and valuable task. This classification supports applications like recommendation systems, inventory management, and product search. In this project, we explored three different approaches to fashion classification, each using a unique data type. The first method focused on computer vision, applying both Vision Transformer (ViT) and VGG-16 models to predict multiple labels from product images. The second approach used structured data including features like color, season, and gender to classify items into master categories using a custom Random Forest implementation, with hyperparameter tuning performed through both grid search and Bayesian optimization. The third strategy relied on natural language processing, using Bag-of-Words and TF-IDF vectorization as well as Support Vector Machines (SVM) to classify products based on textual descriptions. Across our experiments, ViT demonstrated stronger generalization than VGG-16 for image classification, Bayesian tuning led to the best Random Forest performance with an accuracy of 73.37%, and SVM achieved the highest accuracy of 98.1% for description-based classification. These results highlight the importance of tailoring machine learning methods to the structure of the input data when building accurate classification models.

Introduction

Apparel classification plays a key role in e-commerce, supporting tasks like product search, recommendation systems, and inventory management. Fashion data varies widely in form; images, structured attributes, and text descriptions, each requiring different machine learning strategies. In this project, we explored three distinct approaches to classifying clothing items, each using a different data type and algorithm.

The first approach used deep learning on images to perform multi-output classification, predicting gender, article type, color, season, and usage. Two models were tested—VGG-16 and Vision Transformer (ViT)—both taking 224×224×3 images as input and outputting class probabilities for each attribute.

The second approach relied on structured tabular data (base color, season, gender, year, and usage) to predict the master category using a Random Forest classifier. This included custom implementation, manual one-hot encoding, oversampling for class balance, and hyperparameter tuning using both grid search and Bayesian optimization.

The third approach focused on text-based classification using product descriptions. These were preprocessed and converted into feature vectors, then classified using a Support Vector Machine (SVM) to predict clothing categories.

By tackling the same problem through different data modalities and algorithms, we were able to compare the strengths and limitations of each method and gain a deeper understanding of fashion classification in real-world scenarios.

We chose to use the Fashion Product Images dataset from Kaggle, created by Param Aggarwal, which contained 44,000 images and metadata entries. It has a variety of categories and descriptions, as well as images of each fashion item, making it well-suited for tasks such as classification, recommendation, and multimodal learning.

Background

Artificial Neural Networks

Artificial Neural Networks (ANN) are models that function similarly to the network of neurons in a human brain. Similarly to biological neurons, artificial neural networks consist of interconnected nodes. These nodes can be divided into distinct layers, including an input layer, at least one hidden layer, and an output layer. As input travels through the layers of the neural network, each connected node multiplies its incoming inputs by a corresponding weight. The node then calculates a weighted sum of its inputs (adding an additional bias value) as follows:

$$z = \Sigma\ (w_i * x_i) + b$$

The sum z is passed to an activation function so that the neural network introduces non-linearity, allowing it to account for more complex patterns. The resulting value from the activation function is then passed to the next layer via the node's output connections. The final output layer provides the results of the relevant classification task. When training an ANN, loss functions are used to compare the network's output classification to the true classification of the data. Backpropagation, the process of feeding the calculated error backwards through the network, is utilized in training to adjust the various weighted connections within the ANN.

VGG-16: A Convolutional Neural Network

VGG-16 refers to a specific type of CNN proposed by Oxford University's Visual Graphics Group (VGG) (Simonyan & Zisserman, 2015). This network includes 16 layers in total: 13 convolutional layers and 3 dense, fully connected layers. The convolution operation (in this CNN and others) applies sliding 2D filters to a receptive field in the input image to detect local features. The 13 convolutional layers are broken down into five blocks (each 2-3 layers) where the number of filters increases with propagation (first 64, then 128, 256, and ends on 512). Each layer concludes with a rectified linear unit activation function as follows:

$$\text{ReLU}(x) = \max\{0, x\}$$

At the end of each block, an additional max-pooling layer is added to shrink the image while also maintaining features. The 3 concluding dense layers gradually reduce the layer neurons into the number of classes in the relevant categorization task(s). In the final layer, a SoftMax activation function is used as follows:

Given an input vector $\mathbf{z} = [z_1, z_2, \ldots, z_K]$, the softmax function outputs a vector $\sigma(\mathbf{z}) = [\sigma(z_1), \sigma(z_2), \ldots, \sigma(z_K)]$ such that:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, 2, \ldots, K$$

The final output then becomes a probability distribution over all classes. Figure 1 shows a diagram of the VGG-16 network.

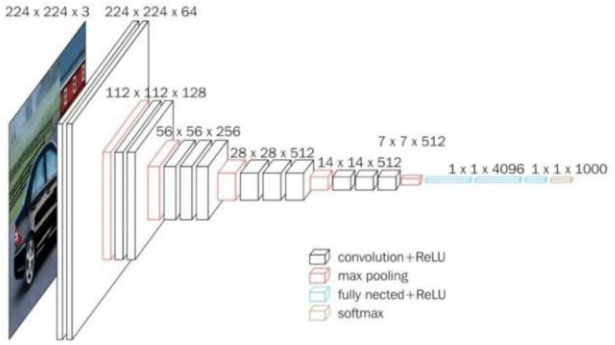

Figure 1: VGG-16 Architecture

ViT: A Transformer Inspired Neural Network

Transformers have been very successful in regard to LLMs (Vaswani et al. 2017), so the ViT model has been introduced to apply a similar architecture to computer vision problems (Dosovitskiy et al. 2021). The input image is broken down into patches (16x16 pixels), which are then flattened into vectors. These vectors are then linearly projected into an embedding space. Since transformers typically deal with text inputs, the positionality of each original patch is then embedded. Additionally, a Classify Token ([CLS]) is prepended to the entire sequence of patch embeddings that will collect information on the global image. The sequence then enters the Transformer encoder which is composed of layers each containing a multi-head self-attention (MHSA) and feed-forward neural network (FNN). The MHSA allows the model to focus on several parts of the image all at once and conclude what patches are more important for classification, aggregating this information within the [CLS] token. The FFN allows for non-linear processing of the image. Each layer concludes with a normalization step. Finally, the sequence has been processed through all of the Transformer encoder layers, a multilayer perceptron (MLP) is used to classify the image based on the information aggregated through the [CLS] token. Figure 2 shows a diagram of the ViT network.

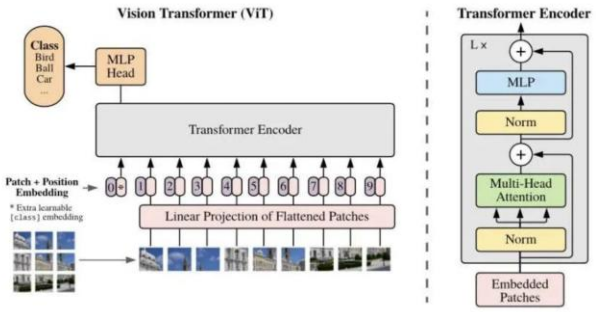

Figure 2: Vision Transformer Architecture

Support Vector Machines

Support Vector Machines (SVM) are a supervised learning method often used for classification. It works by finding a hyperplane that best divides the different classes by maximizing the margins between the points and the plane. Figure 3 below demonstrates a simple SVM model that separates two classes.

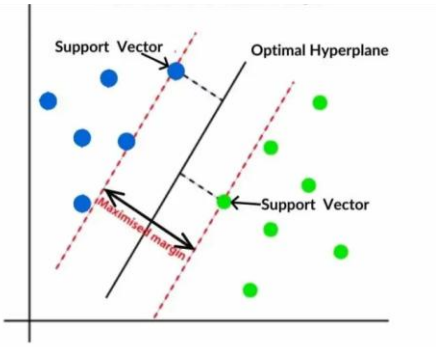

Figure 3: An SVM model showing the maximized margin between two classes of data.

Support Vector Machines rely on Kernel functions to organize the input data into a feature space where the data can become linearly separable, even if it was not previously. Different types of Kernel functions transform the data in different ways to achieve the best possible fit.

When faced with multi-class data, SVM's break down the classes into binary classification problems, either by

comparing pairs of classes, or comparing one class to the rest of classes grouped together, the latter of which we used.

Bag-Of-Words

A Bag-Of-Words model is commonly used to represent textual data for machine learning algorithms, which require numerical values. Bag-Of-Words models transform the text into a vector based on word frequency and are referred to as a bag of words because it does not take into account order or structure of the text when doing so.

Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency model (TF-IDF) functions similarly to a Bag-Of-Words model, however it contains more information about the importance of certain words. The TF-IDF model adds weight to the words, increasing weight if a word appears often in a document, and decreasing the weight of a word if it can be found across multiple documents. The math of this model can be seen below, where the Term Frequency is calculated based on how many times a term appears in a document, compared to the total number of terms in the documents, and the Inverse Document Frequency is found using the total number of documents (*N*) and the number of documents that contain a term (*df*):

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

Decision Trees

Decision trees are supervised learning models that partition data based on features that yield the best class separation, typically measured using the Gini impurity or entropy. While easy to interpret, they can be prone to overfitting. Random forests mitigate this by averaging predictions across multiple decision trees, each trained on a bootstrap sample and a random subset of features, which improves stability and accuracy. One-hot encoding is used to convert categorical variables into binary vectors for training. To fine-tune model parameters such as tree depth and ensemble size, Bayesian optimization offers a more efficient alternative to exhaustive grid search by modeling the objective function and selecting hyperparameters that are likely to improve performance.

In optimizing the random forest's performance, selecting appropriate hyperparameters—such as the number of trees (n_estimators), the maximum depth of each tree (max_depth), the minimum number of samples required to split a node (min_samples_split), and the minimum number of samples required at a leaf node (min_samples_leaf), is critical. Traditionally, Grid Search Cross-Validation (GridSearchCV) is used to exhaustively search across predefined hyperparameter grids. However, this approach can be computationally expensive and inefficient, especially when the search space is large.

To address this, our project employed Bayesian optimization, a more intelligent and sample-efficient strategy. Instead of blindly evaluating all parameter combinations, Bayesian optimization models the performance of the objective function (in our case, classification accuracy) as a probabilistic surrogate function, such as a Gaussian Process or Tree-structured Parzen Estimator (TPE). It iteratively selects the most promising hyperparameter values to evaluate next by balancing exploration of uncertain regions with exploitation of known good regions. This approach dramatically reduces computation time while still identifying high-performing configurations.

Related Work

The use of artificial intelligence is on the rise in the field of fashion, and there are many different methods being used to classify clothing products, especially for use in the online retail sector.

Much of the research done on fashion classification algorithms revolves around classifying images, as clothing is a physical object, and is therefore often represented through pictures. Other researchers often choose to use deep neural networks to classify clothing images. In particular, (Abbas et al. 2024) aimed to use the GrabCut algorithm to remove clothing images from their backgrounds, then employ ResNet152 and EfficientNetB7 architectures to create a precise classification system. The images in the dataset we used were real world RGB images, similar to those used by (Abbas et a. 2024), however the backgrounds of the images in the dataset we used were already removed, eliminating the need for a GrabCut algorithm. While a ResNet Neural Network could have been used for image classification of clothing items for this dataset, a ViT can outperform ResNet and a VGG is more straightforward.

In terms of classifying data based on attributes and descriptions, a naive bayes classifier was another popular option for each of those. However, random forest algorithms can handle more complex data and reduce over fitting more effectively than naive bayes algorithms, which is why it was chosen to classify the data based on attributes.

Naive bayes are often used for text classification problems due to their speed and efficiency and could have been used to classify products based on product descriptions. However, SVM models are able to take into account interactions between words in a way that a naive bayes model, which views each word as independent cannot. This seemed

important in fashion product data, where words within a description likely have a strong connection with one another to accurately describe a product.

Project Description

Algorithm 1

Given an RGB image *x* with a size 224 x 224 pixels, the computer vision algorithm will perform multi-output classification over the five categories of gender, article type, color, season, and usage. Formally, this problem is defined as follows:

$$f\Theta(x) = (\hat{y}_{gender}, \hat{y}_{article\ type}, \hat{y}_{color}, \hat{y}_{season}, \hat{y}_{usage})$$

In the output, each $\hat{y}_i$ is a predicted probability distribution over class labels for each category. Two separate versions of this algorithm were developed: one using a ViT model and the other using a VGG-16 model. Comparisons of these two models will be discussed in the Experiments section.

For data cleaning purposes, only entries with the master category of "apparel" and an associated image path were used. Additionally, only entries from the year 2012 were used, leaving a total of 8,631 entries. To begin, a Pandas dataframe was created that encoded the five label categories into numerical values. The Scikit-learn library was used to randomly divide the dataframe (seed 42) into a training set of 80% of the entries and a test set of 20% of the entries. Data preprocessing was defined with a transform variable that resized images to the ImageNet standard of 224 x 224 and converted PIL images to a tensor. A custom PyTorch dataset class (pseudocode below) was given the training entries + transform variable and the test entries + transform variable to create a train set and test set, respectively.

Algorithm 1 Custom PyTorch Dataset Class

Require: DataFrame `df`, Transform `transform`
  Define class `CustomDataset`
    Initialize with `df` and `transform`
    Store `df` and `transform`

    Function `len()`

  return number of rows in `df`

    Function `getitem(index)`
      Load image using `df[index]`
      Get corresponding labels from `df[index]`
      If `transform` is not None, apply it to image
      Convert image and labels to PyTorch tensors

  return image tensor, label tensor

For the ViT version, data loaders (trainloader and testloader) were created using DataLoader imported from torch.utils.data. For the VGG-16 version, flow_from_dataframe was utilized from the Keras library. Both versions of the algorithm used a batch size of 64. It is important to note that ViT architecture does not include inductive biases like CNNs do, requiring them to process an extensive dataset for successful training. Since we are working with a slightly small dataset, we fine-tuned a pre-trained ViT from the Hugging Face library (pre-trained from the ImageNet-21k dataset). To account for our five different categorizations, five linear classifiers were added to the ViT model. We used the classifiers on the [CLS] to output probability distributions for each category. The VGG-16 model was made from scratch using the architecture described in the Background section. A custom multi_output_generator was created to output a dictionary of our five categories, and a custom output_signature was also created so that we could use our custom generator seamlessly with TensorFlow. Both models utilized an Adam optimizer initially configured to a learning rate of 0.0001 to account for our small dataset. Additionally, both models utilized a scheduler that reduced the learning rate on a plateau that monitored test loss with a patience of 3 (minimum learning rate of 0.00001).

Algorithm 2

This algorithm centers on classifying fashion items into predefined categories, referred to as "masterCategory," based on structured item-level attributes. These categories represent coarse-grained labels for clothing types such as "Shirts," "Footwear," "Accessories," and others. The problem is formulated as a supervised multiclass classification task.

Let the dataset be defined as:
$S=\{(x1,y1),(x2,y2),\ldots,(xn,yn)\}$ where $xi \in Rd$ is a feature vector for the ith item, and $yi \in Y$ is the corresponding categorical label from a finite set of master categories Y.
Each input vector xi includes features such as:
xi,1 : baseColour
xi,2 : season
xi,3: gender
xi,4: usage
xi,5: year (numerical)

Categorical features were manually encoded using one-hot encoding to convert each unique category into binary indicators. Numerical features were used directly. The target is to learn a function: $f:X \rightarrow Y$ that maps the encoded input space $X \subset Rd$ to one of the master category labels Y.

We first cleaned and preprocessed the dataset by removing irrelevant columns (such as 'Unnamed: 10' and 'Unnamed: 11') and filtering records between 2011 and 2012. Categorical features (e.g., baseColour, season, gender, usage) were manually transformed using one-hot encoding, resulting in binary features for each unique category value. The dataset was then split into training (80%) and testing (20%) sets using a reproducible permutation-based split.

A Decision Tree algorithm was built such that at each node, a subset of features was randomly selected, and the

best feature-threshold split was chosen based on Gini impurity:

Gini Impurity at node is calculated as:

$$G(t) = 1 - \sum_{i=1}^{c} p_i^2$$

where p_i is the probability of class i at node t, and C is the number of classes. Splitting continues recursively until a stopping condition is met (e.g., maximum depth or homogneity of labels).

Algorithm 1 DecisionTree

```
function DecisionTree(X, y, max_depth, max_features)
    if max_depth is reached or all y have the same label then
        return Leaf node with most common label in y
    end if
    Select random subset f ⊂ F of features
    best_gini ← ∞
    for each feature f_i in f do
        for each threshold t in X[f_i] do
            Compute Gini impurity of the split
            if Gini impurity < best_gini then
                Update best feature and threshold
            end if
        end for
    end for
    Split data into left and right based on best threshold
    Recursively build left subtree with left data
    Recursively build right subtree with right data
    return Internal node with best feature and threshold
end function
```

We extended our tree implementation into a Random Forest classifier; an ensemble of multiple decision trees trained on different bootstrap samples and random feature subsets. The final prediction is the mode of individual tree predictions.

Algorithm 2 RandomForest

```
function RandomForest(S, F, B)
    H ← ∅                                   ▷ Initialize ensemble of trees
    for i = 1 to B do
        S^(i) ← BootstrapSample(S)
        h_i ← DecisionTree(S^(i), max_features, max_depth)
        H ← H ∪ {h_i}
    end for
    return H                                ▷ Return the forest of decision trees
end function
```

We initially explored performance tuning with grid search over the parameter space of tree depth, number of estimators, and splitting thresholds. However, this method was computationally expensive and yielded only marginal improvements.

To improve this, we used Bayesian Optimization (Optuna framework), which adapts its search space based on past evaluations.

Algorithm 3 Bayesian Hyperparameter Optimization

```
function Objective(trial)
    n_est ← trial.suggest_int('n_estimators', 5, 50, step=5)
    samp ← trial.suggest_float('sample_ratio', 0.5, 1.0, step=0.1)
    mf ← trial.suggest_int('max_features', 1, D)
    md ← trial.suggest_int('max_depth', 1, 5)
    model ← RandomForest(n_est, samp, mf, md)
    model.fit(X_train, y_train)
    preds ← model.predict(X_val)
    return accuracy_score(y_val, preds)
end function
```

This formal setup allowed for rigorous model comparison and performance optimization.

Algorithm 3

Overall, given a textual description of a series of clothing items, this algorithm aims to classify them into a master category of either "Accessories", "Apparel", "Footwear", "Personal Care", or "Free Items". Within this process, multiple NLP algorithms are used that take in a product description and output a numerical vector. This numerical vector is then inputted into a Linear SVM model, which then returns an array of strings, where each string correlates to the predicted category of the product, and can be used to find the accuracy of the model.

Before the algorithm could be implemented, the data needed to be cleaned. First, rows with missing values were deleted, as well as any data prior to 2016 to ensure classification was done on more current products. Once this was completed, the "Accessories" category had over 2500 more values compared to the other categories in the dataset. To fix this, undersampling was performed to even out the dataset, randomly removing 2500 of these samples. Next, human-readable product descriptions from JSON files whose names correlate with the ID category of each product were extracted, any html tags in the descriptions were removed, and they were added as a column to the dataset. This was done to ensure they were accessible for classification purposes.

Following this, the descriptions were converted into vectors based on word frequency, twice with a Bag-Of-Words approach using the Scikit-learn function Count Vectorizer and once using the Scikit-learn Tfidf Vectorizer. For one of the Count Vectorizers, as well as the Tfidf vectorizer, a list of stop words was provided, which is a list of common English words, such as "the", "as", and "is" that are excluded from the vector. These vectors could then be provided to a Linear SVM model to be classified.

Algorithm 3 Text Processing and Linear SVM

```
text = Process Text                          ▷ Prepare the text to be vectorized
X train, X test, y train, y test = Train Test Split(Descriptions, Categories)
vector = Vector Algorithm                    ▷ Create vector converter
fitted vector = vector.fit(X train)          ▷ Fit the description to the vector
model = LinearSVC().fit(fitted vector, y train)
predictions = model.predict(X test)
```

After comparing Scikit-learn SVC and Linear SVC functions, along with a Scikit-learn SVC functions that used a custom linear kernel, it was determined that the Scikit-learn Linear SVC function ran the fastest, without sacrificing accuracy, with a C value of 0.1. It is worth noting that while called SVC as opposed to SVM, these methods function the same and will be used interchangeably going forward. These models were chosen for comparison as the kernel used by SVC, a Radial Basis Function kernel, is one of the most widely used kernel functions, while also providing insight on if the data needs to be transformed into a higher

dimension for separation. Meanwhile the kernel used by a Linear SVC, a Linear kernel, is most often used for text classification problems, since they tend to have many features and are linearly separable to begin with. Using a linear kernel also provides insight into whether the data is separable without a transformation into a higher dimension. The formulas for Linear and RBF Kernels can be seen here:

Linear kernel	$k(X_i, X) = X_i \cdot X$
Radial basis kernel (RBF)	$k(X_i, X) = e^{-(\lvert X_i - X \rvert^2 / 2\sigma^2)}$

The data was randomly split into train and test sets with 80% of the data being the train set and 20% being the test set. After the data was split, the product descriptions in both the train and the test sets were turned into vectors with their respective word frequency algorithms. The Linear SVM model was then used to classify the data into the appropriate master category.

To classify the data, the Support Vector Machine algorithm aims to find the linear hyperplane, whose margins maximize the distance between the points of different classes. The algorithm uses a hinge loss function to award a penalty for violations. Mathematically the hinge loss function is defined below where $f(x_i)$ is the decision function value output, and y_i is the true class label for the data point, 1 if classified correctly, and –1 if not:

$$\text{Hinge Loss}(x_i) = \max(0, 1 - y_i \cdot f(x_i))$$

This function makes it so a correct classification within the margin will always result in no penalty, while a point that is misclassified or within the margin will have a larger penalty. The SVM algorithm aims to optimize the balance between maximizing the margin and minimizing the penalty provided by the hinge loss function. It does this through optimization of the equation below. The first half of the function aims to maximize the margin by minimizing the weight vector. The second half contains a C value that balances the tradeoff between maximizing the margin and minimizing the penalty, as well as the loss function that determines the penalty:

$$\underset{w,b}{\text{minimize}} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \zeta_i$$

Below is the pseudocode for an SVM can be seen, where the SVM first checks if the current point violates the margin, then updates the weight accordingly based on if the point is misclassified.

Algorithm 3 SVM Expanded

```
while values not converged do
    for value in vector of description do
        if Category * dot(vector, weight) < 1 then
            weight = update weight with hinge loss function
        else
            weight = update weight with regularization
```

Experiments

Algorithm 1

To gain insight into the different types of computer vision algorithms, several loss and accuracy metrics were compared between the ViT version of the algorithm and the VGG-16 version of the algorithm. When training both versions of the algorithm, it became clear that the ViT model required an even smaller learning rate due to the pre-training of the model. The Adam optimizer was updated to a learning rate of 0.00005 for the ViT model, and the VGG-16 ultimately remained at 0.0001. These small learning rates account for careful updates of both models, but the pre-trained ViT model requires more caution for its fine-tuning purposes. It also appeared that the VGG-16 version initially achieved higher accuracy and smaller loss much faster than the ViT version, indicating a risk of the VGG-16 quickly overfitting the data. To account for this, the early stopping and under-sampling of dominant classes were both introduced into the VGG-16 version of the algorithm. Training and testing on the model would stop if the test loss did not improve for more than 5 epochs. Additionally, under-sampling reduced disproportionately dominant classifications such as “women” for gender or “summer” for season.

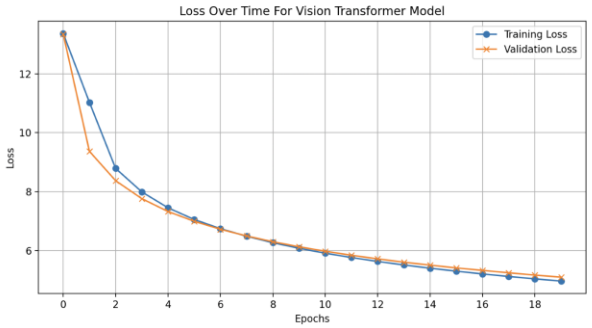

Figure 4.1: Training and test (validation) loss for the ViT version of the algorithm

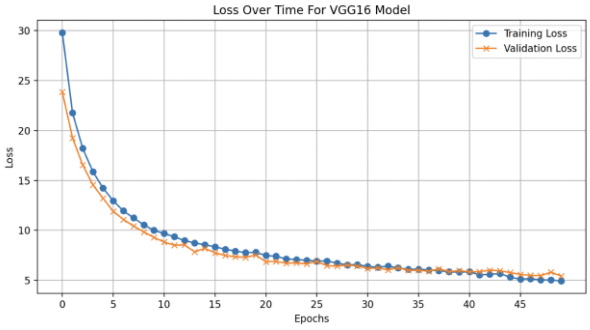

Figure 4.2: Training and test (validation) loss for the VGG-16 version of the algorithm

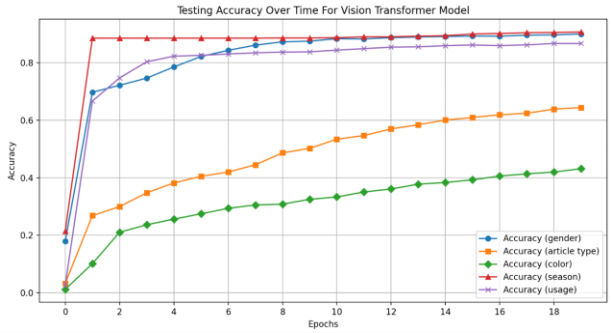

Figure 5.1: Test accuracy for all five categories of the ViT version of the algorithm

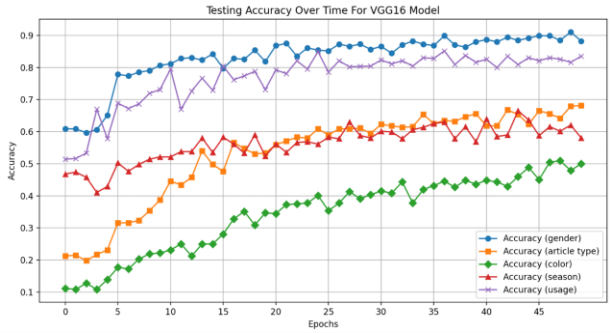

Figure 5.2: Test accuracy for all five categories of the VGG-16 version of the algorithm

Figures 4.1 and 4.2 show both the training loss and test loss for the ViT version and the VGG-16 version, respectively. Figures 5.1 and 5.2 show the testing accuracy across the five categories for the ViT and VGG-16, respectively. When comparing Figure 5.1 to Figure 5.2, it is evident that the pretrained ViT model resulted in a smoother increase in accuracy due to the nature of fine-tuning a model v. training from scratch. Figures 6.1 and Figure 6.2 depict the predictions of the ViT algorithm and VGG-16 algorithm, respectively, on randomly sampled images. Evidently, the VGG-16 version of the algorithm is still prone to overfitting despite the modifications over early stopping and under-sampling. Although the measured metrics of loss and accuracy are relatively similar for both models, it can be concluded that the ViT version of the algorithm is more reliable due to its ability to maintain self-attention over the entire image rather than utilizing convolutional filters on local features like the VGG-16 version. Notably, VGG-16 architecture includes max-pooling, meaning that the model can risk losing smaller details as inputs shrink across each layer. The ViT model's approach of using locationally embedded patches with a summarizing [CLS] token means that it does not risk the same loss of detail. Although a more diverse dataset and further tuned sampling could improve the reliability of the VGG-16 model, these results indicate that ViT models offer a more contextually comprehensive version of our computer vision algorithm.

Figure 6.1: Category predictions of random product images using the ViT version of the algorithm

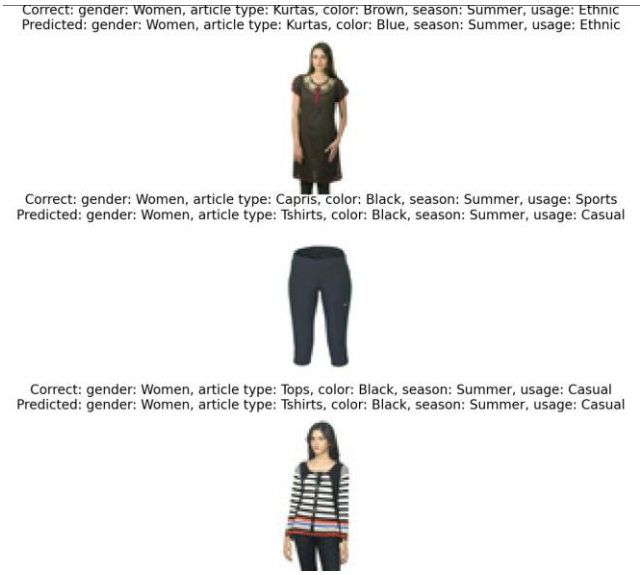

Figure 6.2: Category predictions of random product images using the ViT version of the algorithm

Algorithm 2

We began by computing a baseline accuracy: predicting the most frequent class (Apparel). This resulted in an accuracy of 60.92%. Using a single custom decision tree with a maximum depth constraint, we observed an improvement over the baseline. However, decision trees were prone to overfitting, especially on imbalanced classes.

Figure 7.1 shows the results of hyperparameter tuning using GridSearchCV for a random forest model. The best-performing model, tuned across a range of parameters (n_estimators, max_depth, min_samples_split, and min_samples_leaf), achieved a test accuracy of 60.92%.
The confusion matrix reveals strong performance in predicting the Apparel category (2,187 correct), but notable misclassifications between Apparel, Footwear, and Accessories. Less frequent classes like Free Items and Sporting Goods were rarely predicted correctly, reflecting challenges with class imbalance. These results suggest that while the model handles dominant categories reasonably well, further improvement is needed for minority classes and similar-looking items.

Figure 7.2 presents the confusion matrix for a random forest model trained with GridSearchCV-tuned hyperparameters, achieving a test accuracy of 61.75%. The tuning involved experimenting with n_estimators, max_depth, min_samples_split, and min_samples_leaf.

The matrix shows that the model performs best in classifying the Apparel category (2,255 correct predictions), indicating it has learned dominant patterns for this class well. However, there is substantial overlap in predictions among Apparel, Footwear, and Accessories, with many true Apparel instances being misclassified as Footwear (553) or Free Items (384). Suggesting that visual or categorical features between these classes may be quite similar, leading to confusion.

While the model does reasonably well on major classes, it struggles with minority ones like Free Items and Sporting Goods, which have very low true and predicted counts. This reflects ongoing challenges with class imbalance.
Bayesian optimization was used to find the best hyperparameters for the random forest model. Instead of testing every possible combination like grid search, Bayesian optimization focused on the most promising areas in the search space. This made the process faster and more efficient, while still improving accuracy.

In Figure 7.4, the heatmap shows how different values of n_estimators and max_depth affected the model's accuracy. The best results—above 73% accuracy—were achieved when using a higher number of trees (n_estimators around 500) and deeper trees (max_depth near 19 or 20). This suggests that letting the model grow more complex helped it better capture patterns in the data.

Figure 7.3 shows the confusion matrix for the best model found by Bayesian optimization. The model did very well in predicting Apparel, with 3,332 correct predictions. It also performed well for Footwear and Accessories, though there was some confusion between the two. Predictions for smaller classes like Free Items and Sporting Goods were fewer but mostly accurate.

Overall, Bayesian optimization helped us find a better-performing model in less time than grid search. It improved both speed and accuracy, making it a more effective method for tuning the model in this project.

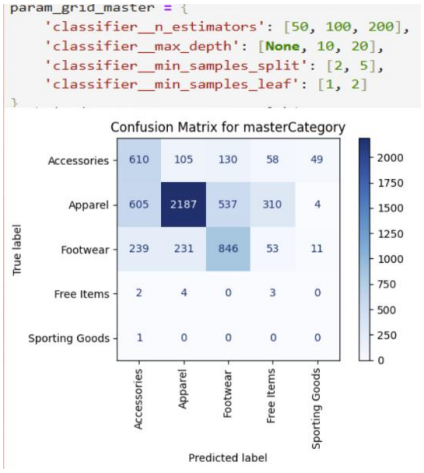

Figure 7.1: Accuracy after hyperparameter tuning

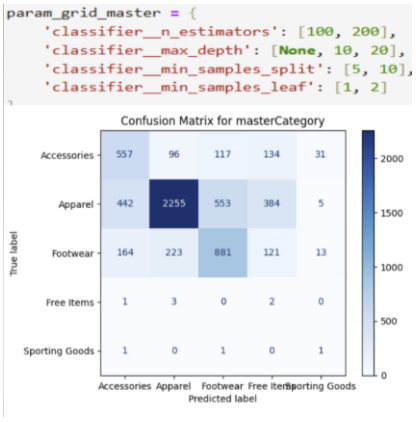

Figure 7.2: Confusion matrix for random forest model with tuned hyperparameters

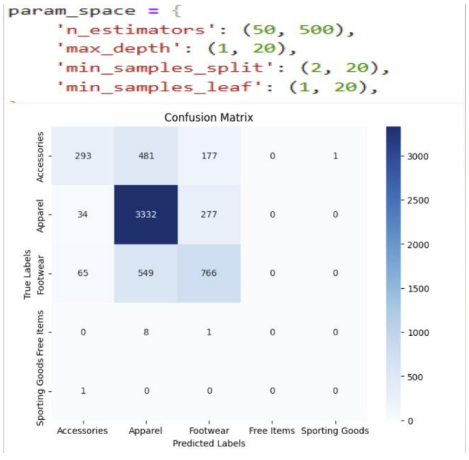

Figure 7.3: Confusion matrix for best model using Bayesian Optimization

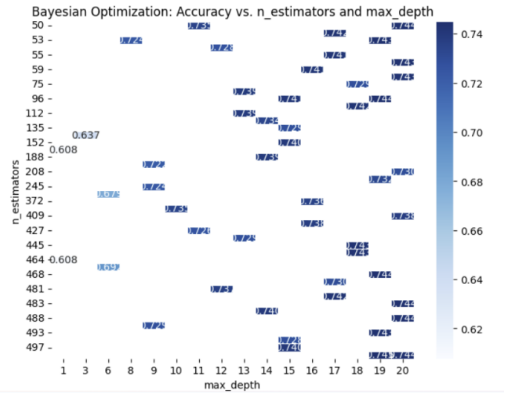

Figure 7.4: Heatmap demonstrating how different parameters affected the model's accuracy

Algorithm 3

To determine the best way to accurately predict the category of a product based on its description, different factors were experimented with, including kernel, C value, and vectorization techniques. When comparing an SVC function, an SVC function with a custom-made Linear kernel, and a Linear SVC function with a built-in linear kernel, it became clear that the latter was the best choice, as it ran the fastest without sacrificing any accuracy, as seen in Figures 8.1 and 8.2.

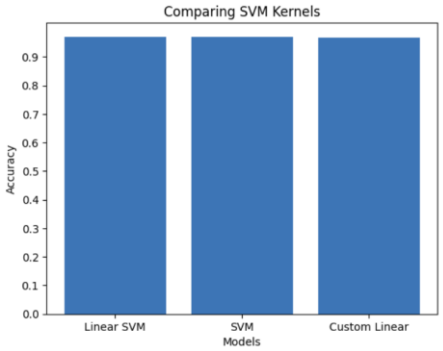

Figure 8.1: Comparing the Accuracy of SVMs using different kernels

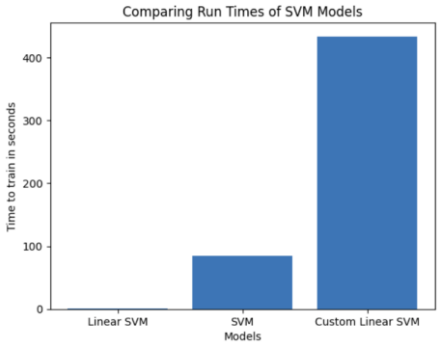

Figure 8.2: Comparing the runtime of SVM's using different kernels

This is likely because the data is already linearly separable, so transforming it to a higher dimension takes longer, without any impact on the accuracy as it was already separable to begin with.

Once this was determined, the best C value needed to be calculated. The C value determines how strict the penalty should be in order to find a balance between the greatest margin and lowest penalty. A higher C value will enforce a stricter penalty for misclassifications. To find the best C value, 5 separate Linear SVC models were run with separate C values which were 0.001, 0.01, 0.1, 1.0, and 3.0, respectively. The accuracies were then plotted and can be seen in Figure 9 below.

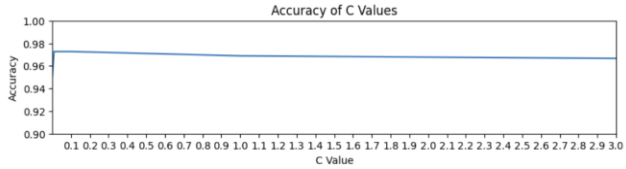

Figure 9: The accuracy of the model over different C values

While the accuracies were not wildly different, there was a small jump between 0.001 and 0.01 from 95% to 97% accuracy, as well as a small decline from 0.1 to 1.0 from 97% to 96%. Much larger values, such as 100, were also tried, but removed and not represented in the graph as it made it harder to see the trends, and remained consistent with the conclusion that a C value around 0.1 was most accurate. It is understandable that much smaller values, as well as larger values could have lower accuracies, since they are either imposing a penalty that is too strict or are too lenient to accurately classify the data.

Once it was determined a Linear SVC model with a C value of 0.1 was most accurate, it was time to compare different vectorization strategies to turn the human-readable product descriptions into machine-readable numerical vectors. A Count Vectorizer that included stop words such as "the" and "is", a Count Vectorizer without these words, and a TF-IDF Vectorizer without these words were all compared. The accuracies among these different vectorizers remained relatively consistent, however the TF-IDF performed best in the end with 97.2%, 97.6%, and 98.1% accuracies respectively. This makes sense as TF-IDF vectors go a step further than Count Vectorizers in adding weights to the text to provide a better understanding of which words should be more influential than others. The confusion matrices for the predictions using a model with each of these vectorizers can be seen in Figures 10.1, 10.2, and 10.3 below showing that each of the models that used the different vectorization techniques performed well and classified most of the products correctly.

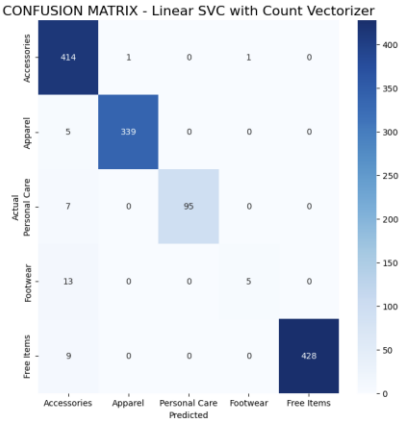

Figure 10.1: A confusion matrix demonstrating how many of the products a Linear SVC using a Count Vectorizer with no stop word removal was able to classify correctly

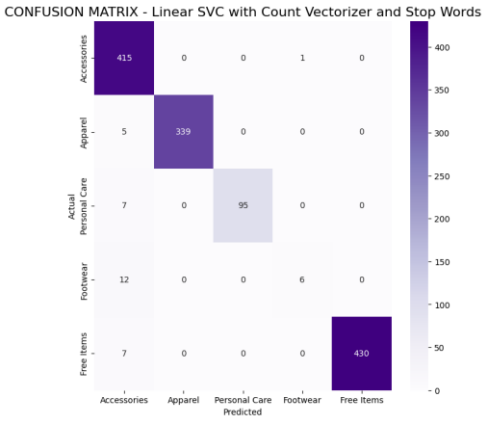

Figure 10.2: A confusion matrix demonstrating how many of the products a Linear SVC using a Count Vectorizer with stop word removal was able to classify correctly

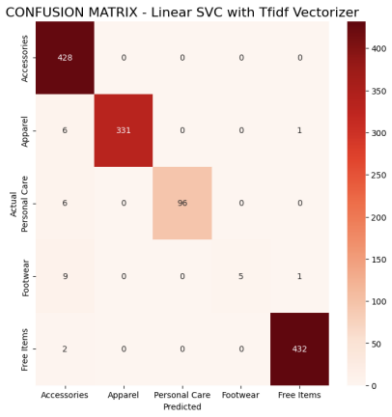

Figure 10.3: A confusion matrix demonstrating how many of the products a Linear SVC using a TF-IDF vectorizer with stop word removal was able to classify correctly

It is worth noting that stop words did not seem to have a large effect on the accuracy of the model when used in the vectorization, so they likely do not carry a lot of influence over the results. This could be due to the model being good at filtering out noise, and other terms having a stronger presence for classification than these terms. Other text preprocessing steps were also attempted, such as stemming words to remove their prefixes and suffixes, however this significantly reduced accuracy to around 46%, indicating that it may have reduced the meaning or connection between words. The confusion matrix for this inaccurate model can be seen in Figure 11 below.

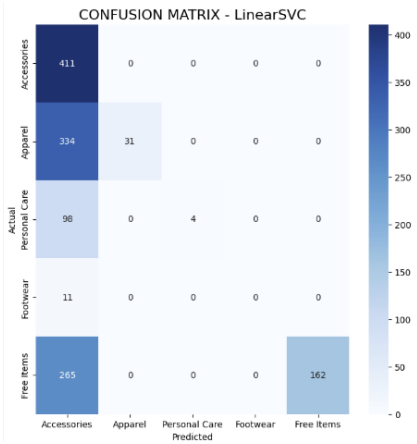

Figure 11: A confusion matrix showing the model performed poorly and misclassified most products as Accessories when stemming preprocessing steps were applied to the text

Conclusion

In this project, we analyzed fashion data across the various forms it comes in: images, categorical labels, and product descriptions. For our first algorithm, the computer vision algorithm, our experiments compared the training loss, test loss, and test accuracy across two versions of the algorithm. The first version utilized a vision transformer (ViT) model, while the second utilized a VGG-16 convolutional neural network. Our results showed comparable results for the two versions, with both achieving over 50% accuracy in four out of five categories. However, the VGG-16 model was prone to biasing frequent classes. Therefore, we concluded that the ViT version of the algorithm was more accurate and reliable in categorizing product images.

For our second algorithm, the Random Forest classifier, various hyper-tuning methods were compared to develop an accurate model that identified the master category based on product attributes. Our results showed Bayesian optimization to be the most effective method of hyper-tuning, resulting in a model with an over 73% accuracy rate.

For our third algorithm, the Support Vector Machine, various kernels, C-values, and vectorization methods were compared to develop a model that best product category based on its description. Our results showed that a model that incorporated built-in kernels, a C-value of 0.1, and TF-IDF Vectorizer was most accurate with a rate of 98.1%. However, our experiments showed that the Vectorizer alterations were nearly negligible, as the SVM maintained extremely high accuracy across all conditions. Overall, our experiments showed that different formats of the same data require unique attention when creating accurate models. Data format must be heavily considered when constructing machine learning solutions since all formats are not the same.

Contributions

Julia LaRosa worked on image classification with neural networks, Meroska Gouhar worked on random forest classification on clothing attributes, and Charlotte Stieve worked on SVM classification of product descriptions. The paper was split up equally between all members.

References

Abbas W.; Zhang Z.; Asim M.; Chen J.; Ahmad S. 2024. AI-Driven Precision Clothing Classification: Revolutionizing Online Fashion Retailing with Hybrid Two-Objective Learning. https://www.mdpi.com/2078-2489/15/4/196

Abhyankarallhad. Medium. 2025. Comparative analysis of VGG-16 Convolutional Neural Network and Vision Transformer (ViT). https://medium.com/@abhyankarallhad/comparative-analysis-of-vgg-16-convolutional-neural-network-and-vision-transformer-vit-388c2a21b178

Aggarwal, Param. Kaggle. 2019. Fashion Product Images Dataset. https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset/data.

Atharva. 2025. Kaggle. Ecommerce Text Classification. https://www.kaggle.com/code/atharva1311/ecommerce-text-classification.

Breiman, L. 2001. Random forests. https://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE. arXiv:2010.11929.

Gaffar, Syed Abdul. 2024. Analytics Vidhya. A Guide to Building an End-to-End Multiclass Text Classification Model. https://www.analyticsvidhya.com/blog/2021/11/a-guide-to-building-an-end-to-end-multiclass-text-classification-model/.

Hadad, Arash. Medium. A Comprehensive Guide to Support Vector Machine (SVM) Algorithm. https://python.plainenglish.io/a-comprehensive-guide-to-support-vector-machine-svm-algorithm-76dbcf18b5ae.

Simonyan, K and Zisserman, A. 2015. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. arXiv:1409.1556.

Singh, P. 2019. A conceptual explanation of Bayesian model-based hyperparameter optimization for machine learning. https://medium.com/data-science/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f.

VanderPlas, J. 2016. Random Forests. In *Python Data Science Handbook*. https://jakevdp.github.io/PythonDataScienceHandbook/05.08-random-forests.html.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. arXiv:1706.03762.