Project Final Report

Erin Jones, Ellis Martin, Isidora Rollán Zegers

Computer Vision; April 19, 2024

GitHub repo (see README for repo contents)

1 Dataset

The Fabric Dataset

Intelligent Behaviour Understanding Group (iBUG)

Department of Computing, Imperial College London

1.1 Description

The original dataset consists of images of the following classes of fabric samples: acrylic, artificial_fur, artificial_leather, blended, chenille, corduroy, cotton, crepe, denim, felt, fleece, fur, leather, linen, lut, nylon, polyester, satin, silk, suede, terrycloth, utilities, velvet, viscose, wool. Each fabric sample was imaged under four different lighting conditions using a photometric stereo sensor to allow for material classification beyond textural classification [1]. The non-rigid material nature of various fabrics presents an interesting problem: how to classify different fabric types, especially when fabrics may be moving or changing shape. This scenario is especially true in the event the fabric is being imaged on a conveyor belt in the context of producing or recycling textiles. The imaging methods of the dataset under different lighting qualities create opportunities to use reflectance alongside textural identification methods.

Due to the limited number of samples associated with certain fabric types and the samples per category requirements associated with this project, our team had to examine ways to increase the sample sets and eliminate particular categories. We decided to resample each image into 4 sub-images of resolution 200x200 px, representing quadrants of the original image and eliminate those categories which did not contain enough samples to meet assignment requirements. As such, we were left with the following categories and their associated image counts after resampling:

Figure 1- Distribution of Images per Class after Subsampling

Class	Image Count
Blended	6576
Cotton	9408

Denim	2592
Polyester	3616
Wool	1440

The remaining dataset of 23,600 images presents us with challenges related to the size and distribution of samples per fabric type.

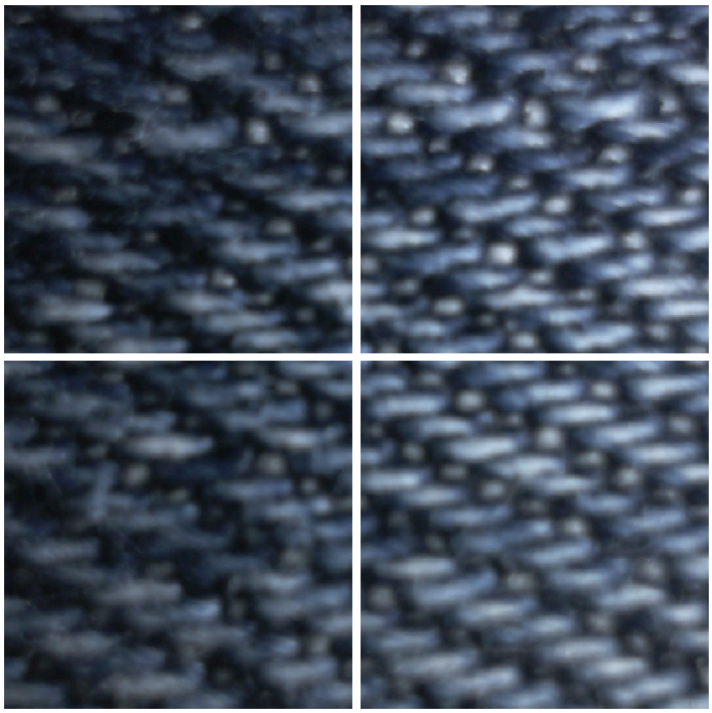
1.2 Research Question

Can we correctly classify textile materials (wool, polyester, cotton, blended, denim) using reflectance and microgeometry?

1.3 Example Images

Figure 2 - Wool_22 image 1a-d (image 1 divided into 4 subimages)

Figure 3 - Denim_1 image 1a-d (image 1 divided into 4 subimages)



2 Preprocessing

To make sure that we had an adequate amount of data, we preprocessed the images as mentioned above. First, we split each image into four 200x200 px subimages. Then we renamed the files to include their material type in the name, creating a unique ID for each image.

2.1 Train-Test Split

To adhere to data science best practices, we split the data into train-test groups immediately after our preprocessing was complete. We performed a stratified split, pulling 80% of each class into a train folder and the other 20% into a test. The test images were not accessed until the end of the project.

Figure 4 - Distribution of Images in Train-Test-Validation

Train	18903 images
Test	4729 images
Total	23632 images

2.2 Data Augmentation

After splitting the data, we augmented the training data by flipping all images in the wool and denim classes horizontally. We did this because the classes were very unbalanced in our original data, with cotton as the largest class. Now, as shown below, the data is more balanced. We attempted to further balance the classes with additional augmentation with vertical flipping but found that the size of the balanced dataset was computationally-unwieldy and abandoned the second phase of augmentation.

Figure 5 - Distribution of Training Images per Class after Subsampling & Augmentation

Class	Start Count	Final Count
Blended	5260	5260
Cotton	7526	7526
Denim	2073	4146
Polyester	2892	2892
Wool	1152	2304
Total	18,903	22,218

2.3 Resizing and balancing train data

As will be explained in detail in section 5.1.3, as part of our feature engineering efforts, we experimented with both scalar and vectorized features. For the vectorized features, we are referring to features which generated a numerical value corresponding to each pixel value in the image, which we then used np.ndarray.flatten to produce a one-dimensional vector. After flattening images to vectorize certain features and perform dimensionality reduction, the train dataset for the vectorized features was of ~ 22,000 x 6000 which was incredibly computational dense. After seeing that the results weren't good, we retried this version with a downsampled and balanced down dataset.

Figure 6 - Distribution of Training Images per Class after resizing and balancing

Class	Start Count	Final Count
Blended	5260	2892
Cotton	7526	2892

Denim	4146	2892
Polyester	2892	2892
Wool	2304	2831
Total	22,218	14,399

3 Feature Extraction

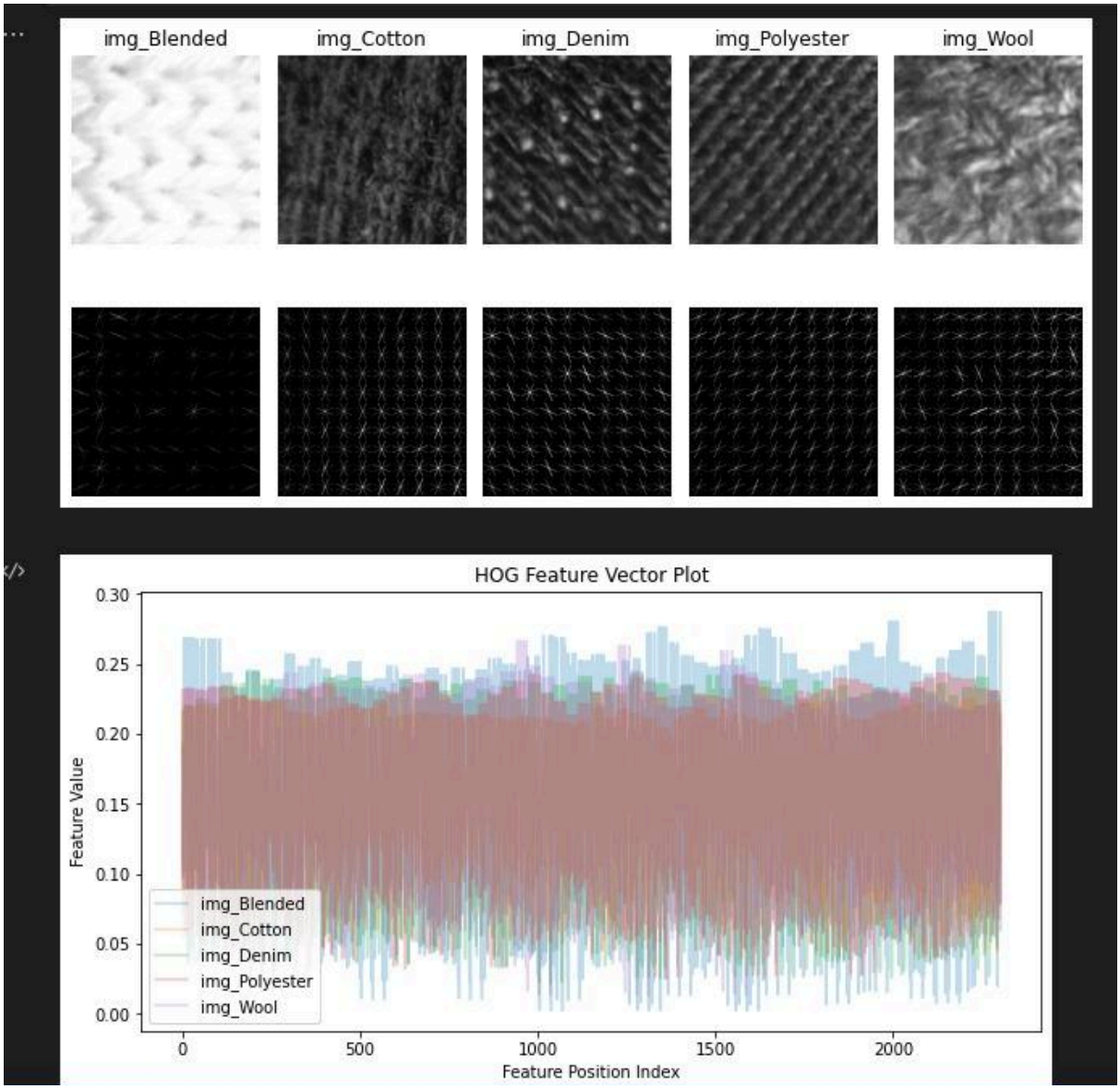
Feature Area	Features [2,3,4,5,6]
Edge Detection and Gradient Features Highlighting transitions in microgeometry and patterns common in machine-woven textiles	• Laplacian of Gaussian (LoG) • Histogram of Oriented Gradients (HOG)
Texture Features Characterizing surface structure and microgeometry	• Normal Maps • Gabor Filters
Frequency Domain Features Identify structures and patterns not easily visible in the spatial domain	• Wavelet Transforms
Statistical Measures Capture distribution and variation in pixel values possibly correlated with microgeometry	• Bag of Visual Words (BoVW)

3.1 Edge Detection and Gradient Features

3.1.1 Histograms of Oriented Gradients (HOG)

Histograms of Oriented Gradients are used to capture the distribution of edge orientations of our samples. This is done through the following process: Normalize the image to [0,1], compute the gradients (edge orientations) at each pixel in the image and divide the image into small, overlapping sections. Finally, a histogram of the gradient orientations is created within each cell and those histograms are concatenated to form the final HOG feature vector.

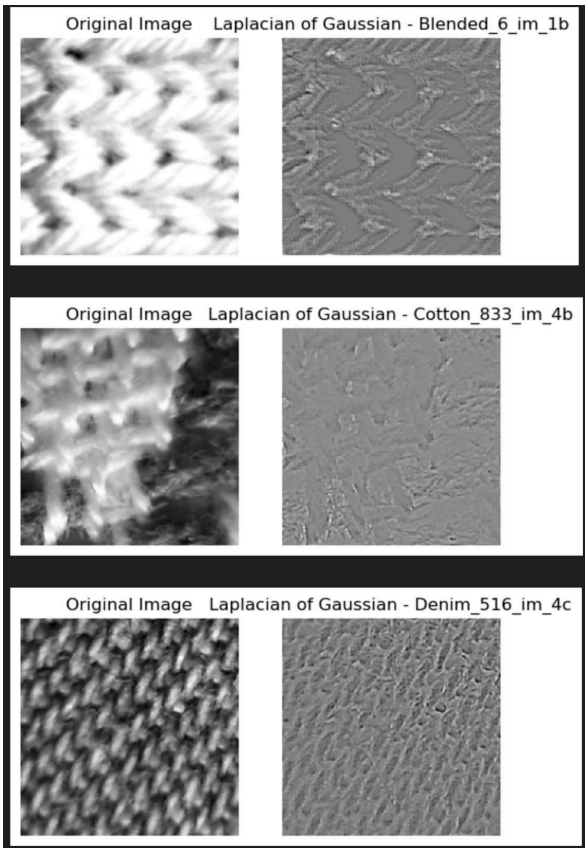
Figure 7 - Histogram of Gaussian Features Visualized



We chose a cell size of 4x4 pixels to create a reasonably sized feature vector, a grid of 50x50 cells, that can be efficiently processed by the model. Larger cell sizes would lead to a smaller feature vector, but may miss important textural details whereas smaller cell sizes would result in a much larger feature vector, which could potentially lead to overfitting or increased computational expense.

3.1.2 Laplacian of Gaussian (LoG)

Figure 8 - LoG Features Visualized



The Laplacian of Gaussian (LoG) feature has been implemented to consider texture features where noise and high-frequency components have been somewhat smoothed out. LoG works by applying a Laplacian operator (second derivative) to a Gaussian function to detect edges irrespective of directions. The Gaussian function is defined by its standard deviation, represented by a sigma value in the implementation. We set the sigma value to 0.7 so it would be more sensitive to finer details and textures in the fabric images. This enabled us to capture the intricate patterns and weaves that likely characterize the different fabrics classes.

What started to become clear with many of our edge-related features, we were picking up on a lot more information about the weave of the fabric. There is likely some correlation between the weave and the type of fabric. For instance, denim is just a very particularly woven and treated type of cotton. Wool largely has a thicker weave, represented by wider distances between the edges due to the nature of wool thread and what can be done with it. So, in some sense, the edge features may characterize a fabric type. However, in instances where the weave can be quite similar, for instance, with blended fabrics and cotton, this information is corollary but not definitely indicative of a particular fabric type.

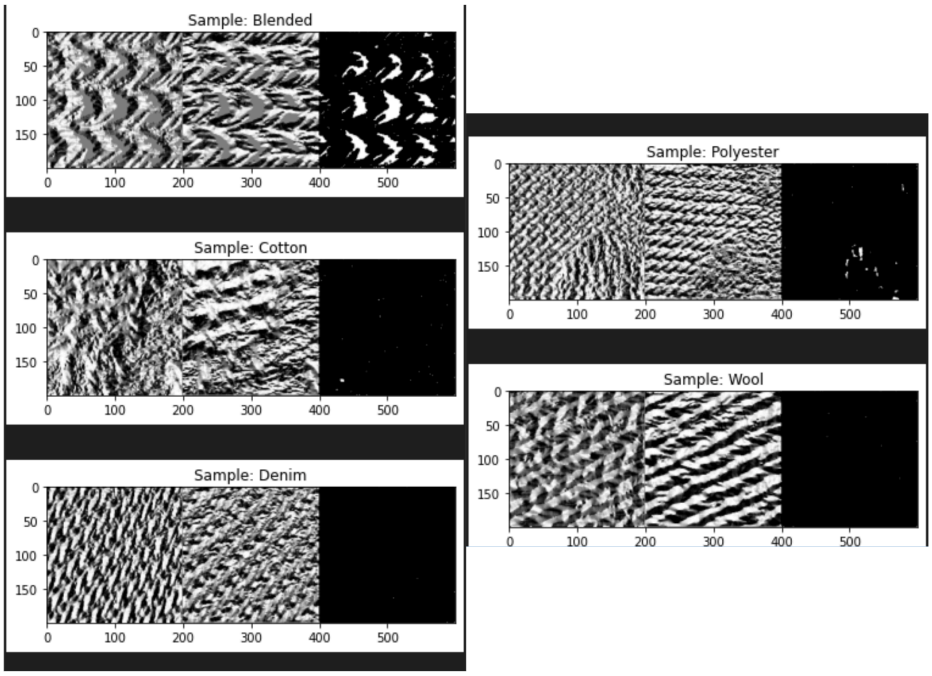
3.2 Texture Features

3.2.1 Normal Maps

Normal maps are useful for capturing fine surface details in images. They can capture information about the 3D shape and structure of the fabric's surface. We expect this feature to be useful because

different fabric types have distinct surface structures that are a result of their manufacturing processes and material properties. This feature pairs really well with the HOG feature as normal vectors encode 3D surface structure and HOG features capture the 2D distribution of edge orientations.

Figure 9 - Normals Features Visualized



In our classification task, texture is a major element of our feature space. Each pixel's surface orientation is calculated through the following process: Images are normalized to [0,1], converted to grayscale and then gradients in the x and y directions are computed using Sobel for edge detection. Next, normal vectors of the image are calculated by dividing the gradients by their Euclidean norm. Finally, the 3D normal vectors are flattened by concatenating the three dimensions and reshaping it into one vector to fit into the feature pipeline. Normal maps are less sensitive to minute changes to lighting which further helps our classification since the fabric samples are imaged under four different lighting conditions and divided into 4 subimages.

3.2.2 Gabor Filters

We've also implemented Gabor Filters, linear filters that are designed to respond to specific spatial frequencies and orientations within an image. Gabor filters use localized, oriented, and frequency-selective receptive fields much like the human visual system.

To create the filter, a sinusoidal wave is modulated by a Gaussian envelope. The filter is characterized by parameters such as the spatial frequency, orientation, and bandwidth. We used a filter with a spatial frequency parameter of 0.6: we wanted a response to medium spatial frequencies, somewhere between fine and coarse details. We set the orientation of the Gabor filter to the horizontal axis:

patterns that are horizontally-aligned will be detected more so than vertically-aligned. We chose this orientation as we were augmenting primarily over the horizontal axis, the directional sensitivity of the Gabor filter is less-significant. Finally, we set the sigma value for both x and y directions to 1 to expand the Gaussian kernel to have a broader spatial support and be responsive to a wider range of spatial frequencies.

To extract Gabor features from the images, each image was convolved with each of the Gabor filters in the filter bank. This results in a set of filtered images, where each image represents the response of the image to a specific spatial frequency and orientation.

3.3 Frequency Domain Features

3.3.1 Wavelet Transforms

Wavelet transforms were also utilized by representing images using a set of basis functions called wavelets, localized in both the spatial and frequency domains. This allows wavelet transforms to capture both local and global features of the input data.

We implemented the feature by applying a 2D wavelet transform to the input fabric image, decomposing it into multiple frequency subbands. Then we extracted statistical features (mean, variance, skewness, kurtosis) from the wavelet coefficients in each subband, creating a feature vector that represents the fabric texture. The statistical features computed from the wavelet coefficients provided a concise yet informative representation of the fabric texture.

3.4 Statistical Measures

3.4.1 Bag of Visual Words with SIFT

For our complex feature, we've implemented Bag of Visual Words (BoVW) with Scale-Invariant Feature Transform (SIFT). BoVW is useful for this classification task because it is relatively insensitive to the spatial arrangement of the visual features within the image, as it only considers the frequency of occurrence of the visual words. Additionally, SIFT detects distinctive interest points in the image that are invariant to scale and rotation. It is adept at capturing the textural properties of the fabric, such as the weave patterns, fiber structures, and surface irregularities.

Our process began with extracting distinctive, scale and rotation-invariant keypoints from a collection of images using the sift library SIFT, where each keypoint was described by a robust 128-dimensional descriptor capturing local gradient information. These descriptors were then clustered across the entire training set using K-means to create a visual vocabulary, with each cluster centroid forming a

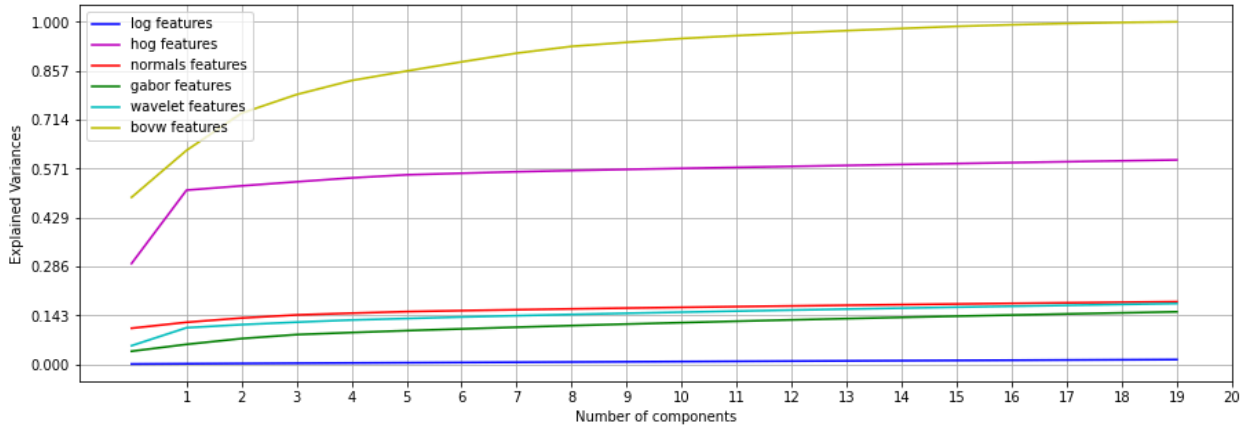
'visual word' that represents common patterns identified across all images. This vocabulary allowed for the quantization of feature descriptors in new images, whereby each SIFT descriptor is assigned to the nearest visual word, and the image is subsequently represented as a histogram of these visual word occurrences, effectively transforming the image into a fixed-length feature vector.

For K-Means, we used a cluster parameter of 20 because we wanted a codebook (number of clusters) that was not too coarse but there are some signs of overfitting. Based on differences between model performance on test/train in the models in the context of the scalar dataset where the BOVW features represented ~65% of the feature dimensions, it is possible that the vocabulary was too descriptive of the training set. We ran out of time to further explore this feature, however, in the future, performing cross-validation based hyperparameter tuning for the BOVW feature would have likely resulted in a smaller k and a more generalizable visual vocabulary.

4 PCA

We used Principal Component Analysis (PCA) to detect which features were most useful for our classification task.

Figure 10 - PCA for Feature Vectors



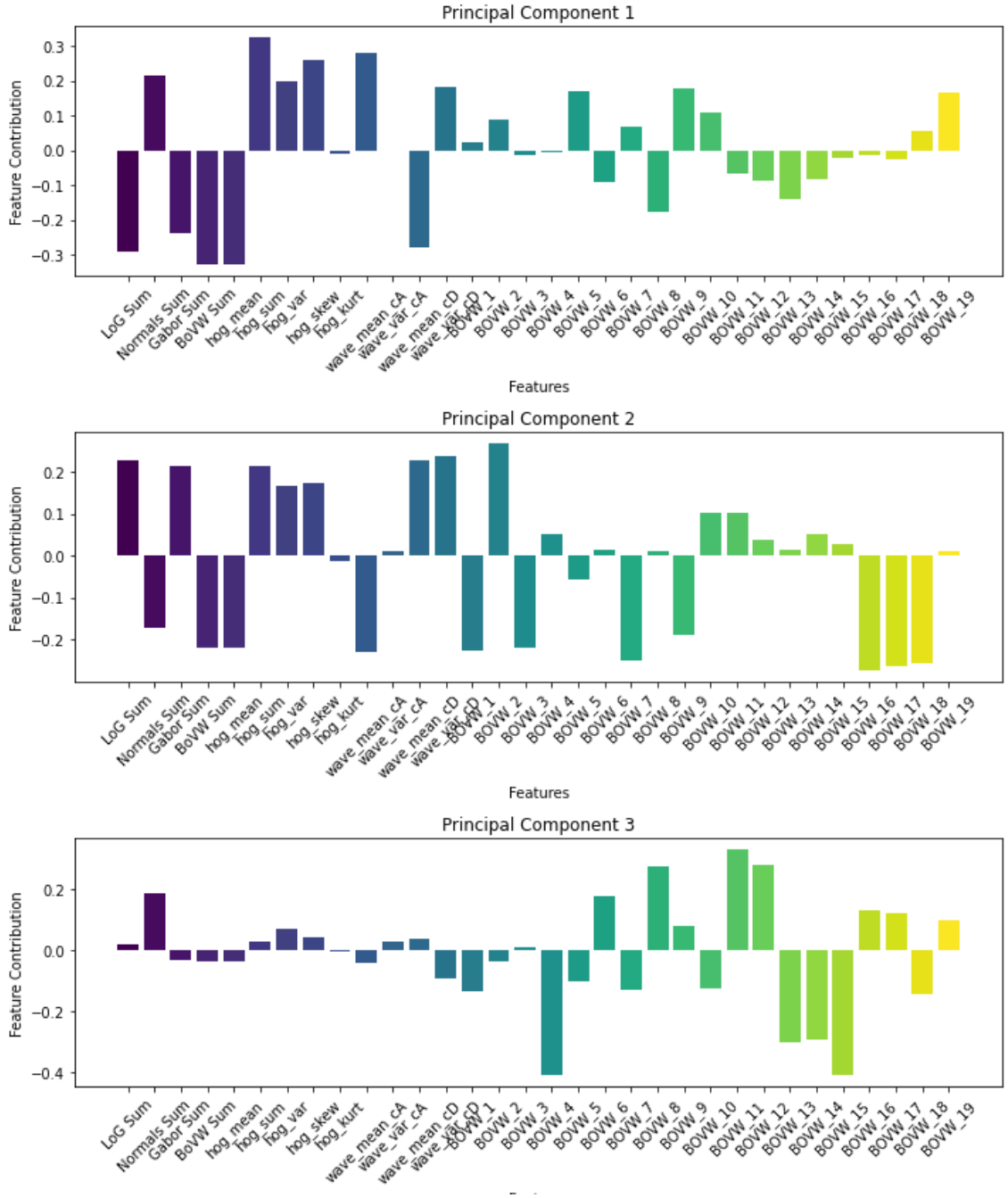
The results show that HOG and BoVW explain most of the variance of our dataset. We used this analysis to decide how to transform the train dataset to run the classifier according to which features were most useful.

From the dimensionality reduction tSNE wasn't very clear on which features were more explanatory nor PCA, so we decided to follow the Explained Variances graphs to set our features for the vectorized model. (See Appendix for images)

Since we were also analyzing the case of building a classifier with scalars, we performed an additional PCA where we wanted to see for each component what scalar was contributing the most (Figure 8).

Finally, one of our versions of the classifier with feature vectors used dimensionality reduction to reduce the number of columns from ~60,000 to ~6,000 columns with BoVW, Normals and HoG features.

Figure 11 - PCA for Scalars



5 Classification

This classification problem bore several challenges – we were starting with an imbalanced dataset, oftentimes fabric weave and content similarities across fabric groups result in decision boundaries that are difficult to place, and vectorized features proved to be incredibly computationally intensive.

We started by implementing a support vector machine, which was run with both scalar and vector features. We then moved on to train a random forest model. In both cases, the classes were left imbalanced and in the case of random forest, the vectorized feature set, which was questionably heavy with the SVM, became untenable with the decision tree. It was at this stage that we decided to abandon the vectorized features in the exploration of a balanced random forest and XGBoost in the interest of producing results in <24 hours.

Each classifier exploration via grid-search cross validation with five folds can be found in its respective notebook (see the GitHub README.md[1]). The cross-validation was conducted with verbose=3 to ensure ample data was produced from within each fold as the process occurred. Notes on each model are included in the notebooks with findings summarized below.

5.1 Support Vector Machine

We first started with a support vector machine in two forms: scalar with all our features and with select vector features.

5.1.1 Scalar Features & Tuning

For this classifier, we implemented scalar representations of HOG (mean, sum, var, skewness, kurtosis), Wavelet (cA mean, cA variance, cD mean, cD variance), LoG (sum), Gabor (sum), Normals (sum), and bovw_feature_vector. The bovw_feature_vector for each image was a 20x1 vector produced by the extract_bovw_features function in the feature_utils.py file in the utils folder on GitHub. It represents a histogram of visual word occurrences for an image, where each entry in the vector corresponds to the normalized frequency of a particular 'visual word' within the image. These visual words are derived from key local features identified by SIFT and quantized using a pre-trained k-means clustering model, which for both the training and test sets was only instantiated and trained on the training data. The cross-validation and results code can be found in the notebook 3_svm.ipynb on the GitHub linked before

[1] https://github.com/elliswmartin/fabric

After performing cross validation, we found that these were the best parameters:

Figure 12 - Best Hyperparameters SVM for Scalar Dataset

Best Parameters*	Cross-Validation	Implications/Interpretation
C	10	The C parameter in an SVM serves as a regularization parameter, influencing the trade-off between achieving a low error on the training data and minimizing the model complexity for better generalization to new data. Ten is a pretty high regularization team, meaning the SVM model works to classify all training examples correctly by giving a smaller margin. This can lead to a lower bias but potentially higher variance, increasing the risk of overfitting if the data is noisy or if the dataset is not large enough to justify a complex model, which may very well be the case in the context of our dataset, especially for those sparse classes like wool or denim.
gamma	scale	The gamma parameter defines the influence of individual training samples on the learning process. It determines the radius of influence of the support vectors in the Radial Basis Function (RBF) kernel and affects the decision boundary. When set to scale, the setting automatically adjusts gamma to be inversely proportional to the number of features in the dataset (1 / (number_of_features * X.var())). This can be particularly useful for feature normalization, ensuring that gamma is scaled appropriately across datasets of varying feature scales and distributions. It typically helps in achieving a balance where each feature contributes appropriately without dominating the kernel's behavior.
kernel	rbf	The kernel parameter specifies the type of kernel function to be used in the SVM, transforming the input data into a higher-dimensional space where a hyperplane can be used to separate classes. The RBF kernel can model complex, non-linear decision

		boundaries. The kernel's effectiveness depends significantly on the gamma parameter. An RBF kernel with a suitable gamma and C can effectively handle situations where the relationship between class labels and attributes is nonlinear.

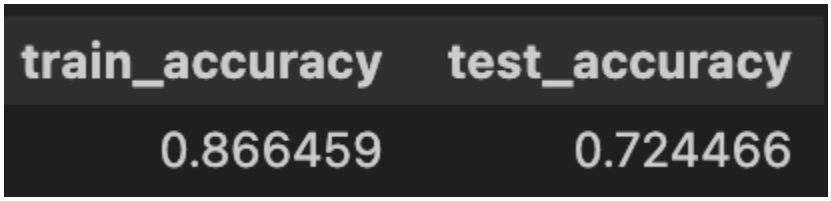
* - best for SVM with vectorized features too

Based on the above suggested parameters produced by the grid-search cv, (C=10, gamma=scale, kernel=rbf), this SVM model is configured to use a complex model (due to higher C), aiming to classify the training data as accurately as possible while risking overfitting. Additionally, employing an RBF kernel that can handle non-linear patterns in the data, with gamma automatically adjusted to suit the scale of the data, serves to counterbalance the high C as it provides a more robust approach across datasets due to the scaling of gamma.

Within the folds of the cross-validation, which can be seen in the notebook, we see a significant amount of instability associated with the performance of this model. Generally this points to possible overfitting to the training set and a lack of generalizability.

5.1.2 Scalar Performance

Figure 13 - SVM w/ Scalar Feature Performance

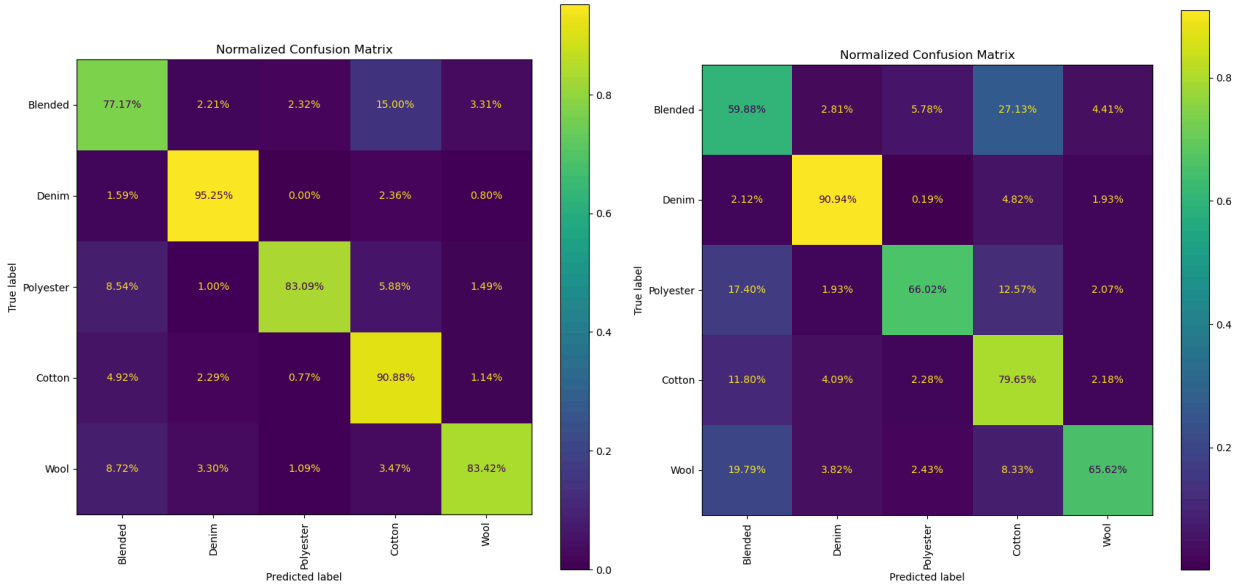


On the training set, this model hit ~86.6% accuracy, which dropped to ~ a 72.4% accuracy on the holdout set. A comparison of its training time, inference time, CPU usage and memory usage are compared in section 6, the conclusion.

The scalar SVM model performed the best at classifying denim and cotton when purely looking at the accuracy scores as a percentage (see confusion matrices below). The next best were polyester and wool, and finally blended classification was the worst performing for this model. We find in this model a couple of classification challenges that recur throughout most of our models: differentiating between cotton and blended, and differentiating between cotton and polyester. We hypothesize that these classes are not as accurately classified for three reasons: blended & polyester fabric may have

the fabric structures and other qualities that are very close to cotton, cotton is overrepresented in the dataset so the model may skew towards classifying the largest class, and the underrepresentation of the other classes may mean that edge cases confuse results / there are just less good, clear class examples.

Figure 14 - Confusion Matrices for Scalar SVM Train (Left) and Test Data (Right)



5.1.3 Vector Features & Tuning

For this classifier, we first chose to do it with BoVW, HoG and Normals vectors concatenated so each row represents the features of each image. We did three trials of this:

a. *Full dataset with dimensionality reduction:* after leaving a dataset of 22,218 rows and 6831 features (columns). We performed the grid search with the SVM to check which parameters were best for this case. The code ran for more than 10 hours, taking approximately 3 hours for each cross-validation. For computational simplicity, we discarded this alternative.

b. *Resized and balanced dataset with dimensionality reduction:* With the same features, we now created a dataset of 14,398 images and 5,657 features. The grid search for this case and took 8 hours to run and had bad results.

c. *Downsampled data with only two features:* We finally vectorized features slightly different from the two previous options, and we only kept BoVW and HoG since these features explained more of the dataset's variance. To have a balanced train set, we balanced down to the class with fewer images leaving a train dataset of 11,520 rows and 2,323 features. We performed the grid search to find the best parameters once more.

5.1.4 Vector Performance

Figure 15 - Option B

Train Accuracy	85%
best cross-validation	42%
held-out validation	47%
runtime	8 hours
Test Accuracy	39.5%
runtime	15 minutes

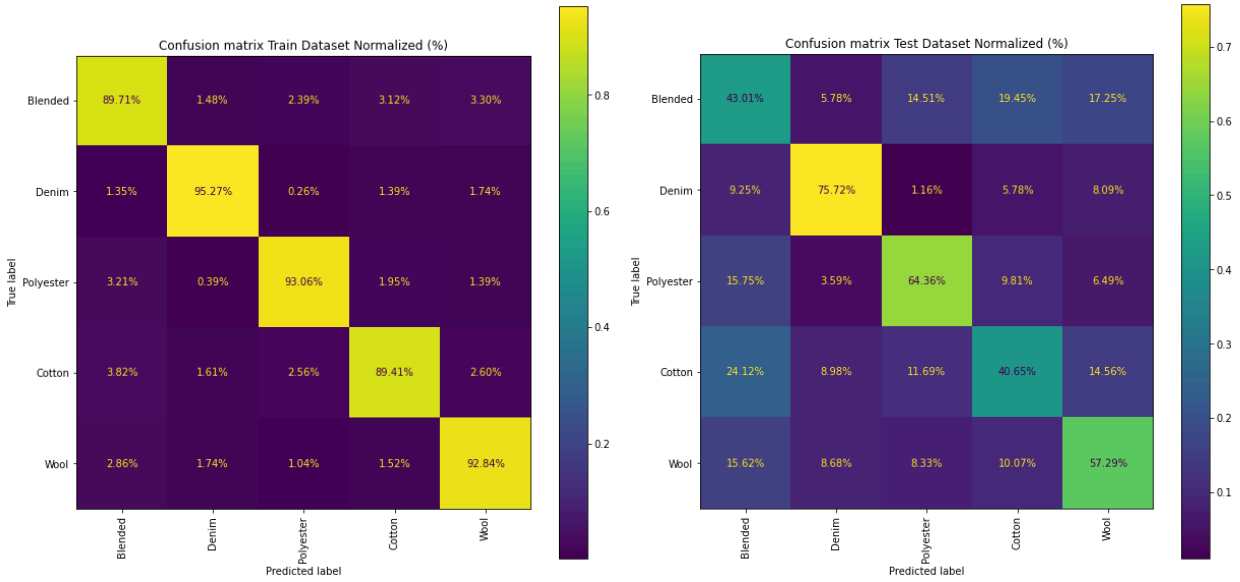
Figure 16 - Option C

Train Accuracy	92.1%
best cross-validation	56%
held-out validation	60%
runtime	3 hours
Test Accuracy	49.8%
runtime	63 seconds

Our SVM classifier is overfitting the train data dramatically, showing that our features do not explain the difference between the five fabrics we are analyzing. From the confusion matrix, we can see that it is doing really well with denim, which has a very standard weave as shown in the HOG feature, but it performs poorly for blended fabrics and cotton mixing each other. Our hypotheses:

a. The features selected are good to see weave (texture) but have trouble identifying other characteristics.
b. SVM classifier is good with high-dimensionality datasets and tends to overfit when there aren't too many dimensions.
c. The SVM with vector features performs relatively well on the polyester fabric when compared to the random forest

Figure 17 Confusion Matrices for Vector SVM Train (Left) and Test Data (Right)



5.2 Random Forest - Two Models

5.2.1 Random Forest 1 - Standard Features & Tuning

The next classifier we trained was a standard random forest model, using the same scalar dataset and cross validation strategy using grid search indicated in section 5.1.1. This meant implementing scalar representations of HOG (mean, sum, var, skewness, kurtosis), Wavelet (cA mean, cA variance, cD mean, cD variance), LoG (sum), Gabor (sum), Normals (sum), and bovw_feature_vector. Random Forest, as is true with most decision tree models, is significantly more memory-intensive and computationally intense than the SVM we started with. We originally tried to also implement this model with the vectorized feature dataset described in section 5.1.3 but it took nearly 18 hours to finish only 20% of the cross-validation and it was continuing to run slower and slower over time. It also was achieving terrible accuracy per the cross-validation performance log produced while running, so we decided to axe it and just run the rest of the race using the scalar dataset.

The cross validation results and interpretation can be found in 3_random_forest.ipynb.

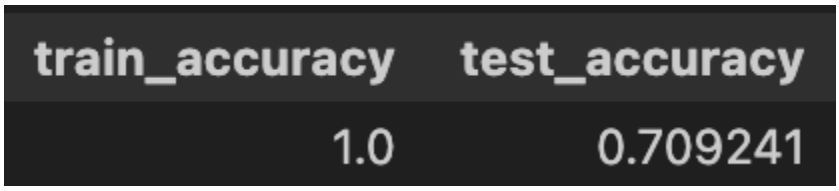
Figure 18 - Best Hyperparameters - Random Forest Imbalanced

Best Parameters	Cross-Validation	Implications/Interpretation
max depth	30	The maximum depth of each tree in the forest, which controls the complexity of the learned models, with deeper trees potentially capturing more detailed data patterns but also risking overfitting if the depth is too excessive for the data. A maximum depth of 30 means that each tree in the forest can have up to 30 splits in its decision nodes.
max features	sqrt	The number of features to consider when looking for the best split at each node. Setting this parameter to sqrt means that the maximum number of features considered at each split will be the square root of the total number of features, which adds randomness to the model making process, promoting model diversity in the ensemble, which can lead to a more robust overall model.
min samples	5	The minimum number of samples required to split an internal node in a tree, meaning at least two samples are required to justify creating a new decision node. This is the smallest number for splitting, allowing the trees to be quite detailed, which increases the risk of overfitting.
n estimators	300	The number of trees in the forest. More trees increases the performance of the model at the cost of increased computational expense. A forest with 300 trees adds robustness to the model, as more trees in the ensemble reduce the variance of the output by averaging more individual model predictions.

The configuration of these hyperparameters intend to produce a robust Random Forest model that has the potential to model complex patterns in the data due to the relatively high maximum depth of the trees, can reduce overfitting despite the deeper trees by using a relatively large number of estimators (trees in the forest) and by considering a limited, randomized subset of features at each split.

5.2.2 Random Forest 1 - Standard Performance

Figure 19 - Random Forest Performance



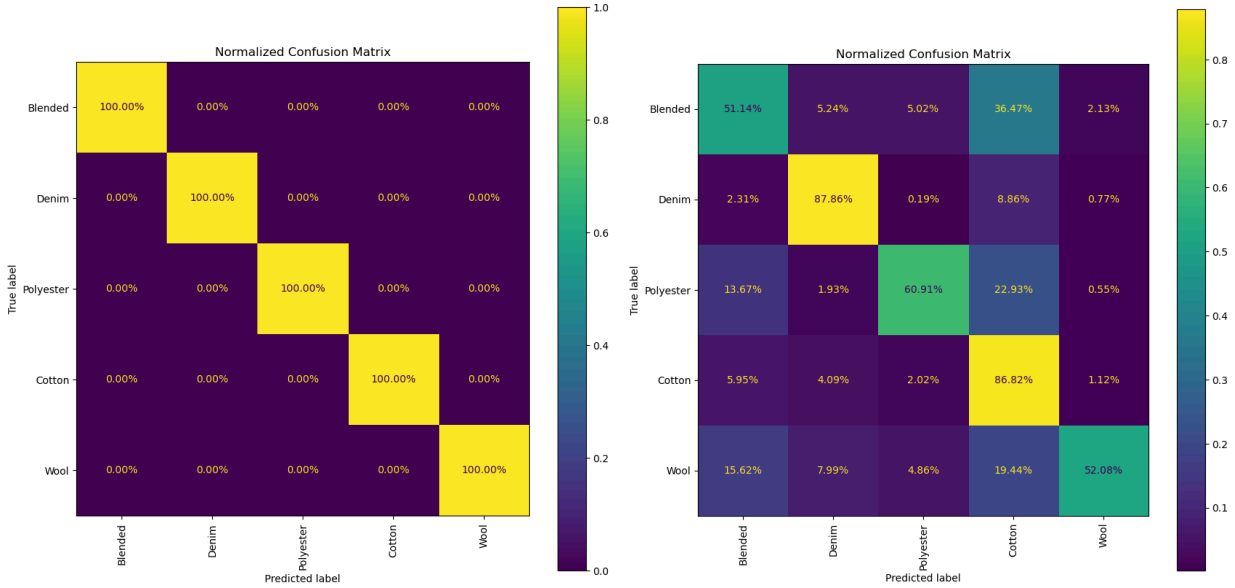
train_accuracy	test_accuracy
1.0	0.709241

Unfortunately, it seems that the parameters selected by our cross-validation round did not effectively control for overfitting, which can be seen in the massive drop in accuracy on the hold-out set. Likely the tree was a little too detailed and wound up 'memorizing' the features associated with very particular fabric samples, meaning that it didn't actually learn and could not generalize well to the test set. 100% Accuracy on the training set should always leave some amount of suspicion in these circumstances.

As seen in the left confusion matrix in fig. 9, the model performed extremely well on the training set. The performance difference between the train and test sets paired with the 100% accuracy on the train set leads us to believe that this model is overfitting on the training data and therefore does not generalize to the test data well. The best performance in the test data again occurs around denim and cotton classification, with the blended and cotton, and cotton and polyester confusion present in this model as seen in previous models as well. Overall this model performs worse than the scalar SVM and better than the feature SVM. A pattern is also beginning to emerge, as it seems that the denim is well described by the features with very few misclassification and high accuracy scores.

This model does however significantly reduce most false negatives and false positives across classes when compared to the SVM, as evidenced by the upper and lower triangles in the test confusion matrix in Figure 20 on the left. This comes with the tradeoff that more instances of cotton appear to have been misclassified.

Figure 20 - Confusion Matrices for Standard Random Forest Train (Left) and Test Data (Right)



5.2.3 Random Forest 2 - Weighted Features & Tuning

For the second version of our random forest classifier, we used the exact same scalar features seen in 5.2.1. This time, we used the balanced class parameter in an effort to account for the serious imbalances in our data, with cotton dominating over other classes. This parameters implements a weighted strategy to properly weight the training data and curtail any disparate impact the imbalances may have. By running this cross-validation and evaluating the results separately, we were also able to compare the impact that this strategy for tackling the imbalances had on the classification results.

There are two types of weighting offered as built-ins on sklearn - balanced and balanced subsample. In cross-validation, balanced subsample won out, which makes sense as it tends to perform better when working to manage imbalances between classes.

The cross-validation results can be found in the 3_RF-BalancedWeighting.ipynb notebook on the repository and featured a similar amount of stability to the stability seen within the random forest model.

Figure 21 - Best Hyperparameters - Random Forest Imbalanced

Best Parameters	Cross-Validation	Implications/Interpretation
max depth	none	The maximum depth of each tree is not restricted,

		meaning there is no limit to how deep the tree can grow. Nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. By allowing the trees to grow without restriction, we can capture highly complex patterns in our model. This setting raises the risk of overfitting, particularly if not controlled by other parameters.
max features	sqrt	The number of features to consider when looking for the best split at each node. Setting this parameter to sqrt means that the maximum number of features considered at each split will be the square root of the total number of features, which adds randomness to the model making process, promoting model diversity in the ensemble, which can lead to a more robust overall model.
min samples	5	The minimum number of samples required to split an internal node in a tree, meaning at least two samples are required to justify creating a new decision node. This is the smallest number for splitting, allowing the trees to be quite detailed, which increases the risk of overfitting.
n estimators	200	The number of trees in the forest. More trees increases the performance of the model at the cost of increased computational expense. A forest with 200 trees adds robustness to the model, as more trees in the ensemble reduce the variance of the output by averaging more individual model predictions.
class weight	balanced subsample	This adjusts the weights of the classes inversely proportional to their frequencies in the input data for each bootstrap sample used to train individual trees within the forest. Balanced sample helps the model address class imbalance more dynamically by recalculating class weights for each tree, thus potentially improving performance on minority classes which might otherwise be overlooked (e.g. wool)

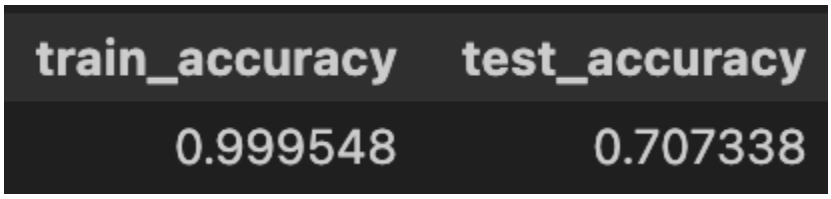
Upon further reflection, it may have been good to simply run the model with the exact same parameters as those used by Random Forest 1 to isolate the impact to the balanced classes parameter,

however this approach allowed us to optimize the model under the auspices that the data was imbalanced and we would be invoking this parameter. As it turned out, the parameters selected were different when invoking balanced sets of training data.

Overall, we produced a model that is more robust to imbalance due to balanced_subsample ensuring the model is more equitable across classes. The suggested hyperparameters produced potentially deep and complex trees with max_depth set to None, however this is somewhat mitigated by min_samples_split being set to 5. By implementing sqrt for max_features and a moderate number of trees (200), we foster diversity among the trees in the model, enhancing its generalization capability.

5.2.4 Random Forest 2 - Performance

Figure 22 - Best Hyperparameters - Random Forest Imbalanced

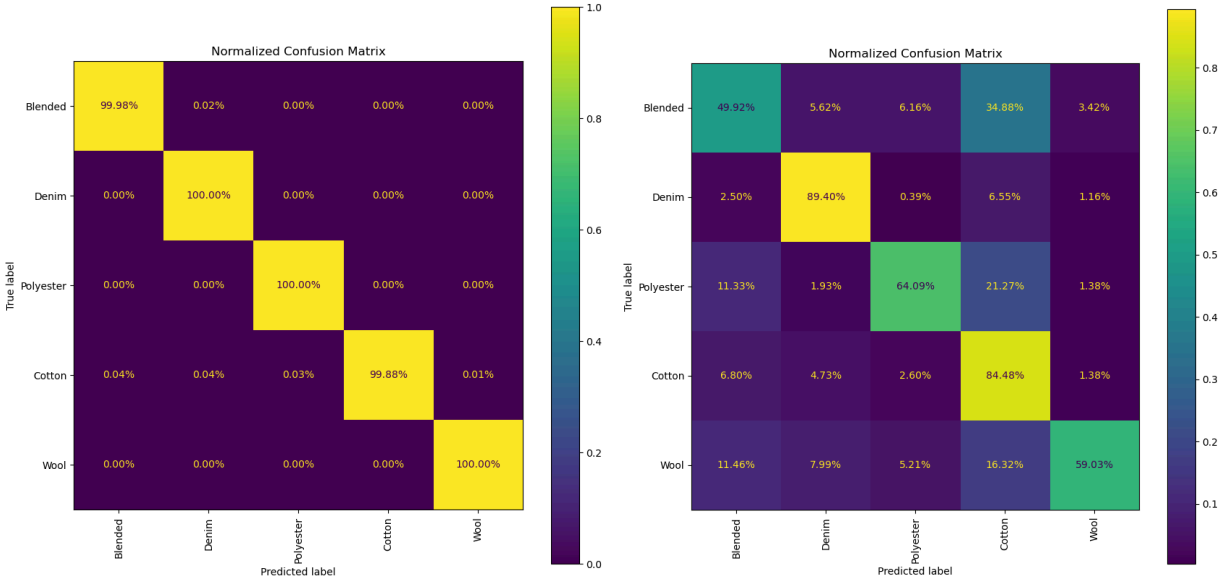


train_accuracy	test_accuracy
0.999548	0.707338

Overall the accuracy decreased slightly, meaning the model wasn't fitting the training data quite so perfectly, which is a good thing in this case, as 100% accuracy is a little too good to be true. That said, 99.9% is still not exactly what we want. There was also a decrease in the test accuracy. The really important information springs from looking at the confusion matrices and comparing test set performance between the two models

To compare to the next best performing model seen so far, SVM scalar, the WRF performs similarly in terms of correctly classifying polyester and denim. It performs better than scalar SVM at classifying cotton but worse than scalar SVM at classifying blended and wool. The key here is looking at the cross section of chart representing cotton against other fabrics. The number of times the model mislabeled blended fabrics as cotton has decreased by several percentage points. This is also true with polyester. Although the model is performing worse overall, the balancing has allowed for less confusion and automatic categorization of certain blended and polyester samples as cotton just due to its heavy presence in the training data.

Figure 23 Confusion Matrices for Weighted Random Forest Train (Left) and Test Data (Right)



5.3 XGBoost Classifier: Optimized for Accuracy

Finally, we experimented with an XGBoost classifier leveraging Synthetic Minority Over-sampling TEchnique (SMOTE) due to the promising results of the balancing and weight manipulation seen in the results of the random forest. SMOTE was leveraged due to its ease of implementation via the imblearn library, as the XGBoost library does not include a balanced subsample parameter unfortunately.

5.3.1 Features & Tuning

For this classifier, we implemented scalar representations of HOG (mean, sum, var, skewness, kurtosis), Wavelet (cA mean, cA variance, cD mean, cD variance), LoG (sum), Gabor (sum), Normals (sum), and bovw_feature_vector – the scalar feature set which has been used for all classifiers apart from the SVM where we implemented the vectorized features.

Again, we implemented a cross-validation pipeline to tune the hyperparameters before testing what was deemed to be the optimal model on both test and training data. The results of this cross validation exercise were quite impressive, achieving high accuracies in the range of 80-90% much more frequently than seen with other models. These results can be seen in the notebook 3_XG-Boost.ipynb. Overall, the results of the cross validation suggested a gap between the average accuracy and that experienced by the hold out set that was much greater than seen with other cross validation exercises and the inner-fold behavior was extremely volatile. That said, given the extremely high accuracy, we were hopeful that even if the model overfit, it still might perform better on the test set than the other models.

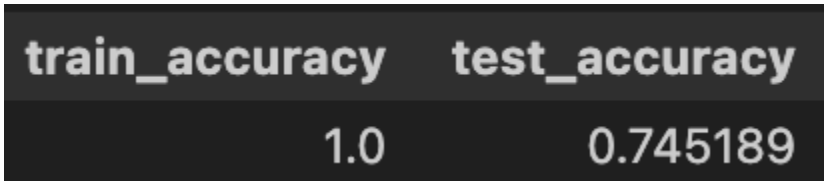
Figure 24 Best Features for XGBoost Algorithm

Best Parameters	Cross-Validation	Implications/Interpretation
col sample by tree	0.7	This sets the fraction of features (columns) to be randomly sampled for each tree, meaning 70% of the features are selected randomly from the total features each time a new tree is built during the training process. By reducing the number of features used in each tree, the model is more robust to overfitting and also speeds up training.
learning rate	0.2	Learning rate scales the contribution of each new tree added to the model, meaning each tree's contribution to the final model is reduced to 20% of its predictions. A relatively higher learning rate (like 0.2) speeds up the learning process, but setting it too high can lead the model to converge too quickly to a suboptimal solution.
max depth	9	This parameter determines the maximum depth of each tree, allowing for 9 levels of splitting in each tree. The higher this is, the more complex of patterns the model learns.
min child weight	1	The minimum sum of instance weight (hessian) needed in a child leaf. By default, it is set to 1, meaning that leaves need to have at least one instance in them. Using the base value gives the algorithm more freedom to make decisions involving fewer samples, increasing model complexity but potentially also noise.
n estimators	300	Specifies the number of trees in the ensemble, with 300 trees being built before the boosting process stops. More trees in the model can better capture complex patterns and improve performance on varied datasets, but too many trees can lead to overfitting.
subsample	1.0	Controls the fraction of the training data to be randomly sampled for each tree. Here, 1.0 means 100% of the data is used for each tree, implying no subsampling.

Overall, this model is configured to be robust enough to model complex datasets with considerable depth and numerous trees while incorporating strategies to maintain generalization through controlled sampling of features and a balanced learning rate. It does this by leveraging complexity and depth (with 9 capturing intricate patterns, 300 estimators, trained on the full subsample of data), regularization and control of overfitting (colsample_bytree of 0.7 and a learning_rate of 0.2) help to mitigate the risk of overfitting.

5.3.2 Performance

Figure 25 Train + Test Accuracy for the XG Boost Algorithm

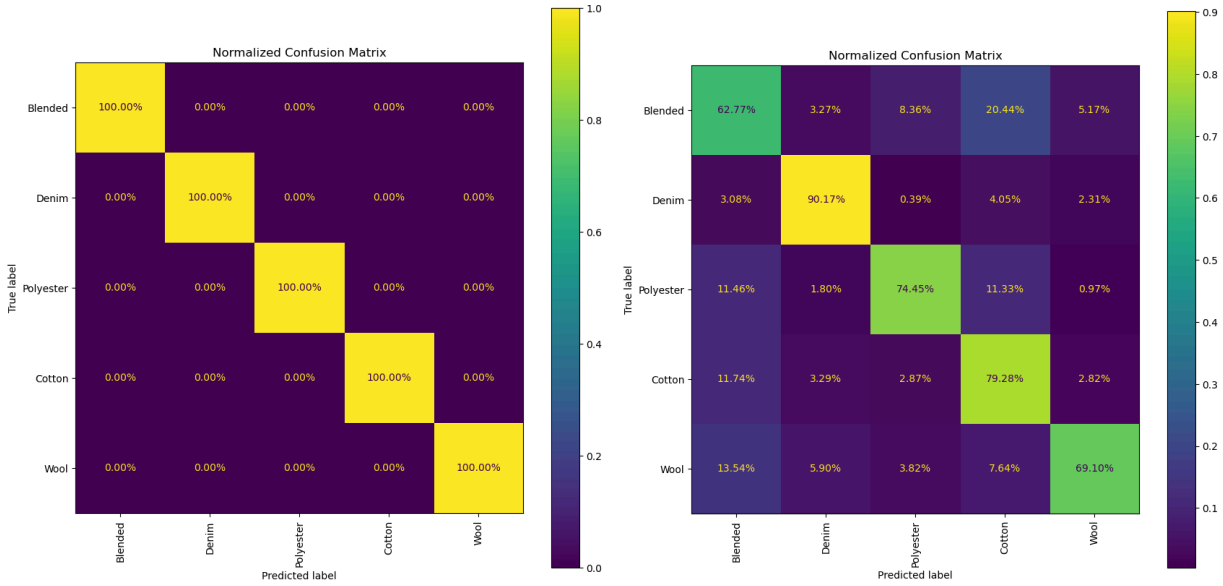


As mentioned with previous models, the disparity in train and test data performance paired with the at or near 100% accuracy on the train data leads us to believe that this model is overfitting on the train data and therefore does not generalize well on the test data, however it still performs several percentage points higher on the hold out set than seen with any other model.

To compare to the next best performing models, XGB performs within 1% point to the same value (denim, cotton) or better than (blended, polyester, wool) scalar SVM at all of the correct classifications. It outperformed scalar SVM in polyester by a large percent (74% versus 66%). Similarly, it misclassified blended and polyester as cotton less than scalar SVM.

XGB outperformed weighted random forest in all categories except for cotton which it classified correctly 79% of the time whereas weighted random forest classified cotton correctly 84%. Overall, a closer examination of the confusion matrices seen in figure 26 shows that the lost accuracy in this category for XGBoost has to do with the algorithm more frequently classifying cotton as blended garments. As previously mentioned, there are physical similarities in these fabric types in the real world. In fact, most blended garments contain cotton in their mix - so the decision boundary is hard to draw. That said, it does point to the effectiveness of SMOTE as an approach to ensuring balance across classes as cotton is the dominant class in this case. This is also explored in a bit more detail in the final section where the precision and recall scores of these two models are compared.

Figure 26 Confusion Matrices for XGBoost Train (Left) and Test (Right) Data



6 Interpretations + Limitations + Conclusions

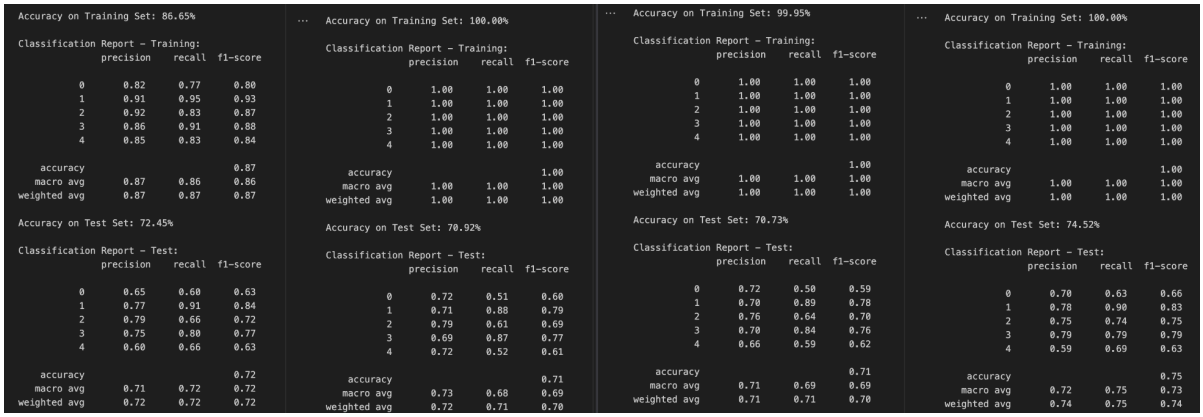
In both figures below, we find data produced in the notebook titled 3_Final_models.ipynb on the repo. This notebook was used to run all of the models with the optimum parameters generated via cross validation on both training and the true hold out set for our data. This notebook also contains the majority of the confusion matrices shown in Section 5.

Overall, the most bang for your buck seems to come from the support vector machine using scalar features. This model was built on unbalanced data and might produce better results were it trained using SMOTE data, but the fact that it wasn't saved cost in terms of CPU during the training process for this model. It was less computationally intensive than the XGBoost. It also showed the least signs of overfitting to the training data, with reasonable performance metrics produced in the 80% range for the training dataset.

This is an important thing to note about Figure 28 – one *must* consider both the CPU and memory, not just the time, when considering computational efficiency. We ran this on the most powerful machine in the group and it proves disparate results when just looking at the training and inference time for each model. This is especially true when comparing SVM which hails from sklearn and xgboost, as they may leverage different techniques as they relate to parallelizing and optimizing performance.

Examining the memory before and after the training as well as the CPU shows that while the XGBoost seems to take less time, it is actually more computationally intensive. For this reason, we selected the XGBoost as our most accurate model, but acknowledge that it does not balance accuracy with computational intensity as well as the SVM well.

Figure 27 Precision + Recall + Accuracy Across Models - SVM (left); Standard RF (left-center); Weighted RF (right-center); XGBoost (right)



Accuracy on Training Set: 86.65%

Classification Report - Training:

	precision	recall	f1-score
0	0.82	0.77	0.80
1	0.91	0.95	0.93
2	0.92	0.83	0.87
3	0.86	0.91	0.88
4	0.85	0.83	0.84
accuracy			0.87
macro avg	0.87	0.86	0.86
weighted avg	0.87	0.87	0.87

Accuracy on Test Set: 72.45%

Classification Report - Test:

	precision	recall	f1-score
0	0.65	0.60	0.63
1	0.77	0.91	0.84
2	0.79	0.66	0.72
3	0.75	0.80	0.77
4	0.60	0.66	0.63
accuracy			0.72
macro avg	0.71	0.72	0.72
weighted avg	0.72	0.72	0.72

Accuracy on Training Set: 100.00%

Classification Report - Training:

	precision	recall	f1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

Accuracy on Test Set: 70.92%

Classification Report - Test:

	precision	recall	f1-score
0	0.72	0.51	0.60
1	0.71	0.88	0.79
2	0.79	0.61	0.69
3	0.69	0.87	0.77
4	0.72	0.52	0.61
accuracy			0.71
macro avg	0.73	0.68	0.69
weighted avg	0.72	0.71	0.70

Accuracy on Training Set: 99.95%

Classification Report - Training:

	precision	recall	f1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

Accuracy on Test Set: 70.73%

Classification Report - Test:

	precision	recall	f1-score
0	0.72	0.50	0.59
1	0.70	0.89	0.78
2	0.76	0.64	0.70
3	0.70	0.84	0.76
4	0.66	0.59	0.62
accuracy			0.71
macro avg	0.71	0.69	0.69
weighted avg	0.71	0.71	0.70

Accuracy on Training Set: 100.00%

Classification Report - Training:

	precision	recall	f1-score
0	1.00	1.00	1.00
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	1.00	1.00
4	1.00	1.00	1.00
accuracy			1.00
macro avg	1.00	1.00	1.00
weighted avg	1.00	1.00	1.00

Accuracy on Test Set: 74.52%

Classification Report - Test:

	precision	recall	f1-score
0	0.70	0.63	0.66
1	0.78	0.90	0.83
2	0.75	0.74	0.75
3	0.79	0.79	0.79
4	0.59	0.69	0.63
accuracy			0.75
macro avg	0.72	0.75	0.73
weighted avg	0.74	0.75	0.74

Figure 28 Time + Computation + Accuracy Across Models

	Model	mem_pre_train	cpu_pre_train	mem_post_train	cpu_post_train	mem_post_inference	cpu_post_inference	training_time	inference_time	train_accuracy	test_accuracy
0	svm	70.5	29.7	69.4	7.1	69.4	5.0	52.743610	4.061114	0.866459	0.724466
1	random_forest	70.0	9.9	69.8	9.8	69.8	7.0	21.359502	0.245032	1.000000	0.709241
2	weighted_random_forest	69.6	10.0	70.2	4.7	69.9	6.0	14.835556	0.170405	0.999548	0.707338
3	xgb	69.2	10.4	70.3	26.9	70.3	26.4	6.327536	0.060745	1.000000	0.745189

In Figure 27, it is interesting to note that across the board, precision and recall were pretty close to balanced, which is fantastic! The XGBoost model has just managed to increase both and as such it is the winner in terms of model accuracy.

Overall, the models performed quite similarly across the board and further hyperparameter tuning likely would not have gotten us that much closer to generalizability and accuracy. Given our confusion matrices, future efforts would likely be best spent exploring features that better delineate classes like blended and cotton or blended and polyester. We feel confident that the features chosen do fairly well for denim, as its weave is quite distinct and well captured by the edge-focused features used. Exploring additional features used by Kampouris and team in their paper and their interactions with the muddled classes would be a great starting point.

Due to time and resource constraints, our exploration of vector features and CNN generated feature sets was limited. Future work may also include leveraging more powerful computational tools and

developing more descriptive features that can be processed using the computational resources. Overall, however, our work did an excellent job pointing out the power of using simpler features over vector features in some circumstances and we were able to produce models that didn't have terrible precision accuracy or recall.

7 References

[1] Kampouris, C., Zafeiriou, S., Ghosh, A., Malassiotis, S. (2016). Fine-Grained Material Classification Using Micro-geometry and Reflectance. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science(), vol 9909. Springer, Cham.

[2] Öztürk, Ş., & Akdemir, B. (07 2015). Comparison of Edge Detection Algorithms for Texture Analysis on Glass Production. Procedia - Social and Behavioral Sciences, 195, 2675–2682. doi:10.1016/j.sbspro.2015.06.477

[3] Li F, Yuan L, Zhang K, Li W. A defect detection method for unpatterned fabric based on multidirectional binary patterns and the gray-level co-occurrence matrix. Textile Research Journal. 2020;90(7-8):776-796.

[4] N. Mohanty, A. Lee-St. John, R. Manmatha, T.M. Rath, Chapter 10 - Shape-Based Image Classification and Retrieval, Editor(s): C.R. Rao, Venu Govindaraju, Handbook of Statistics, Elsevier, Volume 31, 2013.

[5] V.F. Leavers. Shape Detection in Computer Vision Using the Hough Transform (1992) DOI:10.1007/978-1-4471-1940-1 Library of Congress Data available

[6] Carson, C., Thomas, M., Belongie, S., Hellerstein, J.M., Malik, J. (1999) Blobworld: a System for Region-Based Image Indexing and Retrieval (long version). In: Technical Report No. UCB/CSD-99-1041, EECS Department, University of California, Berkeley.

8 Appendix

8.1 PCA and tSNE Dimensionality reduction results