Transforming Features for Date Fruit Classification

Nisala Kalupahana [*1] **Akash Munagala** [*1] **Vikram Meyer** [*1]

Abstract

We apply various transforms to the raw features of a date fruit dataset in order to explore whether this increases the classification accuracy. The first feature transformation explored uses a neural network to extract more complex features from the raw features by using the second to last hidden layer as an embedding layer that represents the transformed feature. The second feature transformation explored uses principal component analysis to transform samples from the input feature space to a lower dimensional feature space that captures a large percent of their variance. We compare the classification accuracy between a neural network, linear SVM, and RBF kernel SVM on the raw feature dataset as well as transformed feature dataset. We use a multi-class date fruit dataset of nearly 1000 samples to compare the methods.

1. Introduction

Agriculture is a prime field for automation, and machine learning is taking an increasingly large role in this endeavor. Especially with the wider availability of automated pickers for crops like fruit trees, it's important to be able to separate out automatically picked foods by properties such as ripeness and variety. This classification is often done in two steps: first, features are extracted from the food, and then those features are used to classify the food (by ripe/not ripe, variety, etc.) (Koklu et al., 2021). One food where this type of classification is being developed is date fruits. Fruit trees are often grafted together, so multiple varieties can be present on the same tree. There are several prominent date varieties grown commercially, so even in orchards without grafted trees, multiple varieties can be present in one area. In order to separate these varieties out after automated picking, machine learning can be used, and datasets have been produced to help with this sort of classification work (Koklu et al., 2021). In the particular dataset we are examining in this paper, the initial raw features have been generated from date images using computer visison techniques. Our goal is to classify the date fruits into their respective classes given only these features, though as we explain in further sections, we test whether transforming these features can increase classification performance.

1.1. Neural Networks as Feature Extractors

Convolutional neural networks (CNNs) have been shown to extract increasingly complex features from an image, especially as you move deeper into the network (Olah et al., 2020). Early layers detect simple features in an image such as edges or curves. Later layers detect more complex features as linear combinations of features detected in previous layers. This allows a hierarchy of feature detectors to be learned using a deep neural network, where the earlier layers learn simpler features like edges, and later layers build on those simple features to detect more complex features such as the presence of faces or objects. While this idea of neural networks as feature detectors has received a good amount of research attention in attempts to add explainability to increasingly accurate computer vision models, the idea of neural networks as feature extractors for just normal numerical feature inputs has not received as much attention in the research literature.

We hypothesize the existence of feature detectors in CNNs implies a similar existence of feature detectors within fully connected artificial neural networks (ANNs) trained on non-image inputs. We propose the features learned by deeper layers in a neural network are better representations of a sample that can be classified easier since they represent more useful information about a sample. This leads us to believe that an optimal margin classifier (SVM) which tries to maximize the margin between classes, generally leading to better test set generalization, should be able to classify these learned features to a higher degree of accuracy compared to just the raw features. We also believe the optimal margin classifier has the potential of classifying the learned features better than the final fully connected layer of the neural network because the the optimal margin classifier uses optimization with hard constraints to find

[*]Equal contribution [1]Department of Computer Science, Vanderbilt University, Nashville, Tennessee, USA. Correspondence to: Nisala Kalupahana <nisala.a.kalupahana@vanderbilt.edu>, Akash Munagala <akash.munagala@vanderbilt.edu>, Vikram Meyer <vikram.j.meyer@vanderbilt.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

optimal decision boundaries, whereas neural networks only use backpropragation of an error signal to learn decision boundaries.

We can view this neural network as a simple mapping from one feature space to another, since a neural network is simply a composition of linear and nonlinear transformations on an input. This perspective will be useful later in the paper. For the purposes of this paper, we will define the mapping

$$\Phi_N : F_1 \subseteq \mathbb{R}^n \rightarrow F_2 \subseteq \mathbb{R}^n$$

where n is the number of raw features in the dataset. This means we are mapping from one feature space of size n to another feature space of size n.

1.2. PCA for Feature Space Transformation

Principal component analysis is a useful statistical technique for determining the components that capture the most variability in a dataset. The perspective we choose to view PCA as in this paper is a transformation from the original feature space to a lower dimensional feature space that potentially represents the data better (Li et al., 2003). We define our PCA transformation to be the mapping

$$\Phi_P : F_1 \subseteq \mathbb{R}^n \rightarrow F_3 \subseteq \mathbb{R}^k$$

where $k < n$. In this paper, we test whether using the PCA transformation on the dataset creates a dataset that is more linearly separable for a support vector machine to classify with a higher degree of accuracy. We also test using a neural network classifier on the PCA dataset. Though a neural network can learn the important input features on its own from the raw features, and can in fact approximate any function including ϕ, applying PCA before training the neural network can help it learn better as it reduces the amount of information about the training distribution that must be learned by already supplying the neural network with the most important features – the ones that capture the most variance in the dataset.

2. Methods

2.1. Dataset

The date fruit dataset (Koklu et al., 2021) is used to test which classification technique performs the best. The dataset was generated using computer vision techniques on a set of labeled date fruit images. It consists of 898 samples that are labeled as belonging to one of 7 classes (genetic varieties) of dates. There is a class imbalance in the data as demonstrated in table 1.

Table 1. Dataset Class Imbalance

CLASS	# SAMPLES
DOKOL	204
SAFAVI	199
ROTANA	166
DEGLET	98
SOGAY	94
IRAQI	72
BERHI	65

While our data exploration did note a class imbalance, we did not perceive the imbalance to be drastic enough (i.e. 100:1) to require additional pre-processing techniques such as oversampling or undersampling to balance the classes. Instead, we made sure to add F1-score as an additional metric since F1 score is a weighted average of precision and recall, in order to ensure our accuracy was not misleading us due to the class imbalance.

The dataset has 34 features such as area, perimeter, solidity, eccentricity, and many other measurements.

The first step taken in understanding this data was determining whether it was separable. To do this, we first started with a pair plot, where we plotted some core features against each other, with the classes highlighted, to get an immediate feel for their separability. (Figure 1)

While some variable pairs are less separable than others, most of the plots have a visible separations between classes that seems to be exploitable for classification purposes. In order to further confirm this, we plotted pairs of three features against each other (an example of such a plot is shown in Figure 2) to ensure the data became more separable as more features were added.

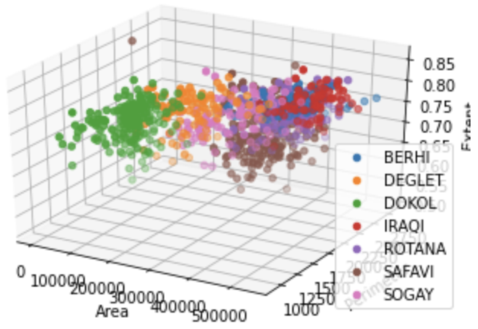

Figure 2. Area vs. Perimeter vs. Eccentricity

Figure 2 demonstrated that the data was separable, especially as more features were added. In addition to this separability analysis, the paper that generated this dataset also performed classification work that we use as a benchmark for our analysis. They (Koklu et al., 2021) ran a logistic regression, which achieved an accuracy of 91%, and created

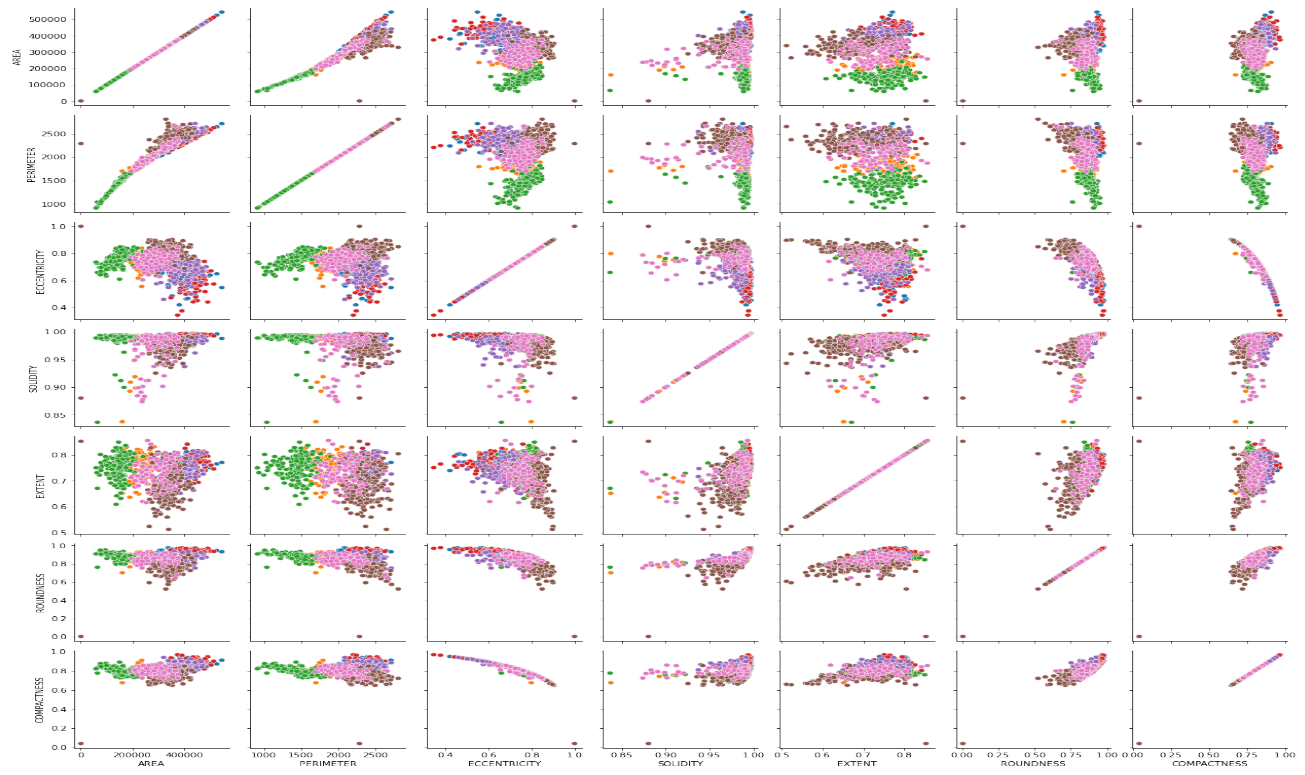

Figure 1. Pair plot showing some core features plotted against each other.

a basic artificial neural network (ANN), which achieved an accuracy of 92.2%. They also created a stacking model that combined the results from both of these models, which achieved an accuracy of 92.8%. This stacking model is complex and likely contains significant redundancies between the ANN and logistic regression, so our aim in this paper is to create a simpler model that can match or exceed the accuracy of these benchmarks.

2.1.1. Pre-processing

Initially, a train/test split of 80/20 was created with the dataset. Next, each of the features in the train set was standardized to have $\mu = 0$ and $\sigma = 1$. This was an important step because the scale of the different features varies by orders of magnitude, so this step gives all features an equal initial weight in our learning algorithms. The μ and σ for each feature in the train set is then used to standardize all the samples in the test set. Leaving the test samples out of the calculation of the μ and σ for each sample is important in ensuring the test samples remain unseen by the learning algorithm until inference time. As all the features are numerical, there is no need to utilize any pre-processing techniques for turning categorical features into numerical features. These standardized features are what we refer to as the raw features in this paper as only standard pre-processing techniques have been applied to them.

Later, in order to verify our initial results more rigorously, we use k-fold validation to create $k = 10$ folds of the dataset. For each train set created with $k - 1$ folds, we apply the same standardization technique from above, with the mean and standard deviation from the train set used to standardize the corresponding test set.

2.2. Classification Techniques

We employed three classification techniques on all the datasets.

1. SVM with linear kernel
2. SVM with Radial Basis Function/Gaussian kernel
3. Neural Network

We decided to use a SVM with a linear and RBF kernel since we were interested in the effect of the additional feature map of the Gaussian kernel from the input feature space to an effectively infinite dimensional feature space. The linear kernel is effectively the identity mapping from the input feature space into the same feature space.

We employed three classification techniques by transforming the normalized features before utilizing them through in three models: a support vector machine (SVM) with a linear kernel, a support vector machine (SVM) with a radial basis function kernel, and a neural network. For each technique, we applied k-fold cross validation, using a k of 10 in order to accurately measure our model performance. For the SVM models, we used the default hyperparameters for the regularization constant c and smoothness parameter γ provided by scikit-learn. For the neural network models, we used the ReLU activation functions for the hidden layers and a softmax activation for the output layer. Our weights were initialized using Kaiming normalization since this is best for ReLU neural networks.

2.2.1. Raw Features

Our first approach just used the original 34 normalized features in the three models. The only transformation utilized was the standardization to $\mu = 0$, $\sigma = 1$ for each of the features. We used this dataset to train and evaluate SVMs with a linear and RBF kernel. We also used this dataset to train and evaluate a neural network classifier. The neural network we used had 3 hidden layers of 1024, 1024, and 34, and an output layer of 7 to match our class size.

2.2.2. Reduced Dimension Samples

Based on previous success in extracting lower dimensional features using principal component analysis as a transformation (Li et al., 2003), we used PCA to project our samples to a lower dimension. The idea is that PCA is able to remove redundant features that might have been generated in the computer vision-based generation of the dataset. After performing PCA, we determined that 20 components capture 99% of the variance so we decided to use these components as the basis for our lower dimensional feature space.

In order to generate a transformed dataset consisting of the samples projected to this lower dimensional feature space, we simply used the PCA transformation from the Scikit-learn library. We fit the transformation using the train set of raw features, and then applied the resulting transformation generated for our desired 20 components to both the train and test set to get new transformed train and test sets. Our new train set was then used to train a SVM with a linear kernel and one with a RBF/gaussian kernel. The new test set was then used to generate predictions from the SVMs that could be used to calculate an accuracy and f1-score for the SVMs' generalization ability on this lower dimensional dataset. We also trained a neural network (same shape as in Section 2.2.1) end-to-end as a classifier on the transformed train set. Then we evaluated the neural network's classification accuracy and calculated an f1-score for the transformed test set.

2.2.3. Neural Network Extracted Features

Based on previous success in extracting features from images to be classified by an SVM (Koklu et al., 2022), we extracted higher level features from the raw input features by passing the inputs through a neural network (same shape as in section 2.2.1). We train our deep neural network using supervised methods and the target class labels. We utilize PyTorch for defining and training our model. Since we are performing multi-class classification, the loss function used for training was CrossEntropyLoss provided by PyTorch. This loss function takes output logits x, applies $\log(\text{softmax}(x))$ and then uses the negative log likelihood loss function on this result to get a loss value that can be backpropagated through the network. As this loss function works with integer encoding, we do not need to perform one-hot encoding of our targets.

We trained the network using stochastic mini-batch gradient descent with a batch size of 128, in order to approximate the true gradient as best as possible without having to compute the loss over the full train set. After training many models and observing the graph of train accuracy vs. test accuracy over the number of epochs, we deemed 10 epochs to be sufficient for achieving optimal test set accuracy without overfitting the train set.

After training a model in a supervised manner to predict the class of a sample, we then set the model to extract features by removing the final layer of 7 neurons. This leaves us with a layer of 34 neurons that acts as an embedding layer for our samples. We chose the size of the embedding layer based on results in 2. The fact that the selected embedding layer size is the same as the number of raw features is convenient in that it allows us to compare whether the features learned by the neural network can be classified with higher accuracy than raw features.

The architecture we used for our experiments was 34 neurons in the input layer, two hidden layers of 1024 neurons each, an embedding layer of 34 neurons (2nd to last layer), and the 7 neuron final layer for class prediction. Note that when using the neural network as a classifier in evaluation mode, we apply $\log(\text{softmax}(x))$ to the logits output in the final layer to get class probabilities.

In order to generate a transformed dataset consisting of the features in the embedding layer, we set the model to feature extraction mode where the final layer is the embedding layer and then used the model to make predictions on all of the train and test set data. This new transformed train set was then used to train a SVM with a linear kernel and one with a gaussian/rbf kernel. Next, the SVMs' predictions on the transformed test set were used to calculate an accuracy and f1-score.

3. Results

The first thing we tested was the effect of the size of the embedding on the classification performance of the SVMs and neural networks classifying the embeddings.

Table 2. Effect of Embedding Size on Classification Performance

Size	Classifier	Accuracy	F1-Score
5	Neural Network	63.12%	0.5527
5	Linear SVM	75.92%	0.7277
5	RBF SVM	75.93%	0.7189
20	Neural Network	91.32%	0.9132
20	Linear SVM	90.77%	0.9070
20	RBF SVM	90.65%	0.9060
34	Neural Network	**92.55%**	0.9255
34	Linear SVM	91.76%	0.9167
34	RBF SVM	91.21%	0.9116
512	Neural Network	91.99%	0.9186
512	Linear SVM	92.21%	0.9216
512	RBF SVM	92.32%	0.9229
1024	Neural Network	91.21%	0.9095
1024	Linear SVM	91.21%	0.9101
1024	RBF SVM	92.10%	0.9203

Table 2 demonstrates that too small of an embedding (e.g. size 5) can lead to poor classification, while increasing the size of an embedding to be arbitrarily large does not necessarily correspond to an increase in classification accuracy. In order to determine the embedding size for further experiments, we took the embedding size that produced the highest f1-score, regardless of the classification method. This embedding size was 34, the number of input features, with the neural network having the highest f1-score of 0.9255. Thus, for the results in the remainder of the paper, all neural network features are those extracted using an embedding layer of 34 neurons.

An interesting result from the embedding of size 5, is that the SVM performs better classification of the embeddings than the neural network itself (it beats it by over 12% in terms of accuracy and 0.17 in terms of f1-score). This supports our earlier hypothesis that an optimal margin classifier can classify the extracted features from a neural network better than the final layer of the neural network itself.

Our first experiments utilized a simple randomized 80%/20% train-test split. This gave us very promising initial results with many of the classification techniques beating the 92.8% accuracy achieved in (Koklu et al., 2021). After receiving these promising results, we sought to verify the rigour of these results as our best performing classification technique (a linear SVM classifying the features extracted from the neural network) yielded a test set accuracy of 95.6%, nearly 3 percentage points higher than the accuracy achieved in (Koklu et al., 2021).

After implementing k-fold cross validation, the results achieved by all of our methods decreased in accuracy and f1-score after averaging the metrics across the k models. All the results can be viewed in the tables below.

Table 3. Raw Features Dataset

Classifier	Accuracy	F1-Score
Neural Network	**92.55%**	**0.9255**
Linear SVM	92.43%	0.9244
RBF SVM	91.10%	0.9100

Table 4. Neural Network Extracted Features Dataset

Classifier	Accuracy	F1-Score
Neural Network	**92.55%**	**0.9255**
Linear SVM	91.76%	0.9167
RBF SVM	91.21%	0.9116

Table 5. Reduced Dimension (PCA) Dataset

Classifier	Accuracy	F1-Score
Neural Network	92.55%	0.9249
Linear SVM	**92.99%**	**0.9300**
RBF SVM	91.10%	0.9100

As evident in all the tables above, the accuracy and F1-score are very close for all entries, indicating that the slight class imbalance discovered in the data exploration stage did not affect the performance of the model drastically. Therefore, we will refer to the accuracy performance metric as it is more intuitive to the reader.

4. Discussion & Conclusions

The best classification method on the raw features was unsurprisingly a neural network classifier trained end-to-end using supervised learning on the target class to get a test set accuracy of 92.55%, very close to the 92.8% target from (Koklu et al., 2021). We say this is an unsurprising result because the neural network of size 34 x 1024 x 1024 x 34 x 7 has 1.1 million learn-able parameters and the dataset has 1000 samples, meaning there is plenty of capacity for learning a classification function for the inputs features.

The best classification method on the features extracted from a neural network was again a neural network, with the same accuracy of 92.55% since the network was the same size and a neural network classifying based on raw features uses the extracted features in the embedding layer to make classifications. Though table 2 shows an embedding of size

Table 6. Classification Method Performance Ranking

Classifier	Features	Accuracy	F1-Score
Linear SVM	PCA	92.99%	0.9300
Neural Network	Raw	92.55%	0.9255
Neural Network	PCA	92.55%	0.9249
Linear SVM	Raw	92.43%	0.9244
Linear SVM	NN	91.76%	0.9167
RBF SVM	NN	91.21%	0.9116
RBF SVM	PCA	91.10%	0.9100
RBF SVM	Raw	91.10%	0.9100

5 could be used by the optimal margin classifier with linear kernel (SVM) to reach a higher classification performance than the neural network classifying this embedding, as we increased the size of the embedding, the linear SVM did not maintain this increased classification performance over the neural network classifier. This result was unexpected to us as we initially thought a learned embedding would contain more useful information for classification using optimal margins.

The best classification method on the features extracted using PCA was surprisingly the SVM with a linear kernel, which achieved an accuracy and f1-score of 92.99% and 0.93 respectively. This was the only method tested using k-fold cross validation that outperformed the 92.8% accuracy reported in (Koklu et al., 2021).

Going into the experiments, we expected the classification methods that transformed the features using either a neural network or PCA to perform better than classification based on the raw features. While the initial results from the 80/20 randomized train-test split strongly supported our hypothesis, the empirical results from the k-fold cross validated models only weakly supported this hypothesis with the PCA transformed features being classified using an SVM performing only marginally better than the stacking model of an ANN and logistic regression in (Koklu et al., 2021).

Setting up the initial neural network mini-batch training for multi-class classification was challenging because we did not realize the Cross Entropy Loss function in PyTorch also applied the log softmax operation before actually applying the loss. Similarly, we were initially unsure as to whether we needed to provide one-hot encoded targets for this loss function, though we eventually discovered the integer encoded targets work. Making sure the tensor dimension of the operations was correct also provided a slight challenge since we were working with mini-batches instead of just single samples.

Another challenge we faced during this project involved increasing the rigour of our results using k-fold cross validation. After doing the k-fold cross validation, the performance of all our classifiers decreased, despite a similar class distribution amongst the k cross validation train/test sets and the initial train/test set used to get better results. This decrease in performance despite similar class distributions confused us, although we eventually realized the stochastic nature of mini-batch updates over different train sets will lead to some of the k models getting trapped in local minima when training, resulting in poor performance that brings down the average performance metric.

Interesting future directions could include testing this hypothesis on other datasets as perhaps this dataset generated using computer vision methods contains a latent structure in the data that is not conducive to the success of these feature transformation methods. Similarly, it would be interesting to explore the success of these methods on datasets with mixes of numerical and categorical features.

Acknowledgements

Nisala found the initial data set and performed the data exploration and visualization. Nisala got the results for the raw features on the SVM. Akash performed PCA on the data set and got the SVM and NN classification results on the reduced dimensionality data set. Vikram used the neural network to create the feature extracted data set and got the classification results using the SVM and neural network. The report, slides, and final paper were written together by assigning an equal number of sections to each person.

References

Koklu, M., Kursun, R., Taspinar, Y. S., and Cinar, I. Classification of Date Fruits into Genetic Varieties Using Image Analysis. *Mathematical Problems in Engineering*, 2021, 2021. doi: 10.1155/2021/4793293.

Koklu, M., Unlersen, M. F., Ozkan, I. A., Aslan, M. F., and Sabanci, K. A CNN-SVM study based on selected deep features for grapevine leaves classification. *Measurement: Journal of the International Measurement Confederation*, 188, 2022. ISSN 02632241. doi: 10.1016/j.measurement.2021.110425.

Li, W., Shi, T., Liao, G., and Yang, S. Feature extraction and classification of gear faults using principal component analysis. *Journal of Quality in Maintenance Engineering*, 2003.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Appendix

The data exploration and initial support vector machine results reported in the interim report was done in `interim-report.ipynb`. Our experiments using only a single 80/20 train test split were done using `preprocessing.ipynb` for the preprocessing of the data and using `experiment.ipynb` for the initial results of the different classifiers. The k-fold cross validated results can be found by running `python3 experiment.py -f <output-results-folder>`. The `Experiment` class loads the original dataset, splits the data in k-folds which are used to then run the method k times and average their performance metrics. This will print out the average accuracy and average F1 score for all the classification methods. Additional information will be printed including the logs from the training of the models as well as the class splits for each of the k train sets. `model.py` contains the definition of our PyTorch neural network module. `datasets.py` contains the two PyTorch dataset classes used in our experiments.