Letter Recognition From Character Image Features

Statistical Learning
Professor: Francesco VACCARINO

Author: Ghassane BEN EL AATTAR

Politecnico di Torino

Contents

1 Introduction

In the era of data-driven decision-making, classification tasks have become pivotal in extracting meaningful patterns from vast amounts of data. Classification, a subset of supervised learning, involves predicting the category of a given data point based on its features.

The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

In our exploration, we evaluated the performance of the following distinct classifiers: Decision Tree, Random Forest, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Logistic Regression and Neural Networks. Each classifier was subjected to a randomized search for hyperparameter tuning, ensuring optimal performance. Furthermore, to gauge the impact of dimensionality reduction on model performance, some classifiers were trained and tested on both the original dataset and its PCA-transformed counterpart.

This report presents a comprehensive analysis of the performance metrics, including accuracy and F1-score, for each classifier. Through this comparative study, we aim to discern the most effective classification algorithm for our dataset and understand the implications of dimensionality reduction on model efficacy.

2 Exploratory Data Analysis

This section is focused in visualizing some statistics and analyzing the high level characteristics of the dataset.

2.1 Dataset Description

The dataset comprises several attributes related to the characteristics of capital letters. Each attribute captures specific information, as detailed below:

- **lettr**: Represents the capital letter. It has 26 distinct values ranging from A to Z. This is the target feature of our classification task.
- **x-box**: Denotes the horizontal position of the box. It is an integer value.
- **y-box**: Denotes the vertical position of the box. It is an integer value.
- **width**: Represents the width of the box. It is an integer value.
- **high**: Represents the height of the box. It is an integer value.
- **onpix**: Total number of on pixels. It is an integer value.
- **x-bar**: Mean x of on pixels in the box. It is an integer value.
- **y-bar**: Mean y of on pixels in the box. It is an integer value.
- **x2bar**: Mean x variance. It is an integer value.
- **y2bar**: Mean y variance. It is an integer value.
- **xybar**: Mean x y correlation. It is an integer value.
- **x2ybr**: Mean of $x \times x \times y$. It is an integer value.
- **xy2br**: Mean of $x \times y \times y$. It is an integer value.
- **x-ege**: Mean edge count from left to right. It is an integer value.
- **xegvy**: Correlation of x-ege with y. It is an integer value.
- **y-ege**: Mean edge count from bottom to top. It is an integer value.
- **yegvx**: Correlation of y-ege with x. It is an integer value.

2.2 Missing Values

Many machine learning algorithms implementations assume each value is numeric and will throw errors if null values are encountered and many other algorithms are negatively biased by their presence. Removing records with missing values can be fast and effective but may result in losing valuable information, therefore smarter techniques exists such has imputing the mean, the median, the most frequent value or using algorithms like KNN. Luckily this dataset contains no missing values and there is no need for further investigation.

2.3 Dataset Values and Distributions

Let's know analyze the values that the attributes have in the dataset. We can see from the summary table (Figure 1) that all the numerical attributes have the same range of values, since they were scaled beforehand, as mentioned in the introduction. The medians are not similar between attributes, while the standard deviations are all close to 2 for all attributes.

Summary Table for Numeric Attributes

Attribute	min	max	50%	std
x-box	0.0	15.0	4.0	1.9132115451305487
y-box	0.0	15.0	7.0	3.304555303560018
width	0.0	15.0	5.0	2.0145732805671623
high	0.0	15.0	6.0	2.261390433303058
onpix	0.0	15.0	3.0	2.190457870579605
x-bar	0.0	15.0	7.0	2.0260354096051847
y-bar	0.0	15.0	7.0	2.325353771151403
x2bar	0.0	15.0	4.0	2.699967875980484
y2bar	0.0	15.0	5.0	2.3808228815642014
xybar	0.0	15.0	8.0	2.4884749190761686
x2ybr	0.0	15.0	6.0	2.631070148533924
xy2br	0.0	15.0	8.0	2.0806190061547643
x-ege	0.0	15.0	3.0	2.3325408522341826
xegvy	0.0	15.0	8.0	1.5467224363197007
y-ege	0.0	15.0	3.0	2.5670725409640314
yegvx	0.0	15.0	8.0	1.6174700524178258

Figure 1: Summary Table for Numeric Attributes

Let's now analyse the distributions of the attributes in the dataset, start-

ing first with the numeric attributes in Figure 2.

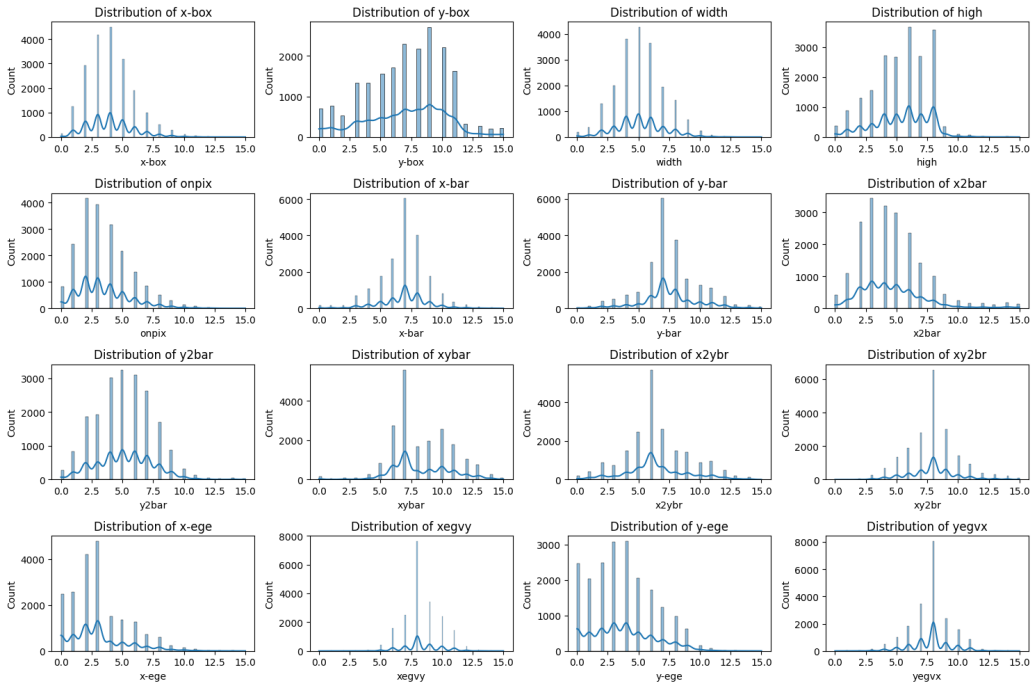

Figure 2: Distributions of Numeric Attributes

The histograms show the distribution of each numerical attribute. From the plots, we can observe the spread and central tendency of each attribute. Some attributes like 'x-box' and 'y-box' seem to have a somewhat normal distribution, while others might be skewed, therefore numerical attributes vary in their distributions.

Next, we'll evaluate the distribution for the categorical attribute 'lettr'.

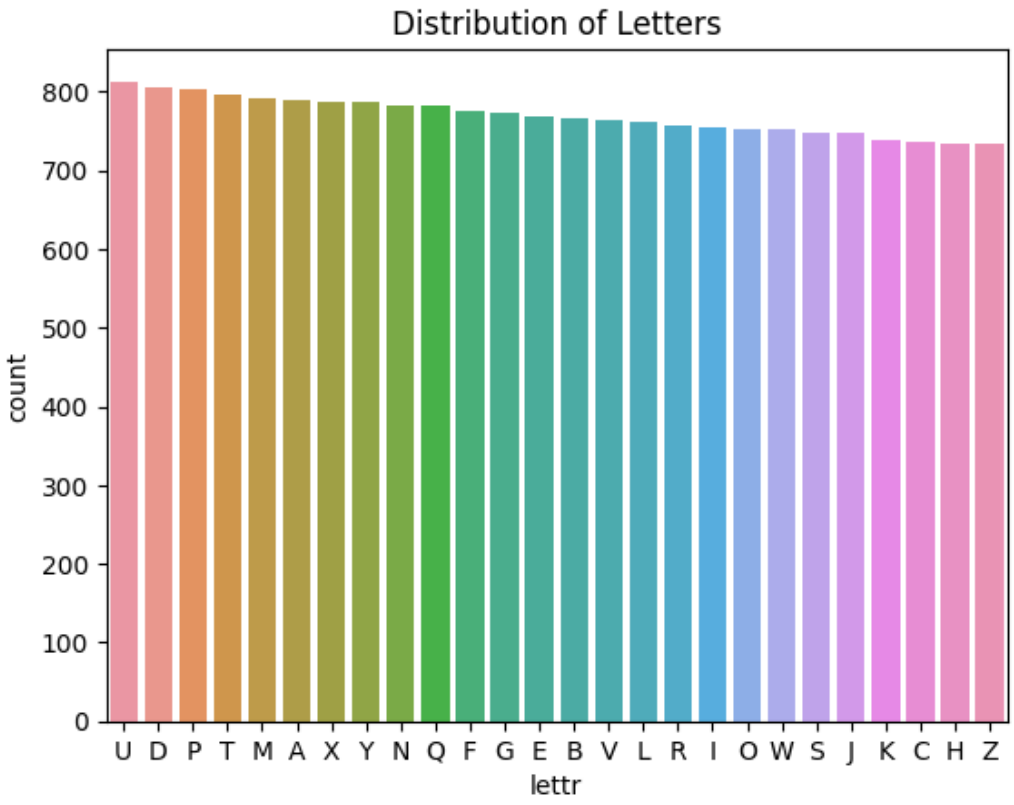

Figure 3: Distribution of Letters

As we can see, the distribution of letters seems fairly even, meaning that undersampling or oversampling steps may be avoided before performing classification.

2.4 Correlation Analysis

Correlation analysis can help in identifying highly correlated features. If two features are highly correlated, they carry similar information, and there might be little benefit in keeping both. Removing redundant features can reduce the dimensionality of the dataset, making models simpler, faster, and potentially more accurate. We can use a heatmap to analyse possible correlations between numeric attributes.

The heatmap below (Figure 4) visualizes the correlation between the numerical attributes. The color intensity and the annotated values represent the strength and direction of the correlation.

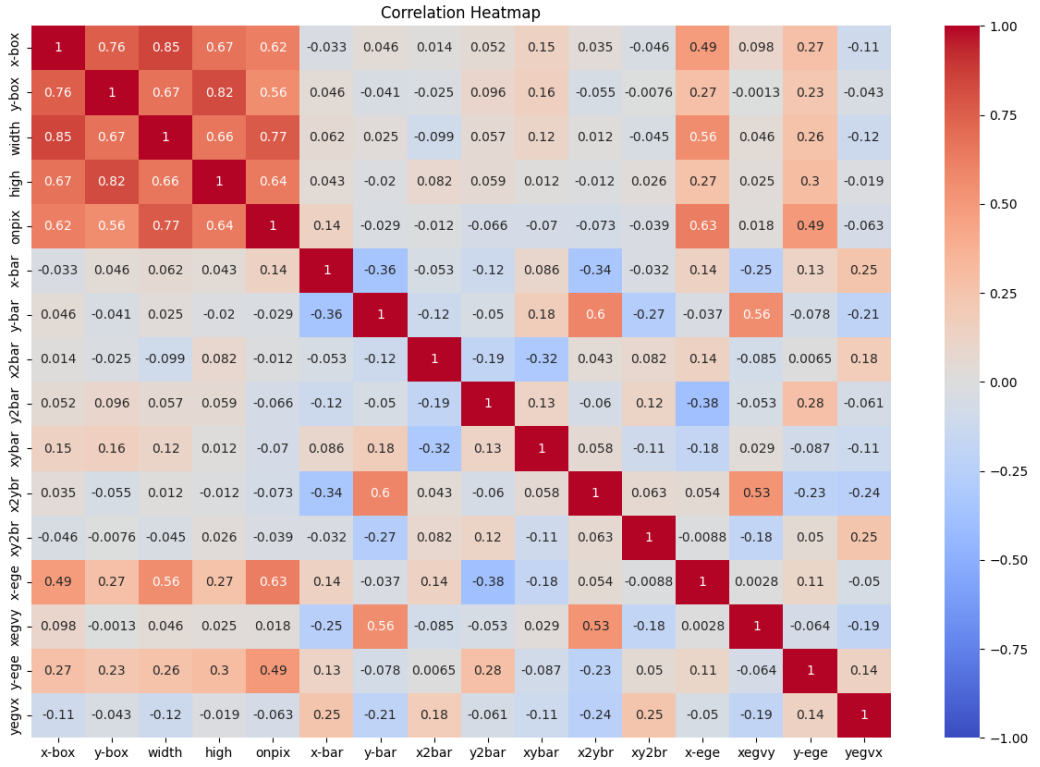

Figure 4: Correlation Heatmap

Some attributes show a high positive correlation, such as x-box with width (0.85) and y-box with high (0.82). This indicates that as one attribute increases, the other tends to increase as well. There are also attributes with a strong negative correlation, but from the heatmap, it seems the negative correlations are not as strong as the positive ones. For the purpose of this classification task, I opted for keeping the attributes, although feature selection in this instance may improve the performance of certain classification models

2.5 Outliers

Outliers are data points that differ significantly from other observations in a dataset. They can be unusually high or low values that do not align with the general trend or distribution of the data. Outliers can arise due to variability in the data, errors in data collection or processing, or genuine extreme observations. Machine learning algorithms (regression in particular) can be sensitive to outliers, which might lead them to produce less accurate models.

Not all outliers are errors though and some might even represent genuine extreme observations, providing valuable insights into phenomena that wouldn't be apparent from the "typical" data points.

To check for potential outliers in our dataset, we can use boxplots. The boxplots in Figure 5 below showcase the distribution of each numerical attribute for different letters. It helps in identifying patterns, outliers, and variations in distributions for different letters.

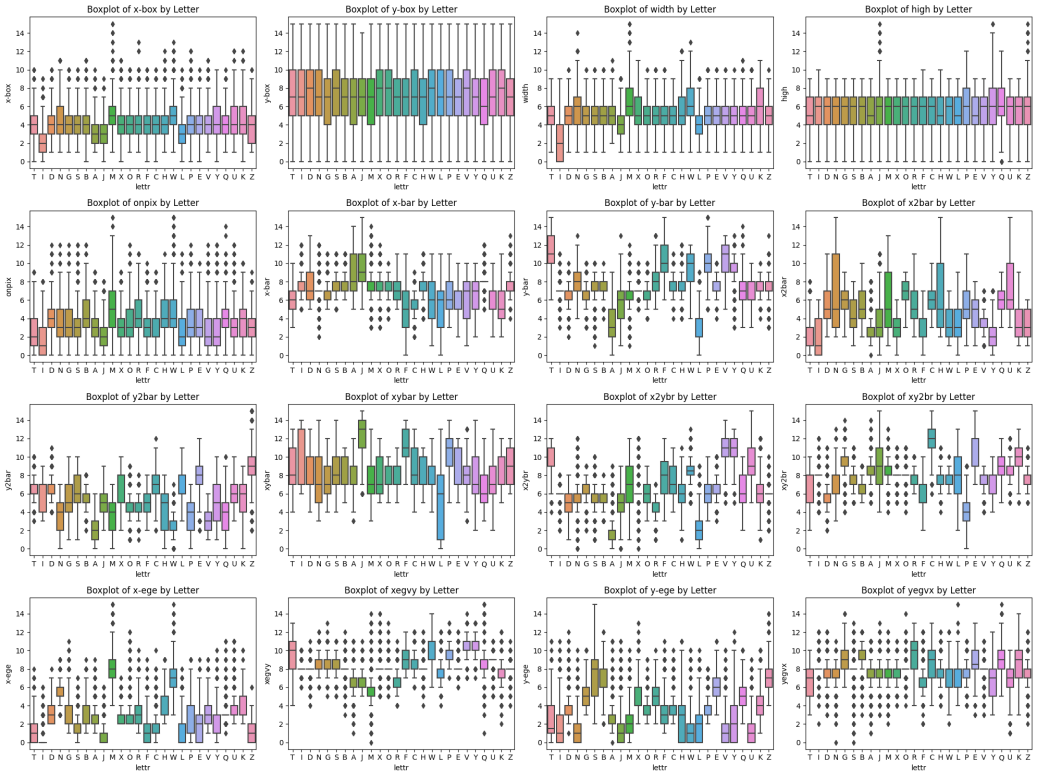

Figure 5: Boxplots

There are noticeable outliers in some attributes for specific letters. For this task, I opted for not removing extreme values from the dataset, as I don't have the necessary knowledge to understand if they are correct or incorrect data.

2.6 Training, Test and Validation Set

Now we'll split the dataset into a training set and a test set, with the common 80% and 20% split. This will allow us to perform our modelling on the training set and evaluate performances on unseen data.

When evaluating different hyperparameters for estimators, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can "leak" into the model and evaluation metrics no longer report on

generalization performance. To solve this problem, yet another part of the dataset can be held out as a so-called "validation set": training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set.

2.7 Cross Validation

By partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k "folds":

- A model is trained using $k - 1$ of the folds as training data;
- the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems such as inverse inference where the number of samples is very small. [1]

We'll use 5-fold CV for each classification algorithm.

3 Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms the original features of a dataset into a new set of orthogonal features known as principal components (PCs). These PCs are linear combinations of the original features and are constructed to capture the maximum variance in the data.

Formally, given a dataset with centered features (zero mean) and standardized variance (unit variance), the first principal component, PC_1, is found by:

$$PC_1 = w_1 x_1 + w_2 x_2 + \cdots + w_p x_p$$

where x_i are the original features and w_i are the loadings of the respective features. These loadings are constrained such that:

$$\sum_{i=1}^{p} w_i^2 = 1$$

ensuring that the variance isn't arbitrarily large.

Subsequent PCs are constructed to be orthogonal to the preceding ones, ensuring they capture the maximum remaining variance while being uncorrelated with previous components.

One of the main advantages of PCA is its ability to mitigate the curse of dimensionality by reducing redundancy and retaining only the most informative features. However, a notable drawback is that the derived principal components often lack clear interpretability, as they don't directly correspond to the original features.

The amount of variance explained by the m^{th} component, termed as the Proportional Variance Explained (PVE), is given by:

$$PVE_m = \frac{\sum_{i=1}^{n} \left(\sum_{j=1}^{p} w_{jm} x_{ij} \right)^2}{\sum_{j=1}^{p} \sum_{i=1}^{n} x_{ij}^2}$$

The cumulative PVE over the first m components represents the total variance retained by using these components. Conversely, the cumulative PVE of the last $n - m$ components denotes the lost variance or the reconstruction error. This perspective aligns with an alternative formulation of PCA that aims to minimize the reconstruction error between the original data points and their projections in the reduced feature space.

Let's now analyse the cumulative explained variance for the principal components after performing the full PCA on the training set.

This indicates how much variance each principal component retains from the original data.

The plot below (Figure 6) shows both the individual explained variance by each component and the cumulative explained variance.

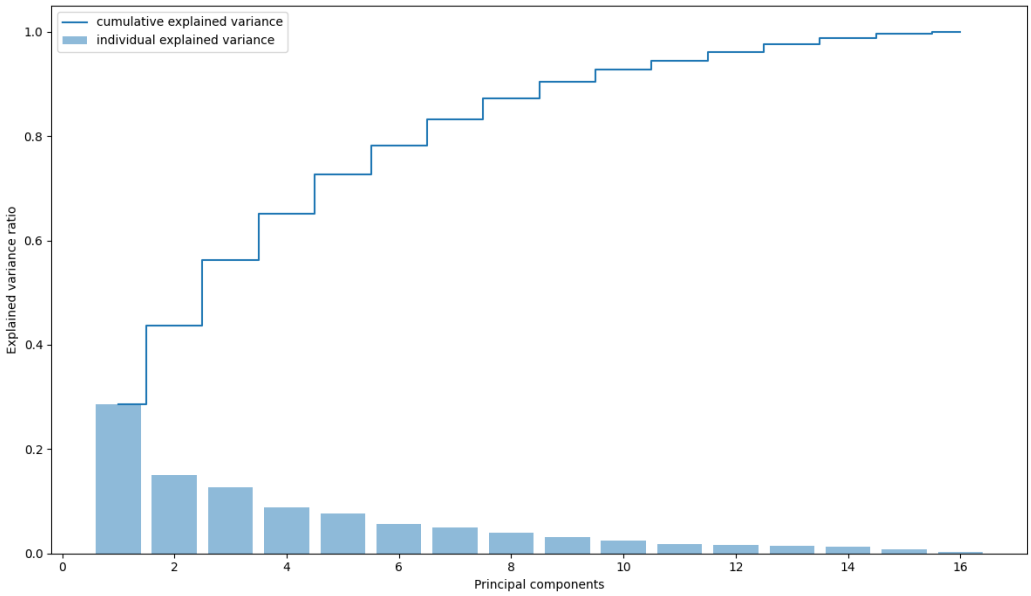

Figure 6: Cumulative Explained Variance Plot

Around 90% of the data's variance is captured by the first 10 principal components. This suggests that we could potentially reduce the dimensionality of the dataset to 10 principal components while still retaining a significant amount of information.

As we move to higher-numbered components, the amount of explained variance decreases. This is typical in PCA, where the first few components capture the most variance, and each subsequent component captures less and less.

Reducing the dimensionality of the data this way can ìmprove but also worsen the performance of a classification model. As mentioned in the introduction, we'll try comparing results for certaing algorithms with and without

dimensionality reduction.

4 Performance Evaluation

4.1 Accuracy and F1-score

In the realm of classification tasks, selecting appropriate evaluation metrics is paramount to understanding and interpreting the performance of algorithms. In this classification task, we will compute the accuracy and the F1-score for the different methods.

Accuracy, denoted as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}},$$

represents the proportion of correctly predicted instances out of the total instances. It provides a general overview of the model's performance across all classes. However, accuracy can be misleading, especially in imbalanced datasets where one class significantly outnumbers the others. In such scenarios, a model might achieve high accuracy by merely predicting the majority class, rendering the metric less informative.

This is where the F1-score comes into play. The F1-score is the harmonic mean of precision and recall, given by

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}},$$

where

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

and

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}.$$

It provides a balance between these two metrics, ensuring that both false positives and false negatives are taken into account. The F1-score is particularly valuable in situations where one type of classification error is more significant than the other or when class distribution is skewed.

4.2 Randomized Search

Hyperparameter tuning is a crucial step in the machine learning pipeline, aiming to optimize the parameters that govern the training process of an

algorithm. One popular method for hyperparameter tuning is *Randomized Search*. Unlike grid search, which exhaustively tries all possible combinations of hyperparameters, randomized search samples a fixed number of hyperparameter combinations from specified probability distributions.

The primary advantage of randomized search lies in its efficiency. For hyperparameter spaces that are large, randomized search can identify good combinations with fewer iterations compared to grid search. Mathematically, given a hyperparameter space $\mathcal{H}$, randomized search samples n combinations, where $n \ll |\mathcal{H}|$, to train the model and evaluate its performance.

To ensure robustness in the evaluation, randomized search will becombined with cross-validation in our analysis.

5 Decision Trees

Decision trees, foundational in the realm of tree-based methods, operate by partitioning the predictor space into distinct, box-shaped regions. Each bifurcation within the tree signifies a decision rule applied to predictions. One of the salient strengths of decision trees is their inherent interpretability, allowing for clear insights into the decision-making process. However, they occasionally lag in performance when juxtaposed with more sophisticated algorithms.

In constructing decision trees, a greedy top-down strategy is employed. This approach circumvents the exhaustive exploration of all conceivable partitions, an endeavor tantamount to tackling an NP-Hard problem. Instead, the tree evolves from the root to the leaves, at each juncture selecting the most advantageous split based solely on the current state. For classification challenges, purity metrics, such as the Gini index or cross-entropy, are pivotal. These metrics assess the homogeneity of a region, with heightened purity observed when a region is predominantly populated by a singular class.

The Gini index is defined as:

$$\text{Gini} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

And the cross-entropy is given by:

$$D = -\sum_{i=1}^{c} p_i \log_2 p_i$$

A pivotal consideration during tree construction is its depth. An overly expansive tree risks overfitting, capturing noise rather than underlying patterns. Conversely, an overly pruned tree may underfit, missing critical patterns. In this analysis, the purity metric (Gini or entropy) and other hyperparameters were optimized via cross-validation with randomized search.

Let's see in Figure 7 below the algorithm results in terms of precision and F1-Score.

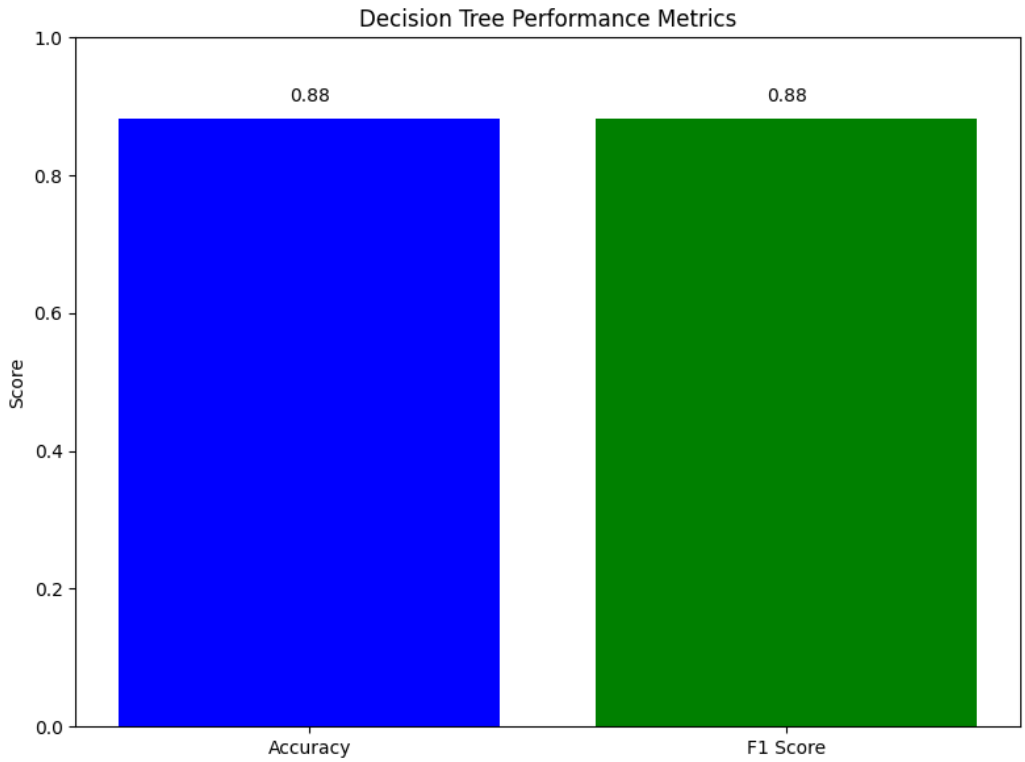

Figure 7: Decision Tree Performance

For methods such as Decision Trees and Random Forest, applying PCA before modeling may be a significant disadvantage as interpretability is lost. Therefore, for this classification task, PCA will not be applied for the above mentioned methods.

6 Random Forest

Ensemble methods involve training multiple classifiers and subsequently aggregating their predictions. In the context of decision trees, a foundational approach is to train a set of trees on distinct portions of the training dataset, often referred to as *bootstrapped training sets*. Classification of test points is then achieved by adopting a majority vote strategy. This methodology is termed *bagging*.

Random forests enhance the bagging technique by introducing an additional layer of randomness. Specifically, the feature selected at each split in the construction of individual trees is chosen from a randomly sampled subset of features. This ensures that the trees within the forest are decorrelated, enhancing the model's robustness.

Mathematically, given a set of features F, at each split, a subset $f \subseteq F$ is randomly selected, where $|f| < |F|$. The best split feature is then determined from this subset.

A pivotal hyperparameter in random forests is the number of trees, often denoted as n_{trees}. The optimal value for this hyperparameter is typically ascertained through cross-validation.

In Figure 8 below, we can notice how the performance of the Random Forest classifier is noticeably higher than the one produced by Decision Tree.

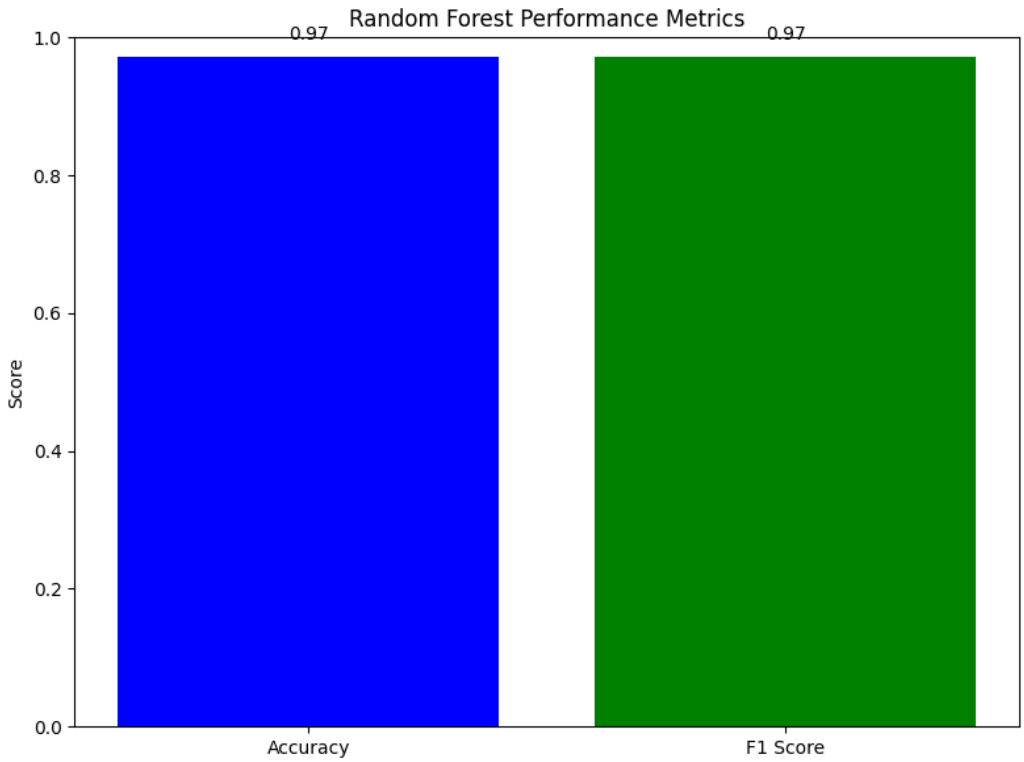

Figure 8: Random Forest Performance

7 K-Nearest Neighbor

The k-nearest neighbor (k-NN) classifier is a non-parametric, instance-based learning algorithm. Given a new, unseen observation, k-NN identifies k training samples that are closest to the point and returns the output value (class label) that has the majority among its k nearest neighbors. Mathematically, the decision function for a new point x can be represented as:

$$y(x) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(x)} y_i$$

where $\mathcal{N}_k(x)$ denotes the set of k points closest to x and y_i is the output value of the i^{th} point.

Since the method relies on the concept of distance it suffers from the curse of dimensionality and PCA might be useful in addressing this problem.

The curse of dimensionality is a consequence of computing distances in high dimensional spaces where most of volume tend to be void and the data become so sparse that the notion of similarity based on distance become statistically insignificant.

For this method, we will compare results of the classification performed on the original dataset and the dataset with PCA applied (10 principal components kept).

We can notice in Figure 9 that in practice applying PCA in our instance reduces the accuracy and F1-Score.

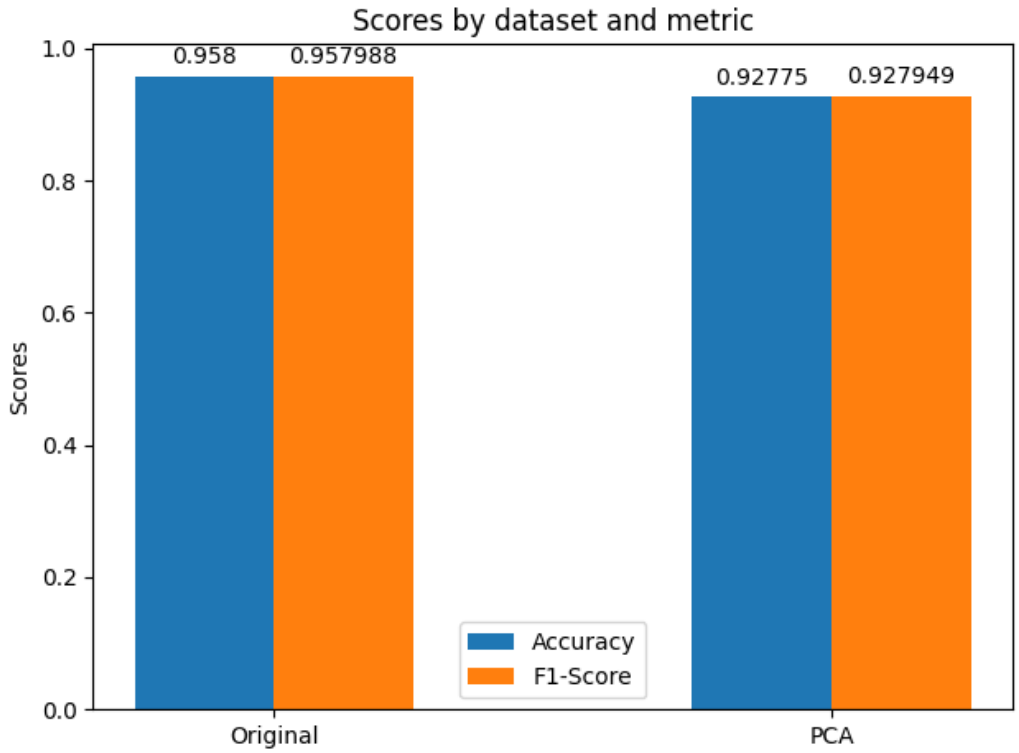

Figure 9: KNN Performance

8 Support Vector Machines

Support Vector Machines (SVM) were originally conceived for linearly separable data. In such scenarios, an hyperplane exists that can distinctly partition data points into two regions, each corresponding to a class label. Given that multiple hyperplanes can achieve this separation, SVM seeks the one that maximizes the margin. This margin is defined as the minimum perpendicular distance between the data points and the hyperplane. The data points lying closest to this hyperplane, which influence its orientation and position, are termed *support vectors*. These vectors are pivotal as the classifier's decision boundary is contingent upon them.

The objective of SVM is to find the hyperplane while maximizing the margin. This can be mathematically represented as:

$$\min_{w,b} \left(\frac{1}{2} ||w||^2 \right)$$

subject to the constraint:

$$y_i(w \cdot x_i + b) \geq 1,$$

where the constraint ensures correct classification of all data points. If a data point lies on the correct side of the hyperplane, the product $y_i(w \cdot x_i + b)$ will always be greater than or equal to 1.

For non-linearly separable data, SVM has been extended using the concept of a *soft margin*. This is achieved by introducing slack variables, ξ_i, for each data point. The modified objective function becomes:

$$\min_{w,b} \left(\frac{1}{2} ||w||^2 + C \sum_{i=1}^{m} \xi_i \right)$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0.$$

Here, C is a crucial hyperparameter that balances the trade-off between maximizing the margin and minimizing classification errors. A larger C prioritizes classification accuracy, potentially at the expense of a smaller margin, while a smaller C allows for some classification errors in favor of a larger margin.

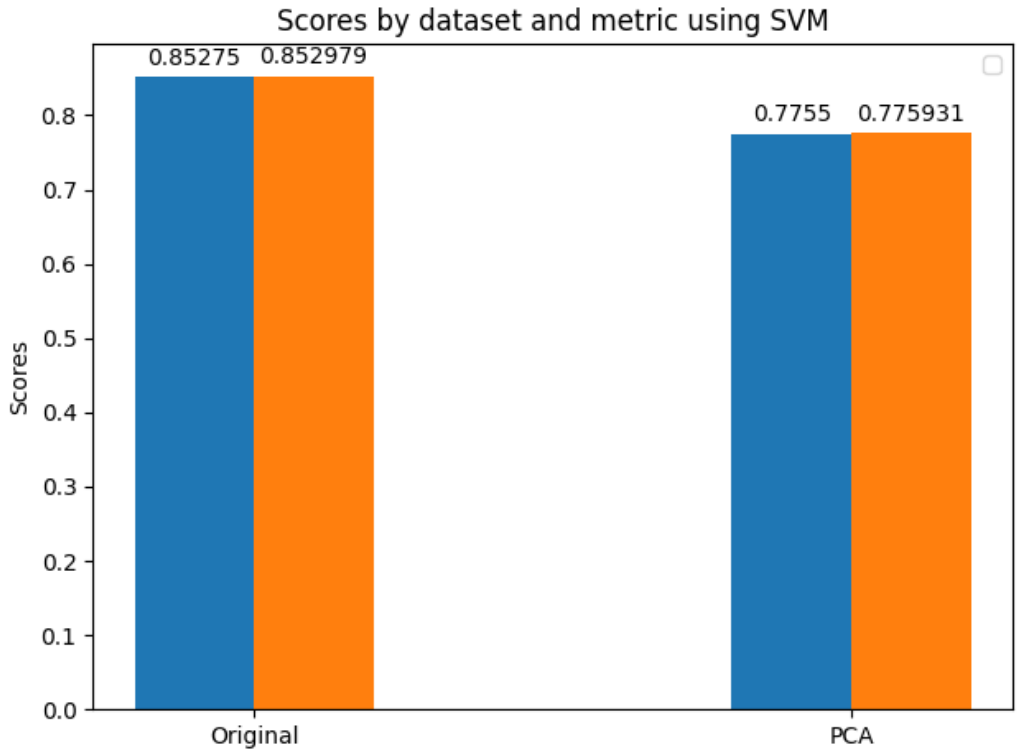

Figure 10: SVM Performance

9 RBF Kernel SVM

Support Vector Machines (SVM) with the Radial Basis Function (RBF) kernel offer a powerful approach for classifying non-linear data. The primary idea behind SVM remains consistent: to find a hyperplane that distinctly partitions data points into regions corresponding to class labels. However, in scenarios where data isn't linearly separable in the original feature space, the RBF kernel transforms the data into a higher-dimensional space where such separation becomes feasible.

The RBF kernel is defined as:

$$K(x, x') = \exp\left(-\gamma ||x - x'||^2\right),$$

where γ is a parameter that determines the shape of the decision boundary.

Given the potential for multiple hyperplanes to achieve this separation, SVM seeks the one that maximizes the margin, defined as the minimum perpendicular distance between the data points and the hyperplane. The data points lying closest to this hyperplane, termed *support vectors*, are pivotal as they influence its orientation and position.

The objective of SVM with the RBF kernel is similar to the linear case but operates in the transformed feature space. The objective is to find the hyperplane while maximizing the margin, subject to constraints ensuring correct classification of all data points.

For non-linearly separable data, the concept of a *soft margin* is introduced using slack variables, ξ_i, for each data point. The modified objective function becomes:

$$\min_{w,b} \left(\frac{1}{2}||w||^2 + C \sum_{i=1}^{m} \xi_i \right)$$

subject to:

$$y_i(w \cdot \phi(x_i) + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0,$$

where ϕ represents the transformation induced by the RBF kernel. Here, C is a crucial hyperparameter that balances the trade-off between maximizing the margin and minimizing classification errors.

We can see in Figure 11 below that RBF Kernel SVM clearly outperforms regular SVM in our instance.

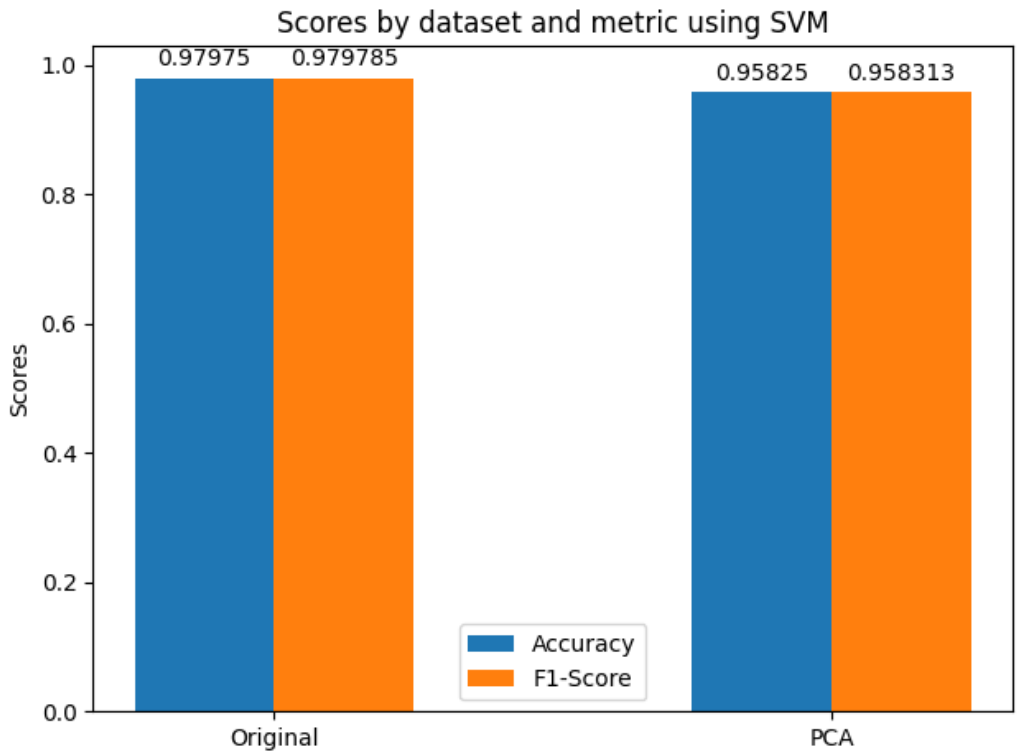

Figure 11: RBF Kernel SVM Performance

10 Logistic Regression

While linear regression can be adapted for classification tasks by interpreting the output as the probability of a data point belonging to a particular class, it has inherent limitations. Specifically, linear regression can predict values outside the [0, 1] range, which are not valid probabilities. To address this, logistic regression is employed, ensuring that predicted probabilities are confined within the [0, 1] interval.

In logistic regression, the probability $p(X)$ that an instance X belongs to a particular class is modeled as:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Equivalently, the log-odds or the logit transformation of $p(X)$ is a linear function of X:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

The parameters β_0 and β_1 are determined using the training data through Maximum Likelihood Estimation (MLE). The objective of MLE is to maximize the likelihood, which is the probability of observing the given set of data.

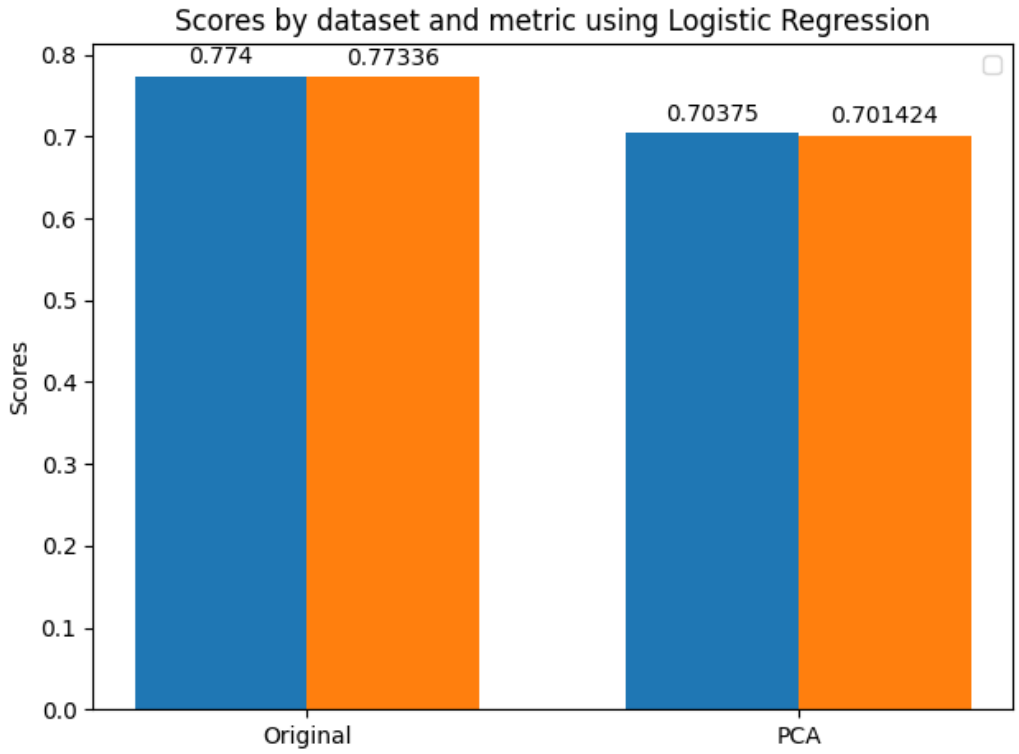

Figure 12: Logistic Regression Performance

11 Neural Networks

Neural networks, a foundational element of deep learning, are machine learning algorithms that draw inspiration from the human brain's structure. They are composed of layers of interconnected nodes, commonly referred to as "neurons." Due to their ability to process large datasets and identify intricate patterns, neural networks are often chosen for sophisticated classification tasks.

For the 'lettr' prediction task, a feedforward neural network was constructed using the TensorFlow and Keras libraries. The architecture of this network is as follows:

- **Input Layer:** Corresponding to the number of features in the dataset.
- **Hidden Layers:** Two layers, each with 50 neurons, employing the Rectified Linear Unit (ReLU) activation function.
- **Dropout Layers:** Introduced after each hidden layer with a dropout rate of 0.5 to prevent overfitting.
- **Output Layer:** Comprising nodes equal to the number of unique letters in the dataset, using a softmax activation to predict the probability distribution over the classes.

The model was trained using the backpropagation algorithm, optimizing the weights based on the sparse categorical cross-entropy loss function. A randomized search was conducted over a range of hyperparameters, including optimizers, neuron counts, dropout rates, batch sizes, and epochs, to identify the optimal configuration.

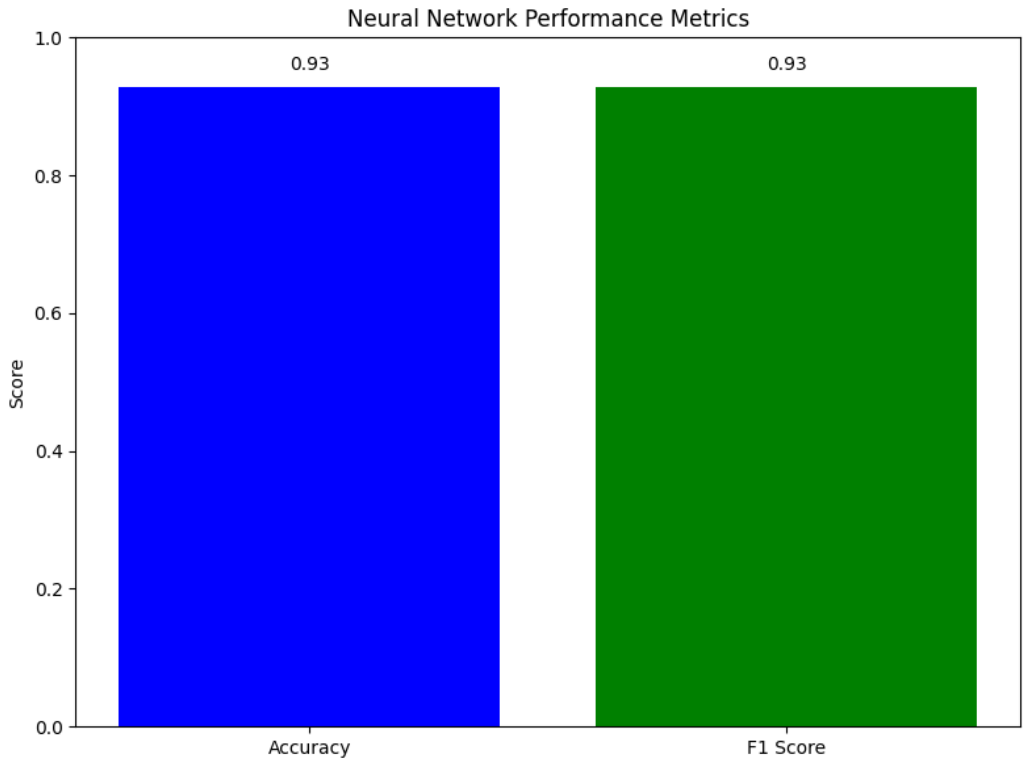

Figure 13: Neural Network Performance

As illustrated in Figure 13, the neural network achieved an accuracy of 0.93 and an F1-score of 0.93. This commendable performance indicates the network's proficiency in discerning the intricate relationships between the features and the target variable.

12 Comparison of Classification Algorithms Results

We evaluated several classification algorithms to predict the attribute 'lettr'. The performance metrics considered were accuracy and F1-score. Here's a detailed comparison of the results:

Decision Tree

The decision tree achieved an accuracy and F1-score of 0.88. This suggests that the decision boundaries for the classes in the dataset might be relatively simple and axis-aligned, which decision trees handle well.

Random Forest

Random forest outperformed the decision tree with an accuracy and F1-score of 0.97. This improvement can be attributed to the ensemble nature of random forests, which aggregates predictions from multiple decision trees, reducing variance and potential overfitting.

K-Nearest Neighbors (KNN)

KNN achieved an accuracy of 0.958 and an F1-score of 0.957988 without PCA. However, with PCA, the accuracy dropped to 0.92775 and the F1-score to 0.927949. The reduction in performance with PCA suggests that some important variance, crucial for classification, might have been lost during dimensionality reduction.

Support Vector Machine (SVM)

Without PCA, SVM achieved an accuracy of 0.85275 and an F1-score of 0.852979. With PCA, the performance dropped to an accuracy of 0.7755 and an F1-score of 0.775931. This decline indicates that the linear SVM might struggle with the transformed feature space post-PCA.

SVM with RBF Kernel

The RBF kernel SVM showed impressive results with an accuracy of 0.97975 and an F1-score of 0.979785 without PCA. With PCA, the accuracy was 0.95825 and the F1-score was 0.958313. The high performance suggests that the data, when transformed into a higher-dimensional space by the RBF kernel, becomes more separable, allowing the SVM to find an optimal hyperplane.

Logistic Regression

Logistic regression achieved an accuracy of 0.774 and an F1-score of 0.77336 without PCA. With PCA, the performance metrics dropped to an accuracy of 0.70375 and an F1-score of 0.701424. The relatively lower performance of logistic regression, compared to other algorithms, might be due to its linear decision boundary, which might not be optimal for this dataset.

Neural Network

The feedforward neural network, designed with two hidden layers and dropout regularization, achieved an accuracy of 0.93 and an F1-score of 0.93 without PCA. The commendable performance of the neural network indicates its capability in discerning the intricate relationships between the features and the target variable.

Overall Comparison

Random forests, SVM with the RBF kernel, and the neural network emerged as the top-performing algorithms. The ensemble nature of random forests, the ability of the RBF kernel to handle non-linear data, and the neural network's deep learning capabilities likely contributed to their superior performance. PCA, while reducing dimensionality, led to a drop in performance for most algorithms, suggesting that careful feature selection or extraction might be more beneficial for this dataset.

Algorithm	Without PCA		With PCA	
	Accuracy	F1-Score	Accuracy	F1-Score
Decision Tree	0.88	0.88	–	–
Random Forest	0.97	0.97	–	–
KNN	0.958	0.957988	0.92775	0.927949
SVM	0.85275	0.852979	0.7755	0.775931
SVM (RBF Kernel)	0.97975	0.979785	0.95825	0.958313
Logistic Regression	0.774	0.77336	0.70375	0.701424
Neural Network	0.93	0.93	–	–

Table 1: Comparison of classification algorithms on the 'lettr' prediction task.

13 Conclusions

The objective of this study was to classify black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet based on 16 primitive numerical attributes. We explored a range of classification algorithms, including Decision Trees, Random Forests, K-Nearest Neighbors, Support Vector Machines, Logistic Regression, and Neural Networks. Each classifier was subjected to hyperparameter tuning using randomized search combined with cross-validation.

Our findings indicate that ensemble methods, specifically Random Forests, SVM with the RBF kernel, and the neural network, outperformed other algorithms. The ensemble nature of random forests, which aggregates predictions from multiple decision trees, the ability of the RBF kernel to handle non-linear data, and the deep learning capabilities of the neural network likely contributed to their superior performance.

Dimensionality reduction using PCA, while reducing the number of features, led to a drop in performance for most algorithms. This suggests that while PCA can be beneficial in certain scenarios, careful feature selection or extraction might be more beneficial for this dataset.

References

[1] James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). *An introduction to statistical learning*. New York: Springer.