Bennett Brain and Anthony Testa

ML Final Project Report

Feature Engineering on the IRIS Dataset

Abstract

We explored possible ways to achieve high performance on the IRIS dataset- a limited classification dataset with only 3 categories, 4 features, and 150 samples. The motivation was exploring how to work with very small datasets, which is why we did not opt for a larger, more comprehensive dataset. We used three classification techniques - K Nearest Neighbors (KNN), Logistic Regression (LR), and a 2-layer Neural Network (NN). We used three dimensionality-reduction/transform techniques – Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and t-Distributed Stochastic Neighbor Embedding (t-SNE) plus a control with none applied. We also did feature engineering to create more features derived from the original four, and then took three subsets of that expanded dataset, plus the original dataset, resulting in 4 different data sets to test on. Across all of these experiments, Neural Networks proved very unreliable, while KNN was our best classifier with LR slightly behind. Expanding the dataset improved performance, and using dimensionality reduction/transformation techniques provided marginal improvements.

Overview

The IRIS dataset is one of the oldest classification dataset [1] It is very small, with only 3 categories (setosa, versicolor, and virginica), 4 features (Length and Width for both the Petal and Sepal), and 150 samples. We set out to see which techniques could do well at tackling classification in such a limited space. While many problems today involve massive quantities of data, sometimes data collection can be challenging, and developers will have to work with small data sets. In those cases, throwing a solution built for large-scale data may prove ineffective, as we show later on with Neural Networks here.

Since our focus here was more on exploration than simply achieving the highest possible accuracy, we focused on testing many possible routes to tackling this problem. We tried multiple classifiers, variations on the dataset, and dimensionality transforms. More details are in the Experiment Setup section, but it totaled 48 total experiment types, each of which we ran multiple times to get mean and variance. We measured train accuracy, test accuracy, and training time, but were mostly focused on test accuracy since that is typically the metric of how well a model generalizes on future data.

We settled on this approach because we wanted to get a broad sense of which methods to use when approaching small datasets like IRIS. With infinite time and resources we could test far more classifiers, dimensionality transforms, methods of feature engineering, etc, but we believed this would balance runtime and time constraints with sampling a broad swathe of approaches. For the classifiers, Logistic Regression is a much higher-bias, lower-variance method, while Neural Nets are the opposite, and K Nearest Neighbors is somewhere in between. For the dimensionality transformations, PCA and LDA are both linear while t-SNE is non-linear, so we have a good range there as well. And for feature expansion, the three expanded datasets are of varying sizes, with the unaltered base dataset being the smallest, so we have breadth in that aspect as well.

One limitation to our approach is that if there are any implementation specifics that could be optimized, they may slip by. For instance, if KNN with k = 7 is better than k = 3 for our purposes, our approach will not catch that. Since we already are testing such a large cross-section of methods, varying one method even more will drastically increase runtime and the quantity of results. That said, our goal here is not to simply fit the IRIS dataset optimally, but to explore which methods are best for small data sets, so getting a general idea of which methods perform well without finding a global maximum is still in line with our goals.

Experiment Setup

We tried three different classifiers- Neural Networks, Logistic Regression, and K Nearest Neighbors. Specifically, we used a Neural Network with 5 hidden layers of 2 neurons each utilizing the lbfgs optimizer for fitting. For Logistic Regression we used an L2 penalty term, and the liblinear optimizer. KNN used k = 3 neighbors and uniform distance weighting. We used the built-in classifiers from SciKit [2].

We also expanded the dataset by adding 15 features:

Feature	Variable Name	Calculation
Petal Area	petal_area	Petal length * petal width
Sepal Area	sepal_area	Sepal length * sepal width
Square Root of Petal Area	sqrt_petal_area	Square root of above
Square Root of Sepal Area	sqrt_sepal_area	Square root of above
Petal Aspect Ratio	petal_aspect_ratio	Petal length / petal width
Sepal Aspect Ratio	sepal_aspect_ratio	Sepal length / sepal width
Petal/Sepal Aspect Ratio	ps_area_ratio	Petal area / sepal area
Petal/Sepal Length Ratio	ps_length_ratio	Petal length / sepal length
Petal/Sepal Width Ratio	ps_width_ratio	Petal width / sepal width
Petal/Sepal Length Difference	length_difference	Petal length – sepal length
Petal/Sepal Width Difference	width_difference	Petal width – sepal width
Petal Length Squared	petal_L2	Petal length * petal length
Petal Width Squared	petal_W2	Petal width * petal width
Sepal Length Squared	sepal_L2	Sepal length * sepal length
Sepal Width Squared	sepal_W2	Sepal width * sepal width

All of these features are derived from the original four. Inspiration for these features was taken from the Expanded Iris Dataset [3] which we found in our initial research on the project, but we built some of our own on top of this. In order to test the effects of expanding the dataset, we took three subsets of this- one with all the expanded features (“edf”), one with only the ratios (“edf_ro”), and one with everything but the ratios(“edf_2”). Additionally, we included the original, un-expanded dataset as a control (“data”).

In addition to this feature engineering, we wanted to perform various dimensionality reduction/transform techniques on our dataset. We also wanted to implement these algorithms ourselves, to increase our understanding of the process and retain more control over the results. Principal component analysis was relatively straightforward, simply being the eigen decomposition of the covariance matrix of our dataset. We first centered the data about the mean. It is important to note, that all measurements are within the same range prior to any transform, essentially removing the necessity for normalizing (as the data is already normalized). This is incredibly important for all methods performed here, but especially PCA and LDA as they look to maximize scatter.

LDA was also straightforward, simply computing the class-wise scatter of the data before performing the eigen decomposition of the dot product between this and the covariance matrix of the dataset. For both methods we then reduced dimension by choosing the top features according to the eigen values associated with these components.

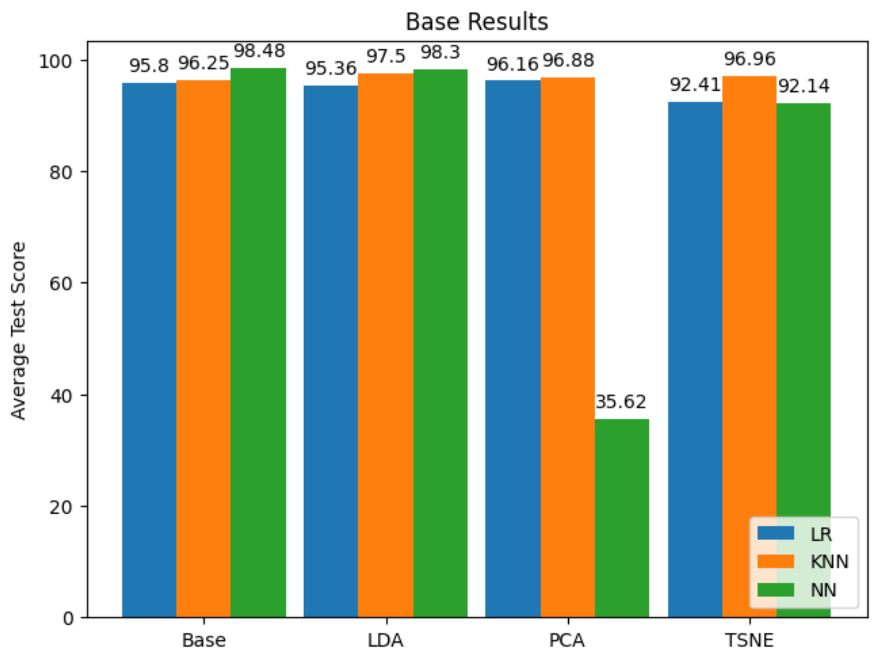
t-SNE was a little more complicated and followed Laurens Van Der Maaten's implementation [4] along with guidance from professors and other articles. It computes the pairwise distance probability based on the distance between any two points, then initializes an identical matrix for the 2-dimensional representation. We then implemented gradient descent with early exaggeration to minimize the Kullback Liebler divergence between these two matrices, in order to create the embedding into two dimensional space.

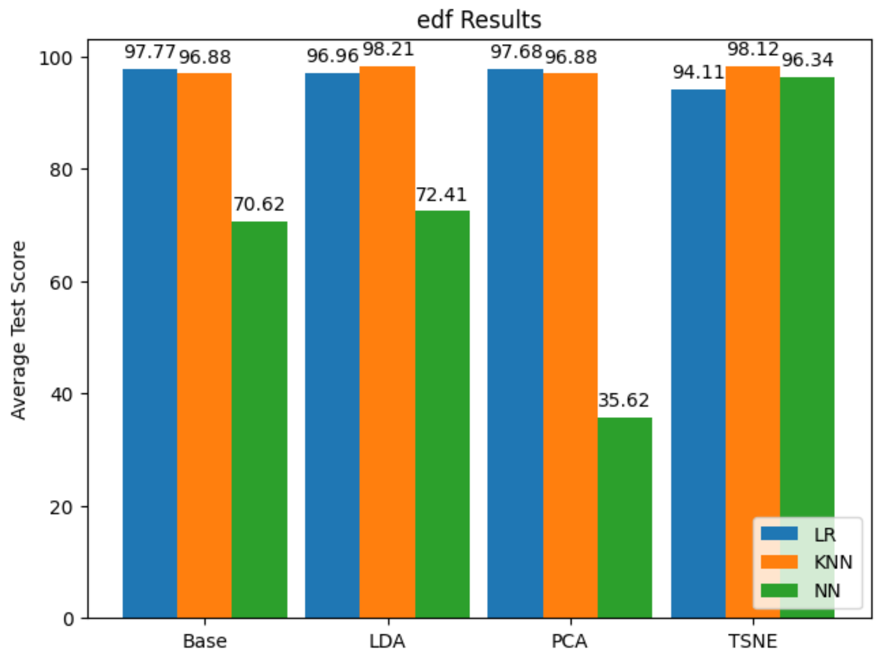
For each dimensionality technique (plus control), dataset, and classifier, we ran the associated combination to see how well they performed. This results in 4x4x3 = 48 experiments to run. We kept the train/test split the same across all experiments to keep the results comparable. However, we also wanted to see how reliable these results were, so we took this large run and wrapped it in a loop (with each iteration of the loop using a different train/test split) that ran 10 times. This creates 480 total experiments, and for each of the original 48 we took the mean and variance of its multiple runs.

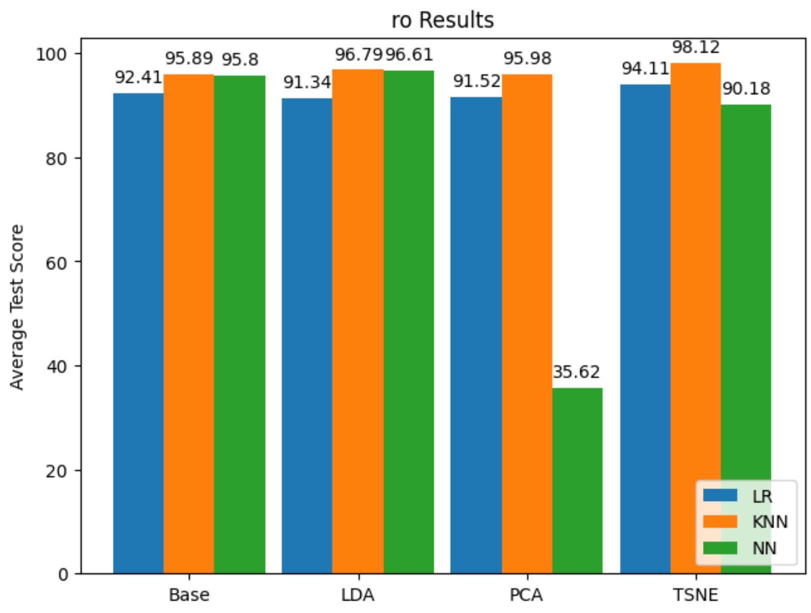
We ran all this on a jupyter notebook, with some helper functions in a python file called "functions.py". Since the dataset is small, the run time (even with 480 total experiments to run) was manageable.

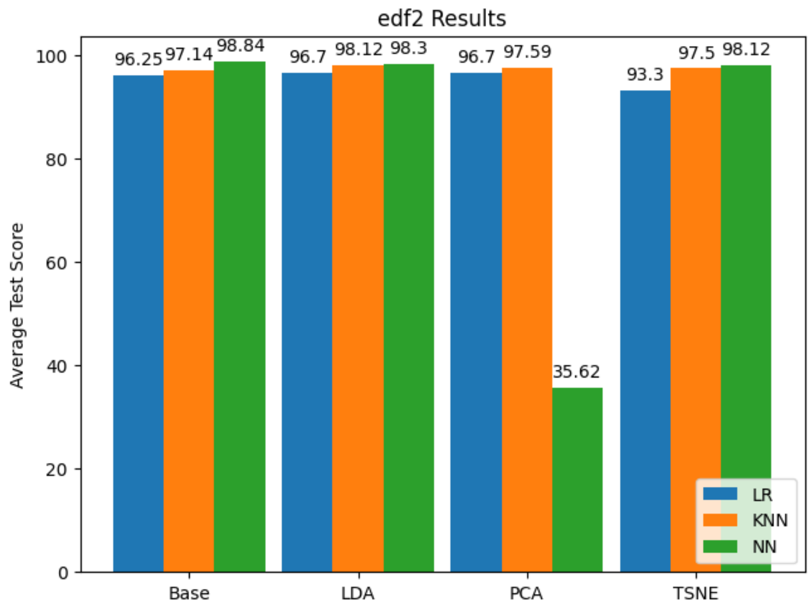
Experiment Results and Discussion

The test accuracy graphs have been reproduced below:


Base Results
Average Test Score
0
20
40
60
80
100
95.8
96.25
98.48
95.36
97.5
98.3
96.16
96.88
35.62
92.41
96.96
92.14
Base
LDA
PCA
TSNE
LR
KNN
NN


edf Results
Average Test Score
0
20
40
60
80
100
97.77
96.88
70.62
96.96
98.21
72.41
97.68
96.88
35.62
94.11
98.12
96.34
Base
LDA
PCA
TSNE
LR
KNN
NN





The best classifier overall was KNN, across multiple different environments. It had the most consistently high performance, never dropping below 95.89% accuracy with most results a few percentage points higher. However, Logistic Regression was not far behind, doing only slightly worse in most environments with an overall minimum of 91.34%. Neural Networks had the worst overall performance- occasionally doing well but with a few massively steep drops with test accuracies down as low as 35.6%. This goes to show that Neural Networks, while they may sometimes do well on small data sets, are highly unreliable and susceptible to overfitting on smaller datasets like the IRIS data set. Meanwhile, K Nearest Neighbors seems to do very well in this environment.

The ratios-only dataset did not perform well overall, generating the lowest performance for Logistic Regression in particular and doing slightly worse for most cases of KNN. The full expanded dataset had the best performances on average for both Logistic Regression and KNN, though it had the worst for

Neural Networks. This makes sense, as adding more features can cause overfitting with Neural Networks to happen faster, while KNN and Logistic Regression have the ability to find stronger signals in the data, but since they are less variable than Neural Networks, adding more flexibility with larger amounts of features does not cause them to overfit yet. The final data set, which was essentially the base data set with a few added features, did slightly better than the base across KNN and Logistic Regression, but not as well as the full expanded data set. This further supports the previous observation, as adding more derived features only served to help KNN and Logistic Regression.

As for the dimensional transformation techniques, there were some mixed results. KNN worked well with both LDA and t-SNE, the latter of which is an expected result since t-SNE essentially attempts to create neighborhoods in the data in low-dimensional space. Logistic Regression seemed to work best with PCA and the unaltered data set. Neural Network performance was unpredictable at best, but it did especially poorly with PCA. The exact reason for this is unclear, and if there were more time it would be prudent to conduct many more trials and permutations before concluding anything particular about Neural Network performance in this environment, as the variance was so high that it's difficult to extrapolate from averaged results. Overall, the dimensional transforms were moderately helpful for KNN, which already was our best performing classifier, so it seems that these transforms can certainly be a useful tool when working with small data sets.

Finally, in addition to producing the best performance, KNN trained the fastest out of all methods, significantly more so than Neural Networks.

Conclusion

To work with small data sets, a variety of methods should be attempted. But methods with lower variance, such as KNN and Logistic Regression, will produce more consistent results than highly flexible models such as Neural Networks. Adding derived features can help those models perform better, and while doing so may cause Neural Networks to overfit, such higher-variance models probably should not be used for small-data problems in the first place. Dimensionality transforms such as PCA and t-SNE may be helpful for small data sets but should not be taken as a given. If time allows, testing both with and without those alterations is prudent. It is impossible to recommend a single classifier or method that should be used for all small data sets - the idiosyncrasies of each data set change which methods will be best - but the general principle of keeping to more legible, lower-variance models is broadly applicable.

References

[1] Fisher,R. A.. (1988). Iris. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76.

[2] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[3] Samy Baladram. (2023). <i>🌺 Iris Dataset Extended</i> [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DS/3916784

[4] https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf