ECE 471: Intro to Pattern Recognition
Project 4

Neural Network, Decision Tree, and Performance Evaluation

Alec Yen
University of Tennessee, Knoxville
Electrical Engineering and Computer Science Department
ayen1@vols.utk.edu

1. Abstract

In this report, procedures for KNN, decision trees, and neural networks were implemented and tested on a forensic glass dataset. All algorithms were evaluated for performance using m-fold cross validation and an 80-10-10 split was also tested on a 3-layer neural network. Each learning technique did similarly well, although KNN performed slightly better than decision trees and neural networks for this dataset. Smaller values of k appeared to achieve higher accuracy than larger values. M-fold cross validation also showed that for the neural network, using more nodes improved the accuracy of the network.

2. Introduction

Some of the most popular pattern classification techniques include decision trees and neural networks. Both fall under the category of supervised learning. However, their methodologies differ from past techniques. Decision trees aims to differentiate elements of a dataset using property queries from parent to child nodes. Neural networks are biologically-inspired models based on the fundamental building blocks of the human brain (perceptrons, axons, dentrites, etc.).

Decision trees have been used in a variety of applications, including modelling rainfall predictions [1] and even evaluating a nation's judicial system [2]. While decision trees have practical applications, neural networks are what has truly caught the eye of the pattern classification field in recent decades. Their usage has been extensive and ubiquitous including applications from reading facial expressions [3] to fault diagnosis [4].

The objective of this project was to implement cross validation for the performance evaluation of KNN, decision trees, and a 3-layer neural network on a forensic glass dataset. KNN was implemented from scratch and the decision tree and neural network were implemented using `scikit-learn` and `keras`, respectively.

3. Technical Approach

3.1. KNN

K-nearest neighbors (KNN) was implemented in a previous project. This algorithm was reused for the purpose of testing cross validation; a summary of the methodology follows.

In KNN, a test sample's Euclidean distance from each training sample is measured. The k nearest training samples' class labels are found and the majority is assigned to the test sample. The posterior probability of a tests sample is justified as in (1), where k_i are the k closest training samples of class i. n_i is the total number of training samples of class i and n is the total number of training samples, and thus $\frac{n_i}{n}$ represents the prior probability. Thus, KNN assumes a prior probability based on the distribution of the training data. If we wish to alter the prior probability to a new desired prior probability

of $P(\omega_i)'$, we must multiply $\frac{k_i}{k}$ by $\frac{P(\omega_i)'}{P(\omega_i)}$. KNN is non-parametric because we do not need a model to calculate maximum posterior probability and perform classification.

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})} = \frac{\frac{k_i/n_i}{V}\frac{n_i}{n}}{\frac{k/n}{V}} = \frac{k_i}{k} \tag{1}$$

3.2. Decision Trees

Decision trees are a non-statistical approach, unlike KNN. All samples start at the root node and are divided up and classified as one progresses through the tree. At each node N, a property query is made to distinguish each sample into two groups. The impurity $i(N)$ of such a decision can be measured using either entropy impurity (2) or Gini impurity (3). Regardless of which type to use, the objective of decision trees is to maximize the change in impurity from each node to the next layer. This change in impurity is defined as in (4). Decision trees in the project were implemented using `scikit-learn`'s `DecisionTreeClassifier` class.

$$i(N) = -\sum_j P(\omega_j)\log_2 P(\omega_j) \tag{2}$$

$$i(N) = 1 - \sum_j P^2(\omega_j) \tag{3}$$

$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L)i(N_R) \tag{4}$$

3.3. Neural Networks

The fundamental building block of neural networks is the perceptron (a single layer network), which are inspired by the neurons of the human brain. They are composed of inputs $\vec{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_d & 1 \end{bmatrix}$ and weights $\vec{w} = \begin{bmatrix} w_1 & w_2 & \dots & w_d & -w_0 \end{bmatrix}$, where w_0 is the bias. The output $z = \vec{w}^T\vec{x}$. If $z > 0$, the perceptron outputs 1. Otherwise, it outputs 0. Assuming the ground truth is $\vec{T}$ and the network's output is $\vec{z}$, gradient descent can be used as in (5) to converge to a solution.

$$\vec{w}^{k+1} = \vec{w}^k + \sum_{i=1}^{n}(T_i - z_i)x_i \tag{5}$$

A 3-layer neural network is simply the combination of these perceptron units with the addition of backpropagation. In this project, the `keras` library was used to implement the neural network.

3.4. Cross Validation

One of the primary goals of this project is to implement a general procedure for m-fold cross validation of a classifier. This serves not to help build a classifier, but rather to

evaluate a classifier's performance and maximize the data available. In this process, the dataset is partitioned into m sets. $m-1$ sets are used for training the classifier while the remaining 1 set is used for evaluating the classifier. The process is then repeated m times so that each set is used for testing at least once. The overall accuracy of the classifier is the average of the accuracies found on each of the m test sets.

4. Experimentation and Results

All algorithms were seeded with fixed seeds for reproducibility. The dataset was also normalized before each test.

4.1. The Dataset

The `fglass` data set from Ripley's book was used [5]. It is a dataset that classifies different types of glass (window glass, containers, tableware, etc) using various numerical features of the glass, including the refractive index and the weight of different oxides (sodium, magnesium, etc). In total, there are 9 features and 6 possible classes (numbered 1 through 7 and omitting 4). There were 214 total samples and [5] also provided the indexes for ten groups for usage in cross-validation experiments.

4.2. Cross Validation and KNN

The first task was to implement cross validation using KNN, where $m = 10$. The cross-validated accuracy for each k is plotted in Figure 1. The exact values are shown in Table 1. Through m-fold cross validation, it was found that k = 2 performed the best overall. The standard deviation of the accuracies while performing cross validation is shown in Figure 2 and Table 2.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.68	0.70	0.69	0.66	0.61	0.63	0.63	0.65	0.63	0.64	0.62	0.64	0.63	0.64	0.64

Table 1: Cross validation accuracy of KNN at different values of k

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.06	0.06	0.13	0.08	0.10	0.13	0.09	0.11	0.09	0.08	0.11	0.10	0.08	0.07	0.07

Table 2: Cross validation standard deviation of accuracy of KNN at different values of k

4.3. Cross Validation and Decision Tree

The second task was to use the existing cross validation procedure to evaluate the accuracy of decision tree is performing classification on this dataset. Using the `DecisionTreeClassifier`

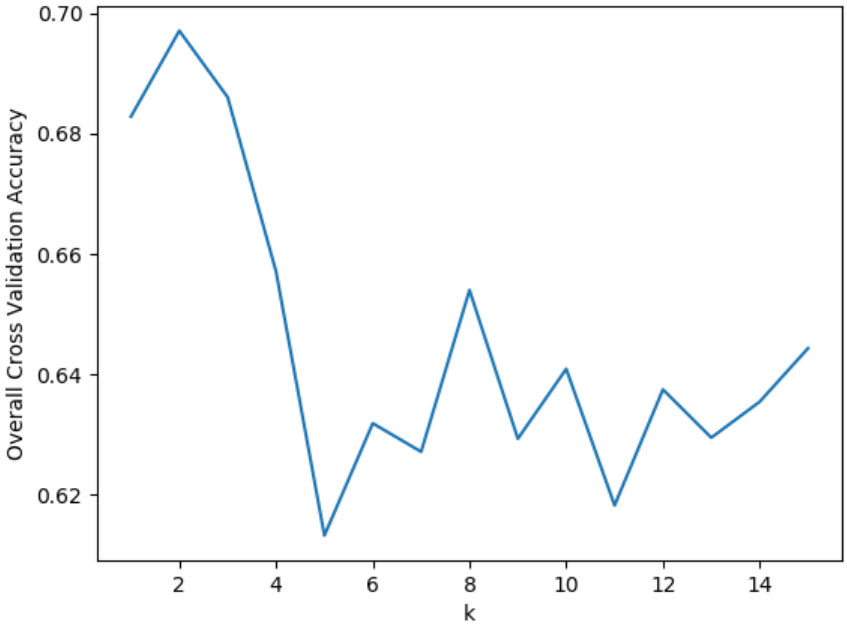

Figure 1: Cross validation accuracy of KNN at different values of k

	m=1	m=2	m=3	m=4	m=5	m=6	m=7	m=8	m=9	m=10
k=1	0.67	0.67	0.71	0.71	0.59	0.81	0.64	0.67	0.65	0.72
k=2	0.60	0.71	0.79	0.71	0.59	0.77	0.68	0.75	0.65	0.72
k=3	0.40	0.67	0.82	0.71	0.53	0.81	0.59	0.79	0.77	0.78
k=4	0.53	0.67	0.75	0.65	0.53	0.65	0.59	0.71	0.77	0.72
k=5	0.53	0.67	0.71	0.65	0.41	0.65	0.50	0.62	0.77	0.61
k=6	0.47	0.76	0.64	0.65	0.41	0.62	0.55	0.62	0.77	0.83
k=7	0.53	0.76	0.68	0.59	0.47	0.62	0.55	0.62	0.73	0.72
k=8	0.60	0.76	0.68	0.65	0.41	0.62	0.64	0.62	0.73	0.83
k=9	0.53	0.67	0.75	0.59	0.47	0.62	0.59	0.62	0.73	0.72
k=10	0.60	0.71	0.71	0.65	0.47	0.58	0.59	0.62	0.69	0.78
k=11	0.60	0.67	0.75	0.59	0.35	0.58	0.59	0.62	0.65	0.78
k=12	0.53	0.71	0.75	0.59	0.47	0.65	0.59	0.62	0.62	0.83
k=13	0.53	0.67	0.71	0.65	0.47	0.62	0.59	0.62	0.65	0.78
k=14	0.60	0.67	0.71	0.59	0.53	0.65	0.55	0.62	0.65	0.78
k=15	0.60	0.67	0.68	0.71	0.53	0.62	0.59	0.62	0.65	0.78

Table 3: Matrix of accuracy of KNN at different values of k using each group m for cross validation. Table 1 was obtained by taking the average of each row.

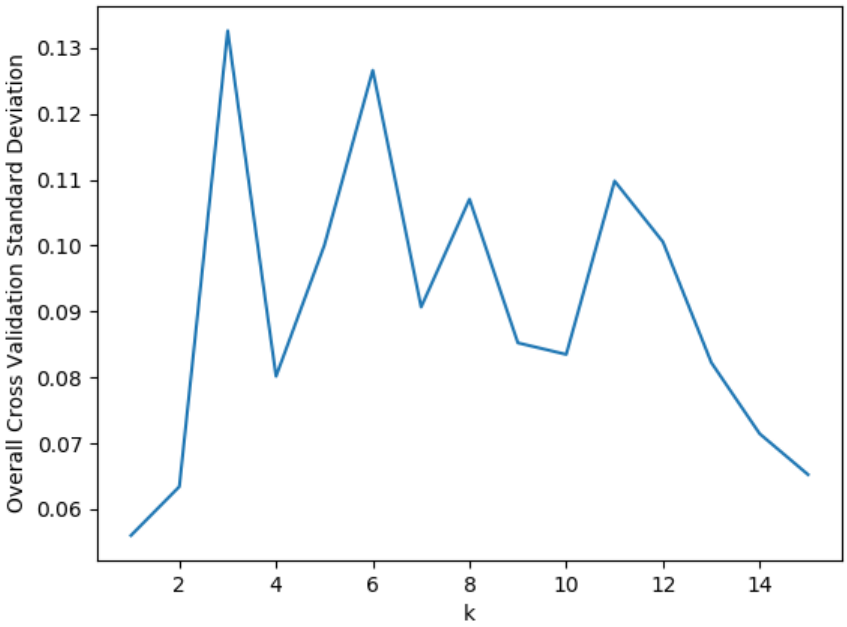

Figure 2: Cross validation standard deviation of accuracy of KNN at different values of k

class, the accuracy of the decision tree was 0.667 . The standard deviation was 0.097 . The code to implement the decision tree is seen in A.1.

4.4. 80-10-10 Split with Neural Network

The third and final task was to train a 3-layer neural network. There were 9 inputs, n nodes in the hidden layer, and 8 outputs. Although there are 6 classes, one-hot encoding is used where the outputs are zero-indexed and the one with the largest output weight is the network's decision. Because the numbers are from 1 to 7, 8 outputs are used for one-hot encoding.

We were tasked with implementing the network using 80% of data for training, 10% for validation, and 10% for testing. More specifically, of the 10 grouped sets we were provided, 9 groups were used for training/validation and 1 was used for testing. Because the seed was fixed, group $m = 8$ was used for testing each time. This turned out to be a good middle ground test set in terms of accuracy (referring to Table 3).

Using the code in A.2, a 3-layer neural network was trained with these specifications using `keras`, with varying hidden layer nodes. Using `keras`, the training set was divided into 8/9 training and 1/9 validation (for 80% and 10%). The performance curve in Figure 3 demonstrates the accuracy using varying number of neurons in the hidden layer. The performance curve in Figure 4 demonstrates the accuracy of the training set over

time.

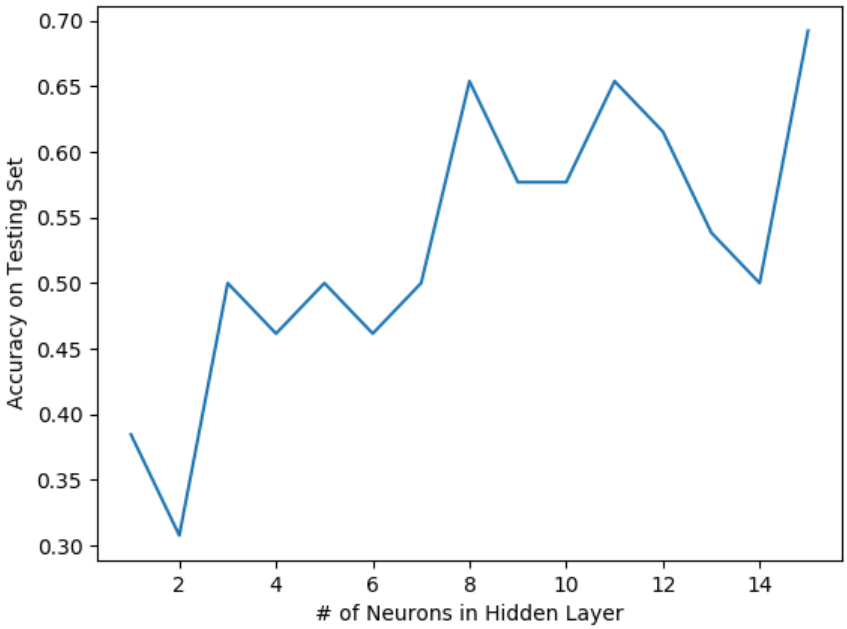

Figure 3: Accuracy using 80-10-10 split with neural network, using a variable number of hidden nodes

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.38	0.31	0.50	0.46	0.50	0.46	0.50	0.65	0.58	0.58	0.65	0.62	0.54	0.50	0.69

Table 4: Accuracy using 80-10-10 split with neural network, using a variable number of hidden nodes

5. Discussion

From the results, we can draw several conclusions regarding the accuracy of each learning technique on the dataset. KNN achieved an accuracy of 0.697 when k = 2 . The decision tree achieved an accuracy of 0.667 using cross validation. The neural network achieved an accuracy as high as 0.692 using a 15 nodes in the hidden layer.

From Table 3, it is clear that some of the groups were more difficult to test on, resulting in lower accuracies. In particular, group $m = 5$ had very low accuracies regardless of the k used. On the other hand, $m = 10$ had very high accuracy by comparison.

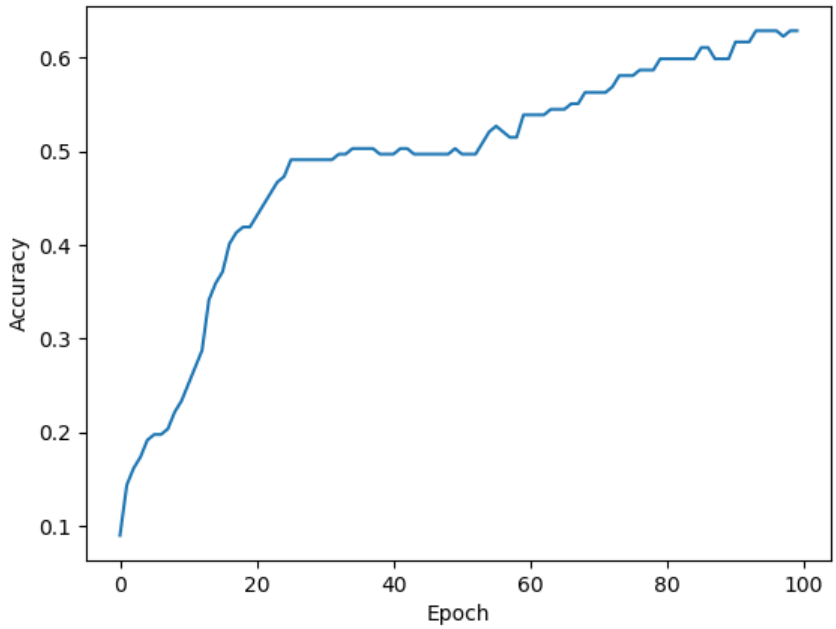

Figure 4: Performance curve of accuracy on training set using 80-10-10 split with neural network, using 15 hidden nodes, over time

Smaller values of k appeared to do better. This may indicate that in the hyperspace of the test samples, there is a large degree of mixing between the samples, so there are no clearly defined clusters. However, the strong performance of KNN indicates that the nearest neighbors did indeed serve as a good classification method.

The decision tree performed comparably to KNN. `scikit-learn`'s method achieved an accuracy of 0.667 which is slightly under than of KNN's best performance. Nonetheless, it is very comparable which suggests that the tree was able to obtain relatively high purity property queries.

A greater number of nodes in the neural network clearly helped the accuracy, as seen in Figure 3. The greater complexity allowed the network to become more nuanced and develop more decision boundaries that helped address the nonlinear nature of the data. Of course, using a larger number of nodes risks over-fitting, but that effect can be mitigated with the usage of a validation set.

6. Summary

In this project, three different classifiers have been implemented and evaluated using m-fold cross validation. It is observed that KNN has the strongest performance for smaller values of k and that decision trees can obtain a comparable accuracy. Neural networks

struggled significantly if not given enough hidden nodes, but with greater complexity, the 3-layer neural network also achieved fairly high accuracy.

The classifiers and algorithms in this project have been very thoroughly tested. The modular design and implementation of the Python algorithms benefited this process nicely. Future work on this project could be done to test other parameters of the decision tree and neural network classifiers, including using different activation functions or perhaps introducing more layers. Overall, this project served as a meaningful and instructive way to introduce students to decision trees, neural networks, and m-fold cross validation.

References

[1] A. Geetha and G. Nasira, "Data mining for meteorological applications: Decision trees for modeling rainfall prediction," in *2014 IEEE International Conference on Computational Intelligence and Computing Research*. IEEE, 2014, pp. 1–4.

[2] C.-F. Tsai and J.-H. Tsai, "Performance evaluation of the judicial system in taiwan using data envelopment analysis and decision trees," in *2010 Second International Conference on Computer Engineering and Applications*, vol. 2. IEEE, 2010, pp. 290–294.

[3] H. Kobayashi and F. Hara, "Dynamic recognition of basic facial expressions by discrete-time recurrent neural network," in *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, vol. 1. IEEE, 1993, pp. 155–158.

[4] S. K. Gupta, S. Ayub, and J. Saini, "Fault diagnosis of rc-coupled amplifier using slope fault feature and comparision with different neural networks," in *2015 Fifth International Conference on Communication Systems and Network Technologies*. IEEE, 2015, pp. 1163–1166.

[5] B. Ripley, "Pattern Recognition and Neural Networks by B.D. Ripley," 1996. [Online]. Available: http://www.stats.ox.ac.uk/pub/PRNN/

A. Appendix

A.1. Decision Tree

```
clf = DecisionTreeClassifier().fit(tr[:,:-1],tr[:,-1])
Z = clf.predict(te[:,:-1])
```

Listing 1: Implementation of decision tree

A.2. Neural Network

```
model = Sequential()
model.add(Dense(units=hidden_layer_inputs, activation='relu',
    input_dim=9))
```

```
model.add(Dense(units=8, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd',
    metrics=['accuracy'])
model.fit(x_train, y_train, epochs=100, batch_size=32,verbose=
    False,validation_split=1/9)
loss_and_metrics = model.evaluate(x_test,y_test,batch_size=128,
    verbose=False)
```

Listing 2: Implementation of 3-layer neural network using `keras` with 9 inputs, variable nodes in the hidden layer, and 8 outputs using one-hot encoding