Machine Learning and Pattern Recognition Final Project

Jose Paz Amaya
Department of Bioengineering, Temple University
tul16325@temple.edu

Introduction: For the class final, we participate in a competition where we compete to get the best performance on classifying a particular data set (*Fig. 1*). The amount of overlap between classes presents a challenge when classifying, especially given the data only has 2 dimensions. We classify the data using two different approaches: a non-neural network-based approach and a neural network based approach. I chose to use the K-Nearest Neighbor algorithm from SciKit Learn and a Multi-Layer Perceptron using TensorFlow Keras, respectively. KNN is a machine learning algorithm where decisions are based on a majority vote. It is an algorithm that is frequently used for classification since it is able to follow non-linear surfaces better than other algorithms. MLP, on the other hand, consists of a feed-forward neural network that has at least three layers of neurons. The value in MLPs comes with their ability to learn representations of the data and use mathematical mappings to predicting outputs.

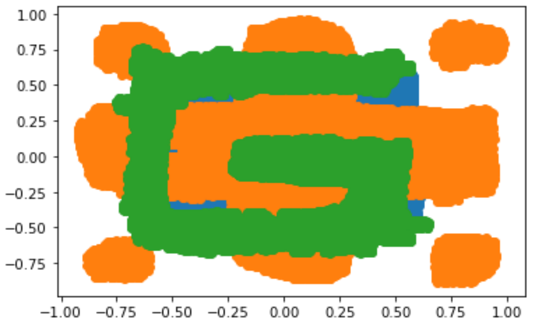
Figure 1. Three different classes in the training data set.

KNN Algorithm: Initially, I tried to stay away from KNN because of its popularity in previous semesters. However, after quickly trying out several algorithms (SVM, LDA, K-means), KNN performed best with minimal tunning, showing potential for improvement once parameters were carefully tunned. I used GridSearchCV from SciKit Learn to sweep through parameters and check performance on each combination using a 5-fold cross validation. The parameters tested were: weight function, number of neighbors, and power parameter for the Minkowski metric. Because of the range of cases that needed to be tested to find an optimal set would have taken a significant amount of time every time it was ran, I swept through range 30-150 by increments of 10 and once the optimal value was found (110), I then narrowed the search down around that value by sweeping 90-120 by increments of 1. Uniform weight function and the Manhattan power parameter were consistently giving the best performance during cross validation. Finally, the optimal number of neighbors to maximize performance was found to be 112. To obtain a parameter estimate that would generalize better, GridSearchCV ran on the training and the development data set concatenated together. Model training was done only on the training data set (with the parameters N_neighbors = 112, weight='uniform', distance_metric='Manhattan') and then the model was used to predict the training,

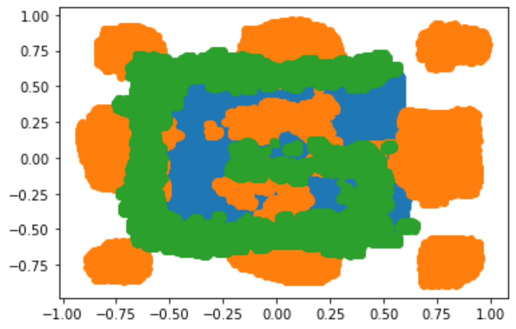
Figure 2. KNN results on training data

The same process described above was repeated after transforming the data using LDA to try to reduce the amount of overlap and increase performance. However, the results suggested overfitting since the error on the training data set was lower but the higher on the development set. From this observation, I predicted this system would perform worse than without the LDA so I stuck to KNN alone. PCA was tested again but the transformation was not significant and there was no change in performance when compared to KNN alone.

MLP Algorithm: For the neural network algorithm, MLP was implemented using TensorFlow Keras. Initially, different network architectures were tested to see what design would fit the classification problem best. As a result, a convergent network was used to construct the model, which consisted of and input layer of 2 neurons and then, 300 neurons in the first layer, 30 in the second layer and 3 in the output layer. The activation function for every layer was set to rectified linear unit (ReLu) except the last layer where it was set to SoftMax. For the optimizer, I used Stochastic Gradient Descent (SGD) since it proved to work better than Adam for my model. In addition, I tried several categorical loss functions but the one that worked best with the model was the Spare Categorical Cross Entropy. The learning rate was set to 0.01, the epoch was set to 60 and the batch size to 200 as this promoted convergence within a reasonable amount of time (~5 min.) while avoiding overfitting.

Most of these metric and parameters where found through an exhaustive search. Although, some of these parameters/metrics where initially selected according to the nature of the data set, I quickly found that accuracy could be improved by changing parameter values in ways that were not necessary logical or made sense in the context of the problem or data set. However, that is fine since we are engineers and do not lose sleep (I did) over proving or making sense out of things like these, we just look for the parameters that will yield the best performance.

Results: KNN scored 25.35% and 25.81% error on the training and development sets respectively. There was a sharp change in performance when scored on the evaluation set, where error increased to 64.30%. Using LDA to reduce de overlap in the features resulted in similar performance scoring 25.34% error on the training set and 25.95% on the development set.

	Data Set		
Algorithm	**Train**	**Dev Test**	**Eval**
KNN (k=112)	25.35%	25.81%	64.30%
LDA + KNN (k=111)	25.34%	25.95 %	-

Table 1. The transformation of the data led to a decrease in performance in the development set.

The neural network-based approach was able to achieve better performance, scoring 62.64% on the evaluation data set. Interestingly, the MLP also scored worse than the LDA KNN and KNN alone at 28.79% for training and 28.88% for development sets..

	Data Set		
Algorithm	**Train**	**Dev Test**	**Eval**
SciKit: KNN	25.35%	25.81%	64.30%
Keras: MLP	28.79%	28.88%	62.64%

Table 2. MLP was able to achieve a better performance in the evaluation set while performing worse in the train and dev sets.

Conclusions: In conclusion, KNN has shown to be a great algorithm but performance on the evaluation set but MLP performed better due to its flexibility. KNN does a great job at classifying data with non-linear decision surfaces and consequently, it was a great fit for the data we had for the competition. Considering the results, the Gridsearch performed on the pooled data (training + development) overfit the model more than expected. This becomes evident when you see the low error rate achieved on the training set (25.35%) and on the development set (25.81%). Perhaps a higher K value could have smoothed out the decision surfaces and generalized better. In addition, the transformation of the data using LDA before training and testing the model did not result in a significant difference in performance at a reasonable confidence level. However, the slightest decrease in error on the training set and increase on the development set suggested to me the KNN alone might have a better chance of generalization for the competition where every decimal counts.

On the other hand, MLP had significant difference in performance when compared to KNN. Performing significantly worse than KNN on the training and development sets at a 99% CL, MLP was able to generalize better and outperform KNN when scored on the evaluation set. MLP achieved a 62.64% error whereas KNN got 64.30%. This difference is significant at a 99% CL. The MLP, however, had to train for

~5 minutes while KNN was usually done training in about 10-15 seconds, making the trade-off in computational complexity and performance clear. In conclusion, MLP's ability to map non-linear representations of the data and its higher complexity resulted in better significantly performance and better generalization than KNN.