Evaluations and Comparisons of Supervised Learning Algorithms

Joseph Su

October 21, 2016

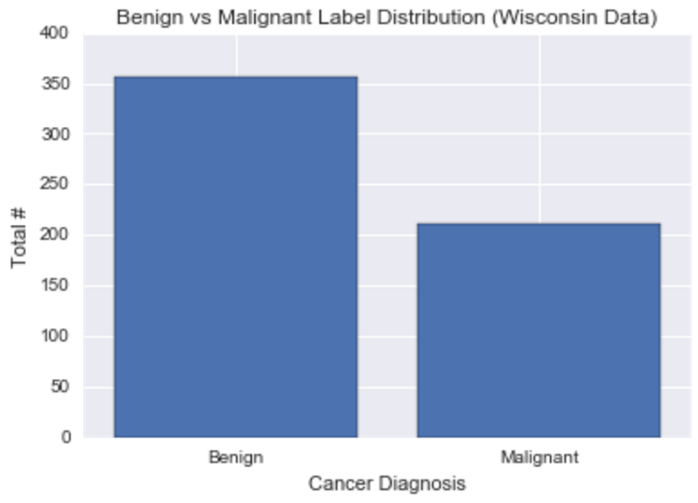
1 Introduction

This paper compares and contrasts five supervised learning algorithms, Decision Trees with pruning, Neural Networks, Boosting, Support Vector Machines, and k-Nearest Neighbor, on the two distinct datasets from the online UCI databases. The distribution of their classification identifiers is shown in Fig. 1.1.

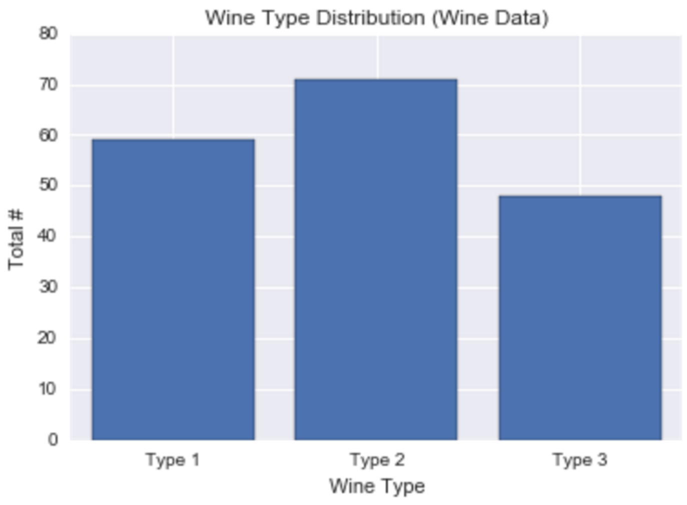
- Wisconsin Breast Cancer dataset (binary)
- Wine dataset (multiclass)

Wisconsin Dataset The set comprises of breast cancer cell features of either benign or malignant nature, retrieved from the UCI's Wisconsin Breast Cancer dataset, a canonical classification data source with a class distribution of 357 benign, and 212 malignant cells [1]. The process of identifying and making prognosis of each cell's classification is a labor intensive and costly task, attributing to analyzing and comparing more than 30 features exhibited in each cell's characteristics - a non-trivial job frequently performed by a clinical pathologist. Some of these features, such as a cell's texture in grayscale, depends on other features such as local variation in radius length (smoothness) or severity of concave portions of the contour (concavity), to name a few. These datasets exhibits an high degree of complexity, inter-dependency, and good quality, which makes discriminating them from the measurements of their cell characteristics simple but challenging. Lastly there are inferences (such as the chance of cancer recurrence) one may draw from these data by running various algorithms through them. The above reasons are what make these datasets interesting.

Wine Dataset This dataset is retrieved from UCI's Wine database with multi-class data points [2]. The motivation in picking this dataset comes from its continuous, real-valued and multi-class features. These features include the quantity of each wine's constituents: alcohol, ash, malic acid, total phenols, hue, etc. These features are categorically different, and their values wholly agnostic of ones from the prior dataset. The use of this dataset ensures pattern independence that minimizes any biases that may come from source similarity. Lastly the dataset encompasses 13 features, a good enough fitting entry point to admit decent, expedient, exploratory analysis and comparisons of the algorithms in cross validating and tuning their efficacies.



(a) Wisconsin Dataset



(b) Wine Dataset

Figure 1.1: Classification Label Distribution

DIMENSIONALITY The dimension space for the Wisconsin dataset is $X \in \mathbb{R}^{s\times f}$ where $s = 569$ samples and $f = 30$ features. The target label vector is $y \in \mathbb{R}^{f}$. The objective is to derive a hypothesis, $h \in H$, to approximate the target concept *c(x)*, where $x \in \mathbb{R}^{s}$. The optimal hypothesis, h', is one that minimizes the error function to map each $x \in X$ to either of the classification labels, benign or malignant (Fig. 1.1a).

The dimension space for the Wine dataset is $s = 178$ and $f = 13$. The classification labeling is multi-class for this dataset where a wine can be of either one of the three types (Fig. 1.1b).

COMPUTING FACILITIES This work makes use of various libraries, toolsets, and modules in Python to arrive at its studies and results: `scikit-learn 1.19.dev0`, `nolearn`, `numpy`, `scipy`, `lasagne`, `IPython`, `seaborn`, and `panda`. Experiments are performed on a 2015 Mac OSX 2.5 GHz Intel Core i7 with 16GB DDR3.

2 DATA PRE-PROCESSING

Preprocessing ensures that data are labeled, trimmed, mapped, encoded and partitioned where necessary. Many learning algorithms require input data on the same scale for optimal performance. Additionally data are dimensionally reduced where necessary to improve performance and reduce noise. We chain all of these tasks in a pipeline to fit and transform data into a predicative model, by applying slightly different data pre-processing treatments on our datasets.

WISCONSIN DATASET The Wisconsin data has a binary feature called diagnosis, which partitions into either B (benign) or M (malignant). We map this ordinal feature into integers of 0 or 1 ($B = 0$ and $M = 1$). The other 29 features in this dataset are already in float32 so they won't need to be encoded. Partitioning this dataset is done by random shuffling them into 80% as the training set, and the other 20% held out as the test set. Data is uniformly scaled down to zero mean and unit variance with the `StandardScaler` module in Python; it computes transformation from the training set before propagating its effect to the test set.

WINE DATASET We apply an ensemble algorithm, `RandomForestClassifer`, to discern features with their relative importance from this dataset. The algorithm generates an averaged impurity decline, or relevant importance, for each feature by training a forest of 10,000 decision trees arriving at our results in Table 1. The top entry represents the most relevant feature.

```
Flavanoids :        0.164608770043
```

Proline :	0.162986172295
Color intensity:	0.156852440723
Alcohol :	0.123624289427
OD280:	0.110142172424
Hue:	0.0785045066346
Malic acid:	0.0467945118113
Total phenols:	0.0452893124467
Alcalinity of ash:	0.0278184354568
Proanthocyanins:	0.0250553428069
Magnesium:	0.0242121514148
Ash:	0.0217115931256
Nonflavanoid:	0.0124003013907

Table 1: Feature Importances from the Wine Data

We remove the least important label, "nonflavanoid", from this dataset before admitting them to cross validation.

3 Cross Validation

This is a key process that assesses a learning algorithm's performance on data it hasn't seen before. To minimize sub-partitioning of training data in a dataset of limited size, we opt for a validation method called k-fold cross validation instead of the traditional holdout method to cross validate our algorithms.

k-Fold Cross Validation k-fold minimizes sensitivity of holdout estimation by resampling data at the expense of more computations. In our experiments, we employ `GridSearchCV` to exhaustively evaluate over a range of specified parameters an algorithm's optimal parameter set, which generates the best performance over the average of all cross-validation runs for that algorithm. Using this method, our dataset is split into k subsets, each of which is used as a test fold against the other k-1 training folds without replacement. At the end of cross validation each fold in the dataset is used exactly once, which yields a lower variance estimate of the model than the holdout approach.

k Value For both of our datasets, k=5 is used to ensure that each test set preserves the labeling distribution of the $k-1$ training set. We feel that a small k gives us an accurate estimate of the average performance of the model as our datasets are not sparse enough to warrant a larger k.

$F1$ Score In our experiments, we apply F_1 scoring to measure algorithmic performance. This score indicates how accurate a model is by looking at how good its predictions are on average, using both precision, p, and recall, r, equally. F_1 score is defined as the harmonic mean of precision and recall:

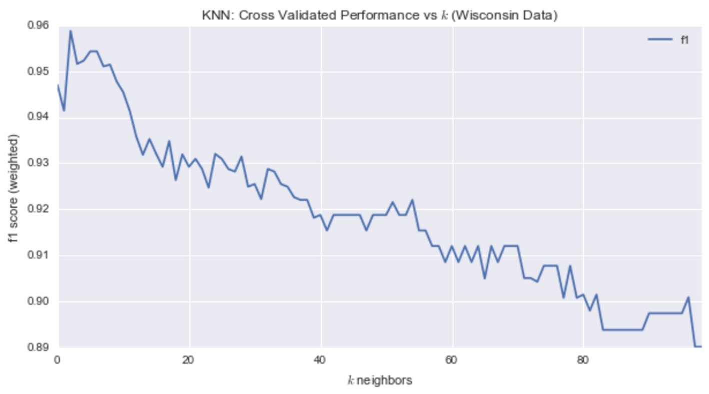
$$F_1 = 2 \cdot \frac{p \cdot r}{p + r} \tag{3.1}$$

F_1 reaches its best value at 1 and worst at 0. Thus an algorithm with a moderately good performance receives an higher score than another with an extremely good or poor performance.

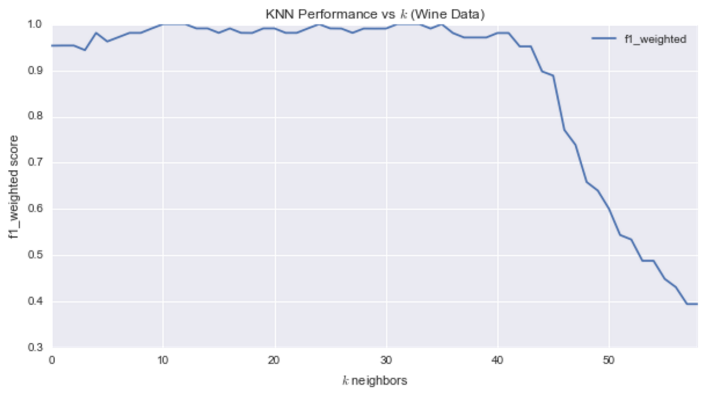
Stratification We apply stratified k fold cross validations across all algorithms to arrive at the most optimal k for each model before we begin the fitting experiments.

4 Algorithms and Evaluations

In this section we describe each of the algorithms and how they are applied to our datasets.



(a) Wisconsin Dataset



(b) Wine Dataset

Figure 4.1: KNN Performance Charts

4.1 K-Nearest-Neighbors

KNN defines a distance metric among training samples to find the K closest samples to a target point. In our experiments, `KNeighborsClassifier` is used along with the Minkowski distance metric guaranteeing each of the k neighbors having equal vote whatever its distance from the target point. Minkowski, or p distance, is defined as

$$d(x_i, x_j) = \sqrt[p]{\sum_{r=1}^{n} |x_{ir} - x_{jr}|^p}. \tag{4.1}$$

where $p = 1, 2, \cdots, \infty$. We employ `GridSearchCV` with a 5-fold validation rule to assess each k neighbor in the range of $1 \cdots 100$ iteratively.

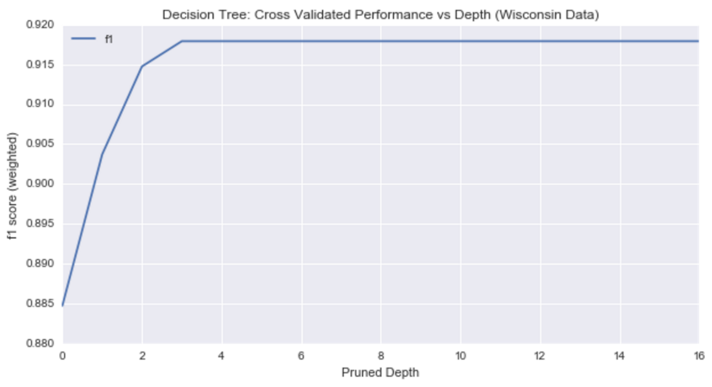
Wisconsin Dataset   The optimal k value is determined to be 3 via Minkowski metric for this dataset. Note that this model's performance decreases almost linearly as k increases, as seen in Fig. 4.1a. This trends comes as no surprise as generally speaking, noise decreases as k becomes greater; but doing so also makes the boundaries between classes less distinct, driving down accuracy.

Wine Dataset   This dataset shows contrast and more resiliency than the Wisconsin trend. In Fig. 4.1b one observes that KNN maintains a near perfect performance level until k reaches 42. In fact, for this dataset the optimal $k = 11$ with an F1 score of 0.944. This resiliency connotes to the quality of splits and integrity of the data source. Applying both Euclidean and Manhattan distance metrics exhibit similar trends with this dataset.

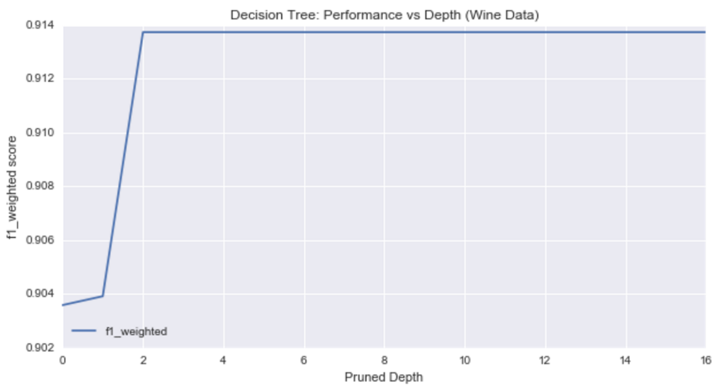
4.2 Decision Trees

Using Decision Trees as weak classifiers, the decision criterion is based on the quality of a split, i.e., the GINI coefficients that replace the entry for the information gain in our experiments. The implementation of decision trees proceed similarly with different pruning depth values. In each iteration of the depth tested, an F1 score is generated to compare and contrast with the global score until all depths have been exhaustively evaluated. In our experiments the search grid covers the maximum depth range of 3 to 20. Note that after the optimum depth is reached the effect of pruning has no effect to the overall accuracy of the model as it generates the same optimized version irrespectively of the depth.

Wisconsin Dataset   The optimal depth is 3 for this dataset which maxes out at depth 4; its un-pruned version is over 15 deep. Fig. 4.2a depicts a rather flat line after the optimal depth. The current test takes about 2 seconds to train.

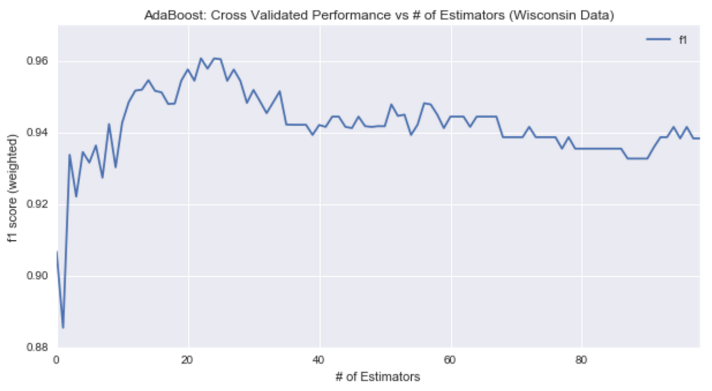


(a) Wisconsin Dataset

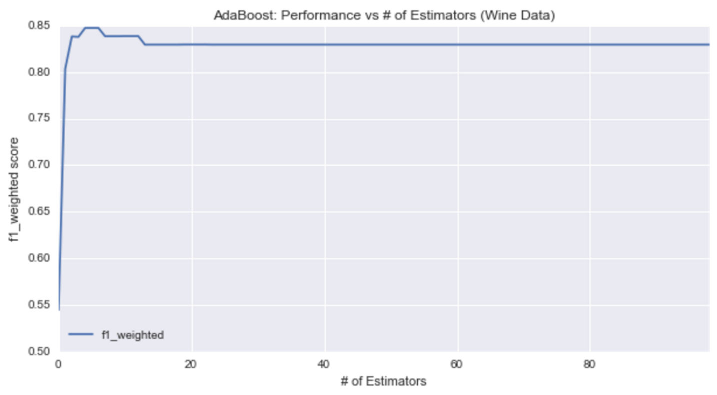


(b) Wine Dataset

Figure 4.2: Decision Trees Performance Charts



(a) Wisconsin Dataset



(b) Wine Dataset

Figure 4.3: AdaBoost Performance Charts

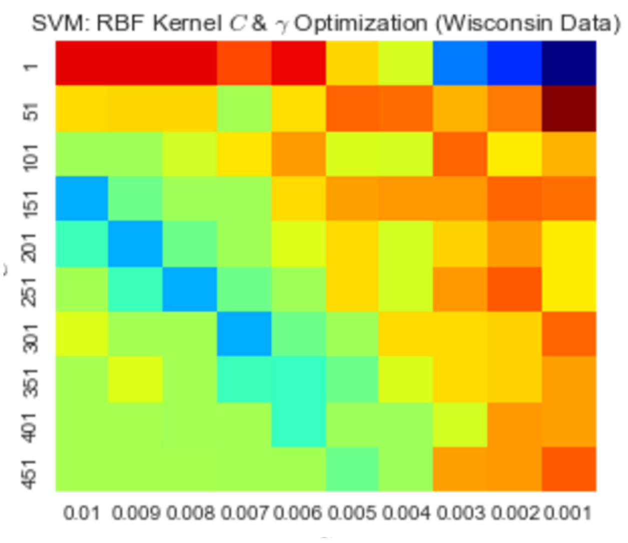
WINE DATASET The optimal depth is 2 for this dataset whereas the un-pruned version is over 8 deep. The max depth is 5. Fig. 4.2b is similar to Fig. 4.2a in trend after the optimal depth is reached. Training time (1.1s) for this dataset is much faster than the Wisconsin time (2s) due to the current dataset's reduced dimensionality and size.

4.3 ADABOOST

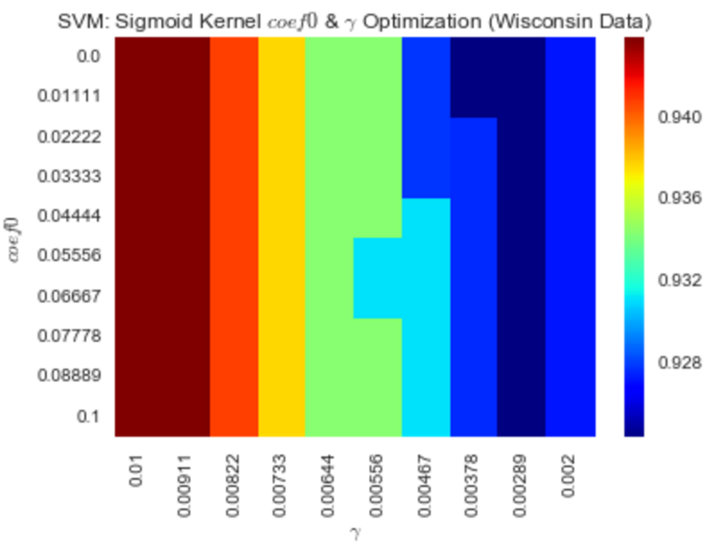
Adaptive Boosting (AdaBoost) selectively boosts dataset features to improve the model's predicative power, by decreasing execution time and dimensionality of the features in a dataset to converge to a stronger learner [4]. In our experiments we run the number of estimators iteratively (on a 5-fold CV) from 1 through 100 in search of the optimal number.

WISCONSIN DATASET The boosted version optimizes at 43 estimators, with a learning rate of 1.0, on a SAMME.R technique using the predicted class probabilities [8]. The total execution time for this algorithm is 65s.

WINE DATASET In contrast to the AdaBoost trend in Wisconsin, this dataset peaks at 5 estimators, before the model stabilizes at $F1 = 0.83$ indefinitely. The grid search for this set turns out to be rather exhaustive taking more than 45s to complete, much faster than the Wisconsin set given, again, the current set's dimension and size.



(a) RBF



(b) Sigmoid

Figure 4.4: SVM Heatmap: Wisconsin Dataset

4.4 SVM: RBF & Sigmoid Kernels

Two Support Vector Machines are implemented, one on a Radial Basis Function (RGB) kernel, and the another on a Sigmoid kernel. The two hyper-parameters for RBF are C, the misclassification penalty coefficient, and γ, which denotes how far the influence of a single training example reaches. Values of C are grid-searched from 1 through 500 with an interval of 50, where as γ values searched from 0.01 down to 0.001, in the decrement of 0.01. `GridSearchCV` fits 5 folds for each of the 100 candidates, totaling 500 fits for each of our datasets. This process is repeated for each of the kernel choices, RBF and Sigmoid.

Wisconsin Dataset The performance heatmaps for both RGF and Sigmoid kernels are depicted in Fig. 4.4. The orange to red patches indicate higher performance whereas deep blue regions indicate low performance. With Sigmoid kernel the best margin is within $\gamma > 0.0073$ with any values of $coef0$, whereas with RBF, the best margin is when $\gamma = 0.001$ and $C = 50$, or with any $\gamma > 0.006$ and $C < 10$.

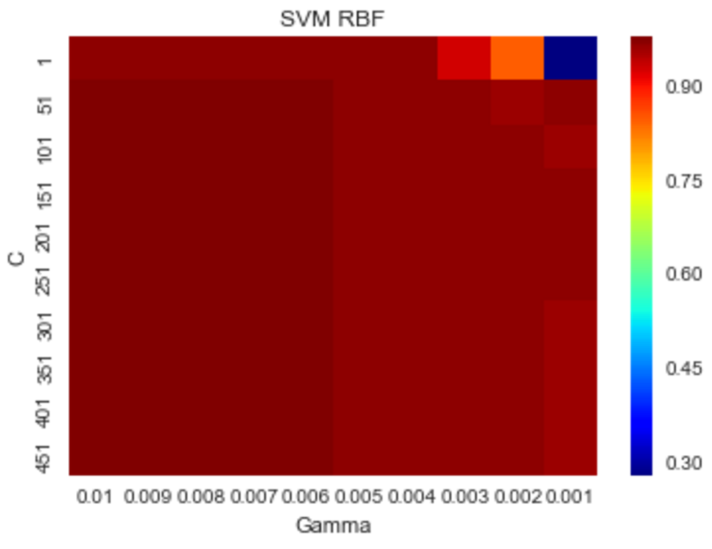
Wine Dataset The performance heatmaps for both RGF and Sigmoid kernels are depicted in Fig. 4.5. These graphs exhibit the dataset's inherent resiliency across a broad margin of γ and C and $coef0$ for both kernels. This trend connotes to SVM's inherent strength to predict the data well.

4.5 Neural Networks

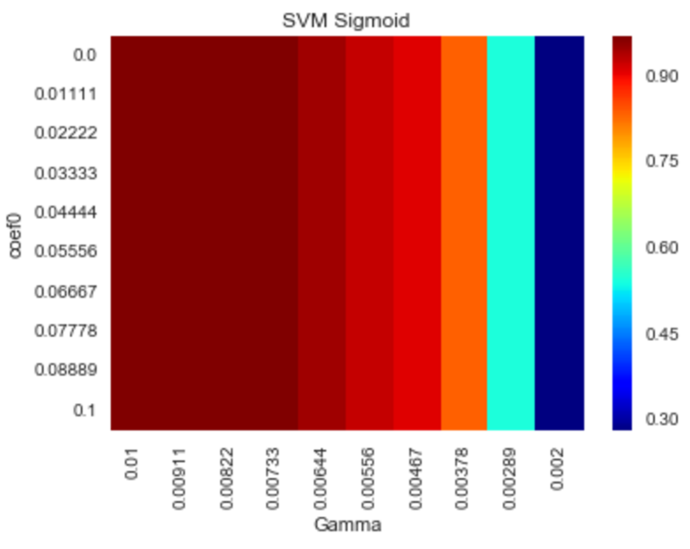
We apply convolution neural networks (CNN) in deep learning to both datasets. CNN is a type of feedforward artificial neural network, an variant of multi-layered perceptron algorithm inspired by biological processes Literature for this subject is abundant [5, 6].

4.5.1 Methods

We employ a Python wrapper, `nolearn` that works with the popular `Lasagne` and other machine learning utilities in our experiments. We define 3 layers, `input`, `hidden`, and `output` with the `nesterov_momentum` gradient descent optimization method as our update function, to redistribute the weights of our network after each batch. `nesterov_momentum` works well with many learning applications [7]. Our initial gradient descent step size, or the learning rate (α) is 0.01 and its momentum (p), 0.9. The hyperbolic tangent function, $tanh$, and not the standard sigmoid transfer function, is used to allow for faster updates and inclusion of less-weighted outlier data that we wish to include in the analysis.



(a) RBF



(b) Sigmoid

Figure 4.5: SVM Heatmap: Wine Dataset

We run two experiments training 4 different networks to ensure that, for each dataset, a network with a good generalization is found:

1. Static CNN with $\alpha = 0.01, p = 0.9$.

2. Dynamic CNN with decreasing α from 0.03 to 0.0001, and increasing p from 0.9 to 0.999.

Each network has various unit numbers, ranging from 25, 50, 75, to 100 in its hidden layer. All other parameters stay the same in all 4 networks. A larger unit cardinality does not make sense in light of our data size < 500 rows.

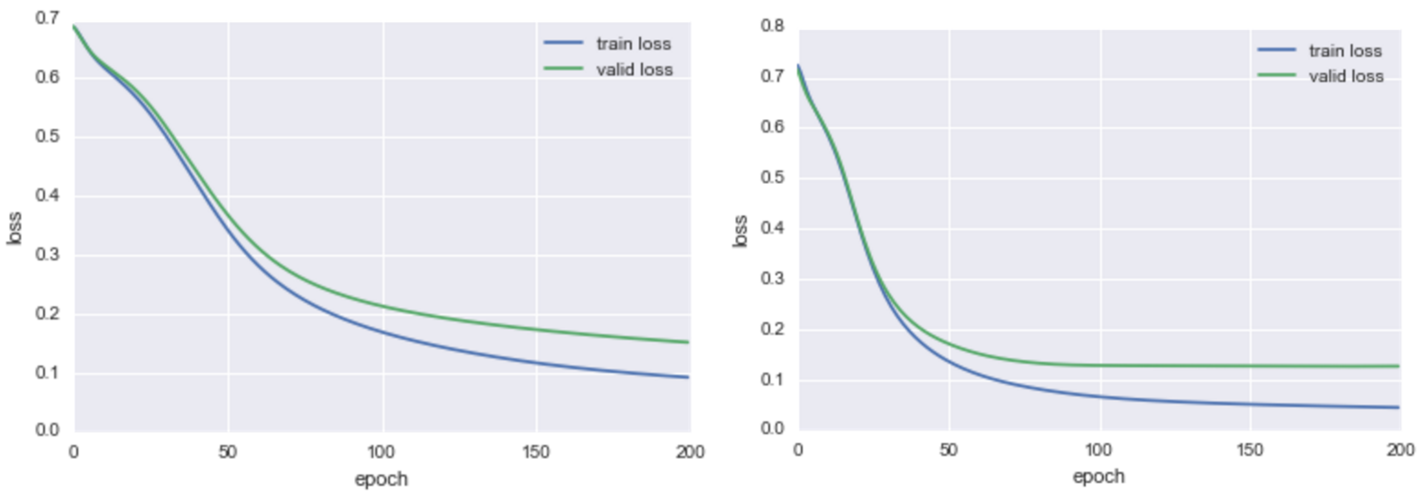
For the first experiment we run with $\alpha = 0.01$ and p=to 0.9 on over 200 epochs; we repeat this experiment on all 4 networks. For the second experiment, we run with a *boosted* α and p, to dynamically calibrate our deep learning by increasing our gradient descent's p while decreasing its α for each epoch.

The ideal hidden unit number for both datasets hovers around 25, which produces the least amount of training and validation losses. Moving forward, all of our experiments apply this number. All CNN diagrams exhibit loss (error) vs. epoch trends.

WISCONSIN DATASET As seen in Fig. 4.6b, it appears that after 175 epochs the network stops learning anything useful. Also, Fig. 4.6b's errors flatten out more quickly than the static version, Fig. 4.6a. The boost (online updating of hyper-parameters) drastically speeds up our training by over 500 percent; it takes roughly 800 more epochs for Fig. 4.6a to arrive at the boosted results (see Fig. 4.6).

This network produces overfitting on the Wisconsin dataset as indicated by the train/validation loss ratio in Table 2, which indicates that the static CNN takes much longer time with more epochs to equilibrate to the same loss ratio as the dynamic CNN version does. This connotes to the idiosyncrasies of the test and training data splits in our dataset. At even higher epochs of 5000 and beyond, which takes more than 8 minutes on a 2015 Macbook Pro running on 2.5 GHz Intel Core i7 to complete, the training and validation errors start to diverge.

Convolutional neural networks usually require a large amount of training data in order to avoid overfitting; this is when dropout should be used to minimize overfitting. We implemented dropout but due to curse of dimensionality the technique quickly exhausted our stack memory space to accommodate addition recursions, even on our small Wisconsin dataset. To overcome this debacle we apply a small convolutional filter of 2x2 as a decent regularizer to the network, arriving at an expedient and accurate result.



(a) Learning Curve on Static CNN

(b) Learning Curve on Dynamic CNN

Figure 4.6: Learning Curves for CNN

WINE DATA In contrast to the learning curves for Wisconsin, this dataset exhibits very little overfitting. The validation curve is just above the training loss making this trend promising. Additionally at epoch > 100 it seems that both curves will equilibrate to a flat line indefinitely. This connotes to an important fact: the algorithm continues to learn a good approximation to the target hypothesis regardless of the training set size in Wine.

Static CNN with fixed parameters

epoch	train loss	valid loss	train/valid
25	0.52732	0.53223	0.99079
50	0.34824	0.35620	0.97765
...			
200	0.09345	0.13775	0.67843
250	0.08129	0.12937	0.62836
300	0.07018	0.12314	0.56991
...			
498	0.04968	0.12598	0.39438
499	0.04963	0.12595	0.39403
500	0.04958	0.12593	0.39368
...			
997	0.03351	0.12353	0.27129
998	0.03349	0.12354	0.27110
999	0.03347	0.12354	0.27091
1000	0.03345	0.12355	0.27072

Dynamic CNN with fluctuating parameters

epoch	train loss	valid loss	train/valid
25	0.33731	0.34507	0.97751
50	0.13994	0.17417	0.80344
...			
197	0.04523	0.12678	0.35675
198	0.04512	0.12680	0.35583
199	0.04501	0.12682	0.35490
200	0.04490	0.12684	0.35398

Table 2: Train / Valid Loss Wisconsin Data: Static vs Dynamic CNN

5 LEARNING CURVES: ANALYSIS

To evaluate overall performance for each of our algorithm models, we run each model with optimum k over different training and test set sizes, and plot the results. We iterate the process through each model to arrive at a consolidated performance comparison chart for each dataset.

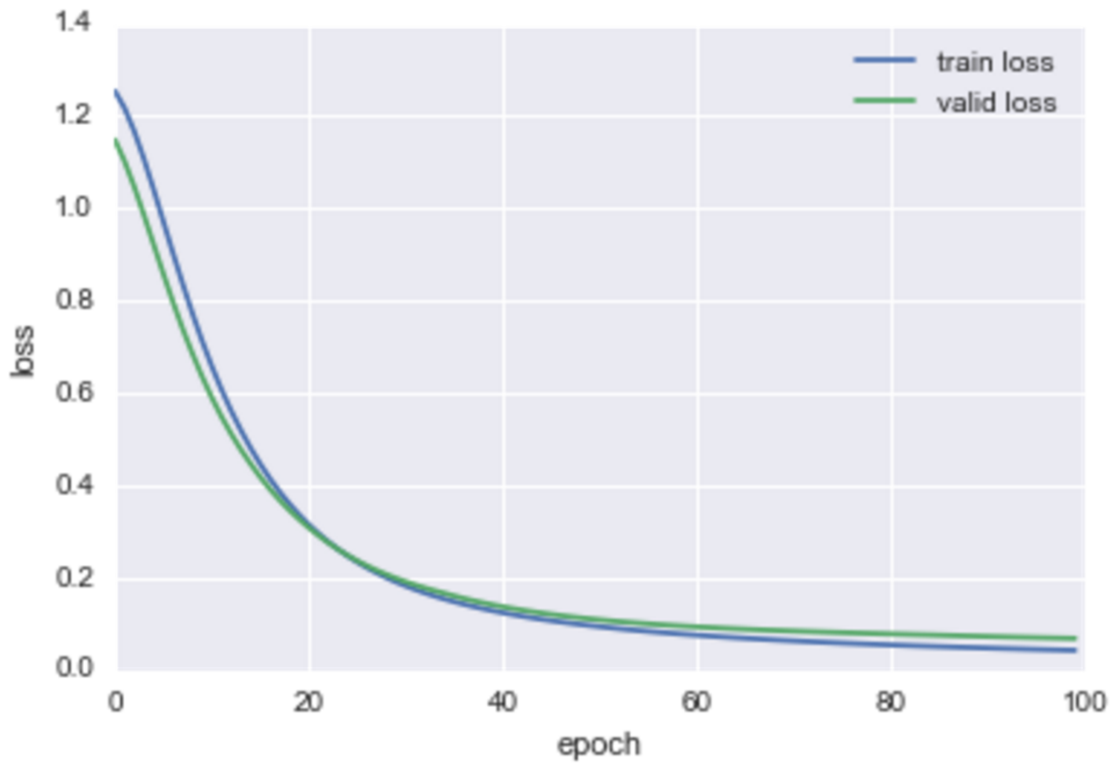


Figure 4.7: Learning Curves for CNN (Wine Dataset)

WISCONSIN DATASET The performance chart for this dataset is depicted in Fig. 5.1. Fig. 5.2 depicts the aggregated performance statistics across all the algorithms on the Wisconsin's test set. Note that SVM generally outperforms neural network, which is the second contender of all others. Specifically, SVM on RBF kernel is the best performer model of all of the models tested. In contrast, decision trees does not fare so well, ranking last. The boosted version improves the 5 weak learners in decision trees for this dataset, as evident in its 8% performance gain ahead of the decision trees.

WINE DATASET The performance chart for this dataset is depicted in Fig. 5.3. The best performance gain is again achieved by SVM running on a RBF kernel. Neural network (CNN) does not do so well. Given the lower size and dimensionality of the dataset as compared to Wisconsin's, the KNN algorithm turns out to be the winner for this dataset as evidenced from Fig. 5.3. Note that all of the learning curves with the exception of AdaBoost show parallel trend between test and training sets as the sample number increases. With AdaBoost, the test divergence between sample # 65 to 90 may be indicative of an incompatible bias in the algorithm with respect to the data.

The confusion matrices for all of the algorithms are depicted in Fig. 5.5. The x-axis corresponds to three wine types (Type 0, 1, and 2); the y-axis represents the same. This means the diagonal line, namely (0,0) to (2,2), represents the degree of match between the two types in our dataset; red indicates high degree of match ($F1$ score) and blue represents low degree of match. The matrices insinuate that our models match most of our Type 0 wines correctly, whereas the non-match mostly occurs with either Type 1 and Type 2 in the dataset. For future work, one may analyze and focus on either types and see if improvement can be made to the model to increase its accuracy and precision by balancing out its recall strength on these types of wine in the dataset.

6 CONCLUSION

We have greatly explored, and experimented with, many use cases involving the 5 representative learning algorithms in Supervised Learning, with 2 distinct classification datasets of either binary or multi-class in nature. We observe that SVM - RBF achieves the best outcome of all algorithms, with KNN following suit.

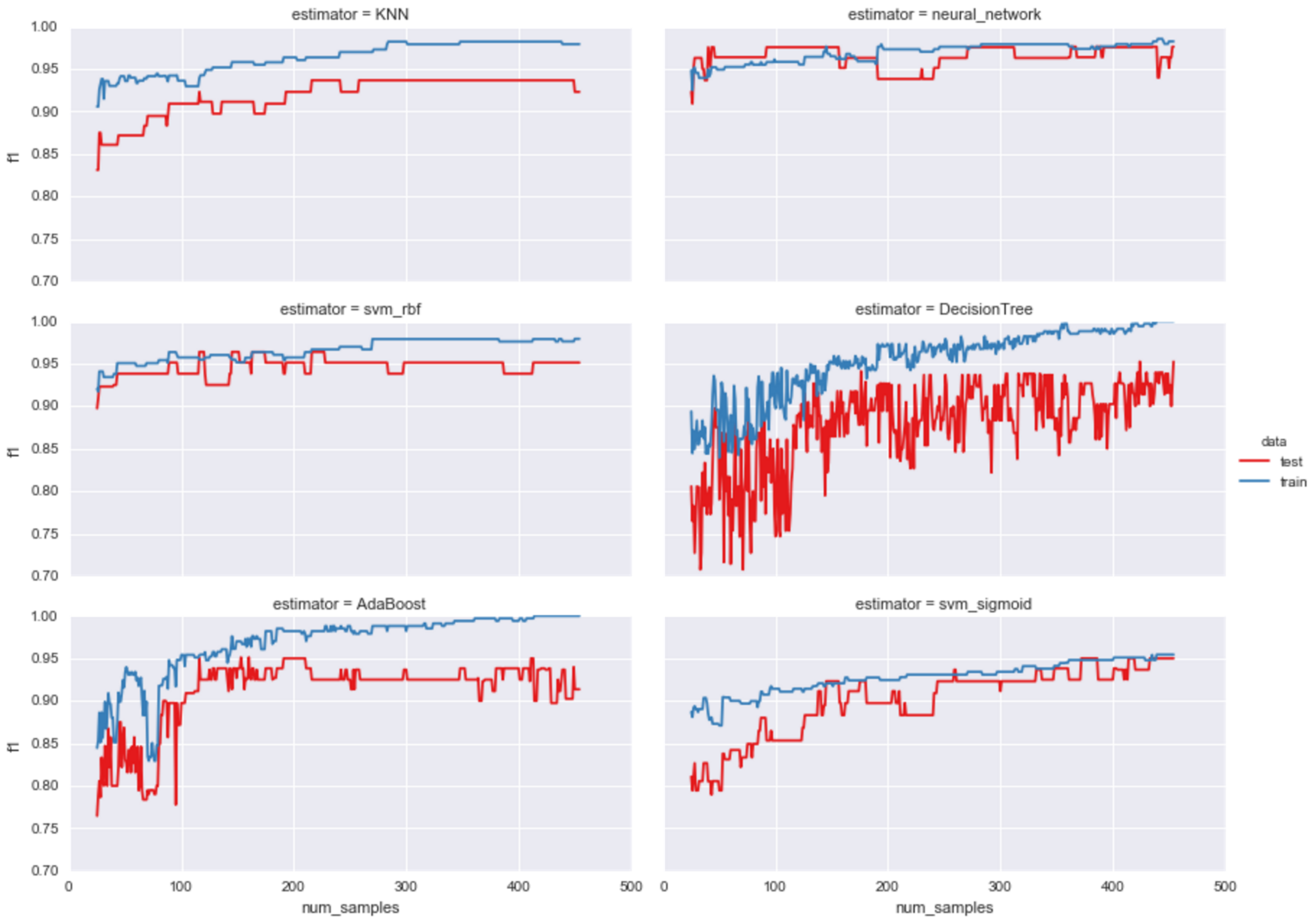


Figure 5.1: Algorithmic Accuracy Comparisons (Wisconsin Dataset)

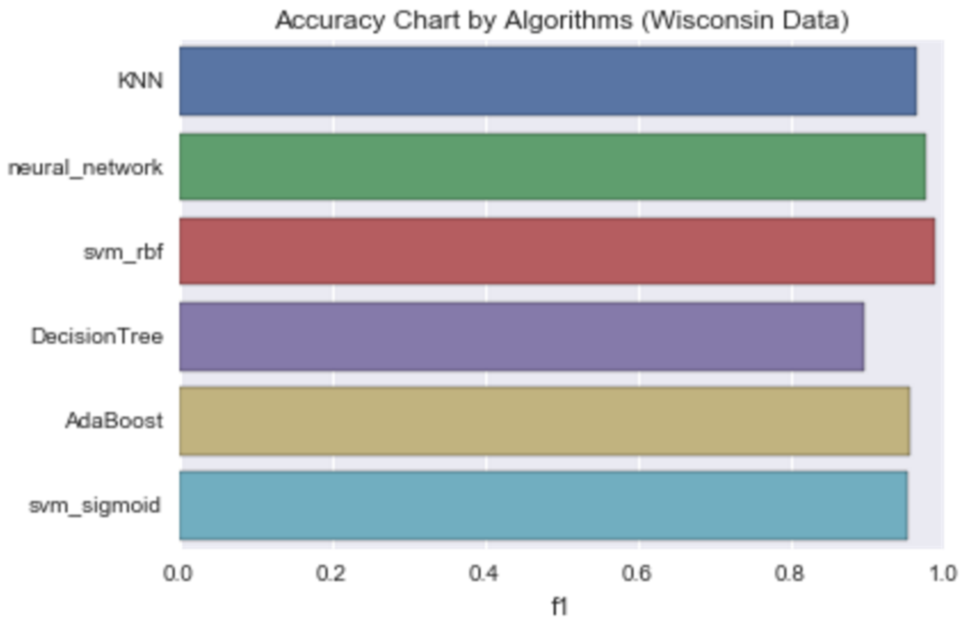


Figure 5.2: Accuracy Chart by Algorithms (Wisconsin Dataset)

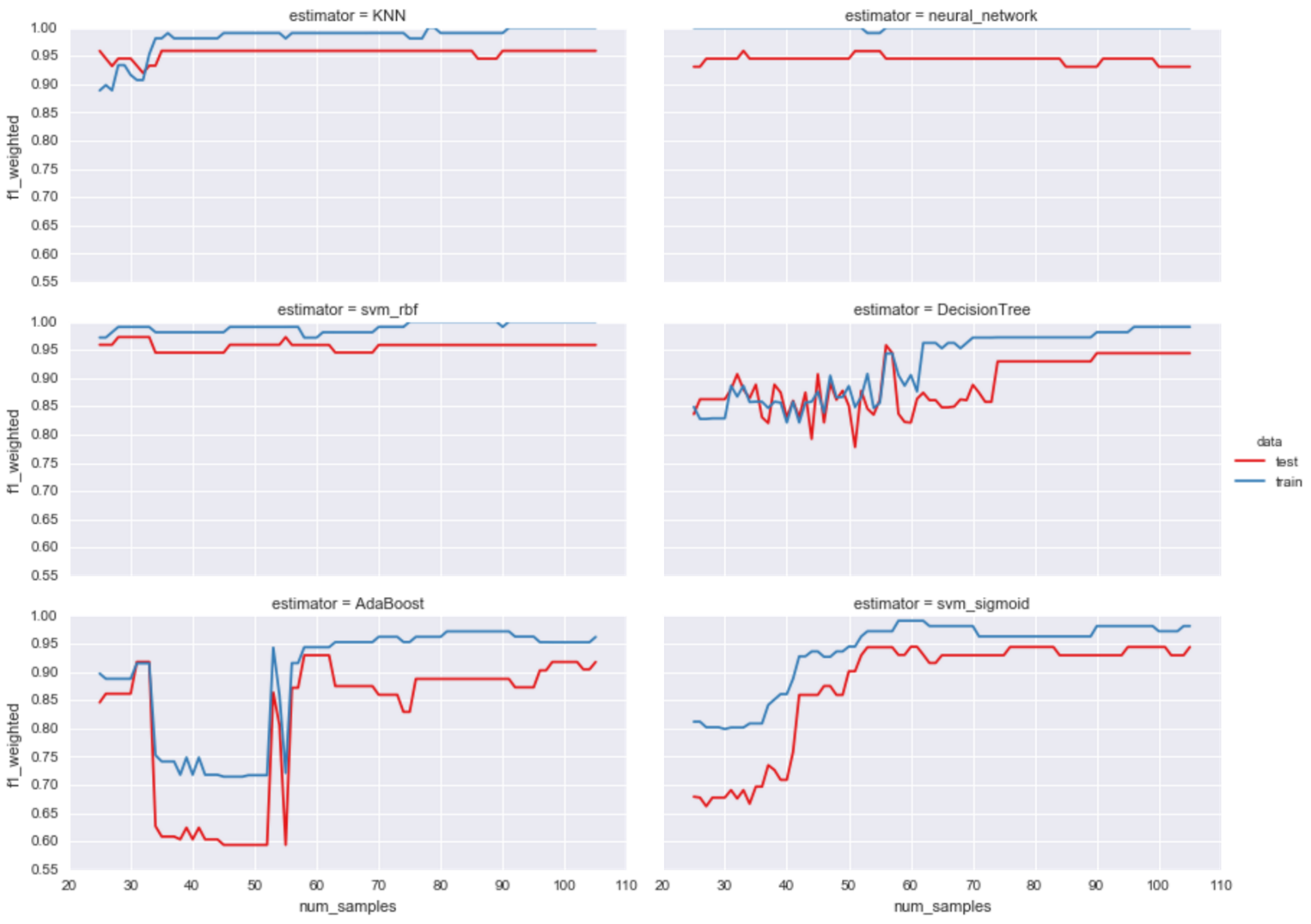


Figure 5.3: Algorithmic Accuracy Comparisons (Wine Dataset)

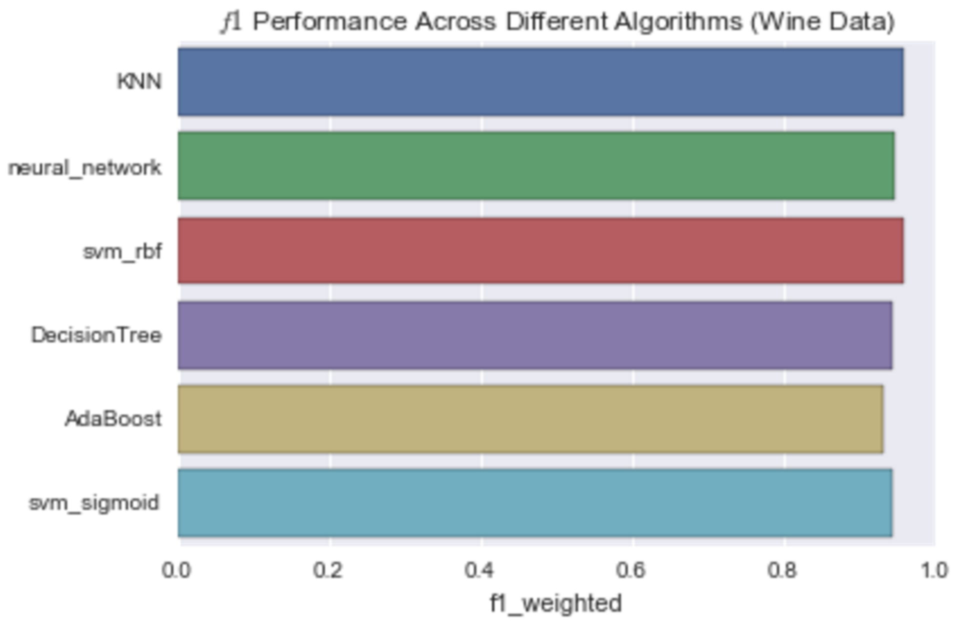


Figure 5.4: Accuracy Chart by Algorithms (Wine Dataset)

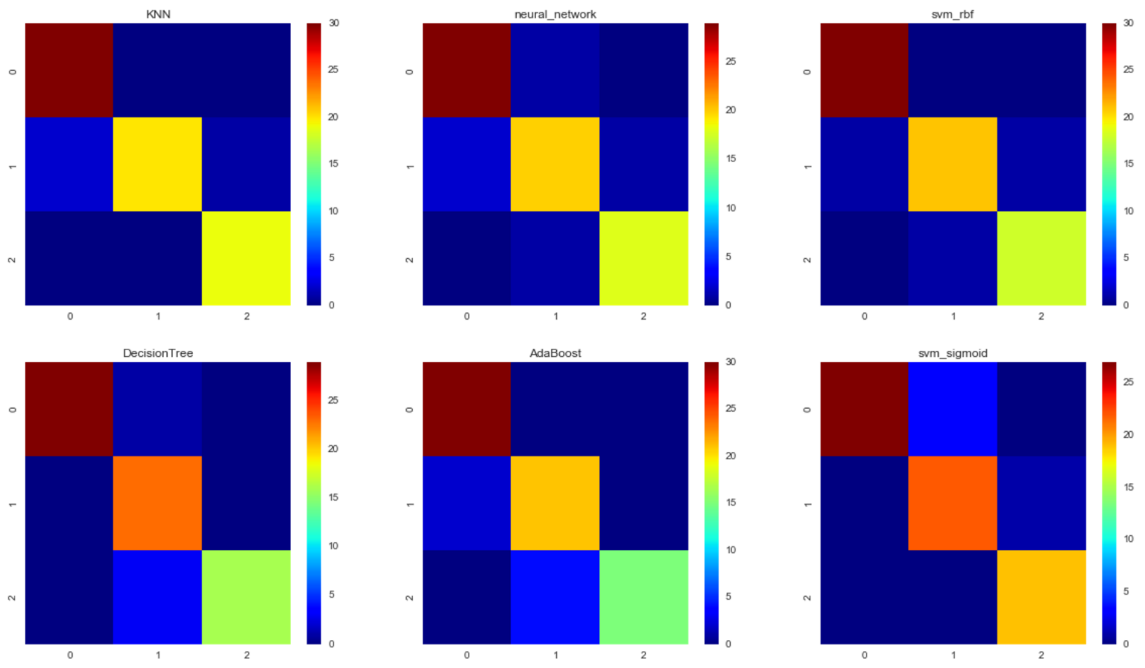


Figure 5.5: Confusion Matrix Representation of Algorithmic Performance (Wine Dataset)

The performance of SVM on this kernel and KNN indicate that they may be suited to relatively small datasets (<1000 rows) with less than 30 features as we have defined it in this paper.

References

[1] UCI Machine learning repository. (2005, Feb 1996) Wisconsin Diagnostic Breast Cancer (WDBC) dataset, UCI Machine learning repository *http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)*. Web. 01 Sep. 2016.

[2] UCI Machine learning repository. Wine Data Set, UCI Machine learning repository *http://archive.ics.uci.edu/ml/datasets/wine)*. Web. 01 Sep. 2016.

[3] Wikipedia contributors. (2016, September 16). Precision and recall. *Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia.* 16 Sep. 2016. Web. 20 Sep. 2016.

[4] Wikipedia contributors. (2016, September 14). AdaBoost. *Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia.* 14 Sep. 2016. Web. 21 Sep. 2016.

[5] Wikipedia contributors. (2016, September 22). Convolutional neural network. *Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia.* 22 Sep. 2016. Web. 23 Sep. 2016.

[6] Wikipedia contributors. (2016, September 22). Convolutional Neural Networks (LeNet) ? DeepLearning 0.1 documentation. Retrieved 31 August 2013. *DeepLearning 0.1. LISA Lab.* Retrieved 22 September 2016.

[7] Nouri, D. (2014) Using convolutional neural nets to detect facial keypoints tutorial. *http://danielnouri.org/notes/category/deep-learning/*.

[8] Zhu, J, Rosset, S. Hastie, T. (2009) Multi-class AdaBoost