CSE 347/447 Final Project Report: Classification with SVM, CNN, and XGBoost

Maximillian Machado, Theo Olsen, Isaac Howenstine

October 18, 2021

Abstract

The final project report implementing 3 different classification algorithms on 3 different datasets.

1 Introduction

The purpose of this project is to implement and evaluate some of the classification algorithms taught in CSE 347/447 on real datasets.

1.1 Classification Algorithms

We decided to implement Support Vector Machines, Convolution Neural Networks, and XGBoost.

1.2 Datasets

In the project scope we were allowed to choose from among the following datasets: PIE, YaleB, USFGait, CIFAR-10, MNIST, Iyer, and Cho. There are 3 different types of data in these datasets: 1-dimensional vectors, 2-dimensional data, and 3-dimensional data.

We decided to primarily focus on vector-based methods where the data is 1-Dimensional. Since Iyer and Cho are the only two datasets of the listed options that are inherently 1-Dimensional, we chose to implement our algorithms on those two. The other dataset we chose was PIE, which is photographs of faces, which can be vectorized using various methods. Vectorizing a 2-dimensional dataset comes with certain assumptions and restrictions, coming from the fact that the spacial relationships between data are destroyed. When a 32 by 32 pixel image from the PIE dataset is converted into a $32^2 = 1024$ point vector, the ordering of data points in the x and y dimensions of a photo are no longer considered. This can cause problems, particularly in the case when we have to deal with images of different sizes. Initially at least, we did not consider the negative impact vectorization has on such analyses. An image like this could simply be unstacked into a long vector, or it could be deconstructed using a hilbert curve, and other methods exist as well.

We decided not to use any 3-dimensional datasets, such as those in USFGait and CIFAR-10 for sake of making the project more manageable.

1.3 Evaluation Methods for Classification Models

The overall performance of these different methods will be evaluated via the following metrics: accuracy, F1 score and AUC.

With regards to classification models, accuracy is the number of accurate predictions divided by the number of all predictions. This would be the sum of diagonals along a confusion matrix divided by all predictions. This is also the metric used to evaluate the optimal parameters for our models.

The F1 score is the harmonic mean of precision and recall:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{1}$$

Recall that precision pertains to a model and class, where for some class precision is the number of correctly predicted class items out of the total number of predicted class items. Recall also pertains to a model and class, where for some class, recall is the number of accurately predicted class items out of the number of actual members of the class. These allow us to calculate the per-class F1 score using the equation above. To get the overall F1 score, we can simply average the class-wise scores together. This is the method that sklearn.metrics uses.

AUC stands for Area Under the Curve, the area under the ROC curve (Receiver Operating Characteristic curve), and describes a cumulative characteristic of the ROC curve. These are used for binary classification. The related method for multiclassification are OAA and OAO methods. We implement the method referenced in lecture 19 by Eric Plog on `https://medium.com/@plog397/auc-roc-curve-scoring-function-for-multi-class-classification-9822871a6659`.

2 Support Vector Machines (SVM)

The Support Vector Machine, or SVM, is a fundamental classification method. A trained SVM classifier describes a hyperplane in n-dimensional space. That hyperplane is used to classify testing data points into one of two groups, depending on which side of the hyperplane those test points are on. The two sides of the hyperplane are usually referred to as positive and negative, though this has no inherent numeric meaning. While an SVM creates a single hyperplane which can separate only two distinct classes, a system of many SVMs can be used together to classify data with more than two classes. When faced with more than two classes (which is often the case), there are two methods to use SVM to identify classes:

1. One versus rest (ovr): Given c different classes, one can use c SVMs, where each SVM separates one class' data points from *all other class' data points*.

2. One versus one (ovo): Given c different classes one can use 2^c SVMs, where each SVM separates *each pair of classes*.

The c different hyperplanes in the n-dimensional vector space are used to divide up the space into chunks which are used to associate test points with a class. For the "one versus rest" method which we use in our analyses, to assign a testing point to a class, use each trained SVM to evaluate that testing point, and the SVM that has the maximal distance from the testing point is the one that is used to classify that data point (via slide 54 of recording for lecture 16).

In the process of training an SVM classifier, the goal is to find the hyperplane that maximizes the width of the buffer around that hyperplane to the nearest local training point on either side (positive or negative) of the hyperplane. It is often the case that a linear hyperplane cannot cleanly separate the two groups of data, i.e., the "linear separability" is too strong. Sometimes, the space the points occupy can be warped or raised into a different vector space to make the points easier to linearly separate with something called a kernel function. Another useful method for dealing with this is a buffer of violatable points called "slack," where some violation of the SVM hyperplane can be allowed with a penalty (called a soft margin).

Assuming the factors above are not an issue (or have been dealt with via a kernel function and soft margin, the search for the optimal classifier can actually be defined according to some set of the

training data points. This is called the Dual SVM Derivation. The essence of this method is in the following equation:

$$\max_{\alpha \geq 0} \min_{w,b} \frac{1}{2}||w||^2 - \sum_j \alpha_j[(w * x_j + b)y_j - 1] \tag{2}$$

The α vector is such that each element of the alpha vector corresponds to one of the training data points. The alpha vector is used to identify the "Support Vectors," which are some set of the (+/-) training data points. The support vectors are combined to define the optimal hyperplane separating the two classes. The support vectors, will lie on the margin boundary along the separating hyperplane. The rest of the points/vectors will not be used to define the boundary at all.

2.1 Overall Method

For each of the three data sets: PIE, Iyer, and Cho, we do the following.

1. Separate into testing and training sets (if not already done)
2. Normalize the training data (and will ultimately normalize the testing data using the same normalization method)
3. Identify relevant and interesting kernels, and for the kernels identify the different important parameters for each kernel method.
4. Use gridsearchcv to determine the optimal parameters for each method, evaluating the parameter performance using the "accuracy" metric.
5. Given the optimal parameters for each method, try each method on the test data.
6. Evaluate the methods in terms of accuracy, F1 score and AUC metrics.

2.2 Kernels and Parameters

All kernels use the parameter C, which represents the tradeoff between the penalty due a number of slack variables violating the hyperplane boundary, and the size of the margin that the SVM finds. For a smaller C value, more slack variables are permitted to violate the hyperplane boundary in order to have a wider support vector margin around the separating hyperplane, and for a larger C value the model will choose a smaller support vector margin if it means cleanly separating the classes.

We use the Radial Basis Function (RBF) kernel for our SVM. Using the RBF method, each training data point is lifted into one higher dimension which we can call z, and the n-dimensional vector space is lifted into an $n+1$ dimensional space. Each point has a radius of influence around it where the positive points pull the local n-dimensional fabric into the positive z-space, and the negative points pull the local n-dimensional fabric into the negative z-space. The size of the radius of influence around each training point is determined by feature gamma.

So for the SVM method using the RBF kernel, the two critical parameters are C and gamma. A significant effort is put into searching for the optimal pairing of these two parameters, and this is done using only the training data. Once the appropriate pairing of parameter values is found, we will use that method to fit an SVM around the training data and assign values to the testing data.

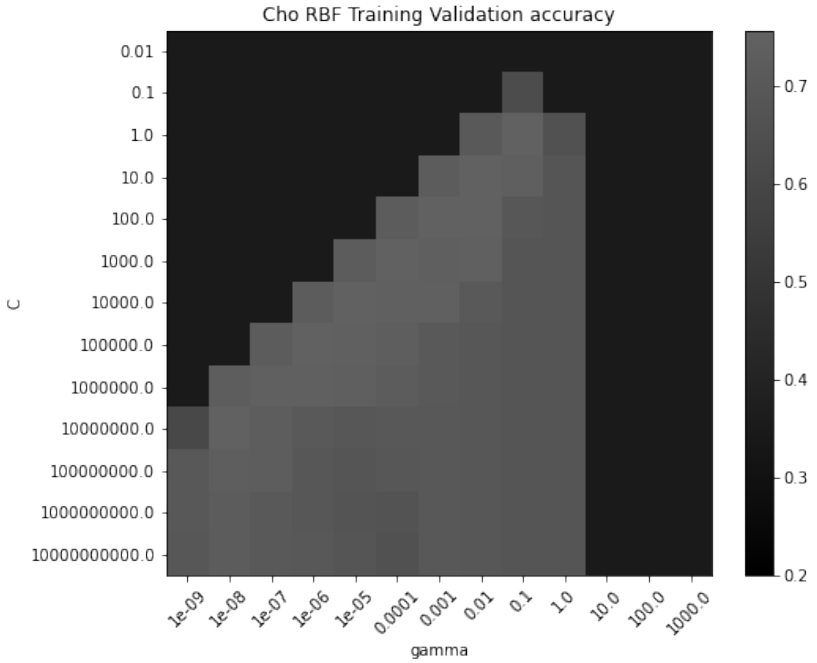


Figure 1: Shows an example of the method used for finding the optimal set of parameters for the RBF method with Cho. The optimal performance was found for a gamma value of 1e-07 and a C value of 1,000,000.

2.3 Figures and results

2.3.1 Cho

See figure 1 to view the method used to find the optimal set of parameters for Cho. The optimal set of parameters from this method were: gamma=1e-07, C=1,000,000. The performance metrics of the method can be seen below.

- Accuracy (RBF Kernel): 71.79%
- F1 (RBF Kernel): 68.88%
- AUC Accuracy (RBF Kernel): 80%

2.3.2 Iyer

We used the same method as that used for the Cho dataset (see figure 1. We removed the outliers from the dataset before performing the analysis. The optimal set of parameters from this method were C: 1,000,000,000 and gamma: 1e-06. The performance metrics of the method can be seen below:

- Accuracy (RBF Kernel): 80.77%
- F1 (RBF Kernel): 81.54%
- AUC Accuracy (RBF Kernel): 86%

2.3.3 PIE

Due to the large vector space of the PIE data of 1024 dimensions, the gridcv method timed out on our machines. As a consequence of that we decided to use gamma and C values that closely matched the optimal parameters found in the Iyer and Cho methods. We used a gamma value of 1e-06 and a C value of 100,000,000. The resulting performance of the method is shown below.

- Accuracy (RBF Kernel): 67.36%
- F1 (RBF Kernel): 68.33%
- AUC Accuracy (RBF Kernel): 83%

3 CNN

In order to understand the importance of the CNN, one must begin with a more simple model for deep learning. An Artificial Neural Network (ANN) uses a network of perceptrons at varying layers to classify a data point. This is also known as Feed-Forward Neural Networks since the inputs are processed only in one direction. In general, there is an input later, hidden layer, and output layer. Some advantages of an ANN is that it can learn any nonlinear function. This gives the name Universal Function Approximator. This is motivated by the activation function; this component introduces nonlinear properties to the network and help learn any complex relationship between a set of inputs and outputs. That being said there are challenges to ANN. For example, large inputs with high dimensions (such as a 3-dimensional data set) can increased the parameters exponentially. These requires many trainable parameters to increase precision. There is also a vanishing and exploding gradient problem where back-propagation observes wildly changing gradients. Convolutional Neural Networks (CNN) are better suited for solving problems related to image and video data. CNNs use filters (which are kernels) to extract the relevant features of the input. This is done through the convolution operation. There are many advantages to this, but most notably the CNN learns the filters without being designed with them. These filters help in extracting the most valuable features. This can capture the spatial features from an image, which can tell the model important information on how to classify the input data. The filters can also be used to simply the memory required to solve the problem by removing the data from the image that does not provide valuable information. In comparison. CNN can handle image data with spatial relationships better than ANN but both suffer from vanishing and exploding gradient problems.

3.1 Overall Method

For each of the three data sets: PIE, Iyer, and Cho, we do the following.

1. Separate into testing and training sets (if not already done)
2. Normalize the training data (and will ultimately normalize the testing data using the same normalization method)
3. Decide upon how many many hidden layers are needed in the CNN model. Since two of the data sets were 1D and one data set was 2D, we decided to use 2 different approaches for the models. Iyer and Cho featured a 1D CNN, but PIE used a 2D CNN.
4. Identify which parameters would need to be hyper tuned by running random inputs with some realistic expectation. Once the parameters are correctly identified, prepare ranges fro a gridserach cv.
5. Use gridsearchcv to determine the optimal parameters for each method, evaluating the parameter performance using the "accuracy" metric.
6. Given the optimal parameters for each method, try each method on the test data.
7. Evaluate the methods in terms of accuracy, F1 score and AUC metrics.

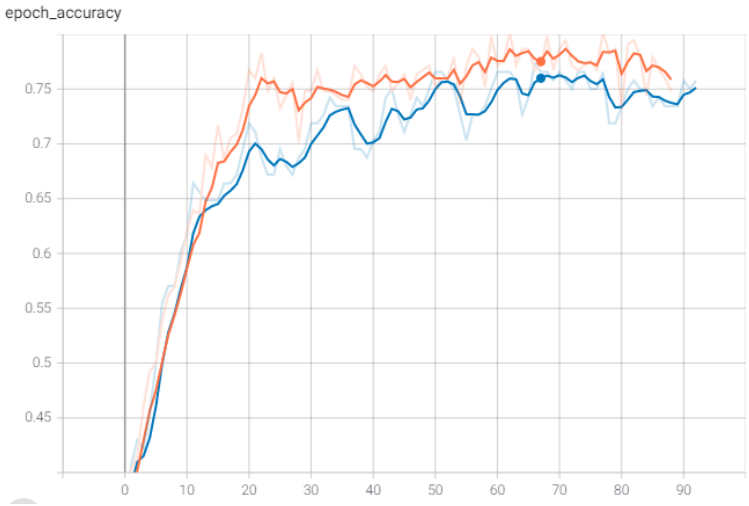


Figure 2: Here is a display on how the 1D CNN learns its cho data with each epoch. This is shown by the accuracy increasing on the training set, until the validation set begins to slow down. The algorithm does better with more epochs on the training set but not on the validation.

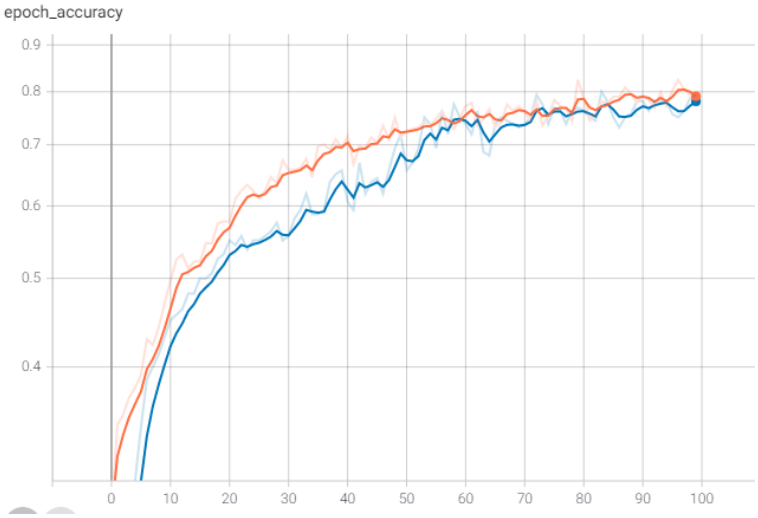


Figure 3: Here is a display on how the 1D CNN learns its iyer data with each epoch. This is shown by the accuracy increasing on the training set, until the validation set begins to slow down. The algorithm does better with more epochs on the training set but not on the validation.

3.2 Parameters

For the CNN model, the are several hyper-parameters to tune in order to measure a change in the models' accuracy, F1 score, and AUC. In our case we decided to just focus on the number of filters, kernel size, dropout rate, and size of dense layers. We used a modest range (as show in the code) to cover reasonable ranges of parameters. The results are seen in the following sections.

3.3 Figures and results

3.3.1 Cho

- Accuracy: 78.00 %
- F1: 77.78 %
- AUC Accuracy: 86.04 %

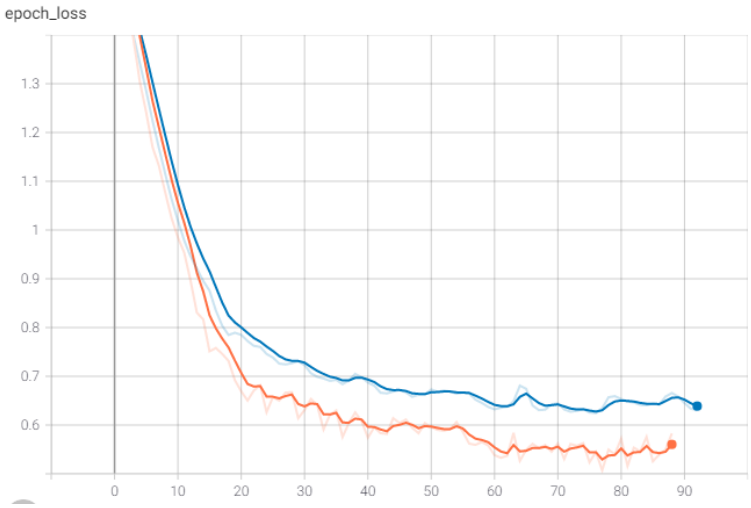


Figure 4: Here is a display on how the 1D CNN learns its cho data with each epoch. This is shown by the loss increasing on the training set, until the validation set begins to slow down. The algorithm does better with more epochs on the training set but not on the validation.

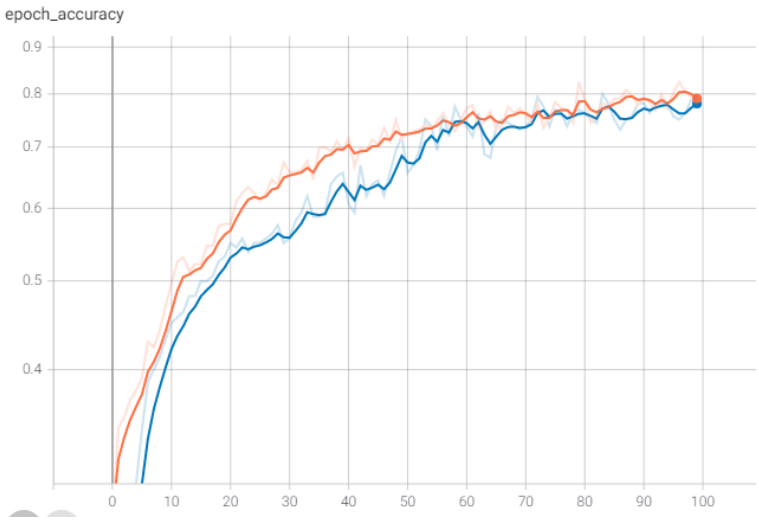


Figure 5: Here is a display on how the 1D CNN learns its iyer data with each epoch. This is shown by the loss increasing on the training set, until the validation set begins to slow down. The algorithm does better with more epochs on the training set but not on the validation.

3.3.2 Iyer

- Accuracy: 75%
- F1: 58.08%
- AUC Accuracy: 78.71%

3.3.3 PIE

- Accuracy: 58.70%
- F1: 57.72%
- AUC Accuracy: 77.56%

4 XGBoost

XGBoost is an implementation of gradient boosting. Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, in this case decision trees. Its weak prediction models are updated by the error of the previous model, by increasing the weight on train cases the model gets wrong. Gradient boosting will then train the next weak model on the re-weighted train set.

4.1 Overall Method

For each of the three data sets: PIE, Iyer, and Cho, we do the following.

1. Separate into testing and training sets (if not already done)
2. Normalize the training data (and will ultimately normalize the testing data using the same normalization method)
3. Identify relevant and interesting kernels, and for the kernels identify the different important parameters for each kernel method.
4. Use gridsearchcv to determine the optimal parameters for each method, evaluating the parameter performance using the "accuracy" metric.

4.2 Parameters and Figures

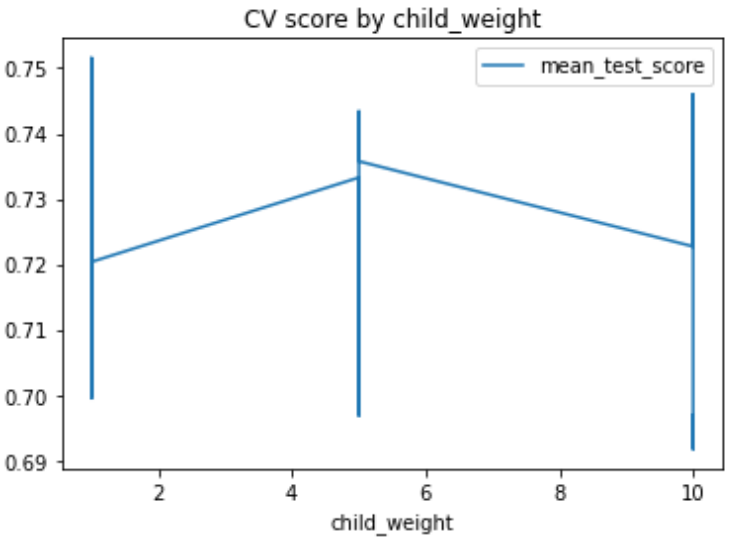
Parameters that were tuned during cross validation searching were gamma, child weight, max depth, and the column sample.

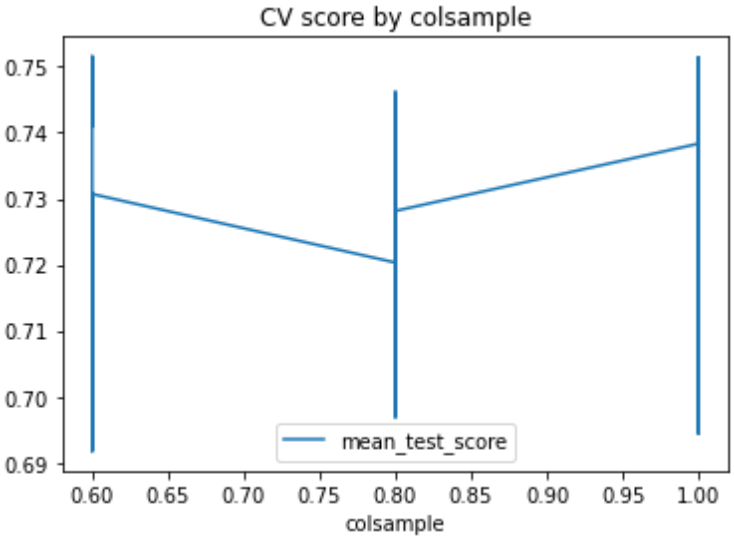
Below there are examples of the parameters being tuned with cross validation.

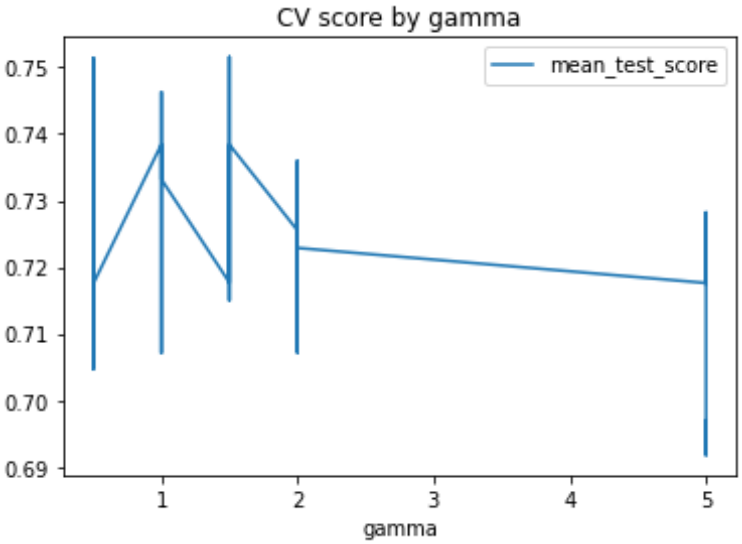
4.3 Data sets and results

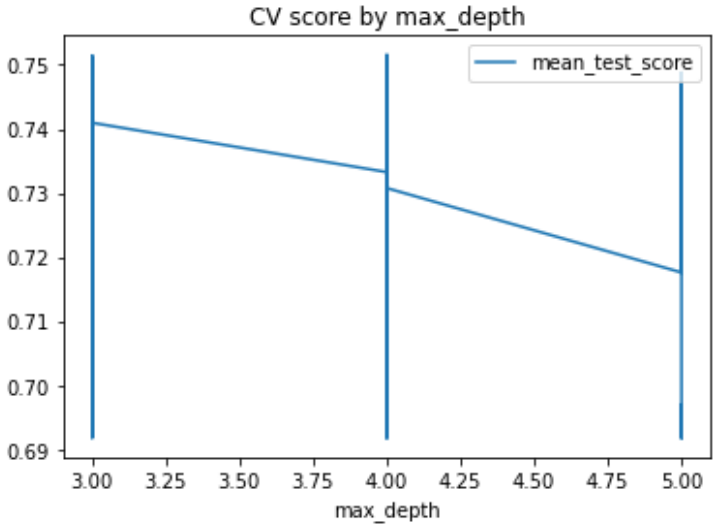
4.3.1 Cho

- Accuracy: 69.07%
- F1: 68.86%
- AUC Accuracy: 81.00%


CV score by child_weight
mean_test_score
0.75
0.74
0.73
0.72
0.71
0.70
0.69
2
4
6
8
10
child_weight


CV score by colsample
mean_test_score
0.75
0.74
0.73
0.72
0.71
0.70
0.69
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00
colsample


CV score by gamma
mean_test_score
0.75
0.74
0.73
0.72
0.71
0.70
0.69
1
2
3
4
5
gamma



4.3.2 Iyer

- Accuracy: 87.6%
- F1: 86.95%
- AUC Accuracy: 89%

4.3.3 PIE

There were significant limitations with training the PIE data set due to spatial information being lost when collapsing it to a vector.

- Accuracy: 9.26%
- F1: 8.33%
- AUC Accuracy: 54.00%

5 Conclusion

We have evaluated the Cho, Iyer, and PIE datasets using the three different methods SVM, CNN, and XGBoost. The overall performance of these methods can be seen in table 1 below. None of the methods performed particularly well. The PIE dataset proved to be particularly problematic. This is likely due to a number of factors: the necessary destruction of the implicit spatial data of a photograph in order to use a vector-based method, and the high dimensionality of the resulting data vectorized data. Another issue was that due to the very large dataset size, we were unable to use the grid-based search method to find the optimal parameters to use for the dataset.

The SVM method performed consistently with an accuracy between 67% and 81% for all datasets. The CNN method had a wider variety of performance and actually performed overall worse than the SVM, with accuracy between 58% and 78%. Our XGBoost implementation had the widest range of performance from 9% accuracy up to 87% accuracy. The fall off of performance with XGboost is due to the fact that due to time constraints we cut off some of the columns of the data in order to speed up the learning. This further destroyed the spatial data and caused very inconsistent performance. The consistent performance of SVM meets our expectations. CNN also performed well overall, although it is particularly well suited for 2D data so in the future we would like to perform the analysis using a non-vectorized format.

Table 1: Final results comparison

		Cho	Iyer	PIE
SVM	Accuracy	71.79%	80.77%	67.36%
	F1	68.88%	81.54%	68.33%
	AUC	80.00%	86.00%	83.00%
CNN	Accuracy	78.00%	75.00%	58.70%
	F1	77.78%	58.08 %	57.72%
	AUC	86.04%	78.71%	77.56%
XGBoost	Accuracy	69.07%	87.60%	9.26%
	F1	68.86%	86.95%	8.33%
	AUC	81.00%	89.00%	54.00%