Music Symbol Recognition:
A comparative study

Drew Erikson & Carter Mintey

April 27, 2020

Abstract

One overlooked yet rather complex issue requiring the heavy use of artificial intelligence-based methods is Optical Music Recognition (OMR). Given proper sheet music that could be interpreted and played by a human, it is the computer's task during OMR to learn how to read and interpret a scanned sheet music document and produce a machine-readable version of the written score with little-to-no errors. This practice is not only intriguing to musicians worldwide, but it also serves implications in the realms of archiving historic compositions and making music more accessible in an interconnected age. A descendant of Optical Character Recognition (OCR), OMR is in need of further iteration. As more and more music is in need of archiving, the scale of the problem increases. Our work examines several methods of symbol recognition so as to automate this process as we study K-Nearest-Neighbors, Support Vector Machines, Logistic Regression with Stochastic Gradient Descent, and Deep Learning algorithms as potential accurate and efficient solutions to this problem, offering different layers of accessibility, accuracy, and efficiency to an oft underrated field of study.

1 Introduction

Music is creative, artistic, and inspiring—quite often, it inspires those across all spaces. As millions of musical scores are written for performance and archiving, music inspired computer scientists to capture its art and preserve its importance in a digital form. Paper is less persistent than a solid-state drive, and thus is born the idea of Optical Music Recognition.

Optical Music Recognition (OMR) was pioneered by Pruslin and Prerau in the late 1960s [13]. Since then, there have been many developments in the realm of music recognition as new technologies and methodologies have emerged. As Rebelo et al. notes in [2], the bi-dimensionality of music makes classification much more difficult than optical character

recognition, or OCR. In the past, OMR was classified as an OCR task, but in recent years has morphed into its own category of research. Literature in this space seeks to optimize and operationalize a process that is integral to the preservation of a growing pool of artwork.

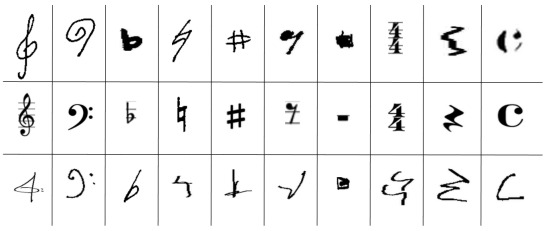

Figure 1: An example of the complexities and nuances between handwritten and printed music symbols

With this ever-increasing scale emerges an issue: keeping up. More classical techniques as explored by Pruslin and Prerau could not possibly stay at pace with the explosives rate that music is currently being written in both electronic and handwritten contexts. This introduces the technical need for automation in the OMR process, and the need for machine intelligence therefore follows close behind. Our review of recognition algorithms in the OMR process seeks to explore the current space developed by experts in machine learning, analyzing several cutting-edge algorithms. Buzz-words such as 'neural networks' or 'support vector machines' tend to garner agreement for the best general performance; however, we aim to take a closer look at several different approaches including these. In this vain, we hope to optimize the OMR process beyond the approach taken to most modern problems using modern machine learning libraries and a careful experimental design.

2 Literature Review

The OMR task is quite complex, requiring several steps to split the data up into easier tasks [2]. One such task is the recognition of music symbols and is arguably the most important task at hand. To solve this task, researchers have used several different techniques throughout the years including, complex pattern recognition [5] to more advanced AI algorithms. In this paper, we will focus our attention on 4 machine learning approaches to classification: K-Nearest Neighbor, Support Vector Machines, Logistic Regression with Stochastic Gradient Descent, and Deep Neural Networks. Each of these algorithms poses different issues and potential benefits–especially within the field of OMR–and it is therefore critical to examine each approach and all of their unique studied qualities within this space.

2.1 K-Nearest Neighbor

Our first classification algorithm that we used for musical symbol recognition is the *k-nearest neighbor algorithm*. One of the most fundamental and simple classification models, Peterson et al. has shown that this should be one of the first choices for a classification study like ours when there is little to no knowledge about the data distribution [11]. In essence, the algorithm is described as literally comparing an example in question to the "k nearest" neighboring samples to decide its classification. For example, with $K = 4$, an unknown sample can be binarized as 'Class A' if the majority of its four nearest neighbors belong to class A, as shown in Figure 2.

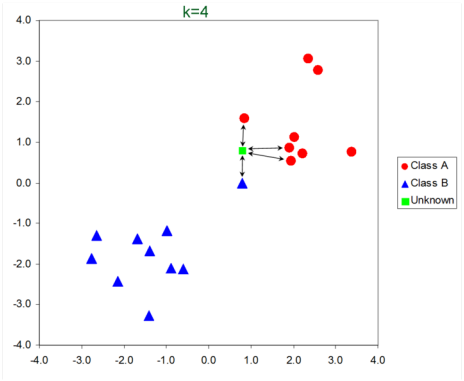

Figure 2: Graphical representation of a KNN example [11]

As examined by both Rebelo et al. and Jorge Calvo-Zaragoza et al. the k-nearest neighbor recognition algorithm is a good fit for OMR [2, 6]. Musical symbols share similar features (stems, note-heads, dots, lines, etc.) that make each one distinguishable yet similar to its peers. In this sense, there is a 'neighborly' relationship between each symbol. Symbols that look similar and share similar features will be 'closer' together, in theory. Therefore, with absolutely no prior intuition of the shape of our training set provided by [9] – our sample of which containing 15200 symbols spread across 32 symbol classes – we use a framework established by previous work with confidence of high accuracy in our results. As is with all of our algorithms, we consult the professionally optimized versions of the k-nearest-neighbor algorithm as designed within the Scikit-Learn package for Python [10].

2.2 Support Vector Machine

To the contrary, Support Vector Machines are much more complex. With this trade-off comes more flexibility in classification–particularly, we are no longer bounded to strictly linear classification. Classes which are linearly separable allow their samples to be classified

based on which side of a straight line they are located. Unfortunately, we see in Figure 3 that not all distributions follow this pattern. Although we recognize linear relationships and 'neighborhoods' within musical symbol data, there are many elements of noise and unique marks that cause the data-set to potentially exhibit nonlinear behavior, as posited by prior work in this field [1].

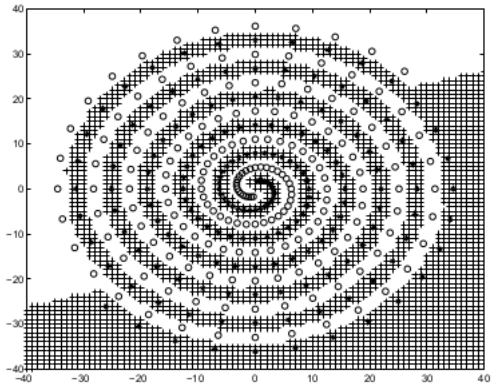

Figure 3: Graphical representation of a classic spiral classification problem, where different shaded points represent each class [15]

Suyken et al. gives a detailed explanation and derivation in [15] as for why support vector machines are uniquely tailored to handle non-linear classification problems, such as those that concern the recognition of musical symbols in sheet music. As mentioned earlier, the related work in this field done by Rebelo et al. acknowledges this need and shows that support vector machines are actually one of the best performing recognition algorithms when it comes to separating music data into each contingent class [2].

Additionally, non-linear recognition can be useful in modifications to OMR as a practice. Substantially different frameworks such as a measure-based approach by Wang et al. [14] examines recognition at a measure level–that is, in a musical piece written in common time, this approach attempts to classify groups of four 'beats' at a time. It is rare that two measures share a lot in common unless a musical piece is highly repetitive, and this quality renders a K-Nearest-Neighbors approach as rather unhelpful. However, a non-linear method such as Suyken's description of a support vector machine could easily adapt to this modification.

2.3 Logistic Regression with Stochastic Gradient Descent

One of the most popular machine learning classification algorithms, logistic regression with stochastic gradient descent (SGD) presents an easy, adaptable, and understandable approach to the OMR task. Hearkening back to a more linear approach–recognizing the similarities between different music symbols–there has been extensive research and application using this

algorithm. Particularly, this method is scalable. As examined by Bottou et al. this methodology is relatively accurate–using regression to separate classes linearly–and extremely efficient, no matter the scale of the data [3]. In terms of our own work, we hope to cite stochastic gradient descent as a uniquely efficient classification algorithm, for when both accuracy and efficiency of a particular OMR practice is at stake.

Setting this algorithm apart is its separation of samples in its training. Each example musical symbol, for example, is examined alone, tuning the recognition model each time a new sample is passed in. Notably, the literature inspiring this work by Rebelo et al. does not employ the use of logistic regression, possibly as a direct result of this feature [2]. This stochastic nature usually decreases accuracy in large sets as mentioned by Bottou, and for this reason serves as a 'black sheep' in our study: an algorithm that has been reviewed extensively as less accurate, more adaptable, and incredibly efficient.

2.4 Deep Neural Network

Neural networks have become quite common for the OMR classification task. Based on the structure of the human brain, neural networks can be very flexible in their ability to learn complex structures within the data [8]. Specifically, convolutional neural networks, or CNNs, are a very powerful tool that can classify very large data-sets much better than linear methods. It has been shown that deep CNNs have performed very well in image recognition tasks [7], so it makes it an obvious contender to be able to recognize the complex structure of neural networks.

Convolutional neural networks work by extracting a subset of features from the data-set to get a new representation of the data. Then, this new representation can be fed into another convolutional layer for more feature extraction. This can be repeated for many layers until the data has been represented in a more linear fashion for classification. This flexibility allows for many different combination of networks, each with varying levels of complexity and accuracy. Lee et al. and Pereira et al. [7, 12] have done comparative studies of different neural networks and found that the results vary significantly from one another. Their findings show that deep CNNs are among the best classifiers for the OMR classification task because of their ability to classify highly non-linear data.

Unlike some of the previous classifiers, the necessity for many layers of convolutions lead to a much more complex algorithm. While we gain a much more accurate classifier, we lose out on the efficiency benefits, especially if this classifier is run on a weak system without a GPU. It also comes with the cost losing interpretability which makes it harder to understand what is actually being done to the data for classification. However, for our task of simply

wanting to classify our data, this loss of interpretability is not as significant to us as is the performance hit of both training and then classifying our samples.

3 Approach

This work took a very direct approach to comparing the different symbol recognition algorithms, particularly focusing on the context of musical symbols. As discussed, we compared the k-nearest neighbors algorithm (KNN), a support vector machine algorithm (SVM), a logistic regression with stochastic gradient descent algorithm (SGD), and a neural network algorithm.

For the KNN, SVM, and SGD classifiers, we used Scikit-Learn, a robust machine learning package for the Python programming language [10]. For the deep learning model, we utilized the algorithm found in the GitHub repository[1] based off of related work done by Pacha et al. [9]. In order to adapt these implemented approaches, we wrote several scripts to train and test each algorithm.

Each of these scripts that we wrote utilized several modern Python packages to fit and run each model on our music symbol data-set, collect efficiency and accuracy data, and build tables/figures for analysis on each model, stratified by model parameters, music symbol class, and overall algorithm. At the core, we studied performance and time-efficiency for the *note recognition* portion of the OMR procedure.

Since the pre-processing and encoding steps are beyond the scope of this project, we used the HOMUS data-set provided by the research done by Pacha et al. In this data-set, 15200 symbols are spread across 32 classes. We designed two compressed sets of symbol images with 96x96 pixels and 112x112 pixels respectively, sacrificing minimal performance in each model for a much more reasonable training and testing duration. This dual approach ensured that we would compare different resolutions of image recognition while maintaining a relatively small data format for processing. This further opened up different levels of context for our study.

In an attempt to avoid introducing bias on our data-set, each image was then randomly rotated and scaled to design a more realistic and robust model. The data was randomly split into training (80%) and test (20%) subsets. For the testing of our models, we combined the validation and test sets to get a more holistic view of the accuracies produced by our models. Because the dimensionality of our data-set is so high, we also introduced a dimension-reduction step for our KNN, SGD, and SVM models. We decided upon projecting our data into a space of 100 components as this offered reasonable results, while keeping our

[1] https://github.com/apacha/MusicSymbolClassifier

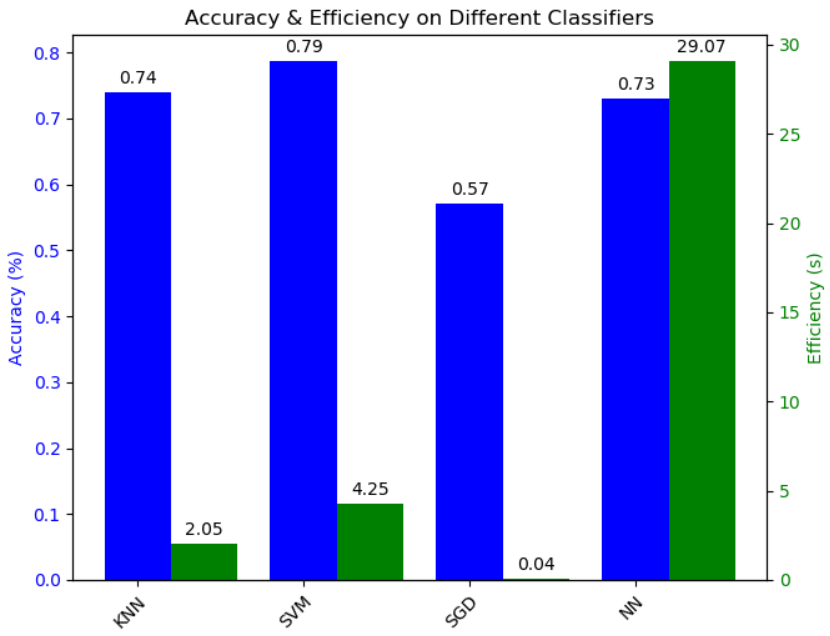

Figure 4: Accuracy vs. efficiency of different classifiers

efficiency relatively high. Using principal component analysis (PCA), we found that the top 100 components for the 112x112 pixel images explained 50% of the variance in our data. On the other hand, for the 96x96 pixel images, we found that the same 100 components explained around 60% of our variance.

After this careful design and analysis, we aimed to draw conclusions about the different algorithms and their parametrizations such that we can choose an optimal recognition algorithm specifically for the OMR recognition process. Although it is known that algorithms such as SVM's or neural networks tend to perform best for general issues, we ask whether or not this general property holds for our particular issue.

4 Design & Results

In order to develop a reliable and robust experiment, we developed an intuitive interface for importing the HOMUS data-set and, for each model, fitting the model to this data-set, testing for accuracy and efficiency, and displaying results in tabular and graphical formats[2].

An important part of our study was fine-tuning the parameters of each model. Specifically, tuned parameters influence how well the model is learned. These 'hyper-parameters' induce different penalties and rates for learning algorithms, particularly SGD and SVM,

[2]This project is open-source. Source code is available at github.umn.edu/eriks074/csci-4511w-omr-proj

and the best way of optimizing these is through a cross-validation methodology called 'grid-search'. We partitioned the training data into sub-groups, through which provided hyper-parameter combinations could be tested, alternating which group of 10% is used as test data against the remaining training data. As a validation effort, we employed the usage of two different machines to ensure that algorithm results were consistent across machines and that parameters were tuned consistently and optimally.

The most obvious and general analysis we took towards this problem was to measure, compare, and contrast accuracy and efficiency in terms of seconds and percent correctly classified, respectively. The results for this general study is shown in Figure 4. Recall that all results have been adequately pre-processed on our end—that is, we have compressed the images to an easily usable format and applied PCA to reduce the dimensionality of each symbol example. Even though the 96x96 pixel compression format had a better explained variance after PCA, we ultimately decided to use the 112x112 compression format, as there were negligible improvements in performance from each model (specific results can be viewed in the source repository; see appendix (8) for details).

In terms of measures of accuracy, there is small variance. Our SVM classifier emerged as the best at music symbol classification, with our KNN classifier as a close runner-up. However, the conversation becomes more complicated when discussing efficiency. The neural network takes the longest and therefore is the least efficient, taking almost 30 seconds to classify the test set. To contrast, the low-performing SGD classifier almost immediately carries out its classification. Uniquely, KNN is second-best in both accuracy and efficiency, and it also is the simplest algorithm to adapt for implementation. This wealth of benefit prompted us to look at a closer parametrization of the algorithm, testing different numbers of neighbors to gauge accuracy and efficiency at a more micro-management scale. This can be seen in Figure 5. Not surprisingly, as K increases, as does the time elapsed to classify the test set. However, there is an obvious peak in accuracy at $k = 4$, and knowing this, we chose this parameter assignment for the entire project.

Our final and most involved stratification and design involved different symbol classes. As discussed, different algorithms are more or less tailored for linear relationships. Many musical symbols share common components, such as stems, note-bodies, dots, and lines; these symbols, we posit, would be better classified by linear algorithms such as KNN. Conversely, symbols that do not fit into this mold—notes such as rests, time signatures, and accidentals—would be better handled by a non-linear algorithm such as SVM. The HOMUS data-set includes 9 time signature symbols, 7 note symbols, 7 rest symbols, and a handful of miscellaneous music symbols. We separated the accuracy each algorithm exhibited for each symbol class, and the results can be seen in Figure 6.

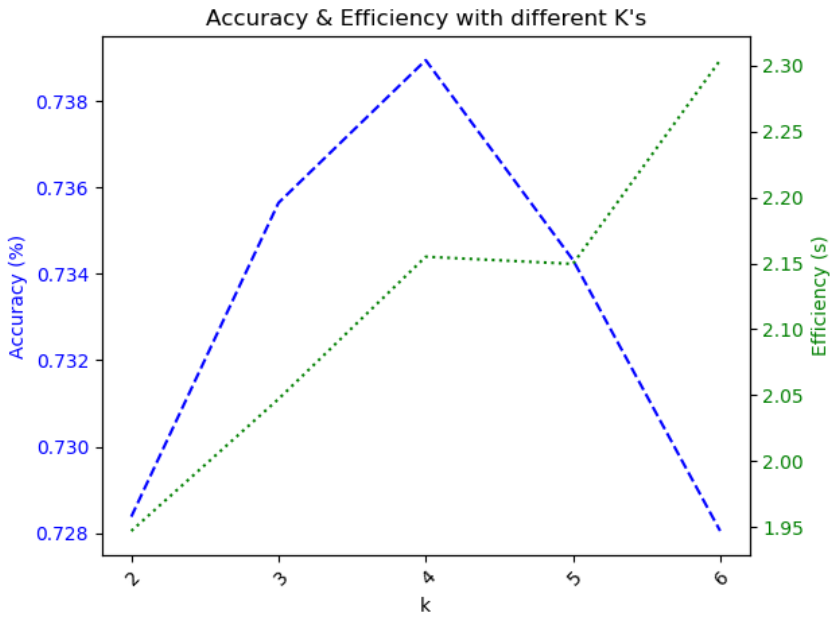

Figure 5: Accuracy vs efficiency of KNN with different values of K

5 Analysis

From our results, we can see that in terms of efficiency, SVM is our clear cut winner, gaining a 5% advantage over KNN. However, KNN is 2 seconds faster than SVM at classifying our 3,000 symbols. Arguably, with this number of symbols to classify, 2 seconds is not much of a difference when looking at classifying a single sheet of music, but it does pose an issue when looking to classify an entire score, of which there could be several more thousand symbols to classify. Still, 2 seconds is not much longer to wait, and I would imagine that waiting a few seconds longer to get a better, more accurate classification would lead to less manual adjustment, thus saving more time in the long run.

While we originally hypothesized that KNN would do better on linear classification for symbols such as stems and note-bodies, we observed that KNN did just as well, if not better than SVM on some non-linear symbols such as time signatures. This is probably in part to how we initialized our KNN model. Instead of using the standard K-Nearest Neighbors algorithm, we set up our model to be weighted. Namely, we used a weight of inverse distance. This means that neighbors closer to our sample point have a larger effect on how that point is classified than a neighbor who is farther away. This allows for our classification to be less linear since the classification boundaries are molded around the clusters of similar data points, thus providing us with a very competitive classifier.

One of the biggest surprises to us was that our neural network algorithm did not really

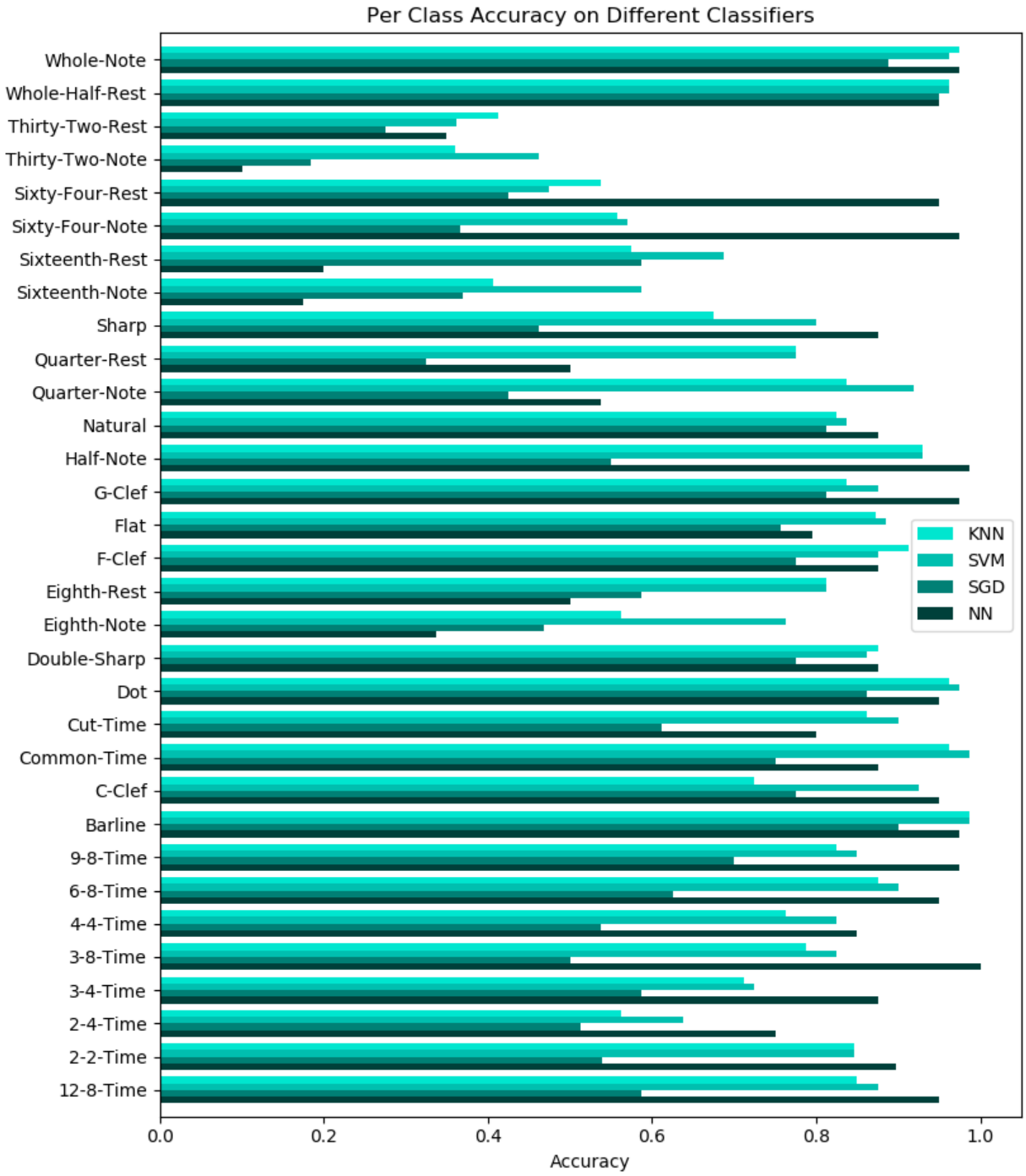

Figure 6: Accuracies of each class for the different classifiers

stand out among our other models. When we originally looked into this model, we were finding studies with reports of 96% accuracies and were really excited about what we would be able to do with it. However disappointed we were, our results for the neural network are not surprising. We were working with pictures of 112x112 pixels, which leads to very high dimensional data–a dimensionality of 12,544, to be precise. Training with only 12,160 sample points, our neural network simply did not have enough data to be able to report hyper-accurate results and was most likely overfitting. The biggest issue with using neural networks as classifiers is the massive amount of data that is required to produce accurate results, and we simply did not provide our model with the best environment to do so.

One result that is not surprising is the logistic regression classifier. We knew that music data is highly non-linear and that a linear classifier would not fare very well against our other models, but nonetheless, it is interesting to observe why this happened. In our data, there are lots of images that look very similar and if we could plot this data we would quickly find that it is not linearly separable. This means that a linear classifier has a very hard time to accurately find it's decision boundary. For those symbols that are very similar and would be clustered close together, this decision boundary could actually end up splitting this cluster down the middle as to minimize the error. Taking this into consideration, the accuracy of only 57% makes sense. Because the decision boundary is linear, it also makes for a very quick classification leading to a highly efficient model.

While some of these results align with related work, we found that other parts did not, and this serves several implications. The logistic regression classifier, for example, is a novel extension of the work done in this field. Although lacking in complete accuracy, this technique is easy to set up and rewards almost immediate results that, although not the best, are passable dependent on the context. This ease of access adds a layer of context to a once-very-complicated work, as a certain level of understanding is necessary to fine-tune and operate an SVM or neural network machine.

In this sense, we do not offer one unique optimization to "rule them all", but instead offer context-dependent optimizations. A quick-start approach would utilize the logistic regression classifier, for example. Conversely, a study done concerning the more linear relations among symbols would be optimized using the KNN algorithm; a more careful and studied nonlinear approach would be best-fitted for the usage of and SVM or neural network. This divide allows professionals and hobbyists alike to access this space and utilize it to archive more music, which seems more useful than any "complete" optimization could ever be.

6 Discussion & Future Work

In hindsight, there are ways to fine-tune our neural network to better deal with data that has a dimensionality that exceeds the number of training samples. One such way is to generate new data from existing data by randomly rotating, scaling, and cropping our images. Usually, these data points then have to have less weight when training the model, but it has been shown in some cases to gain more accurate results. Another way to prevent our model from over-fitting is to do as Pacha et al. did in their study [9] and combine several large data-sets to train the model on. In their case, they had a training sample of over 90,000 images, both handwritten and printed. Along with an increase of data samples, playing around with parameters is key for neural network learning and is something that we could improve on in the future. Pacha's Github page provides a spreadsheet[3] of the results of different parameters on several different models.

Additionally, we acknowledge a clear inconsistency between our work and the related work in terms of symbol size, compression, and dimensionality. Rebelo et al. uses a 20x20 compression set in terms of pixels per symbol image [2]. This is almost a sixth of the size of our scheme, and could explain and complicate the impressive results found in these related studies. Smaller image size allows for works like these to not apply PCA, as 100% of variance can be explained by using every dimension of the data. With low dimension size—400, in terms of Rebelo et al.–it is reasonable in terms of time duration to do so. Our approach is more realistic in that we use thousands of dimensions in total, while using modern techniques such as PCA to increase efficiency of model fitting and training.

One way to try to improve our results is to try to use multiple models to classify a single symbol. This theory was tested in [4] by Byrd et al. From Figure 6, we can see that while SVM and KNN often traded blows on each symbol, there are some cases when the neural network comes out with a demanding improvement on accuracy. In situations such as Sixty-Four-Rest or 3-8-Time, it is possible that we could see a great improvement in accuracy if we were to implement a voting system between SVM, KNN, and NN to get a holistic view of all models. This of course becomes a problem with classification efficiency as the neural network takes the longest time to classify a symbol. As is, it might be worth the efficiency hit for a much better accuracy. Because the neural network that we are using has around 30 layers, running these tests on a GPU could yield better results in terms of time-efficiency. However, as we have stated before, neural networks are very extensible and our neural network in no way claims to be the end-all-be-all in this study and as such, can be tuned to be much more efficient and accurate.

[3] https://bit.ly/2zy652b

It is also worth noting that prior work done in this space explores problems similar to those explored by this work. Rebelo et al. is the most notable of these works, accentuating the minutia of each step of the process of OMR, as well as several outstanding road-blocks to the optimal and complete solution of the problem [2, 1]. However, we posit that this work serves to complicate the pre-existing work done in this field, and that our intensive study of strictly the symbol recognition problem in OMR is unique in this way. Not only do we obtain different results from the overlapping algorithms tested, but also we conduct these experiments with different operationalizing techniques. Rebelo et al. continues to lean on the assumption that a general solution to the OCR problem is sufficient for the OMR problem, which we argue is not the case. Due to the different degrees of linearity in sheet music components that other works do not recognize, our revision of the natures of these algorithms is an important expansion of the conversation being held in this field.

7 Conclusion

With this work we aim to help better the practice of Optical Music Recognition. While true that other reviews attempt to optimize the difficult practice of OMR, we refined our focus to a diverse set of data and an integral part of the problem: music symbol recognition. This allows for a unique *contextualization*; that is, our multi-faceted analysis allows the music hobbyist and the professional composer to take tailored approaches to the issue. Whether it be the requirement of easy set-up, quick results, or the best possible accuracy in recognition, this work adds context to the problem of OMR and adds a layer of accessibility to an otherwise heady field of study. This is ever-important to a society that rapidly writes music and develops technology, and has future implications to other sects of OCR as well.

Our review of OMR and different modern machine learning algorithms to complete its recognition step is an intensive overview that, while showing promising results, requires future iteration and development as technological capabilities expand. Utilizing several deep learning tactics such as a variety of models, unique pre-processing, and component analysis, we engage with existent works in this space to better the field of OMR for many different audiences and applications. In our review, we hoped to have built a methodology to best advocate for the preservation and appreciation for all works of music in the foreseeable future.

8 Contributions

All writing and framing of the project was completed collaboratively. Otherwise, each author did the following technical work:

Drew Erikson:

- Implemented driver program for running each recognition algorithm
- Wrote and fine-tuned experimental code for KNN, SVM, SGD measures
- Designed & produced accuracy and efficiency visualizations for KNN hyper-parameters
- Designed & produced accuracy and efficiency visualizations for holistic analysis
- Nuts and bolts (Serialization of models, 'refit' command line arguments
- Wrote Introduction, Literature Review, Approach, Results, and Conclusion

Carter Mintey:

- Obtained HOMUS data-set and split into training, validation, and testing subsets
- Trained and tested the Neural Network
- Implemented several helper scripts for data collection and analysis
- Designed & produced the per-class accuracy graph
- Cleaned and fine-tuned code
- Wrote Introduction, Literature Review, Analysis, Discussion, and Conclusion

References

[1] Ana Rebelo, Ichiro Fujinaga, F. P. A. R. M. C. G., and Cardoso, J. S. Optical music recognition: state-of-the-art and open issues. *International Journal of Multimedia Information Retrieval 1*, 3 (2012), 173–190.

[2] Ana Rebelo, G. C., and Cardoso, J. S. Optical recognition of music symbols. *International Journal on Document Analysis and Recognition (IJDAR) 13*, 1 (2010), 19–31.

[3] Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[4] Byrd, D., and Schindele, M. Prospects for improving omr with multiple recognizers. In *ISMIR* (2006), pp. 41–46.

[5] Coüasnon, B. *Segmentation and recognition of documents guided by a priori knowledge: Application to musical scores*. PhD thesis, Ph. D. thesis, IRISA, France, 1997.

[6] Jorge Calvo-Zaragoza, J. H. J., and Pacha, A. Understanding Optical Music Recognition. Submitted through Cornell University arXiv.org, August 2019.

[7] Lee, S., Son, S. J., Oh, J., and Kwak, N. Handwritten Music Symbol Classification Using Deep Convolutional Neural Networks. In *2016 International Conference on Information Science and Security (ICISS)* (Dec. 2016), pp. 1–5.

[8] Liljenström, H. Neural stability and flexibility: a computational approach. *Neuropsychopharmacology 28*, 1 (2003), S64–S73.

[9] Pacha, A., and Eidenberger, H. Towards a universal music symbol classifier. In *14th International Conference on Document Analysis and Recognition* (Kyoto, Japan, 2017), IAPR TC10 (Technical Committee on Graphics Recognition), IEEE Computer Society, pp. 35–36.

[10] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[11] Peterson, L. E. K-nearest neighbor. *Scholarpedia 4*, 2 (2009), 1883.

[12] Pinheiro Pereira, R. M., Matos, C. E., Braz Junior, G., de Almeida, J. D., and de Paiva, A. C. A Deep Approach for Handwritten Musical Symbols Recognition. In *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web* (Teresina, Piauí State, Brazil, Nov. 2016), Webmedia '16, Association for Computing Machinery, pp. 191–194.

[13] Pruslin, D. H., Read, G., Prerau, D., Hastings, W., Mahoney, J., Widrow, B., Hoff, M. E., Rumelhart, D., Hinton, G., Williams, R., et al. *Automatic recognition of sheet music*, vol. 21. Taplinger, 1966.

[14] Raphael, C., and Wang, J. New approaches to optical music recognition. In *ISMIR* (2011), pp. 305–310.

[15] Suykens, J. A., and Vandewalle, J. Least squares support vector machine classifiers. *Neural processing letters 9*, 3 (1999), 293–300.

Appendix

Source Example: KNN.py

Listed is an example of our source code, all of which is available at github.umn.edu/eriks074/csci-4511w-omr-proj.

```
from sklearn.neighbors import KNeighborsClassifier
from AccuracyHelper import *
import numpy as np
import time
import joblib
import matplotlib.pyplot as plt

def KNN(X, y, X_test, y_test, refit=False):
    ks = range(2,7)
    a = [0 for x in range(5)]

    # for k=2...10, run KNN classifier
    accuracies = []; end = []; start = [];
    for k in ks:
        start.append(time.time())
        print(f"Running KNN with k={k}...")

        if refit:
            knn = KNeighborsClassifier(n_neighbors=k, weights='distance',
    n_jobs=-1)
            print("Fitting the data...")
            knn.fit(X, y)

            print("Saving model... ")
            joblib.dump(knn, f"models/knn-{k}.sav")
        else:
            print("Loading model...")
            knn = joblib.load(f"models/knn-{k}.sav")

        print("Getting predictions...")
        y_pred = knn.predict(X_test)

        print("Calculating loss...")
        accuracy = CalculateAccuracy(X_test, y_test, y_pred)
        a[k - 2] = accuracy[-1,1]
        DisplayAccuracy(accuracy)
```

```
36          accuracies.append(accuracy)
37          end.append(time.time())
38          print(f"Loss: {accuracy[-1,0]}\t Time: {end[-1]-start[-1]:.2f}s\n"
    )
39
40      # errors for k=2...10
41      a = np.array(a)
42      best = np.argmax(a)
43
44      fig, ax = plt.subplots()
45      PlotKNN(ks, accuracies, np.subtract(end, start), ax)
46      fig.tight_layout()
47      plt.show()
48
49      # take the best
50      return accuracies[best], best+2
51
52
53  def PlotKNN(ks, accuracies, efficiencies, ax):
54      acc = [x[-1,1] for x in accuracies]
55      x = np.arange(len(ks))
56
57      ax2 = ax.twinx()
58      ax2.plot(x, efficiencies, linestyle='dotted', label="Efficiencies",
    color = 'g')
59      ax2.set_ylabel("Efficiency (s)", color = 'g')
60      ax2.tick_params(axis='y', labelcolor='g')
61
62      ax.plot(x, acc, linestyle='dashed', label="Accuracies", color = 'b')
63      ax.set_ylabel("Accuracy (%)", color = 'b')
64      ax.tick_params(axis='y', labelcolor='b')
65
66      ax.set_title("Accuracy & Efficiency with different K's")
67      ax.set_xticks(x)
68      ax.set_xticklabels(ks)
69
70      plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode=
    "anchor")
```