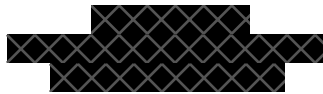


Leveraging Model Averging, Label Noise & The Polysemy of Mars Spectrometry Gas Chromatography



Abstract

My solution is an ensemble of ten identical Deep Neural Networks (DNNs), each using a combination of Label Distribution Learning (LDL) and a novel model averaging algorithm. First, I will outline several key model design choices followed by an explanation of the vector features used. Here, we construct the feature vectors using peak detection features alongside n-difference features inspired by the derivatives produced by the SG algorithm (Savitzky & Golay, 1964). Next, I will introduce *FixedSoup*, a neural network weight-averaging algorithm inspired by "GreedySoup" (Wortsman et. al., 2022) that improves the performance of a single model above the performance of an ensemble of fixed size. Finally, I discuss the potential existence and possible sources of label-noise in the given data and give possible solutions to address them for future research.

1 Modeling

Network Architecture

Given a feature vector we use a modified three-layer DNN to predict all classes simultaneously. After each of the two hidden layers, BatchNorm (Ioffe & Szegedy, 2015) and LayerNorm (Ba et. al., 2016) are sequentially applied before a ReLU activation function. Then after the final layer we add a bias of $\log(\bar{y}^i / (1 - \bar{y}^i))$ before using a sigmoid activation. Where \bar{y}^i is the average of the i^{th} label in the training set. We trained the model using the Adam optimizer (Kingma et. al., 2014) using a three-component loss:

$$L(y, p) := \text{LogLoss}(y, p) + \lambda_h \text{HingeLoss}(y, p) + \lambda_l \text{LCGLoss}(y, p)$$

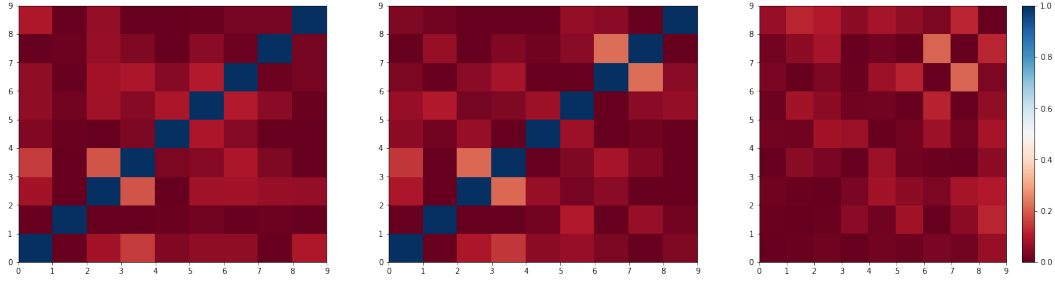
where LogLoss and HingeLoss are the log-loss and hinge-loss respectively. LCGLoss is the *Label Correlation Grid* (LCG) loss proposed by Guo & Zheng (2022). These losses are weighted with $\lambda_h = \lambda_l = 0.1$

Label Correlation Grid

Given a vector $t \in \mathbb{R}^9$ which can represent a label or network output. We then sample $\mathcal{N}(t, .5\mathbf{I})$ nine times and concatenate the samples along a new axis, resulting in $t' \in \mathbb{R}^{9 \times 9}$. We then construct the correlation grid using the correlation function $d(t) := \text{Corr}[t', t'^T]$. Where $d(t)_{i,j}$ is correlation between the i^{th} and j^{th} elements of t . The final LCGLoss loss is absolute the difference between batches of correlation grids constructed from the labels y and the network outputs p . That is, $\text{LCGLoss} := \|d(y) - d(p)\|_1$.

Unlike Guo et. al. (2022) learning from any compression of the label space into a smaller representation did not improve results in any of my experiments.

Figure 1: The label correlation grids computed over all training labels (left), the validation labels (center), and the absolute difference between them (right).



Results

Here we outline the results of the various design choices mentioned above. Here we present the results in Table 1 for the validation and test sets. We train ten models using the training set and compute the average and standard deviation of the log loss for the validation set results. I used the private leaderboard score for the test set score. The result on the private data is the result of training ten models on distinct folds of the complete dataset, containing both the training and validation set.

Table 1: **Log Loss Results for design choices**

model design choice	Validation	Test
base	0.18203 ± 0.0029	-
+hinge loss	0.18369 ± 0.0062	-
$+\log(\bar{y}^i / (1 - \bar{y}^i))$ output bias	0.18249 ± 0.0064	-
+BatchNorm	0.18249 ± 0.0035	-
+LayerNorm	0.15444 ± 0.0036	-
+LCGLoss	0.15080 ± 0.0016	-
+FixedSoup	0.14228 ± 0.0014	0.1536

2 Data

Each feature vector in the constructed dataset is derived from the information contained in the corresponding CSV file. We summarise a CSV file by standard peak features and n-difference features using a subset of m/z values. Moreover, we round each m/z to the nearest integer and subtract the minimum intensity for each intensity signal, for all samples.

peak features: For each rounded m/z value, we compute the average and maximum intensity before applying the log transform: $\log(x + 1)$. I then obtain a 'retention time' for each m/z by determining the time of occurrence for the maximum reached intensity. If there are no intensity values for a given m/z, we impute every feature dependent on this m/z with -1.

n-difference feature: Inspired by the first and second-order derivatives given by the SG algorithm (Savitzky & Golay, 1964) we compute a k-difference time series for each raw intensity signal. We compute $\text{diff}^{(1)}(x; k)$ whose i^{th} element is given by $x_i - x_{i-k}$ for the intensity signal $\{x_i\}_{i=1,2,\dots}$ and $k = 1, 2$ throughout. Moreover, we also compute $\text{diff}^{(2)}(x; k)$ whose i^{th} element is given by $\text{diff}^{(1)}(x; k)_i - \text{diff}^{(1)}(x; k)_i$ for only $k = 1$. Where $\text{diff}^{(1)}$ and $\text{diff}^{(2)}$ are analogous to the first and second derivatives of the intensity sequences modified by k . Next, we average groups of elements $\text{diff}^{(1)}$ and $\text{diff}^{(2)}$ based on the associated m/z value for intensity x_i . The resulting values are concatenated and transformed using the signed log transform: $\text{sign}(d) \log(|d| + 1)$ before inclusion in the final representation.

m/z subset selection: To reduce the final dimensionality of our vector representation, we reduce the number of m/z values used. Firstly, we restricted the m/z values to between 0 and 350 and excluded 4. Next, for each m/z value, we group the associated intensity values and compute the group's variance

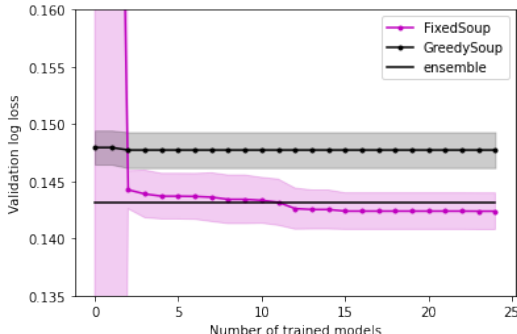
across the entire training dataset. Finally, we remove the ten m/z that result with the lowest computed variances.

3 Fixed Soup

Here we introduce the FixedSoup algorithm, a novel weight averaging framework that improves the final performance of individual neural networks over ensembles. FixedSoup is based on "GreedySoup" outlined by Wortsman et. al., (2022). Generally, these algorithms iteratively find an optimal set of weights that, when averaged, enable a model to perform well on validation data. The main difference is that FixedSoup uses a fixed total (i.e. five in all instances) of weights used for averaging, unlike its predecessor. I have provided python code in the appendix.

The algorithm is as follows. First, we initialize a mixture of model weights or "soup" with k fine-tuned models trained on the same training-validation split. Next, we continuously train models using the same data split. We continue until we encounter a new model with a better validation log-loss than one of the k used in the current soup. Then we construct a new soup using this new model in place of the worst performing model in the old one. We only keep this new soup if the validation log loss of the new soup is better than the previous. Finally, we iterate until convergence. See the Appendix for a python implementation. We test this algorithm against GreedySoup (Wortsman et. al., 2022) and an ensemble of 10 models on the validation set in Figure 2 below.

Figure 2: Comparison of the iterations of GreedySoup (Wortsman et. al., 2022) and FixedSoup.



The FixedSoup algorithm improves the performance and inference time of other types of neural networks including convolutional neural networks (CNNs) and transformers. Namely, Wortsman et. al. (2022) showed that GreedySoup gave the top performance to various convolutional and transformer networks on ImageNet image classification. Usually, the inference of k models using ensembling methods typically outperforms any single individual model but requires the weights of these k models to be stored in memory (simultaneously). However, it can be expensive to store numerous models or several large modern CNNs and transformers. Single models produced by GreedySoup (Wortsman et. al. (2022)) and FixedSoup use much less inference compute and yield similar/better results than a comparable ensemble, without any additional training or modification.

4 Discussion

For the remainder of this paper, I will discuss the potential existence and possible sources of label-noise in the given data and give possible solutions to address them for future research. Moreover, we will only discuss the label distribution component of the implementation and argue for its effectiveness in combatting **isobaric interferences**. There are several reasons to suggest that label noise exists in the data. Where here we roughly define label noise as random perturbations in the label space. I will outline these reasons and suggest ways to address them.

For one, the competition website states the data is: "from sequential experiments on the same instrument". Hence, compounds from earlier experiments have the potential to influence others. However, it is unclear how much cross-sample contamination has or has not occurred.

Secondly, the effectiveness of LDL suggests to me that the data might have the label uncertainty that the LCGLoss was designed to solve. In the study of emotion recognition, researchers found that LDL techniques are effective when distinguishing the "polysemy" or the multitude of possible human emotions resulting from a single facial expression (Zhao et. al., 2021). Subsequently, Guo et. al. (2022) noted that LDL is an effective tool when solving a more general notion of "uncertainty in label space" (Guo et. al., 2022). However, this is only an implicit argument for the existence of label noise.

Finally, I hypothesize the existence of isobaric interferences in the data. Indeed, isobaric interferences are often a problem in mass spectrometry as they often affect several steps in the mass spectrometric process (Wieler, 2014). Various endeavors, including the SAM suite mission and plume sampling on Europa, often require techniques to distinguish and reduce isobaric interferences (Franz et. al., 2017; Mahaffy et. al., 2021). Here, I roughly define isobaric interferences to refer to ions from a given m/z that result from more than one compound. However, even if I'm wrong in thinking that isobaric interferences exist in the data I hope to suggest using them as a possible focus for future research.

My suggestions, therefore, are intended to address label noise given the causes outlined above. Simple solutions could involve creating specialized data designed to teach models to better interact with the presence of isobaric interferences. Alternatively, another solution could involve additional features and resources that are directly or indirectly related to isobaric interferences. For example, in liquid chromatography mass spectrometry, Dührkop et. al. (2020) use a spectral library and structure library features to train models that generalize well. These additional features are not directly related to isobaric interferences however, these features perhaps make the task of distinguishing between ions easier and perhaps might be the ideal way forward.

References

- [1] Avice, G., Parai, R., Jacobson, S.A., Labidi, J., Trainer, M.G., Petkov, M.P. (2022). Noble Gases and Stable Isotopes Track the Origin and Early Evolution of the Venus Atmosphere. *Space Science Reviews*, 218.
- [2] Ba, J., Kiros, J.R., Hinton, G.E. (2016). Layer Normalization. *ArXiv*, abs/1607.06450.
- [3] Dührkop, K., Nothias, L., Fleischauer, M., Reher, R., Ludwig, M., Hoffmann, M.A., Petráš, D., Gerwick, W.H., Rousu, J., Dorrestein, P.C., Böcker, S. (2020). Systematic classification of unknown metabolites using high-resolution fragmentation mass spectra. *Nature biotechnology*.
- [4] Franz, H.B., Trainer, M.G., Malespin, C., Mahaffy, P.R., Atreya, S.K., Becker, R.H., Benna, M., Conrad, P.G., Eigenbrode, J.L., Freissinet, C., Manning, H.L., Prats, B., Raaen, E., Wong, M.H. (2017). Initial SAM calibration gas experiments on Mars: Quadrupole mass spectrometer results and implications. *Planetary and Space Science*, 138, 44-54.
- [5] Guo, Q., Zheng, Z., Jia, X., Xu, L. (2022). Label distribution learning via label correlation grid. *ArXiv*, abs/2210.08184.
- [6] Ioffe, S., Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv*, abs/1502.03167.
- [7] Kingma, D.P., Ba, J. (2015). Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980.
- [8] Mahaffy, P.R., Weng, M.M., Arevalo, R., Benna, M., Summons, R.E., Brinckerhoff, W., Cable, M.L., Craft, K.L., Eigenbrode, J., Farrell, B., Garvin, J.B., Grunsfeld, J.M., Templeton, A.S., Johnson, S., Hoehler, T.M., Murray, A., Sparks, W.B. (2021). Agnostic Biosignature Exploration at Europa through Plume Sampling.
- [9] Savitzky, A., & Golay, M.J. (1964). Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*, 36, 1627-1639.
- [10] Wortsman, M., Ilharco, G., Gadre, S.Y., Roelofs, R., Gontijo-Lopes, R., Morcos, A.S., Namkoong, H., Farhadi, A., Carmon, Y., Kornblith, S., & Schmidt, L. (2022). Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *ArXiv*, abs/2203.05482.

- [11] Wieler, R. (2014). 15.19 – Noble Gas Mass Spectrometry.
- [12] Zhao, Z., Liu, Q., & Zhou, F. (2021). Robust Lightweight Facial Expression Recognition Network with Label Distribution Training. AAAI.

Appendix

Pseudocode

```

import numpy as np

def train_soup(X_train, X_test, y_train, y_test,
               n_iter=40, n_soup=5):
    soup = []
    scores = []
    best_score = np.inf
    best_soup = []

    for i in range(n_soup):
        w, s = train_model(X_train, X_test, y_train, y_test)
        soup.append(w)
        scores.append(s)

    for j in range(n_iter):
        w, s = train_model(X_train, X_test, y_train, y_test)

        if s < max(scores):
            i = np.argmax(scores)
            del soup[i]
            del scores[i]
            soup.append(w)
            scores.append(s)

        soup_avg = np.mean(soup, axis=0)
        _, val = train_model(X_train, X_test, y_train, y_test,
                           initial_weights=soup_avg)

        if val < best_score:
            best_score = val
            best_soup = soup_avg

    return best_soup, best_score

```