X-Ray Photoelectron Spectroscopy Enhanced by Machine Learning

Alexander Gabourie (gabourie@stanford.edu), Connor McClellan (cjmcc@stanford.edu),
Sanchit Deshmukh (sanchitd@stanford.edu)

1. Introduction

X-Ray photoelectron spectroscopy (XPS) is a technique for identifying individual elements in a mixture/compound. Samples are irradiated by X-rays and the kinetic energy of ejected electrons is measured. Ejected electrons are captured by a spectrometer and measured intensities are plotted versus kinetic energy (**Fig. 1)**. Each element appears as a series of peaks distributed as Gaussians/Lorentzians. From these peaks, material properties can be estimated. Current software for XPS systems attempts, but often fails, to correctly classify the sample. We would like to enhance this classification using machine learning (ML) algorithms.

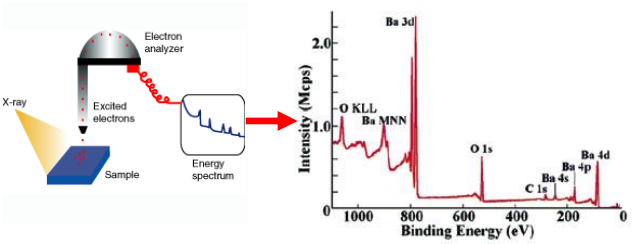

Fig. 1: (Left) A drawing of the XPS characterization technique. (Right) An example XPS measurement. Note that Ba has multiple peaks, each for different atomic orbitals [1].

The ultimate goal of this work is to create an algorithm that could correctly classify any compound given an XPS spectrum (**Fig. 1)**. This algorithm can be separated into two different tasks: First, Gaussians/Lorentzians are fit to the spectrum so to extract physically significant peaks. Second, a multi-class classification algorithm uses those peaks to identify the compound. This report addresses the second of the two tasks as it is better suited for a machine learning project.

For this task, binding energies from elements/compound's XPS spectrums are used as input to multiple different learning algorithms. These algorithms will then output a predicted element/compound to classify the input signal. The softmax regression, Support Vector Machine (SVM), and Naïve Bayes multi-class models will be trained to make accurate predictions. Initially, we develop algorithms to classify pure elements and follow that work with full compound classification.

2. Related Work

For better understanding of our problem and possible solutions, we looked for similar work on analyzing materials from spectral analysis data using ML techniques. We found that researchers tend to use SVM and Artificial Neural Network (ANN) algorithms for analyzing spectral data since both are robust classification techniques [2],[3]. However, a ANN would require a larger training set size than we could gather, leading us to using SVMs for our XPS classification.

We further looked at Naïve Bayes and softmax regression because of both techniques do not inherently require large training set sizes. Naïve Bayes has been previously used for classification of Near-Infrared Spectroscopy, which is a similar technique to XPS, suggesting the algorithm could be used for XPS data classification [4]. We also found that lasso logistic regression has been used for binary classification of spectral data [5]. However, since our problem requires multiclassification, we chose to use softmax regression as a multinomial generalization of logistic regression.

3. Data Collection and Refinement

The U.S. National Institute of Standards and Technology (NIST) has compiled an extensive XPS database consisting of all reported XPS measurements that have been tabulated in academic journals, books, and webpages [6]. This database is public, but only available in an online entry-searchable format not suitable for data processing. Our initial efforts focused on aggregating all XPS information from the database with a custom web-scraper.

Each of the 33,369 entries in the XPS database consists of a 31-row table, where each row presents itself as a potential feature. Since the intention of this project is to help XPS users identify elements/compounds from an XPS spectrum, and only binding energies, peak intensities, and Gaussian/Lorentzian widths of each peak can be extracted, only those three quantities from the database may be used in our learning algorithms. Unfortunately, peak intensities and widths are seldom provided ruling those quantities out as features. This leaves binding energy as the only quantity we can use in our learning algorithms.

As seen in **Fig. 1**, an XPS spectrum is composed of multiple peaks; however, each database entry contains only one peak. An element/compound's complete spectrum can still be constructed by combining multiple different entries from the database, although this reduces the effective dataset size. The list of peaks for each element/compound then acts as a feature vector for learning algorithms. Ultimately, we have a relatively size-restricted dataset, but an alternative is not available.

4. Methods

4.1 Algorithms

The following three learning algorithms were used to classify elements/compounds. Different assumptions on the input data are used but the underlying algorithms remain constant.

4.1.1 Softmax Regression

Since we are trying to solve a multinomial classification problem, a natural algorithm to choose would be the softmax regression. The scikit-learn [7] logistic regression model implements a softmax regression with the multinomial option

enabled. Softmax regression is a generalized multinomial version of logistic regression that can classify an arbitrary number of classes. We train the softmax regression algorithm by finding the w parameters that minimize the cost function:

$$J(w) = \mathrm{C}\left[\sum_{j=1}^{k} log(e^{-y_i(x_i^T w+c)} + 1)\right] + \min_{w,c} \frac{1}{2} w^T w$$

$$y^{(i)} \in [1, 2, \ldots, k]$$

$$x^{(i)} \in \Re^n$$

Where $y^{(i)}$ is the classification vector for the k number of elements, $x^{(i)}$ is the training parameter vector that constitutes the peak positions of each element/compound, w is the learned parameter, and C is the penalty on the loss. The scikit-learn softmax implementation automatically includes L2 regularization in the cost function with the $\min_{w,c} \frac{1}{2} w^T w$ term.

In our project, we used the solvers Stochastic Average Gradient (SAG) and Newton Conjugate Gradient (Newton-CG) for training. SAG is implemented similarly to Stochastic Gradient Descent, where w is updated by iterative steps scaled by a learning rate α/m, where m is the number of training examples, as the $J(w)$ reaches the global minimum [8].

$$w := w + \frac{\alpha}{\mathrm{m}} \sum_{i=1}^{n} a_i^k, \quad a_i^k = \begin{cases} \nabla_w J(w) \ \ if\ i = i_k \\ a_i^{k-1} \ \ \text{otherwise} \end{cases}$$

The SAG learner can find a global minimum quickly in large data sets, but it may fail with small data sets if the global minimum is passed. Newton-CG computes the gradient of the convex cost junction $J(w)$ to find a w that minimizes $J(w)$.

$$w := w - \frac{J(w; x^{(i)}, y^{(i)})}{\nabla_w J(w; x^{(i)}, y^{(i)})}$$

4.1.2 Support Vector Machine (SVM)

SVMs are a popular learning algorithms for classification. SVMs are designed for binary classification, but can be generalized to work with multinomial classification problems through one-vs-one and one-vs-rest schemes. From the scikit-learn package, we decided to implement the C-Support Vector Classification (C-SVC) using two different kernels. In C-SVC, we want to solve the following optimization problem as laid out by [9]:

$$\min_{\theta,b,\xi} \left\{ \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \right\}$$

where w is our parameter, C is a penalty on the loss, b is the intercept, and $\xi_i = \max(0, 1 - y_i f(\phi(x_i)))$ is the hinge loss with $y_i f(\phi(x_i))$ as the margin and $f(\phi(x_i)) = (\theta\phi(x_i) + b)$. Here, $\phi(x_i)$ maps x_i into a higher dimension and is relevant to our kernel choice. Since $\phi(x_i)$ can have a high dimensionality, the problem is usually rewritten in terms of a kernel function $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$, but the solution is the same. The kernels we use for our problems are a linear kernel, $K(x, x') = \langle x, x' \rangle$ (Linear SVC, **Fig. 5, 7**), and a Gaussian kernel $K(x, x') = \exp(-|x - x'|^2/n)$ (SVC, **Fig. 5, 7**), where n is number of features.

For multinomial classification, the one-vs-rest (OVR) classification scheme trains a classifier per class, with samples of that class as positive samples, and all others are negatives. The one-vs-one (OVO) classification scheme trains $k(k-1)/2$ classifiers for a k-class problem. Pairs of classes are compared and a voting scheme is used to select the best class.

4.1.3 Naïve Bayes

We attempted a Naïve Bayes analysis of the binding energy data set to determine a classification for identifying the element. The algorithm utilized for classifying elements from spectra is a modification of the classic Naïve Bayes algorithm typically used for applications like spam classification. This modification enables multi-class classification, with multivariate Bernoulli feature vectors. This algorithm works well with discrete data, with the features being Boolean valued quantities. The Bernoulli quantities for each feature are as described below.

$$p(x^i = 1 | y = C_k) = p_{ki} = P(i|C_k)$$

The algorithm allows for a multi-class classification. We let the classifier learn prior probabilities from the training data for each element. This goes into the classifier determining θ vectors from the Bayesian probabilities for each class, from each Boolean valued feature.

$$p(x|C_k) = \prod_{i=1}^{n_{features}} p_{ki}^{x^i} (1 - p_{ki})^{1-x^i}$$

The decision rule for the Naïve Bayes explicitly penalizes for the non-occurrence of a certain input characteristic of that class:

$$P(x_i|C_k) = P(i|C_k)x_i + (1 - P(i|C_k))(1 - x_i)$$

In this case the feature vectors used in the training dataset consist of full spectra of individual elements as seen in the NIST database.

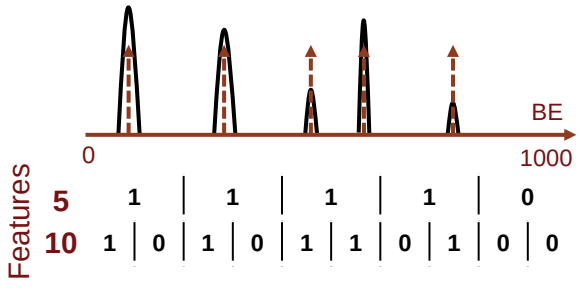

Fig. 2: Feature vector generation for Naïve Bayes from XPS peak data

4.2 Implementations

4.2.1 Element Classification

Softmax & SVM

Our first implementation of multinomial classification of XPS data involved using softmax and SVM algorithms to predict elements. We assume that the XPS user provides all peaks from 0 eV to the binding energy of their largest peak. For training, we used the 76 elements from the NIST database, with training feature vectors being the n different peak positions for each element (ranging from 1 to 14 peaks depending on the element) and training examples as the m different references for each peak, making the training parameters x a m×n sized matrix. To create test examples, we generated 2000 samples from randomly chosen element spectra and added random noise ranging from 0 eV to 5 eV which is the typical variation an XPS user will see when preforming XPS on a new sample.

Naïve Bayes

To generate feature vectors for the Naïve Bayes algorithm as described previously, the peak data is thresholded at some values of minimum and maximum binding energy, typical of most common XPS detector limits (in this case the lower bound was at 0 eV and the upper bound at 1000 eV). The dimensionality of the feature vector corresponded to the number of divisions in the thresholded energy range for the full spectrum. For instance (**Fig. 2**), if the individual energy range was 10 eV, the entire spectrum of binding energies consisted of 1000 eV / 10 ev = 100 features. Presence of a peak within individual energy ranges was considered a Bernoulli random variable, with the probability of having a 1 corresponding to having at least one peak in that energy range. The entire spectrum of an individual element is mapped to a single vector. Downsides of this feature vector construction include loss of information about relative peak heights. This can eventually be taken into account by adding sensitivity factor information for each element.

$$E_i = E_{min} + (E_{max} - E_{min}) \cdot \frac{i}{n_{features}}$$

$$x^i = \begin{cases} 0; & 0\ peaks\ in\ range\ [E_i, E_{i+1}] \\ 1; & 1 \le peaks\ in\ range\ [E_i,\ E_{i+1}] \end{cases}$$

4.2.2 Compound Classification

Softmax & SVM

The next logical step after element prediction was towards full compound prediction. From our dataset, we found that 6,568 different compounds were represented. Since the number of classes we have to predict is large (k = 6,568), we made some reasonable assumptions and a simplification to reduce the complexity of the problem. Our assumptions are that the XPS user is only measuring a continuous subsection of the spectrum (Ex. Between blue lines in **Fig. 3**) and that they are reporting all peak positions correctly.

The simplification can be explained through an example. Let's say we receive an XPS measurement as a vector of binding energies from a spectrum. Here we get two peaks in the energy range between the blue lines in **Fig. 3**. From our set of possible elements (A, B, C), it is clear that we have element C as it is the only element with the two peaks in the energy range. This does not need a learning algorithm to determine. If two peaks are measured in the energy range between the red lines, then it is not immediately clear which element we measured. It is in these circumstances we employ a learning algorithm (SVM, softmax). In general, when this approach to is used, $k_{actual} \ll k_{total} = 6{,}568$ and our classification algorithm is much faster and more accurate; however, this approach does require retraining each test example.

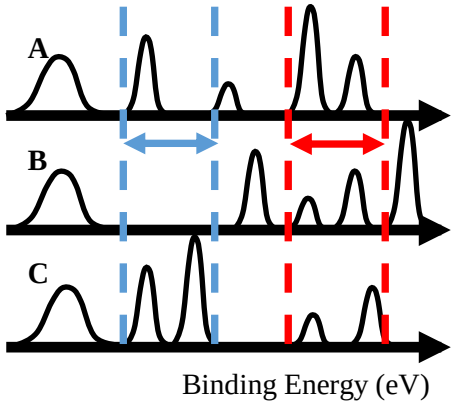

Fig. 3: XPS signals for hypothetical elements A, B, & C. In between blue lines each element has different number of peaks and between the red, they have the same number.

5. Results & Discussion

5.1 Element Classification

Softmax & SVM

Our implementation of Softmax and SVM algorithms resulted in an accurate prediction of elements from XPS data. When training based only on a single binding energy, we only consider lowest energy peak of each element. However, as there is only one training data per element and some elements have peaks close to each other, the test and training error are high (30-60%). To improve our model accuracy, we needed to include a larger number of peaks per element. However, we encountered the issue of a sparse and non-symmetric dataset, as the max number of peaks per element ranges from 1 to 14 peaks, complicating the construction of the x matrix. We overcame this issue by filling in non-existent peaks for elements with the average energy peak of the element using all the training examples. We further tuned the learning by fixing the number of training examples for each element peak to the average number of training examples for the whole data set. **Fig. 4** shows the training and test error after including these new conditions using the Newton-CG solver. We saw a large decrease in both training and test error, such that after including 4 peaks for training, the softmax regression learner was 100% accurate for a range of noise during of testing. **Fig. 4** shows that when incorporating 5 or more peaks our prediction model for elemental XPS data is highly accurate even with large noise added to the test data.

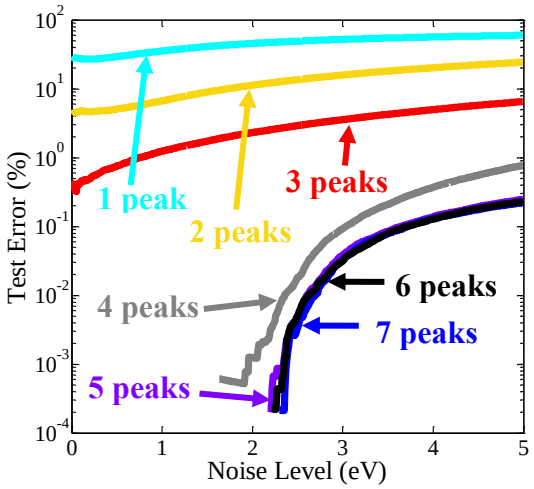

Fig. 4: Test Error for Softmax Newton-CG solver with varying training data size.

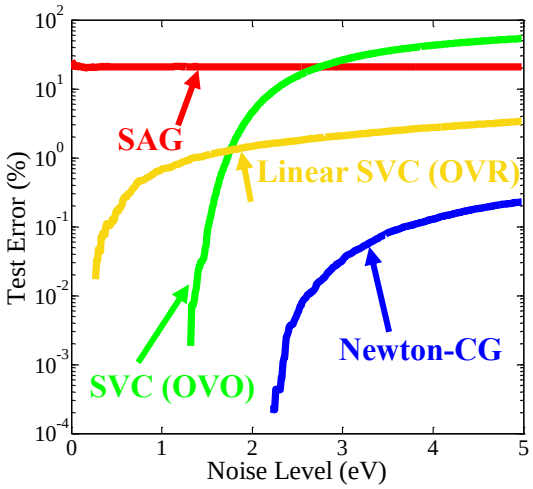

Fig. 5: Test Error for different Softmax and SVM solvers when considering 7 XPS peaks.

We also looked at the SAG softmax solver and SVM models for training and predicting XPS elements. **Fig. 5** shows that the Newton-CG softmax learner achieved the lowest error while the SAG softmax learner displayed a large test error of ~20% independent of noise. The large discrepancy between Newton-CG and SAG is due to the SAG solver failing to converge towards the global minimum. This difference between the two solvers highlights the ineffectiveness of the SAG solver for learning on small a training example size as the learning rate is inversely proportional to number of training examples. The large learning rate for the SAG then results in an "over-shoot" where the SAG solver misses the minimum training error. Only by lowering the SAG tolerance to 10^{-8} was the learning rate decreased enough to achieve a reasonable error of 5%, but the computation time was much longer than the Newton-CG solver, mitigating to main benefit of the "fast" SAG solver.

Using the SVC and Linear SVC solvers, the error rate was lower than the SAG solver but higher than Newton-CG. We observed lower error with the SVC solver than Linear SVC at low noise most likely due to the more robust Gaussian kernel used in SVC over the linear kernel in Linear SVC. However, at higher noise level, the linear SVC shows lower error than SVC. This result indicates that our Gaussian kernel is not properly tuned by hyperparameters. Since Newton-CG already achieved satisfactory prediction accuracy, it was not necessary to optimize any hyperparameters.

Naïve Bayes

With the feature vectors set up as described in **4.2.1**, we train the algorithm with varying feature vector sizes. The training set size is the total number of vectors corresponding to full peak spectrum for each element from each reference. The feature size is varied from 10 to 10000, corresponding to a change in the feature energy range from 100 eV to 0.1 eV. We see that Naïve Bayes always performs better than randomly guessing the most probable element (viz. has less than 88% error). The implementation of the algorithm with feature size of 200 (corresponding to feature energy range of 50 eV) shows lowest training error of ~ 24% (**Fig. 6 (a)**). This indicates that to distinguish one element from the other, most important XPS signatures are contained in distinct 50 eV bins, across the set of all elements in the periodic table. We performed K-fold cross validation with the trained classifier. The training data was shuffled and divided into subsets for cross-validation. The post cross-validation test error remained below 30% for feature vector sizes of 200 (**Fig. 6 (b)**). This indicates low variance for the Naïve Bayes training algorithm for the selected feature size. This also indicates that most of the information from individual element spectra is contained in the minimal subset of spectra containing all element data.

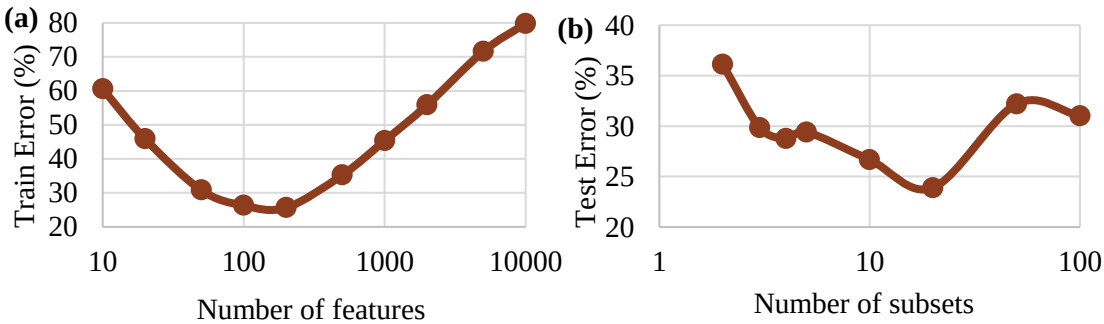

Fig. 6 (a): Train error for Naïve Bayes with changing feature vector sizes; **(b)**: Test error with K-Fold Cross Validation for the test set, K varying from 2 to 1000.

5.2 Compound Classification

Softmax & SVM

As was the case for the element prediction, training data for compounds was not readily available so test cases were created by adding randomly adding noise to spectra of randomly chosen compounds. The resulting binding energy vector was then passed to the prediction algorithms. Our primary and only metric of performance is the overall accuracy of the prediction, ignoring compound specific metrics. Other visuals or quantities, such a confusion matrices, precision and recall metrics, and AUC curves, are not included because they would be ineffective due to the large number of classes we have to represent.

To test the default effectiveness of our learning algorithms, we ran ≈60,000 test cases for each of the softmax (solvers: Newton-CG, SAG) and SVM (linear SVC (OVR), SVC (OVO)) variations. The results can be seen in **Fig. 7**. The test cases pass 1-28 peaks (13 shown) to the model and we see that the test error decreases with increasing number of peaks. This happens for two reasons: First, the use of more peaks shrinks k in the k-class classification problem because more compounds are removed from the possible outcome list due to the simplification described in **4.2.2**. Second, having more peaks means the feature list is larger, improving the uniqueness of the XPS signature. Overall, the best performing learning algorithms are Newton-CG and SVC (OVO) which have very similar test error curves. The SAG method performs poorly here because our dataset is too small [8]. Described in [10], the linear kernel is a special case of the Gaussian kernel meaning linear SVC will not be more accurate than a properly tuned SVC (i.e. Gaussian kernels are generally more accurate). While the linear kernel SVM was fast, it did not predict compounds as accurately as other methods.

In both the softmax and SVM algorithms, there is an adjustable hyperparameter C (see section **4.1**) which penalizes the loss. We decided to forgo optimizing this hyperparameter for compound classification as we retrain the model each test case and did not want to incur heavy computation penalties.

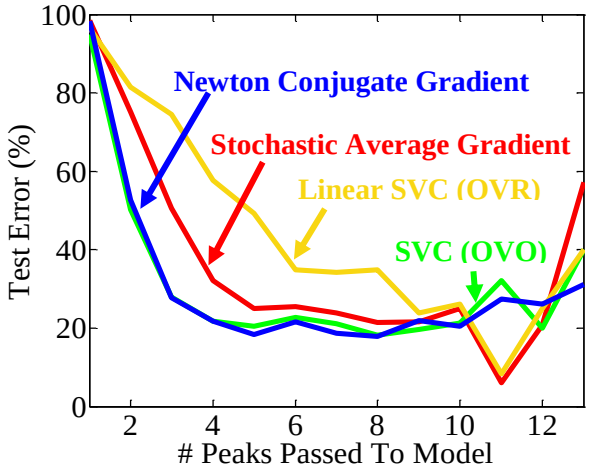

Fig. 7: Total test error of the compound prediction algorithm for multiple different learning algorithms.

Lastly, we must address the large test error for a single peak. Upon inspection of training sets for a single peak, there are generally multiple compounds with peaks either equal or very close to each other making it virtually impossible to train an effective model. The general advice we would give to a user of this XPS classification algorithm would be to provide the algorithm with at least four peaks.

6. Conclusion & Future Work

The goal of this project was to classify compounds from an XPS spectrum given binding energies corresponding to peaks. We addressed the problem using softmax, SVM, and Naïve Bayes machine learning algorithms. Inputs were binding energies and outputs were the classified element/compound. For the elemental classification, we found Newton-CG to correctly predict all elements even with up to 2 eV of noise added to each peak. SVMs, both with linear and Gaussian kernels, could also classify elements without error, but were more sensitive to noise.

For compound classification, Newton-CG and the Gaussian kernel SVM gave the highest accuracy predictions. Accuracy improved as more XPS peaks were input to the algorithm with an error down to ≈20% if more than three peaks are used. A future improvement would be to optimize the hyperparameter C for each method, resulting in excess computation but higher accuracy

For Naïve Bayes, the current approach lets us classify single elements from their spectra with test error as low as ≈24%. This can be expanded upon by doing a complete multi-label classification from a combined realistic XPS spectrum. While Naïve Bayes can be built upon (with OVO or OVR like approaches), future work can consider using decision trees for spectra classification with multi-labels.

This project was ultimately data limited. While the NIST database provided binding energies of XPS peaks, additional features such as peak intensities and Gaussian/Lorentzian widths must be included for accurate compound prediction. Peak intensities and widths can be gathered from current existing XPS spectra for all compounds, but not without significant time and effort. Additionally, there were many elements/compounds with little published information making them difficult to classify. Extra data collection for these could strengthen the dataset.

Overall, our machine learning algorithms successfully predicted elements and performed reasonably well for compound prediction despite any dataset limitations.

References

[1] T. F. S. Inc, "Thermo scientific XPS: What is XPS," 2013. [Online]. Available: http://xpssimplified.com/whatisxps.php. Accessed: Dec. 17, 2016.

[2] T. Zou, Y. Dou, H. Mi, J. Zou, and Y. Ren, "Support vector regression for determination of component of compound oxytetracycline powder on near-infrared spectroscopy," *Analytical Biochemistry*, vol. 355, no. 1, pp. 1–7, Aug. 2006.

[3] M. Madden and T. Howley, "A machine learning appli cation for classification of chemical spectra," in *Applications and Innovations in Intelligent Systems XVI*. Springer London, 2009, pp. 77–90.

[4] M. Blanco and J. Pagès, "Classification and quantitation of finishing oils by near infrared spectroscopy," *Analytica Chimica Acta*, vol. 463, no. 2, pp. 295–303, Jul. 2002.

[5] D. Yu, S. J. Lee, W. J. Lee, S. C. Kim, J. Lim, and S. W. Kwon, "Classification of spectral data using fused lasso logistic regression," *Chemometrics and Intelligent Laboratory Systems*, vol. 142, pp. 70–77, Mar. 2015.

[6] "NIST x-ray Photoelectron spectroscopy (XPS) database, version 3.5," 2012. [Online]. Available: https://srdata.nist.gov/xps/. Accessed: Oct. 1, 2016.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, "Scikit-learn: Machine learning in Python", J. Mach. Learn. Res., vol. 12, pp. 2825-2830, 2011.

[8] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing fi nite sums with the stochastic average gradient," Mathematical Programming, Jun. 2016.

[9] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[10] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with Gaussian kernel," Neural Computation, vol. 15, no. 7, pp. 1667–1689, Jul. 2003.