Software Review

Journal of Educational and Behavioral Statistics
2019, Vol. 44, No. 3, pp. 348–361
DOI: 10.3102/1076998619832248
Article reuse guidelines: sagepub.com/journals-permissions

Machine Learning Made Easy: A Review of *Scikit-learn* Package in Python Programming Language

Jiangang Hao
Educational Testing Service

Tin Kam Ho
IBM Watson

Machine learning is a popular topic in data analysis and modeling. Many different machine learning algorithms have been developed and implemented in a variety of programming languages over the past 20 years. In this article, we first provide an overview of machine learning and clarify its difference from statistical inference. Then, we review Scikit-learn*, a machine learning package in the Python programming language that is widely used in data science. The* Scikit-learn *package includes implementations of a comprehensive list of machine learning methods under unified data and modeling procedure conventions, making it a convenient toolkit for educational and behavior statisticians.*

Keywords: *machine learning; Python;* Scikit-learn

1. Overview of Machine Learning

Machine learning refers to a set of methodologies that allow computers to "learn" the relationship among numerical representations of the data and certain target values. Machine learning methods, such as support vector machine (SVM), maximum entropy, random forest, and artificial neural network, have been widely used in different applications. Studies of these methods cover a wide range of topics that far exceed the scope of the this article. Many excellent texts provide comprehensive coverage of machine learning (e.g., Bishop, 2006; Hastie, Tibshirani, & Friedman, 2009; Witten, Frank, Hall, & Pal, 2016). This article aims to introduce and review the *Scikit-learn* (Pedregosa et al., 2011), a Python package for machine learning. We start with a concise overview of machine learning to highlight certain strengths and limitations of the methodologies, by which we hope to give readers a better understanding about whether the methodologies are suitable for their research needs. Then, we summarize the strengths

and limitations of the *Scikit-learn* package and use specific examples to show how to use the machine learning methods included in the package.

The scope of technologies covered in the concept "machine learning" has seen much evolution over the past few decades. Here, we adopt the notion used over the last few years in the scientific literature and sometimes in popular press. Loosely speaking, machine learning includes algorithms that apply a mapping from a numerical representation of observations (also known as feature representation, typically as a fixed length feature vector) to some target values (as in regression) or categories (as in classification). The inference of such mapping is most commonly accomplished by a process known as supervised learning, and occasionally by semi-supervised learning or, as recently proposed, weakly supervised learning. In a broader scope, machine learning also includes methods for discovering concentrations, associations, or correlations in data, which is usually referred to as unsupervised learning. In addition, there are methods for interactive or online learning, and in a more structured form, goal-oriented exploration in a parameter or state space to optimize a reward function, which is known as reinforcement learning (Sutton & Barto, 1998). Neural networks, as a special type of models involving a stack of nonlinear functions interacting over layers, have been architected for both supervised and unsupervised learning applications. In particular, neural network techniques designed with an emphasis on the intermediate representation generated over multiple architectural layers are known as deep learning (LeCun, Bengio, & Hinton, 2015), which plays an important role in the recent progress in artificial intelligence.

The pursuit of modeling the relationship among data is not unique to machine learning. It is also conducted under other names, and among those with the longest legacy, the discipline of statistical inference. There are, however, some noticeable differences between the machine learning approaches and statistical inference. The first is that machine learning is more prediction-driven, while statistical inference emphasizes model parameter estimation as well as the prediction. This difference largely results from the fact that many popular techniques for machine learning were invented in the context of specific applications, the best known of which is image or speech recognition, especially the recognition of handwritten digits. Typical input in these domains is low-level perceptual signal represented as high-dimensional vectors, such as a set of grayscale values in a raster scan of an image. Components of these vectors are often correlated to some extent, like those corresponding to neighboring pixels in an image. A learning algorithm is usually expected to accomplish an operational goal, such as a prediction for a given observation, whereas the explanation of how one arrives at that prediction is not considered important. On the other hand, in statistical inference, input variables may be in a mixture of numerical, ordinal, or categorical attributes. Statistical models are often built for no more than several dozens of variables. Understanding, explaining, and justifying the relationship via the estimated model parameters are as important as making a good prediction.

The second difference is the emphasis on data modeling versus computation algorithms. Statistical inference emphasizes developing a model that can characterize the probabilistic distribution of the feature or target values and their interactions. The process of model development allows the opportunity for trying various representations of known or assumed data effects. For example, many regression models are written as a function of systematic contributions from the explanatory variables plus a noise term to account for the remaining, unknown effect. The inference methodology relies heavily on several well-studied distributions, such that inference can be supported by rigorous statistical theory. In contrast, there is less emphasis on data models in machine learning but more on computational models that can carry out the inference process efficiently, and the sample size requirement is less explicit. Recent enthusiasm for deep learning, in which features for describing systematic effects are also supposed to be learned in the training process, pushes this to the extreme. Inference in machine learning relies heavily on a few common optimization procedures (e.g., quadratic programming, stochastic gradient descent), especially those effective for a large sample regime. Some models allow for an opportunity for representing procedural knowledge (e.g., in syntactic pattern recognition; Fu, 1981, in which a recognition algorithm is given by a generative grammar and a suitable parser). As the effectiveness of a model often depends on many combinatorial factors, few conclusions are supported by closed-form analytical results. As such, one often resorts to empirical comparisons of the predictions to claim merits of a newly proposed machine learning technique.

The third difference is from the traditions of studying data of a different nature. Statistical inference with a small number of variables often involves many repeated observations, which makes discussion of frequencies and estimation of probability densities possible. On the contrary, in machine learning, the sparsity of data in a high-dimensional vector space results in difficulties in characterizing density, as there are hardly any repeated observations. What matters more is often the geometry of the region in the space that contains the data, for example, the "shape" (or support) of a class. Many statistical analyses are conducted on observations in planned experiments, in which the difference between independent or controlled variables from dependent variables is known before the analysis. In-depth knowledge about the nature of the data also allows for careful treatment of each type of variables, and there are many discussions about differences between measurements in nominal, ordinal, interval, or ratio scales, or in even finer typologies (Stevens, 1946). On the other hand, more typical in machine learning are data mining studies on passive, uncontrolled observations of the joint occurrences of a set of variables that are considered as input or output depending on the needs of the application. For example, in a classification problem, the notion of classes can be fluid, and classes can be redefined until a clean separation is obtained. This sometimes leads to hidden effects that are not accounted for (Blyth, 1972; Chen, Bengtsson, & Ho, 2009).

There is also less attention on the nature of the measurement scale, as any way to convert a measurement to a fixed length feature vector seems to be worth trying.

Despite the different emphases between machine learning and statistical inference, the distinctions are not always clear. Increasingly, machine learning makes use of many techniques from statistical modeling. Examples include many probabilistic models such as Naive Bayes, logistic regression, Gaussian processes, Gaussian mixtures, Bayesian networks, and many sampling techniques for estimation that originated in statistical analysis. Generative models have seen increasing adoption even in neural network settings, which gets closer to the traditional emphasis in statistics on data modeling. An exciting recent development is the generative adversarial nets (Goodfellow et al., 2014), in which a generative model of fake samples is used to drive the optimization of a discriminator. Regularization techniques are brought into many optimization processes to control overfitting. Many of the recent deep learning approaches involve implicit state space models, not unlike the explicit state space models that have long been used in statistical signal processing and control theory. For example, the embeddings (LeCun et al., 2015) obtained in deep learning approaches can be considered as state vectors and are used in conventional statistical models like those for regression and classification.

There is, however, more from statistics that could help enhance methods in machine learning. These include methods for introspection into the models and results: measuring goodness of fit, calculating error bars, performing model diagnostics such as checking against model assumptions such as stationarity and heteroscedasticity; more careful treatment of data of different types (e.g., categorical, ordinal, longitudinal, time series), or methods for experimental design, causal inference, extreme value inference, or process modeling. Some of these may help address issues in machine learning that involve stability and robustness concerns, insufficient systematic experimentation, and the generalization of results to unseen cases.

A healthy development of machine learning in the last couple of decades is that more computing packages that implement machine learning methods have become publicly available. To a large extent, these widely accessible implementations improved understanding of the methods' utility and effectiveness in different contexts and helped expose the limitations through observations by different teams in various research or application domains. Popular packages with such implementations include those with general coverage like *Weka* (https://www.cs.waikato.ac.nz/ml/weka), `R` (https://www.r-project.org), *Scikit-learn* (http://scikit-learn.org/stable), neural-network centric libraries like *Theano* (http://deeplearning.net/software/theano), *Torch* (http://torch.ch), *TensorFlow* (https://www.tensorflow.org), and *Keras* (https://keras.io), plus many application-focused packages like those specialized in computer vision (*OpenCV*, https://opencv.org), Optical Character Recognition (*Tesseract*, https://github.com/tesseract-ocr/tesseract), natural language understanding

(*NLTK*, https://www.nltk.org; the *Stanford NLP* library, https://nlp.stanford.edu/software; and *spaCy*, http://spacy.io), and numerous method-specific packages published on GitHub (https://github.com). Easy access to these packages is increasingly available via cloud computing platforms offered by major companies. Although many options exist, as we will show, the *Scikit-learn* package in Python is highly recommended for some basic machine learning applications.

2. The *Scikit-learn* Package in Python

2.1. Python in a Nutshell

The Python programming language is gaining tremendous popularity among data scientists and software developers (Robinson, 2017). Different from the R programming language that is mainly intended for statistical data analysis, Python shows up in a much wider range of applications such as Internet and website development, database access, desktop GUIs, scientific computation, and software and game development. There are two main Python version series, the 2.x and 3.x versions, and they are not completely compatible even though they are similar in most parts. The 2.x version is a legacy version, whose support and maintenance are scheduled to end around 2020. The 3.x version is a redesign based on the 2.x version and is considered to be the future of Python. Given the fact that there are many existing packages being written using the 2.x version, it is up to the users to decide which Python versions should be used for their particular application. In this review, we will use examples from the 3.x version.

Python is not a compiled language, meaning that it does not precompile the code into binary. Instead, a software environment, Python interpreter, translates the script into binary during the execution of the code in real time. With its distribution, Python comes with some basic functionality but relies on external packages to perform almost all numerical computations. After the natural selection process over the past 10 years, a small set of packages that provide some fundamental computing capabilities have received wide acceptance in the Python community. In Table 1, we listed some of these core packages that are doing the heavy lifting and are maintained by the Python community. In addition to these backbone packages, the front end of the Python interpreter is evolving. At this moment, probably the most user-friendly interface for Python is the Jupyter Notebook, which provides an interactive front end for the Python interpreter and is well suited for most data analytic work. In this review, we show the code snippets in Jupyter Notebook (see Appendix in the online version of the journal).

The easiest way to get Python, the core packages, and Jupyter Notebook is to install them through the anaconda suite (https://www.anaconda.com/download), which garners a selected set of Python packages and provides installers for Windows, MacOS, and Linux. After downloading the anaconda installer and

TABLE 1.
Some Core Packages for Numerical Computation With Python

Package Name	Descriptions	Website
NumPy	Fundamental package for numerical computation	http://www.numpy.org/
Scipy	Enhanced package for scientific computing based on *NumPy*	https://www.scipy.org/
Pandas	Enhanced package for data structure and manipulation	http://pandas.pydata.org/
Matplotlib	Package provides basic plotting functionalities	https://matplotlib.org/
Scikit-learn	Package for machine learning, often shortened as sklearn	http://scikit-learn.org/

installing the Python suites, one should follow the instructions on the anaconda website to launch the interactive front end, Jupyter Notebook. Then, the user is ready to check out the powerful functionalities of Python.

2.2. Overall Evaluation of Scikit-learn

Scikit-learn is the most comprehensive and open-sourced machine learning package in Python. As machine learning is often a component of a more general application (such as a Web service), it is desirable to have it furnished using the same programming language as the other parts of the application for seamless integration. Benefiting from the wide range of applicability of Python, *Scikit-learn* becomes an increasingly popular package for machine learning–related applications.

In addition to the strong backup from the Python ecosystem, *Scikit-learn* itself has many features that make it stand out among machine learning software. The first is its comprehensive coverage of machine learning methods. A community review procedure is in place to identify and decide on which machine learning methods should be included in the *Scikit-learn* package. Such a mechanism ensures a balance between the extensive coverage and selectivity of the machine learning methods contained in the package. The second is that the algorithm implementation of the machine learning methods in *Scikit-learn* is optimized for computation efficiency. Despite that Python is an interpretative programming language, most of the machine learning methods in *Scikit-learn* were based on the compiled binary libraries that were originally programmed in Fortran, C, or C++. These binary-based implementations significantly improve the efficiency of the computation. The third is that *Scikit-learn* has strong community support for documentation, bug tracking, and quality assurance. The *Scikit-learn* community maintains common documentation, a unified bug tracking/fixing process on GitHub, and a rigorous quality assurance procedure based on

a community-wide agreement. Last but not least, *Scikit-learn* imposes a unified input/output data convention and has a fixed model fitting procedure (as we will show in the next section), making the switching from one method to another almost effortless.

In summary, *Scikit-learn* includes a collection of efficiently implemented machine learning methods and is well-documented and maintained by the community. However, readers should also be aware that it is possible that *Scikit-learn* may not include some methods that are used in specialized applications. As such, we recommend readers to consider *Scikit-learn* as long as the needed methods are available but look for other software for those methods not included. In the next section, we will show some specific examples of how to use the machine learning methods in *Scikit-learn*.

2.3. The Scikit-learn *Package*

The *Scikit-learn* package covers four main topics related to machine learning. They are data transformation, supervised learning, unsupervised learning, and model evaluation and selection. The details in each topic can be found in the user guide on the *Scikit-learn* website (http://scikit-learn.org/stable/user_guide.html). In this article, we skip the unsupervised learning part, as the focus there is more on the proper interpretation of the results rather than the methods themselves. The data transformation, supervised learning, and model evaluation should serve our goals to give readers a good sense about the *Scikit-learn* package.

2.3.1. Data transformation. Data transformation is a crucial step in all data analysis. There are common transformations (e.g., whitening transformation, differencing, and log–odds ratios) frequently used for various input variables. *Scikit-learn* provides several convenient functions to perform these methods for data transformation and preprocessing. The core data structure used in almost all *Scikit-learn* functions is the *NumPy* array. The independent variables, also known as features in the machine learning community, are represented by an $N_o \times N_f$ array with N_o as the number of observations and N_f as the number of features. The dependent variables, also referred to as targets or labels, are represented by an $N_o \times N_l$ array with N_l representing the number of labels. In most of the applications for supervised learning, we are dealing with a single label with multiple values, so N_l is one. By convention, the uppercase $\mathbf{X}$ is used to denote the feature array and the lowercase $\mathbf{y}$ is used to denote the label array.

To facilitate our discussion, we will use the iris data set as an example throughout the rest of this article. Each observation of the iris data set comes with four feature variables and a label variable. The four feature variables are the sepal_length, sepal_width, petal_length, and petal_width, which denote the width and length of septal and petal in centimeter. The label variable indicates the species of the iris. There are altogether 150 observations contained in the data

```
In [1]: # --- Data Transformation ---
        from sklearn.datasets import load_iris
        from sklearn import preprocessing
        import warnings; warnings.simplefilter('ignore')

In [2]: data = load_iris()
        X = data.data
        y = data.target

In [3]: X.shape
Out[3]: (150, 4)

In [4]: y.shape
Out[4]: (150,)

In [5]: X = preprocessing.scale(X)
```

FIGURE 1. *Codes snippet corresponds to data loading and transformation.*

set. This data set has been widely used to show how to establish a mapping between the feature variables and the labels in many textbooks and online tutorials on machine learning. This data set is also contained in the *Scikit-learn* package. In the following, we show how to do some basic data preprocessing using *Scikit-learn* on the iris data set.

After starting a Jupyter Notebook, one needs to load the packages that include the functions to be used in the current session. In our example, we first load the iris data set and then perform some preprocessing on the data. As shown in Figure 1, the first input cell (In [1]), we first import the function that loads the iris data set and then import the module for preprocessing. Note that *Scikit-learn* is invoked under the name *sklearn* in the Python interpretative environment. In the second input cell (In [2]), we load the iris data set and create variable **X** to store the feature variables and **y** to store the label variable. In the third and fourth input cells (In [3], In [4]), we check the dimensions of the feature array **X** and the label array **y**. Their dimensions are displayed in the corresponding output cells (Out [3], Out [4]). Then, in the fifth input cell (In [5]), we apply a function, "scale," in the preprocessing module to the feature array **X**. This function standardizes each feature variable by subtracting the mean value of the variable and then scales it by dividing nonconstant features by their standard deviation. It is a recommended preprocessing procedure for most machine learning methods.

The above example is intended to give readers a sense of how to use the functions in the *Scikit-learn* package. In addition to the simple scale function, the preprocessing module includes many other functions for data transformation, such as nonlinear transformation, normalization, and binarization. Readers can refer to the user guide (http://scikit-learn.org/) for more details. In the next subsection, we will demonstrate how to implement multiple classifiers to show how easy it is to fit and compare multiple machine learning methods in *Scikit-learn.*

2.3.2. Supervised learning. Supervised learning refers to a subset of machine learning algorithms that establish a mapping between the feature variables and

their corresponding target variables. The precondition to using supervised learning methods is that both the features and their corresponding labels are known. Supervised learning can be cast into two categories based on the nature of the labels, regression for continuous labels, and classification for discrete labels. In this review, we will focus on a classification case, and the generalization to regression is formally trivial by just replacing the calls of the classification functions with the calls of regression functions. To make the introduction manageable, we focus on four popular classifiers, namely, the SVM, maximum entropy (also called multinomial logistic regression), artificial neural network, and random forest. The details of each of these methods are beyond the scope of this article, and we refer the readers to the corresponding literature or books on supervised learning (Bishop, 2006; Hastie et al., 2009).

A natural question many data analysis practitioners may ask is which of these methods one should use in practice. This is a simple question, but unfortunately, there is not a simple answer. Caruana and Niculescu-Mizil (2006) carried out an extensive empirical study to compare a number of supervised learning methods based on empirical data sets, and their conclusion was that the performance of the methods is essentially dependent on the specific data set, although the SVM and random forest are the top performers for a number of classification tasks. An area of research on what properties of a data set affect the performance of specific classification methods is exemplified by Bernado-Mansilla and Ho (2004) and many subsequent works.

The workflow implemented in *Scikit-learn* for using a classifier includes three steps. First, create the model by specifying the hyperparameters of the model. Second, fit the model with the training data and learn the parameters. Finally, apply the fitted model to the test data to get the predicted labels. The pseudocode for these steps is as follows.

```
model = classifier(hyper-parameters = something)

model.fit(X_train, y_train)
y_test = model.predict(X_test)
```

In the pseudocode, classifier() is an instance of one of the supervised learning methods. In the following snippet (Figure 2), the sixth input cell (In [6]) contains the code to load the aforementioned machine learning methods. Then, we create the corresponding machine learning models in the seventh input cell (In [7]) by using the default hyperparameters of each method. To check the details of the default hyperparameters, we use the input cells (In [8]–[11]), and the corresponding output cells show the hyperparameters in the models.

We are not going to delve into the details of these hyperparameters in this article, and we will refer the readers to the *Scikit-learn* documentation for the details. To apply the machine learning models to the data, we will first need to

```
In [6]: # --- Import Supervised Learning methods ----

        from sklearn.linear_model import LogisticRegression
        from sklearn.svm import SVC
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.neural_network import MLPClassifier

In [7]: model_MaxEnt = LogisticRegression()
        model_SVM = SVC()
        model_RF = RandomForestClassifier()
        model_ANN = MLPClassifier()

In [8]: model_MaxEnt

Out[8]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                  intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                  penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                  verbose=0, warm_start=False)

In [9]: model_SVM

Out[9]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)

In [10]: model_RF

Out[10]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    min_impurity_split=1e-07, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
                    verbose=0, warm_start=False)

In [11]: model_ANN

Out[11]: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
                beta_2=0.999, early_stopping=False, epsilon=1e-08,
                hidden_layer_sizes=(100,), learning_rate='constant',
                learning_rate_init=0.001, max_iter=200, momentum=0.9,
                nesterovs_momentum=True, power_t=0.5, random_state=None,
                shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
                verbose=False, warm_start=False)
```

FIGURE 2. *Codes snippet corresponds to the loading of four machine learning methods from* Scikit-learn.

```
In [12]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

In [13]: tmp1 = model_MaxEnt.fit(X_train,y_train)
         tmp2 = model_SVM.fit(X_train,y_train)
         tmp3 = model_RF.fit(X_train,y_train)
         tmp4 = model_ANN.fit(X_train,y_train)

In [14]: y_pred_MaxEnt = model_MaxEnt.predict(X_test)
         y_pred_SVM = model_SVM.predict(X_test)
         y_pred_RF = model_RF.predict(X_test)
         y_pred_ANN = model_ANN.predict(X_test)
```

FIGURE 3. *Codes snippet corresponds to the training and prediction of different machine learning models.*

split our data into training and testing sets. Then, we fit the data with the training set and test the fitting against the testing set. In the following snippet (Figure 3), Input cell 12 (In [12]) uses a function in the *Scikit-learn* package to randomly split the data set into training and testing sets. The Input cell 13 (In [13]) shows how to fit each of the machine learning models using the training data set. Input cell 14 (In [14]) shows how we apply the fitted model to the testing data set to obtain the predicted labels.

So far, we have introduced how we fit four supervised learning methods from *Scikit-learn*. As one can see, it is almost trivial to generalize this to other methods

```
In [15]: # --- Model evaluation and selection ---
         from sklearn.metrics import accuracy_score, cohen_kappa_score

In [16]: accuracy_score(y_pred_MaxEnt, y_test)
Out[16]: 0.9066666666666662

In [17]: accuracy_score(y_pred_SVM, y_test)
Out[17]: 0.9866666666666669

In [18]: accuracy_score(y_pred_RF, y_test)
Out[18]: 0.9599999999999996

In [19]: accuracy_score(y_pred_ANN, y_test)
Out[19]: 0.9866666666666669

In [20]: cohen_kappa_score(y_pred_MaxEnt, y_test)
Out[20]: 0.8509982286634461

In [21]: cohen_kappa_score(y_pred_SVM, y_test)
Out[21]: 0.97987117552334946

In [22]: cohen_kappa_score(y_pred_RF, y_test)
Out[22]: 0.93961352657004826

In [23]: cohen_kappa_score(y_pred_ANN, y_test)
Out[23]: 0.97987117552334946
```

FIGURE 4. *Codes snippet corresponds to model evaluation.*

contained in the package. Next, we will evaluate the model prediction and single out the best model for the given data.

2.3.3. Model evaluation and selection. Machine learning methods are known to be prone to overfitting. As such, a typical approach to evaluating the performance of a machine learning method for a classification problem is cross-validation. That is, split the data into multiple portions randomly and use some portions as training sets and the others as test (validation) sets. In the previous section, we showed how to get the predicted labels. *Scikit-learn* provides simple metrics to evaluate the agreement between the predicted labels and the true labels. In the following snippet (Figure 4), we show how to evaluate the predictions by checking the accuracy (the fraction of correct classification over the total number of observations) and Cohen's κ. In Input cell 15 (In [15]), we load the corresponding functions, and the results are shown in the Output cells 16 through 23. All these classifiers perform well on the iris data set, even under their default hyperparameters in the *Scikit-learn* package.

In the above example, we checked the performance of the methods by a single training set and a single test set, which is a two-fold cross-validation. *Scikit-learn* provides convenient functions to do more complex cross-validation. In the following snippet (Figure 5), we show the code to perform a 3-fold cross-validation for different supervised learning models introduced above. The output cells (Out [25]–[28]) show the accuracy score of each fold of the cross-validation.

We have shown how to evaluate the performance of the classifiers based on the predicted labels. However, we did not provide details about the selection of

```
In [24]: # --- Cross validation and hyper-paramter search
         from sklearn.model_selection import cross_val_score

In [25]: cross_val_score(model_MaxEnt, X, y,cv=3)
Out[25]: array([ 0.88235294,  0.92156863,  0.89583333])

In [26]: cross_val_score(model_SVM, X, y,cv=3)
Out[26]: array([ 0.98039216,  0.92156863,  0.97916667])

In [27]: cross_val_score(model_RF, X, y,cv=3)
Out[27]: array([ 0.98039216,  0.94117647,  0.97916667])

In [28]: cross_val_score(model_ANN, X, y,cv=3)
Out[28]: array([ 0.94117647,  0.92156863,  0.95833333])
```

FIGURE 5. *Codes snippet corresponds to crossvalidation.*

```
In [29]: from sklearn.model_selection import GridSearchCV

In [30]: hyper_parameters = {'kernel':('linear', 'rbf'), 'C':[1,5,10]}

In [31]: model_grid = GridSearchCV(model_SVM, hyper_parameters,cv=3)

In [32]: model_grid.fit(X,y)
Out[32]: GridSearchCV(cv=3, error_score='raise',
                estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False),
                fit_params={}, iid=True, n_jobs=1,
                param_grid={'kernel': ('linear', 'rbf'), 'C': [1, 5, 10]},
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring=None, verbose=0)

In [33]: model_grid.best_params_
Out[33]: {'C': 10, 'kernel': 'rbf'}

In [34]: model_grid.best_score_
Out[34]: 0.98666666666666669
```

FIGURE 6. *Codes snippet corresponds to hyperparameter searching.*

the hyperparameters in each of the methods we introduced. There are no general rules for choosing the best set of hyperparameters for a given method with respect to a particular data set. A grid search in the hyperparameter space is widely used in machine learning applications. *Scikit-learn* provides a convenient function to allow an easy grid search of hyperparameters and return the best sets based on the performance. We show an example for searching the hyperparameters of the SVM algorithm in the following snippet (Figure 6). In Input cell 30 (In [30]), we first define the hyperparameter space (e.g., the kernel type and the regularization constant C) to search and then implement the search for the SVM model in Input cells 31 and 32 (In [31], In [32]). The Output cell 33 (Out [33]) shows the best hyperparameters based on 3-fold crossvalidation for the SVM model, and the Output cell 34 (Out [34]) gives the best accuracy score.

3. Summary

In this article, we present a concise overview of machine learning and introduce a powerful and well-maintained software package, *Scikit-learn*, written in

the popular Python programming language. We highlight the major differences of machine learning compared to statistical inferences that are more familiar to the community of educational and behavioral statistics, by which we hope researchers can make sensible decisions before they delve into the actual implementation of machine learning methods. Using supervised learning as an example, we show how to carry out some typical machine learning analyses using the *Scikit-learn* package in Python and attach the corresponding Jupyter Notebook for readers to practice the examples themselves. As we stated earlier, the goal of this article is neither trying to provide a comprehensive review of machine learning nor trying to completely cover the *Scikit-learn* package. Instead, we provided researchers with a general sense about how the *Scikit-learn* package can be used and explained why it is worth learning. We encourage the interested readers to check out more details and tutorials at the *Scikit-learn* website.

Declaration of Conflicting Interests

The author(s) declared the following potential conflicts of interest with respect to the research, authorship, and/or publication of this article: The first author prepared the work as employee of Educational Testing Service. The second author prepared the work as employee of IBM.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: Jiangang Hao is funded by the research allocation of Educational Testing Service and Tin Kam Ho is funded by IBM Watson.

References

Bernado-Mansilla, E., & Ho, T. K. (2004). On classifier domains of competence. *Proceedings of the 18th International Conference on Pattern Recognition*, Cambridge, England, August 22–26, Vol. 1, pp. 136–139.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York, NY: Springer-Verlag.

Blyth, C. R. (1972). On Simpson's paradox and the sure-thing principle. *Journal of the American Statistical Association*, *67*, 364–366.

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In W. Cohen & A. Moore (Eds.), *Proceedings of the 23rd International Conference on Machine learning* (ICML '06, pp. 161–168). New York, NY: Association for Computing Machinery.

Chen, A., Bengtsson, T., & Ho, T. K. (2009). A regression paradox for linear models: Sufficient conditions and relation to Simpson's paradox. *The American Statistician*, *63*, 218–225.

Fu, K. S. (1981). *Syntactic pattern recognition and applications*. New York, NY: Prentice Hall.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S. . . . Bengio, Y. (2014). Generative adversarial nets. *Proceedings of NIPS*, pp. 2672–2680.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction*. New York, NY: Springer.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Robinson, D. (2017, September 6). *The incredible growth of Python*. Retrieved from https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Stevens, S. S. (1946). On the theory of scales of measurement. *Science*, *103*, 677–680.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data mining: Practical machine learning tools and techniques*. San Francisco, CA: Morgan Kaufmann.

Authors

JIANGANG HAO is a senior research scientist in the center for next generation psychometrics and data science at Educational Testing Service, Rosedale Road, Princeton, NJ 08540; email: jhao@ets.org. His research interest centers on assessing collaborative problem-solving, game-based assessments, automated scoring, and educational data science.

TIN KAM HO is a senior scientist in artificial intelligence research and development at IBM Watson, Yorktown Heights, NY 10598; email: tho@us.ibm.com. Her research interests are in natural language semantic analysis, conversational systems, machine teaching, and machine learning.

Manuscript received May 18, 2018
First revision received August 7, 2018
Second revision received October 30, 2018
Accepted December 23, 2018