Chapter 9 Supervised Machine Learning*

Jiangang Hao

Educational Testing Service

Abstract

Machine learning refers to a set of methodologies that allow computers to "learn" the relationship among numerical representations of data. In this Chapter, we focus on an important branch of machine learning, supervised machine learning, and introduce three widely used supervised learning methods, the Support Vector Machine, Random forest, and Gradient Boosting Machine. Python codes examples are included to show how to use these methods in practice.

Keywords: Supervised Learning, Bagging and Boosting, Statistical Inference

9.1 Introduction

As highlighted in Chapter 6, one of the core missions of psychometrics is to ensure the observed evidence from assessment or learning tasks can support the claims on the targeted constructs in a valid, reliable, fair and comparable way. In the area of educational assessment, traditional psychometrics has given most attention to developing systematic methodologies to accomplish this mission when the evidence can be easily mapped into some simple forms, such as dichomotous or polymotous scores. The regularity and simplicity of the data make statistical inference well suited for modeling the data. However, digitally based learning and assessment tasks

* The R or Python codes can be found at the GitHub repository of this book: https://github.com/jgbrainstorm/computational_psychometrics

generate much more complex data. The complexity of these data makes it difficult (or sometimes impossible) to represent the information in simple scores in order to apply well-established statistical modeling frameworks, such as the familiar Item Response Theory (IRT). So, there is an intrinsic need for additional methodologies to harness the more complex data from digitally based tasks. Fortunately, the need for "modeling" complex data emerged much earlier in disciplines than psychometrics, and many methodologies have already been developed. In this and the next two consecutive Chapters, we will introduce a set of such powerful methodologies, machine learning, which was developed since late 1950s in the field of computer science and statistics for dealing with complex data.

Machine learning, a term coined by Auther Samuel (Samuel, 1959), refers to a set of methodologies that allow computers to "learn" the relationship among numerical representations of data without explicit instructions by human experts. Based on the learning goals, machine learning tasks are classified into broad categories such as supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. The numerical representation of data usually can be categorized into features (independent variables) and labels (dependent variables). If a task is to learn the relationship between the features and labels, it is a supervised learning task. If the goal is to discover concentrations, associations, or correlations in the features themselves, the corresponding task is unsupervised learning (e.g., see Chapter 10). Semi-supervised learning refers to the task of learning the mapping between features and labels when only part of the training data has labels. Reinforcement learning is to learn optimal strategies by means of a goal-oriented exploration of a parameter or state-space to optimize a reward function (Sutton, & Barto, 1998).

Numerous methods[1] have been developed in each of these categories, and there are many excellent texts that provide comprehensive coverage of them (e.g., Bishop, 2006; Hastie, Tibshirani, & Friedman, 2009; Witten, Frank, Hall, & Pal, 2016). It is worth noting that the boundaries of the broad categories above are not always clear-cut, as many methods can be used in more than one types of machine learning task in practice, depending on the ways one trains the algorithms. The current Chapter focuses on supervised machine learning (or supervised learning), Chapter 10 focuses on unsupervised machine learning, and Chapter 11 focuses on deep learning.

9.2 Supervised Learning

Supervised learning refers to a subset of machine learning algorithms that establish a mapping between features and labels of a dataset. The precondition of using supervised learning methods is that both the features and labels are known. Supervised learning methods can be grouped into two categories based on the nature of the labels: regression for continuous labels, and classification for discrete labels. Two primary questions are frequently raised by researchers with psychometric backgrounds when they are introduced to supervised learning. The first is how supervised learning is different from statistical inference, as they both aim at mapping the relationship between the independent and dependent variables. The second is which, if any, supervised learning methods are consistently superior to the others.

For the first question, there are many excellent discussions from different perspectives (e.g., Hastie, Tibshirani, & Friedman, 2009). Hao & Ho (2019) recently summarized three different

[1] A machine learning method is often referred to as a learner in machine learning literature

emphases between statistical inference and machine learning: machine learning is more prediction-driven while statistical inference emphasizes both prediction and model parameter estimation; statistical inference emphasizes developing a probabilistic model to characterize the data and then estimating the model parameters based on some (usually well-studied) probability distribution functions, while machine learning emphasizes computation algorithms that can efficiently carry out the inference process by minimizing certain loss functions that do not necessarily have a probabilistic underpinning; and statistical inference usually deals with data with a small number of variables obtained through planned *experiments* or quasi-experimental comparisons while machine learning handles sparse and high-dimensional data with a large number of variables (features), usually obtained through passive and uncontrolled *observations*. Despite these different emphases, the distinctions are not always black and white. Increasingly, machine learning makes use of many techniques from statistical modeling.

The second question is simple to state, but has no simple answer. Caruana and Niculescu-Mizil (2006) carried out an extensive empirical study to compare a number of supervised learning methods based on their performance on empirical datasets. The conclusion from the study was that the performance of a method is essentially dependent on the specific dataset, although the support vector machine (SVM) and random forest (RF) are the top performers for a number of classification tasks. In recent years, Gradient Boosting Machine (GBM), especially one of its variants known as XGBoost (Chen & Guestrin, 2016), and Deep Neural Networks (see Chapter 11 for more details) became very popular in the Kaggle community for machine learning competitions (https://www.kaggle.com/competitions) and frequently show up as leading performers. A sensible

recommendation for data analysis practitioners may be that they should first try out these methods if they do not already know which one to use.

In addition, given there are many supervised learning methods, it is always tantalizing to think whether some sort of "average" from an ensemble of learners will lead to better predictive performance. Research studies in this direction leads to ensemble learning (Dietterich, 2000). There are two main leading ideas regarding how an ensemble of learners is constructed, one is called bagging (stands for **b**ootstrap **agg**regat**ing**, Breiman, 1996), and the other is called boosting (Friedman, 2001). The bagging primarily aims to reduce the variance of the prediction while the boosting helps to reduce the bias. In each of these approaches, a base learner is identified first, for example, a decision tree. In the bagging approach, multiple training datasets are created through a bootstrapping procedure (Efron & Tibshirani, 1994), and the base learner will be applied to each of the samples. The outputs from the learners from each sample will be combined (average or voting) to improve the predictive performance. In the boosting approach, the ensemble is constructed sequentially, rather than in parallel, as in the bagging approach. A learner is first trained on the full dataset and subsequent learners are added by fitting the residuals (after applying all preceding learners), by which new learners will assign more weights to the poorly predicted observations by the preceding learners. Random forest and gradient boosting machine (next subsection) are two well-known examples of the bagging and boosting approaches, respectively, both of which use decision trees as learners (e.g., homogeneous learners). In addition to bagging and boosting, there is another ensemble learning approach, known as stacking, which uses the outputs/predictions from several heterogeneous learners (e.g., different machine learning algorithms that do not fall into the same family) as new features and then build a mapping between these features and the labels via another

machine learning method. Compared with the bagging and boosting, stacking could reduce both the variance and bias. In practice, there are numerous different ways to implement stacking, so we (as well as many textbooks on machine learning) primarily focus on bagging and boosting in this chapter.

As the details of each supervised learning algorithms could easily be a Chapter or even a book itself, in the following subsections, we limit ourselves to the basic ideas of some popular methods and refer readers to the textbooks introduced earlier for more rigorous mathematical formulations. We hope that knowing the basic mechanisms behind the methods will help guide practitioners to sensibly choose suitable methods and adjust the corresponding hyperparameters in their applications.

9.2.1 Support Vector Machine

The Support Vector Machine (SVM) was initially introduced in the context of pattern recognition (Vapnik, 1963). It aims at finding a decision hyperplane in the feature space so that data points can be separated into different categories with a maximum margin that is defined as the distance of the closest points (support vectors) from each category to the decision hyperplane. Figure 1 shows a simple case, a two-dimensional feature space, and a binary label, as indicated by the types of the dots, e.g., empty or filled. The left panel shows that there could be multiple ways to draw a straight line (one-dimensional hyperplane) to separate the dots. The right panel shows the straight line that has the maximum margins, which is what SVM is seeking for.

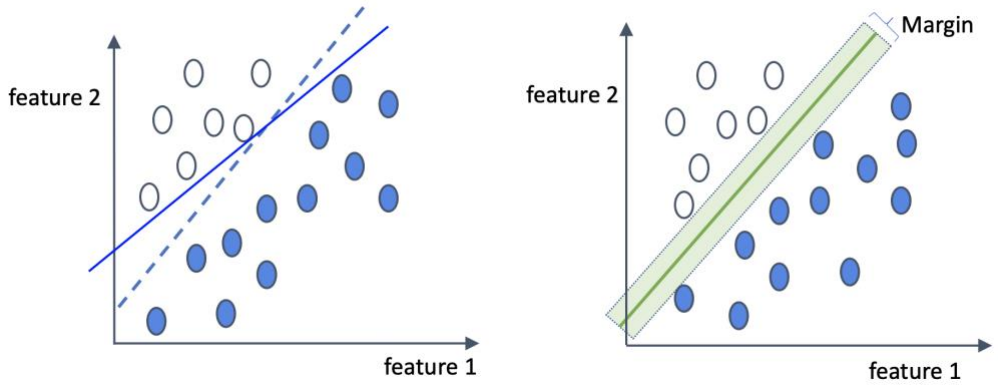

Figure 1. Linearly separable case in a two-dimensional feature space and binary labels. Left: possible hyperplanes. Right: the hyperplane that has a maximum margin. The dots with a green outline are called support vectors.

The case shown in Figure 1 is very simple, not only because of the two-dimensional feature space and binary labels but also because of the dots that are linearly separable (also known as a hard margin). In reality, there could be cases where data of different labels cannot be linearly separated. We show some examples in the left panels of Figures 2 and 3. To address these situations, two additional adjustments to SVM have been introduced; one is called regularization and the other is called the kernel trick.

In the case when there is not a linear separation for the dots, as shown in Figure 2, one needs to choose between a decision hyperplane that has a larger margin but more misclassified data points, and the one with a narrower margin but fewer misclassified data points, as illustrated in the left and right panels in Figure 2 respectively. Qualitatively, a hyperplane with a narrower margin may lead to better classification accuracy based on the training data, but is more susceptible to overfitting, while a hyperplane with a larger margin has lower classification accuracy with respect to the training data, but it is more robust against overfitting. In the algorithmic implementation of SVM, a regularization

term (equivalent to a soft margin or hinge loss function) is introduced to the loss function, and a hyperparameter (usually denoted as *C*) is used to adjust between these two preferences quantitatively. Choosing a larger *C* indicates that one favors a narrower margin with a lower tolerance of misclassified data points. In practice, one needs to decide on the most appropriate *C* based on the particular nature of the dataset, usually by checking the predictive performance of a range of *C* based on a certain cross-validation scheme.

On the other hand, as shown in the left panel of Figure 3, though one can apply the above regularization scheme, there could be a better way. For example, one can introduce a set of new variables such that $x = feature\ 1$, $y = feature\ 2$, $z = (feature\ 1)^2 + (feature\ 2)^2$. The space spanned by the new variables is now three dimensional, and a 2D plane can be found to separate the empty and filled dots, as shown in the right panel of Figure 3. This transformation of the original feature space into a higher dimensional space to make the data linearly separable is done through a procedure called the kernel trick. The characteristic of data being more likely to be linearly separable when being projected into a higher dimensional space via some non-linear transformation is known as Cover's theorem (Cover, 1965). There are many well-studied kernel functions in SVM, such as the linear kernel and radial basis function (RBF) kernel, and the type, as well as parameters of the kernels become additional hyperparameters of SVM. We will skip the detailed mathematical formulations of the kernels and the algorithms for searching the hyperplanes, and recommend interested readers to the texts on machine learning introduced earlier for more details.

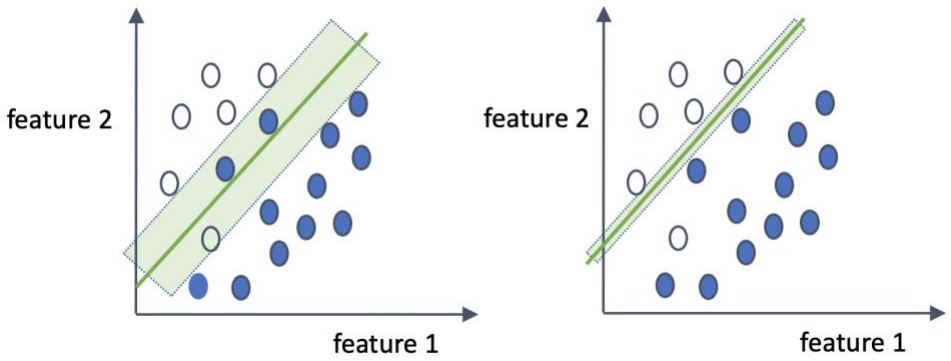

Figure 2. Non-linearly separable cases. Left: larger margin with more misclassified data points. Right: narrower margin with less misclassified data points.

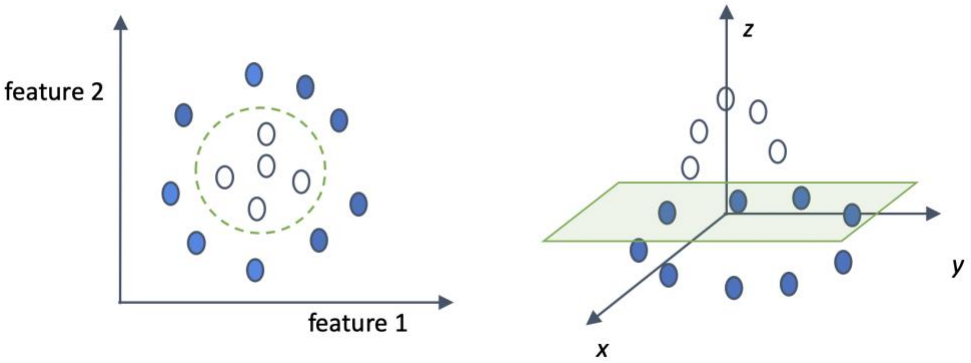

Figure 3. Projection of a 2D feature space to a 3D space to make the data linearly separable.

9.2.2 Random Forest

Random forest (Ho, 1995; Breiman, 2001) was introduced based on tree-structured learners, such as decision trees. A decision tree works by forming decision rules on the features recursively to minimize some loss function of the classification. A schematic of the classification process of a binary decision tree classifier is shown in Figure 4. In a decision tree, the root node represents the entire data under consideration. The decision node is where decisions are made to further split the

node into sub-nodes. The terminal node, or leaf, represents a note that cannot be split further. To avoid overfitting, a procedure to remove the sub-nodes from decision nodes is often performed after the training, which is called pruning. When implementing a decision tree, an important question that needs to be addressed is which loss function one wants to minimize when creating the decision rules. The choice of different loss functions leads to different algorithms for a decision tree. Popular loss functions include the Gini index as used in the CART algorithm (Breiman et al., 1984) and Information Gain as used in the Iterative Dichotomiser 3 algorithm (Quinlan, 1986). After deciding on the loss function, a tree algorithm must also specify a stopping rule, or the condition under which no further splitting will happen. For example, a stopping condition can be set as the minimum leaf size, meaning that if all the nodes have less than a specified threshold, the node splitting process will stop. The stopping rules are usually specified through the hyperparameters in most decision tree algorithm implementations.

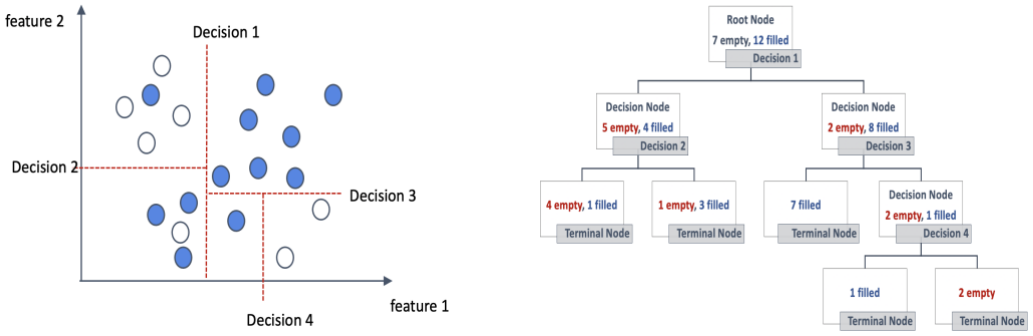

Figure 4. A schematic of decision rules for a binary decision tree classifier. Left is actual decision rules in the 2D feature space. Right is a tree-structured representation.

One of the main drawbacks of the decision tree is the instability of the results, as a small change in a dataset may lead to quite different decision rules. Therefore, the random forest was proposed to

use the "average/voting" of multiple decision trees instead of a single one. Suppose we have *N* observations and *M* features in a dataset, then a typical random forest classifier goes as follows. First, create *K* samples of the dataset using the bootstrapping technique, and each sample has *N* observations. For each of the K samples, a decision tree is applied. In each of the decision nodes of each of the trees, a decision is made based on the most discriminative one of a randomly chosen *F* features from the total *M* features. The number *F* is usually fixed for all trees in most of the random forest algorithms. The selection of a random subset of features is known as the random subspace method or feature bagging (Ho, 1998), which was proposed to reduce the correlation among the estimators in the ensemble. Overall, the random forest method is much more stable and shows better predictive performance compared to a single decision tree method, though the cost paid for this is the lack of interpretability.

9.2.3 Gradient Boosting Machine

When we introduced the SVM and RF, we focused on the classification case because it is easier to convey the basic ideas of the two methods, using as few equations as possible. For the Gradient Boosting Machine (GBM), the regression case turns out to be easier to convey the key ideas of the method, and some equations are necessary to make the description clear. The basic idea behind GBM is the assumption that the ideal mapping between the dependent and independent variables could be approximated by the sum of many functions belonging to the same parametric family. The parameters of these functions will be determined iteratively. First, one fits the first function to the data and determines the corresponding parameters. Then, one determines the second function's parameters by fitting it to the residuals after subtracting the fitting of the first function from the data.

Continue this process to obtrain the parameters of the other functions until a stopping criterion is met. In the following, we introduce how GBM works in details, largely by following the description in Li (2016) with some notation adjustments.

Suppose we have a dataset that consists of *N* observations. Each observation is represented by a feature vector $\boldsymbol{X}_i$ and a target value y_i, and we use $(\boldsymbol{X}, y)$ to denote the data collectively. Our goal is to find a mapping function $F(\boldsymbol{X})$ that can minimize the expected value of a loss function, e.g., $E_{\boldsymbol{X},y} L(y, F(\boldsymbol{X}))$. Given that the functional form of $F(\boldsymbol{X})$ can be anything, a practical strategy is usually to restrict $F(\boldsymbol{X})$ as a weighted sum of a parameterized class of functions, such as regression trees[2]. This way, we can write $F(\boldsymbol{X}) = \sum_{m=0}^{M} \beta_m h_m(\boldsymbol{X}; \boldsymbol{\alpha}_m)$, where $h_m(\boldsymbol{X}; \boldsymbol{\alpha}_m)$ is a regression tree, β_m is the weight, and $\boldsymbol{\alpha}_m$ are the parameters of the regression tree. A procedure to iteratively generate $h_m(\boldsymbol{X}; \boldsymbol{\alpha}_m)$ can go as follows. First, we fit the data $(\boldsymbol{X}, y)$ with a base regression tree (or base learner) $h_0(\boldsymbol{X}; \boldsymbol{\alpha}_0)$. Then, to find a new $h_1(\boldsymbol{X}; \boldsymbol{\alpha}_1)$ that can improve the fitting, we simply fit it to the residuals after substracting the fitted value of $h_0(\boldsymbol{X}; \boldsymbol{\alpha}_0)$ from the target values, e.g., $(\boldsymbol{X}, y - h_0(\boldsymbol{X}; \boldsymbol{\alpha}_0))$. Obviously, $h_0(\boldsymbol{X}; \boldsymbol{\alpha}_0) + h_1(\boldsymbol{X}; \boldsymbol{\alpha}_1)$ will be a better fit to the data than $h_0(\boldsymbol{X}; \boldsymbol{\alpha}_0)$. We can iterate this process to get $h_m(\boldsymbol{X}; \boldsymbol{\alpha}_m)$ by fitting the residual data $(\boldsymbol{X}, y - F_{m-1}(\boldsymbol{X}))$ with $F_{m-1}(\boldsymbol{X}) \equiv \sum_{k=0}^{m-1} \beta_k h_k(\boldsymbol{X}; \boldsymbol{\alpha}_k)$.

In the case of L2 loss function, e.g., $L(y, F(\boldsymbol{X})) \sim \| y - F(\boldsymbol{X}) \|^2$, the residuals relate to the negative gradient of the loss function as

$$y - F_m(\boldsymbol{X}) \propto -\frac{\partial L(y,\ F(\boldsymbol{X}))}{\partial F(\boldsymbol{X})}\Big|_{F(\boldsymbol{X})=F_{m-1}(\boldsymbol{X})} \equiv -g_m(\boldsymbol{X}),$$

[2] Regression tree is a generalization of the binary decision tree to the case where the target variable is real-valued instead of binary.

Where $g_m(\boldsymbol{X})$ denotes the gradient. Now, our previous procedure for obtaining new $h_m(\boldsymbol{X}; \boldsymbol{\alpha}_m)$ by fitting the residual data $(\boldsymbol{X}, y - F_{m-1}(\boldsymbol{X}))$ can be recast as by fitting $(\boldsymbol{X}, -g_{m-1}(\boldsymbol{X}))$, which is why this method is called gradient boosting. This line of reasoning can be generalized to other forms of loss functions that may obscure the initial residual interpretation but is still effective. Note that the above introduction is not a strict mathematical formulation but rather an intuitive way to show the basic ideas of the method. We refer readers to Friedman (2001) for a more rigorous mathematical treatment.

9.3 Hyperparameters

Hyperparameters are those parameters that cannot be estimated through data, such as the regularization and kernel selection in SVM; the number of trees and the size of the feature subspace in RF; and the choice of loss function and the number of boosting stages in GBM. Despite that the choice of hyperparameters often significantly affects the performance of an algorithm, there are actually no fixed rules for choosing the best hyperparameters, though certain default values are assigned to the hyperparameters in most software implementations. Readers need to be aware that these default values are by no means the optimal ones for a particular dataset, and some tuning of the hyperparameters is needed to achieve optimal performance. A brute-force grid search in the hyperparameter space is the most typical approach for deciding the best hyperparameter sets for a given dataset and classification problem. There are many tools or packages for facilitating the hyperparameter search. For example, the Scikit-learn package (Pedregosa et al., 2011) in Python provides functions to run grid search of hyperparameters. Interested readers can find more details from the Scikit-learn documentation website, or from Hao & Ho (2019) for a quick start.

9.4 Examples with Python

In this section, we show some code examples in Python to illustrate how to apply the above machine learning methods in practice. Scikit-learn is an open-source library for machine learning in Python. It includes most of the popular supervised learning algorithms. The general workflow for using a classifier in *Scikit-learn* includes three steps. First, create the model by specifying the hyperparameters of the model. Second, fit the training data to the model and learn the parameters. Finally, apply the fitted model to the test data to get predicted labels. The pseudo-code for these steps is as follows, where `classifier()` is an instance of one of the supervised learning methods. *X* is the feature matrix whose rows index observations and columns index different feature variables. *y* is the array of labels with each element as the corresponding label value.

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

The data used in this example are the labeled chats from an online collaborative problem-solving (CPS) study (Hao et al., 2015). Each turn of the chats has been coded into one of four categories of CPS skills based on a coding rubric (Liu et al., 2015). The goal is to build an automated classifier using machine learning methods to label the chats automatically. Natural language processing (NLP) techniques (see Chapter 14) have been applied to convert each turn of the chats into a numerical vector (for simplicity, we consider only the uni-gram and bigram). In the following Figure 5, we show the Python code for building an automated classifier based on the above dataset by using SVM, RF, and GBT. The detailed meaning of the codes can be found in the corresponding comments of each cell. It is worth noting that only the default values of the hyperparameters in each

method have been used for illustration purposes. Even without any further tuning, the three supervised learning methods still generate reasonably good classification accuracy (defined as the total number of agreements divided by the total number of observations) of about 0.7, which is far better than the baseline (e.g., random assignment of the most frequent category) of about 0.3.

The example code snippet shows the general idea of building an automated text classifier using machine learning methods. Readers should be aware that a real automated annotation engine requires much more effort, such as NLP feature engineering, machine learning model selection, hyperparameter tuning, and cross-validation. Interested readers should consult relevant texts and references. For example, a good starting point is Hao et al. (2017) and references therein.

1. Load the needed libraries

In [1]:
```
from sklearn.svm import LinearSVC # for Support Vector Machine
from sklearn.ensemble import RandomForestClassifier # for Random Forest
from sklearn.ensemble import GradientBoostingClassifier # for Gradient Boosting Machine

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
import warnings
warnings.simplefilter('ignore')
```

2. Instantiate the models and check the details

In [2]:
```
model_SVM = LinearSVC()
model_RF = RandomForestClassifier()
model_GBM = GradientBoostingClassifier()
```

In [3]:
```
model_SVM
```

Out[3]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

In [4]:
```
model_RF
```

Out[4]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators='warn',
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [5]:
```
model_GBM
```

Out[5]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

3. Read in the data file

In [6]:
```
X=pd.read_csv('chat_bigram_feature.csv').values
y=pd.read_csv('chat_label.csv').values.ravel()
```

4. Create the training and validation sets

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
```

5. Train the models and generate predicted labels

In [8]:
```
# train the models with training set
model_SVM.fit(X_train,y_train)
model_RF.fit(X_train,y_train)
model_GBM.fit(X_train,y_train)

# -- get the predicted labels on the test dataset.
y_pred_SVM = model_SVM.predict(X_test)
y_pred_RF = model_RF.predict(X_test)
y_pred_GBM = model_GBM.predict(X_test)
```

6. Check the predictive performance

In [9]:
```
# calculate the accuracy of the predicted labels from SVM
accuracy_score(y_pred_SVM, y_test).round(3)
```

Out[9]: 0.692

In [10]:
```
# calculate the accuracy of the predicted labels from Random Forest
accuracy_score(y_pred_RF, y_test).round(3)
```

Out[10]: 0.644

In [11]:
```
# calculate the accuracy of the predicted labels from Gradient Boosting Machine
accuracy_score(y_pred_GBM, y_test).round(3)
```

Out[11]: 0.642

Figure 5. Implementation of the SVM, RF, and GBM in the Scikit-learn package in Python programming language.

9.5. Summary

In this Chapter, we introduced supervised machine learning and its different emphasis as compared to the statistical inference that is more familiar to researchers with statistic and psychometric backgrounds. By this, we hope readers can make sensible decisions regarding which approaches are

best suited for their application. We introduced the basic ideas behind three popular supervised learning methods, the SVM, RF, and GBM. We further showed how to apply these methods to a real-world problem using coding examples written in Python. As stated earlier, the goal of this Chapter is not to provide comprehensive coverage of supervised learning but more a gentle introduction to provide researchers with a general sense of supervised learning and how to use it in practice. We encourage interested readers to check out more details about machine learning from those dedicated volumes as suggested in the introduction. In the next Chapter (Chapter 10), a similar introduction to unsupervised machine learning will be provided.

Acknowledgement

The author thanks Mr. Andrew Cantine and Robert Mislevy for helpful suggestions and copyediting.

References

Bernado-Mansilla, E., & Ho, T.K., (2004). On Classifier Domains of Competence, *Proceedings of the 18th International Conference on Pattern Recognition*, Cambridge, U.K., August 22-26, 2004, Volume 1, 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.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and Regression Trees (Monterey, California: Wadsworth).

Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-14

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

Caruana, R., & Niculescu-Mizil, A., (2006). An empirical comparison of supervised learning algorithms. In *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.

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). ACM.

Cover, T. M. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE transactions on electronic computers, (3), 326-334.

Dietterich, T. G. (2000, June). Ensemble methods in machine learning. In International workshop on multiple classifier systems (pp. 1-15). Springer, Berlin, Heidelberg.

Efron, B., & Tibshirani, R. J. (1994). An introduction to the bootstrap. CRC press.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of Statistics, 1189-1232.

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

Gerdes, D. W., Sypniewski, A. J., McKay, T. A., Hao, J., Weis, M. R., Wechsler, R. H., & Busha, M. T. (2010). ArborZ: Photometric redshifts using boosted decision trees. The Astrophysical Journal, 715(2), 823.

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

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

Hao, J., & Ho, T. K. (2019). Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language. Journal of Educational and Behavioral Statistics, 44(3), 348-361.

Hao, J., Koester, B. P., Mckay, T. A., Rykoff, E. S., Rozo, E., Evrard, A., ... & Johnston, D. E. (2009). Precision measurements of the cluster red sequence using an error-corrected gaussian mixture model. The Astrophysical Journal, 702(1), 745.

Ho, T. K. (1995, August). Random decision forests. In Proceedings of 3rd international conference on document analysis and recognition (Vol. 1, pp. 278-282). IEEE.

Ho, T.K. (1998). The random subspace method for constructing decision forests. IEEE Trans. Pattern Anal. Mach. Intell, 20(8), 1-22.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436.

Li, C. (2016). A Gentle introduction to gradient boosting. URL: http://www. ccs. neu. edu/home/vip/teach/MLcourse/4_ boosting/slides/gradient_boosting. pdf.

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.

Quinlan, J.R. (1986), Induction of decision trees. Mach Learn 1, 81–106

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

Samuel, A. L. (1959. Some studies on machine learning using the game of checkers. IBM Journal of Research and Development, 3, 211-229.

Steven, 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*. Morgan Kaufmann.

Vapnik, V. (1963). Pattern recognition using generalized portrait method. Automation and remote control, 24, 774-780.