Methods in Open Science

Building and Improving Models

Image from NASA: https://commons.wikimedia.org/wiki/File:Earth_Western_Hemisphere_transparent_background.png#filelinks

WV VIEW

AmericaView
Empowering Earth Observation Education
AmericaView.org - Est. 2003

Building off of the module on predictive modeling and machine learning, we will now discuss key considerations and methods for creating models.

Hyperparameter Optimization

User-Defined Hyperparameters

Algorithm	User-Defined Parameters	Example References
k-nearest neighbor	-Number of neighbors considered (k)	Cover and Hart, 1967; Dudani, 1976; Bailey and Jane, 1978
Multilayer perception (MLP) artificial neural networks with backpropagation	-Number of hidden layers -Number of nodes in each layer -Learning rate -Momentum factor -Weight initialization -Number of iterations	Rumelhart et al., 1986
Decision trees	-Pruning parameters	Quinlan, 1986
Boosted decision trees (Adaboost method)	-Number of iterations or trees (n)	Freund and Schapire, 1996
Random forests	-Number of trees (n) -Number of variables randomly sampled as candidates at each split (m)	Breiman, 2001
Support vector machines using polynomial kernel	-polynomial order (p) -Cost or slack parameter (C)	Cortes and Vapnik, 1995; Vapnik, 1995
Support vector machines using radial basis kernel	-gamma (γ) -Cost or slack parameter (C)	Cortes and Vapnik, 1995; Vapnik, 1995

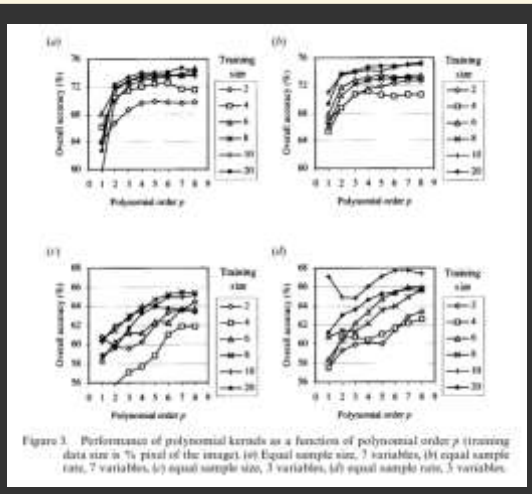
3

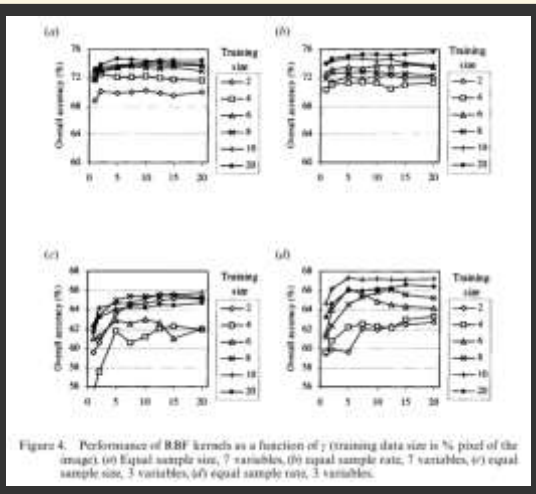
Most machine learning methods have hyperparameters that are not learned during the training process and must be set by the user. I generally think of these as the different knobs that can be turned by the user for a specific learning algorithm.

The specific hyperparameters and number of hyperparameters varies between algorithms. Also, how sensitive an algorithm is to its hyperparameters also varies. It is not generally known in advance if optimizing the hyperparameters will improve the performance of the algorithm for a specific task. So, it is generally recommended that an optimization process be performed in order to select hyperparameters.

This slide provides some example hyperparameters for different algorithms.

Does the algorithm need optimization?





Huang, C., Davis, L.S. and Townshend, J.R.G., 2002. An assessment of support vector machines for land cover classification. *International Journal of remote sensing*, *23*(4), pp.725-749.

4

Again, hyperparameter optimization may or not improve model performance for a specific tasks. Since the impact of the hyperparameters in not know beforehand, it is generally recommended that different settings be tested. This is the process of optimization or hyperparameter tuning.

The figures included on this slide are from a study that explored hyperparameters for the SVM algorithm. For the specific task investigated, when using a polynomial kernel, the order hyperparameter tended to have a larger impact in comparison to the gamma parameter when a radial basis function kernel was used. Further, the impact varied based on the size of the training dataset.

It would be wrong to assume that the results of this study would hold true for other problems; that is why it is generally recommended to optimize hyperparameters.

Grid Search

- If only one hyperparameter is being assessed, all specified values will be tested
- If multiple hyperparameters are being assessed, then you will need to test every combination of parameters
- This can be done using a grid search

	A	B	C	D
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5

5

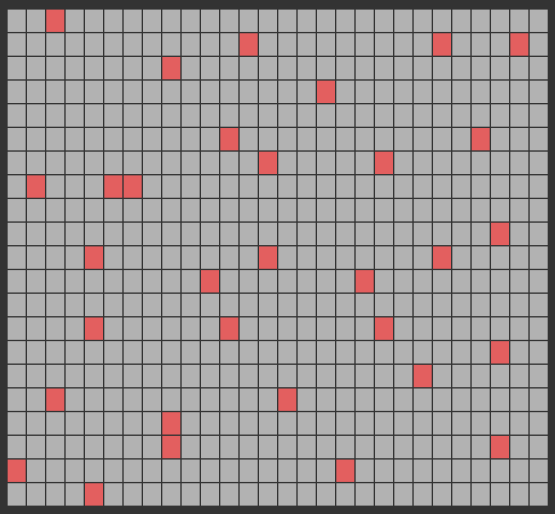
There are different methods available to tune or optimize hyperparameters. Here, we will explore the grid search as an example. The idea here is that all combinations of the hyperparameters are assessed. In the example table, one parameter has four options, A through D, and the other has five options, 1 through 5. So, a total of 20 combinations would be tested.

Note that this process can be slow due to the large number of models that must be trained and assessed. It might take several hours or even days to perform these experiments. One means to speed up the process is to use parallel computing if available.

Another option to potentially speed up the process is to test a coarse set of values then test over a smaller, or more precise, range of values dictated by the results from the coarser search. Essentially, you can obtain a rough estimate of the hyperparameter settings in the first pass then refine them in the second pass.

Hold Out

- Data are split into training and testing/validation sets
- Model is created on training sample
- Model is tested on testing/validation sample
- Hyperparameters that provide the best prediction on the testing/validation data are selected



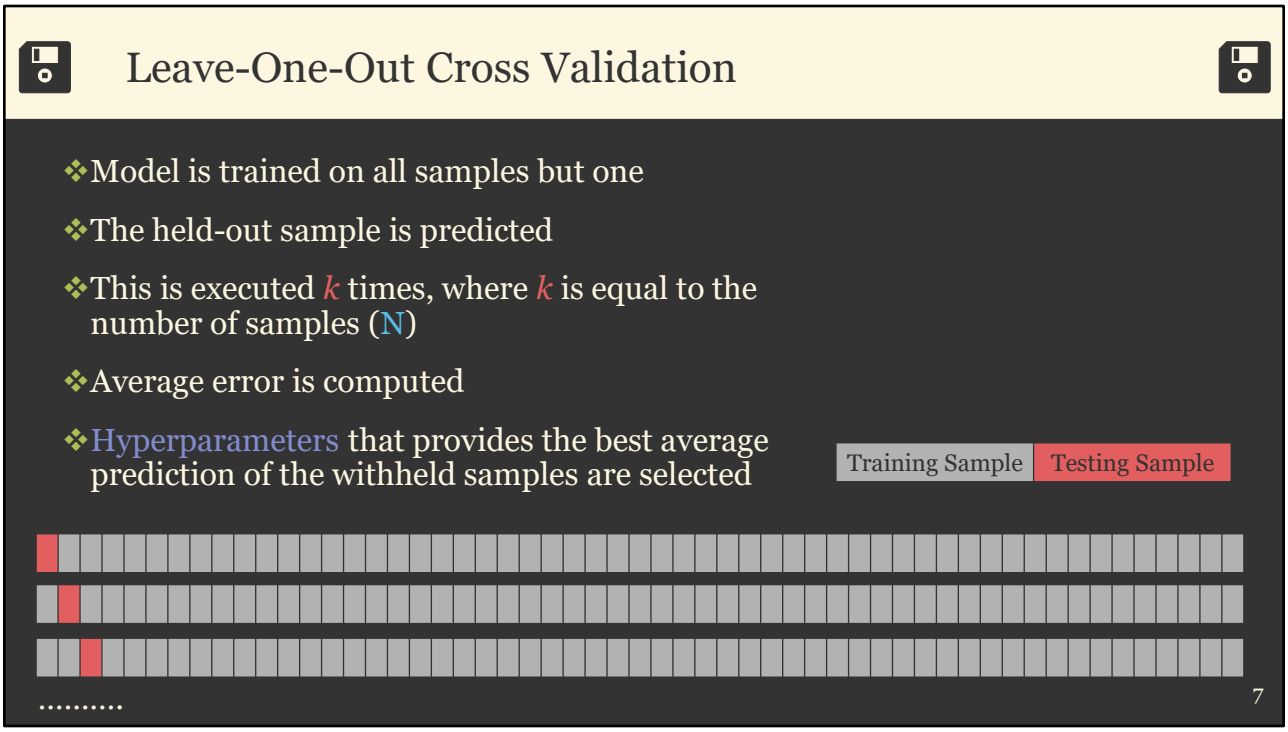
6

When performing hyperparameter optimization, you want to select the hyperparameters that yield the best model generalization. So, you want to test the results on a set of samples that were not used to train the model. Assessing on the training samples will likely bias the analysis due to overfitting.

One option is to split the available samples into separate training and testing/validation sets. This is known as the hold out method. The training set will be used to train the models using different combinations of the hyperparameters while the testing samples will be withheld to compare the models and associated hyperparameters.

In the graphic provided, each block represents a sample, and the entire dataset has been randomly partitioned into training and testing sets.

One issue with the hold out method is that it reduces the number of samples that are available to train the model, which may be a problem especially when the number of training samples are limited. So, other methods to partition the data and test settings have been developed.

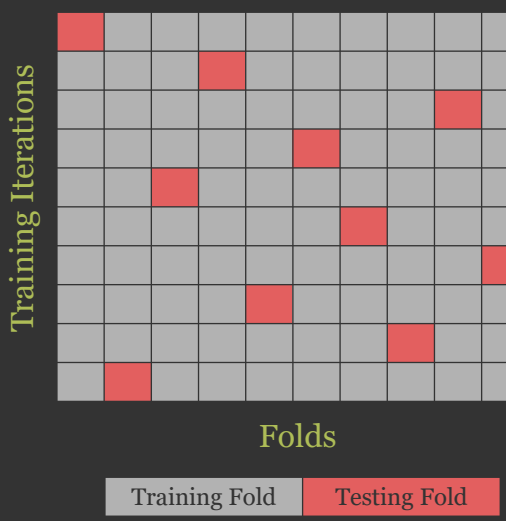


Leave-one-out cross validation consists of training the model on all samples but one. The single withheld sample is then predicted. This process is repeated until all samples have been withheld, and the error is then computed across all the runs. This means that if there are 1,000 samples, 1,000 models will need to be generated for each set of hyperparameters tested in the grid search.

This method is generally extremely slow, so I avoid using it.

k-Fold Cross Validation

- Data are randomly split into *k* folds (for example, 10)
- Model is trained using *k*-1 folds and repeated *k* times, so that each fold is left out once
- Hyperparameters are assessed using the withheld fold each time the model is ran
- Hyperparameters that provide the best average prediction of the withheld fold are selected



8

The *k*-fold cross validation sample partitioning method is my go-to method. The idea here is that the entire dataset is partitioned into *k* folds. The model is than ran *k* times, each time training on *k*–1 folds and maintaining a single fold to validate the results. Using this method, all data points are used to train the model and validate the model. In the graphic, each cell represents a fold. In this case, the data are partitioned into 10 folds, and the model is trained 10 times, each time using 9 folds and testing on the remaining fold. This will have to be repeated for each combination of hyperparameters tested.

It is common to use 5- or 10- folds. If my sample size is small, I tend to use 5-folds so that each fold does not consist of a very small set of samples. For larger datasets, I generally use 10 folds.

k-Fold Cross Validation with Repeats

- Data are randomly split into *k* folds (for example, 10)
- Model is trained using *k*-1 folds and repeated *k* times, so that each fold is left out once
- Hyperparameters are assessed using the withheld fold each time the model is ran
- Process is repeated a defined number of times using different folds
- Hyperparameters that provide the best average prediction of the withheld fold are selected

9

In *k*-fold cross validation, the folds or data partitions are the same throughout the entire tuning process. One means to potentially improve this process is to repeat the experiment using different random data partitions or folds. This is the concept behind *k*-fold cross validation with repeats. For example, the 10-fold cross validation could be repeated 5 times using different sets of folds.

Bootstrapping

- Random Sampling with Replacement
- A sample is drawn with replacement from the full dataset, producing a training set of *n* observations (with duplicates) and an out-of-bag test set consisting of the observations not selected (~36.8% of unique samples)
- This is repeated multiple times, and the hyperparameters that yield the best average out-of-bag performance are selected

https://commons.wikimedia.org/wiki/File:JM_marbles_01.jpg

10

Bootstrap sampling for hyperparameter tuning is a resampling technique in which multiple training and evaluation sets are constructed by repeatedly drawing samples with replacement from the original dataset. In each iteration, a bootstrap sample of *n* observations is drawn from a dataset of size *n*, meaning individual observations may appear more than once within a single training set. Because sampling is done with replacement, approximately 63.2% of the unique observations are selected into the training set on average, while the remaining ~36.8% of observations, those never selected in that draw, known as out-of-bag (OOB) samples, serve as the held-out evaluation set for that iteration. This process is repeated many times, each time training the model with a candidate hyperparameter configuration on the bootstrap sample and evaluating its performance on the corresponding OOB set. The hyperparameter configuration that yields the best average OOB performance across all iterations is then selected, with the repeated resampling providing a stable and relatively low-variance estimate of generalization performance.

What method should you use?

- Number of training samples available
- Algorithm that is being tuned
- Number of hyperparameters
- Number of values/combination of values to assess
- Time/computational requirements (this can take some time)

11

There is not necessarily a best data partition or sampling method for hyperparameter tuning. However, there are some important considerations. First off, if only a limited number of training samples are available, this may make using the hold out method difficult. The number of different hyperparameters, number of values, and combination of values for the hyperparameters that will be tested also have an impact. Testing a large number of combinations may take a long time, especially if it is not possible to use parallel computing.

I generally prefer to use *k*-fold cross validation or *k*-fold cross validation with repeats; however, bootstrapping is also a good option. I generally avoid using leave-one-out cross validation as this method tends to be very slow, especially when a large number of hyperparameter combinations are tested and/or a large number of training samples are used.

Although these processes can sound a bit complicated, modern software tools, such as those available in Python and R, make implementing them straightforward.

What is optimized?

k-NN
- *k* (number of neighbors)

Decision Trees
- cp (complexity parameter/level of pruning)

Boosted DT
- Number of iterations/trees

Random Forest
- ntree (number of trees)
- mtry (number of random variables to select from for splitting at each node)

SVM
- C (slack or cost parameter)
- Kernel parameters
  - Gamma (RBF)
  - Epsilon (RBF)
  - Order (Polynomial)

12

What is optimized will depend on the algorithm being investigated. This slide provides some examples of parameters that are often optimized for different algorithms.

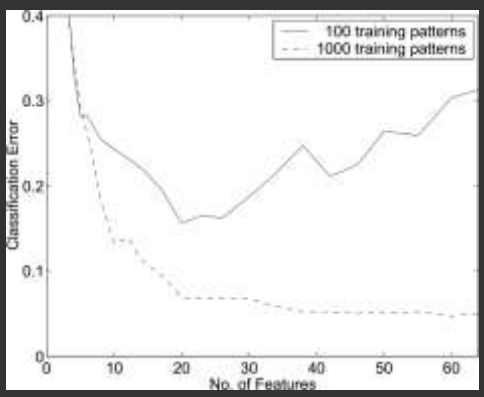
One issue is that some hyperparameters have an infinite number of possible values that could be assessed. For example, the cost parameters for SVM could be set to a variety of values. In contrast, some settings have a more finite set of options. For example, *k* for the *k*NN algorithm is generally tested for a subset of neighbors, maybe varying for 3 to 21, for example. The *mtry* parameter for Random Forest can be 1 through the number of available predictor variables.

Summary of Key Points: Hyperparameters

- Hyperparameters represent user-defined parameters that may impact the performance of machine learning methods.
- It is generally best to tune hyperparameters by testing different values or combinations of values.
- Different algorithms will have different hyperparameters to tune.
- Several methods are available to tune hyperparameters and create data partitions.

Variable Selection

Developing Predictor Variables



Jain, Anil K., Robert P. W. Duin, and Jianchang Mao. "Statistical pattern recognition: A review." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.1 (2000): 4-37.

Hughes Phenomenon

Hughes, G.F. 1968. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory* 14 (1): 55–63. doi:10.1109/TIT.1968.1054102.

"Curse of dimensionality"

Modified from a slide created by Tim Warner

15

As discussed in a prior module, including a large number of predictor variables may cause a decrease in model performance. This is because increasing the number of predictor variables increases the model complexity. So, even though more information is provided, the problem most be solved in a more complex feature space. This issue tends to be more of a concern when only a limited number of training samples are provided. This issue is known as the Hughes Phenomenon or the "curse of dimensionality."

As a result of this issue, you may want to limit the number of variables included in the model. You may also choose to limit the number of features to simplify the model.

Feature Selection vs. Feature Reduction

- Feature Selection = pick from the currently available features
- Feature Reduction = create a new set of features from original features

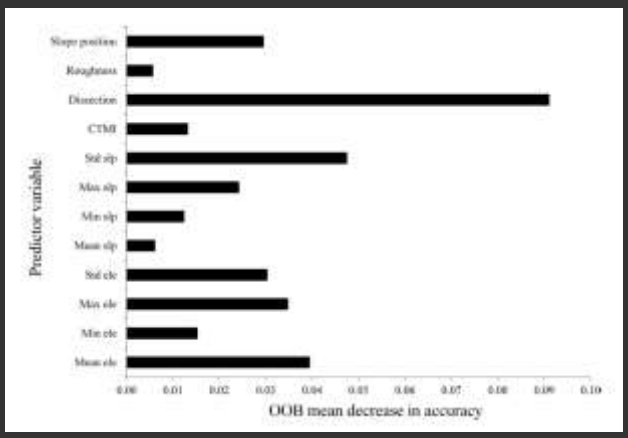
16

There are two broad groups of methods for limiting the number of variables to include in a model. One option is to select a subset of features from the larger set of available predictor variables. This is known as feature selection.

Another option is to create new variables from the raw input predictor variables. This is known as feature reduction.

Variable Selection with Variable Importance

- Randomly permute predictor variable of interest
- Maintain all other variables
- Assess decrease in accuracy for predicting out-of-bag data



Maxwell, A.E., T.A. Warner, and M.P. Strager, 2016. Predicting palustrine wetland probability using random forest machine learning and digital elevation data-derived terrain variables, *Photogrammetric Engineering & Remote Sensing*.

17

If you would like to perform feature selection, how do you decide which variables to keep in the model? Optimally, you would like to maintain the variables that are most associated with the variable of interest or that contribute the most to model performance.

There are a wide variety of methods to accomplish this. Here, I will discuss variable importance estimates based on the Random Forest algorithm.

As mentioned in a prior module, Random Forest is an ensemble decision tree method that uses a subset of the training data to grow each tree and can only select from a subset of the predictor variables at each decision node. Since some of the data are not used in each tree, these out-of-bag (OOB) samples can be use for model validation. They can also be used to assess variable importance.

In order to assess variable importance, the variable of interest is randomly permutated and the resulting decrease in accuracy for predicting the OBB samples is assessed. A greater reduction in model performance with the random permutation indicates that the predictor variable is important in the model. The idea behind randomly permutating the data, which is essentially rearranging the predictor variable values so that they are no longer matched with the correct sample, is to remove the correlation or relationship between

the predictor variable of interest and the dependent variable.

This measure of variable importance estimation is termed OOB Mean Decrease in Accuracy, and greater decreases indicate that the variable is of importance.

Feature Selection using Random Forests

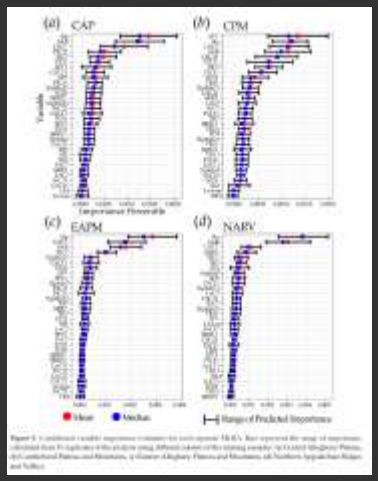
- Based on variable importance as predicted by random forests
- Considers parsimony
- After Murphy et al. (2010)

Murphy, M.A., Evans, J.S. and Storfer, A., 2010. Quantifying Bufo boreas connectivity in Yellowstone National Park with landscape genetics. *Ecology*, *91*(1), pp.252-261.

18

Murphy et al. (2010) proposed an alteration of this variable importance estimation process that also incorporates a measure of parsimony. Specifically, slight decreases in accuracy with a reduced feature space are allowed in an attempt to reduce the model complexity. So, the optimal set of variables depends on both model performance and model complexity. The best variable subset offers a balance between accuracy and complexity.

Variable Importance with RF



- Marginal vs. Partial
- With no variable correlation:
  - Marginal = Partial
- Traditional RF method:
  - More marginal
- Method after Strobl:
  - Use conditional permutation importance
  - Based on conditional inference trees
  - Considers variable correlation
  - Estimate marginal or partial importance

Strobl, C., Boulesteix, A.L., Zeileis, A. and Hothorn, T., 2007. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC bioinformatics*, *8*(1), pp.1-21.

Strobl, C., Boulesteix, A.L., Kneib, T., Augustin, T. and Zeileis, A., 2008. Conditional variable importance for random forests. *BMC bioinformatics*, *9*(1), pp.1-11.

Debeer, D. and Strobl, C., 2020. Conditional permutation importance revisited. *BMC bioinformatics*, *21*(1), pp.1-30.

19

One key consideration is whether you are interested in a marginal or a partial measure of importance. Marginal measures of importance assess the importance of the variable for predicting the dependent variable without considering the other variables in the model. In contrast, partial importance assesses the added value of the variable given the other variables in the model. If there is no correlation between predictor variables, which is a rare occurrence, then marginal and partial importance are equivalent.

Many estimates of importance are not a true measure of marginal or partial importance but are somewhere between these two end members. However, the traditional RF-based variable importance estimation method is generally considered a more marginal estimate of importance.

In a series of studies, Strobl et al. proposed a variable importance estimation method based on conditional permutation importance and conditional inference trees. By considering correlation between predictor variables in the importance estimation process, it is possible to derive a less biased estimate of variable importance and specifically obtain estimates of marginal and partial importance. Also, these methods alleviate some additional biases associated with RF-based variable importance estimation resulting from variable correlation, inclusion of both numeric and nominal predictor variables, inclusion of nominal variables with different numbers of levels, and inclusion

of numeric predictor variables at varying scales.

The methods after Strobl et. al. are my go-to method for variable importance estimation and have been implemented in R via the permimp package.

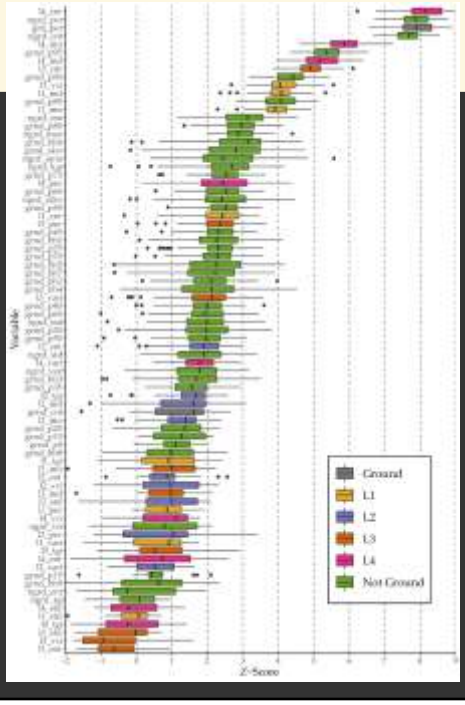
Boruta

Kursa, M.B. and Rudnicki, W.R., 2010. Feature selection with the Boruta package. *J Stat Softw*, *36*(11), pp.1-13.

https://cran.r-project.org/web/packages/Boruta/index.html

Package 'Boruta'

May 21, 2020

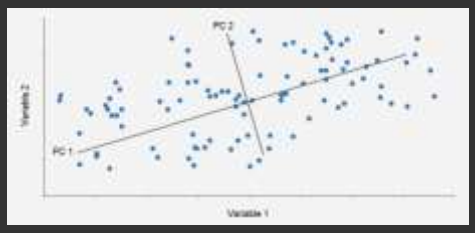


20

Yet another method for variable importance estimation using RF is the Boruta method. This method attempts to determine what features are relevant to the predictive task, as opposed to selecting a minimal optimal set of variables, as is common in other RF-based recursive feature elimination methods. It is a wrapper method that uses variable importance, as calculated by the RF algorithm, to assess the relevance of features relative to randomly generated "shadow variables." The result is a distribution of Z-scores for each variable and a categorization of features as "important", "unimportant", or "tentatively important." This method has been implemented in R within the Boruta package.

Principal Component Analysis (PCA)

- Orthogonal transformation
- Converts *n* correlated variables into *n* uncorrelated variables
- Idea is that variance is information
- Generally, variability in data is described in a smaller number of principal components, thus reducing the number of needed variables



https://www.statistixl.com/features/principal-components/

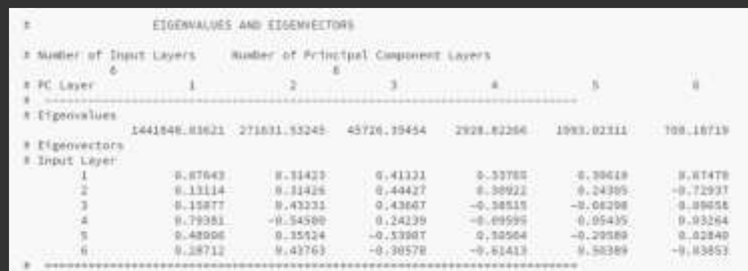
21

As an example of a feature reduction method, we will now discuss principal component analysis (PCA). This is just one example of a feature reduction method. However, it is commonly used.

One common issue with data is that the different features or variables tend to be correlated with each other. This results in redundant information based on the assumption that variability is information. So, it might be desirable to generate new, uncorrelated variables from the original features that are not correlated and, thus, capture the information content available in the original data in a smaller number of variables. This is the purpose of principal component analysis, or PCA.

Principal Component Analysis (PCA)

- Eigenvalues = relates to variance explained by each principal component
- Eigenvectors = Values needed to create principal components from a linear combination of the input variables



EIGENVALUES AND EIGENVECTORS

Number of Input Layers: 6 — Number of Principal Component Layers: 6

PC Layer	1	2	3	4	5	6
Eigenvalues	1441846.83621	271831.53245	45726.19454	2928.82266	1993.02311	708.18719
Eigenvectors / Input Layer						
1	0.07643	0.31423	0.41331	0.53785	0.30618	0.07470
2	0.13114	0.31426	0.44427	0.30922	0.24305	-0.72937
3	0.15877	0.43231	0.43667	-0.38515	-0.08298	0.09658
4	0.79381	-0.54500	0.24239	-0.09595	0.05435	0.93264
5	0.48906	0.35524	-0.53907	0.50564	-0.29589	0.02840
6	0.28712	0.43763	-0.30578	-0.61413	0.50389	-0.03853

PC1 = (0.076*Var1) + (0.131*Var2) + (0.159*Var3) + (0.794*Var4) + (0.489*Var5) + (0.287*Var6)

22

As discussed in previous modules, the process of principal component analysis involves the transformation of the raw variables into a new space in which the resulting features, or principal components, are not correlated. Calculating PCA involves some matrix algebra and the use of a covariance matrix. A mathematical proof of this concept is beyond the scope of this course. The key concept here is that PCA allows for the determination of Eigenvectors, which are effectively coefficients that can be applied to each variable to obtain a new variable or principal component that explains a certain proportion of the variance in the raw input data. The relative amount of variance explained by a single principal component is associated with its Eigenvalue. Larger Eigenvalues indicate a larger proportion of variance explained.

For example, in the provided table, 6 input variables are being used to obtain 6 uncorrelated, new principal component variables. To obtain the first principal component specifically, the following equation will be applied:

PC1 = (V1 * 0.07643) + (V2 * 0.13114) + (V3 * 0.15877) + (V4 * 0.79381) + (V5 * 0.48906) + (V6 * 0.28712).

Other principal component variables can be calculated using their associated coefficients.

In summary, the process of PCA allows for the determination of Eigenvectors which are used to calculate principal components from a linear combination of the input variables.

Principal Component Analysis (PCA)

- Generally, the first principal component explains the most variability in the data
- Explained variability decreases with increased PC
- Can visualize with a Scree Plot

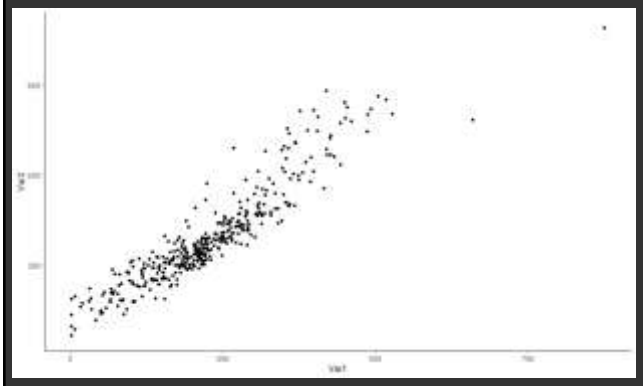
```
#                PERCENT AND ACCUMULATIVE EIGENVALUES

# PC Layer   EigenValue   Percent of EigenValues   Accumulative of EigenValues
        1 1441848.03621            81.6987                  81.6987
        2  271631.53245            15.3813                  97.0800
        3   45726.35454             2.5910                  99.6810
        4    2928.82266             0.1660                  99.8469
        5    1993.92311             0.1129                  99.9599
        6     708.18719             0.0401                 100.0000
#  ------------------------------------------------------------------------------
```

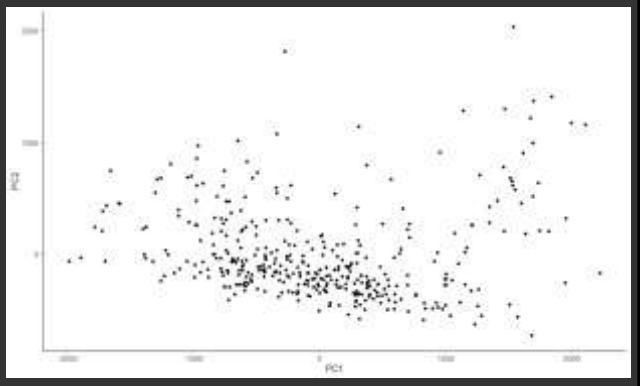
23

Again, one of the key uses of PCA is to reduce the dimensionality of your input data by capturing the information content, or variability, in a smaller number of features.

The Scree Plot provides a means to visualize the explained variance. Again, larger Eigenvalues indicate more explained variance. In the provided example, PC1 captures 81.7% of the variance, PC2 captures 15.4%, and PC3 captures 2.3%. Cumulatively, PC1 through PC3 capture 99.7% of the variance. So, six variables are reduced to three with only a slight loss in information content.

Principal Component Analysis (PCA)





	Var1	Var2	Var3	Var4	Var5	Var6
Var1	1.0000000	0.9266122	0.9199133	0.6217669	0.7892767	0.7819981
Var2	0.9266122	1.0000000	0.8871722	0.8069255	0.7784212	0.7330903
Var3	0.9199133	0.8871722	1.0000000	0.5707753	0.9069602	0.8962260
Var4	0.6217669	0.8069255	0.5707753	1.0000000	0.5598671	0.4448386
Var5	0.7892767	0.7784212	0.9069602	0.5598671	1.0000000	0.9748336
Var6	0.7819981	0.7330903	0.8962260	0.4448386	0.9748336	1.0000000

	PC1	PC2	PC3	PC4	PC5	PC6
PC1	1.000000e+00	4.486604e-16	3.936256e-16	-2.255816e-16	4.090028e-15	2.452804e-15
PC2	4.486604e-16	1.000000e+00	4.332943e-16	4.056150e-15	3.865131e-15	2.201871e-15
PC3	3.936256e-16	4.332943e-16	1.000000e+00	-1.702617e-15	-2.813617e-16	4.994810e-16
PC4	-2.255816e-16	4.056150e-15	-1.702617e-15	1.000000e+00	-2.529040e-15	4.064909e-16
PC5	4.090028e-15	3.865131e-15	-2.813617e-16	-2.529040e-15	1.000000e+00	-3.099247e-16
PC6	2.452804e-15	2.201871e-15	4.994810e-16	4.064909e-16	-3.099247e-16	1.000000e+00

24

The two graphs on this page demonstrate the concept of decorrelation. The graph on the left shows the relationship between Variable 1 and Variable 2, which have a clear correlation. In the correlation matrix, the Pearson Correlation Coefficient for these two variables 0.923. So, there is significant correlation.

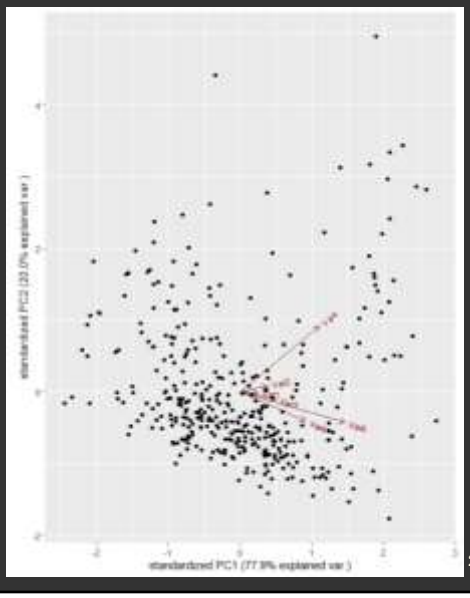
In the graph on the left, I am graphing the first and second principal component. The correlation between these variables is effectively zero, ignoring some rounding error.

So, we are capturing variability while removing or minimizing redundant information.

Principal Component Analysis (PCA)

```
Importance of components:
                          PC1      PC2      PC3      PC4      PC5      PC6
Standard deviation     809.5542 410.7705 112.00948 56.15168 34.79928 25.56569
Proportion of Variance   0.7787   0.2005   0.01491  0.00375  0.00144  0.00078
Cumulative Proportion    0.7787   0.9791   0.99404  0.99778  0.99922  1.00000
```

```
Standard deviations (1, .., p=6):
[1] 809.55420 410.77054 112.00948  56.15168  34.79928  25.56569

Rotation (n x k) = (6 x 6):
             PC1          PC2          PC3          PC4          PC5          PC6
Var1 0.1146256 -0.007213278 -0.466466721  0.05201695 -0.4481221  0.752129368
Var2 0.1781144  0.079202334 -0.526378320  0.04057881 -0.5034498 -0.655606449
Var3 0.2327137 -0.095578242 -0.616236444 -0.31273043  0.6775808  0.006764803
Var4 0.5108610  0.836558322  0.129088331  0.09049894  0.1010786  0.064191154
Var5 0.6790373 -0.388023328  0.330032766 -0.48445177 -0.2114380  0.005005776
Var6 0.4229775 -0.366261969  0.001372416  0.80929827  0.1780066 -0.017037424
```



25

This graph shows a plot of the first two principal components. The red lines indicate the original variables. The direction of the red line is associated with the correlation between the variable and the two principal components whereas the length of the line relates to the magnitude of correlation.

For example, based on the Eigenvectors, all six variables are positively correlated with the first principal component. That is why all the red lines are pointing to the right. In contrast, some of the original variables are positively correlated with the second principal component while others are negatively correlated. This is why some of the lines point downward and some point upward.

In the graph, each point represents one record or data point from the original dataset mapped to the principal component values.

Kernel Principal Component Analysis (KPCA)

- PCA is limited to linear transformations
- KPCA allows for nonlinear combinations
- Project data to higher dimensional space (kernel trick)

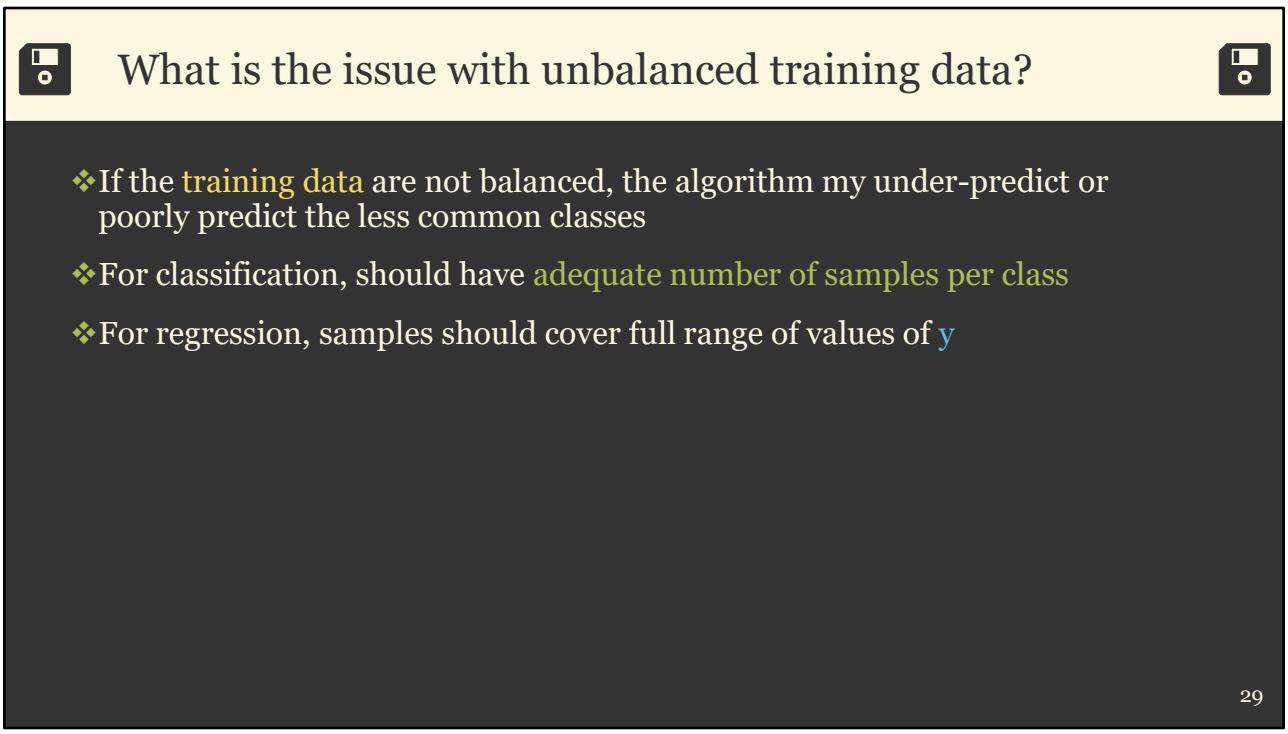
26

It should be noted that there are some augmented versions of principal component analysis. One option is kernel principal component analysis (KPCA). One issue with principal component analysis is that it only allows for a linear transformation. In order to allow for the generation of nonlinear combinations of the predictor variables, they can first be projected to a higher dimensional space. This is the same "kernel trick" applied within Support Vector Machines that allows for more accurately separated classes that are not linearly separable in the input feature space.

Summary of Key Points: Variable Selection

- Including a large number of predictor variables can increase model complexity and reduce model accuracy.
- A subset of variables may be desired to reduce model complexity and increase parsimony even if the large feature space does not negatively impact model performance.
- Feature selection methods select a subset of the existing variables to include in the model.
- Feature reduction methods create new variables from the original predictor variables.
- Principal component analysis is one means to create new variables from input variables. The resulting principal components will be decorrelated, linear combinations of the input variables. Principal component analysis is based on the assumption that variability represent information.

Balancing the Training Data



One common issue in predictive modeling is that the training data may not be balanced or there may be many more samples or examples of some class(es) in the training set as opposed to other classes. This commonly arises because the classes are not balanced in the actual population. For example, the majority of credit card transactions are legitimate. Thus, the proportion of fraudulent transactions are inherently low. This can result in an issue when trying to generate a model to predict the likelihood that a transaction is fraudulent. For the sake of argument, let's say that the rate of fraudulent transactions is around 0.001% (this is probably too high). In this situation, a model could just call all samples legitimate and obtain an overall accuracy well above 99%. However, this model would not be useful.

Generally, when the training dataset is imbalanced the model tends to under-predict or poorly predict the less common class or classes. This problem is generally worse with an increasing level of imbalance. Unfortunately, it is not always easy to obtain a large number of samples from the minority class(es) if they are inherently rare in the population.

This issue of imbalance can also be an issue for regression problems when some ranges of values for the dependent variable are not common in the population and not well represented in the training set.

How can we deal with unbalanced training data?

- Use stratified random sampling
  - Strata defined by classes (classification)
  - Strata defined by ranges/bins of y (regression)
- Randomly under sample majority class(es)
  - Will lower the number of available samples
  - May need large dataset
- Duplicate minority class examples
  - Minority class(es) samples are use in the model multiple times
- Produce synthetic minority samples
  - Can be combined with under sampling of majority class(es)
  - Synthetic data are generated that are similar to existing samples

30

A few methods have been proposed to deal with imbalanced training sets. One option is to collect a sample using stratified random sampling as opposed to simple random sampling. This can allow for generating a training set that is artificially balanced as opposed to representing the true proportions in the population. However, this is not always easy to do, especially when one or more classes, or certain ranges of y for regression problems, are inherently rare.

Another option is to under sample the majority class during the training process. Unfortunately, this would require throwing out some of the training data, leading to training on a smaller training set. Alternatively, you could duplicate the minority samples or use them multiple times in the training process. This has the benefit of allowing you to use a larger training set.

Instead of just duplicating the minority samples, you may decide to generate synthetic samples for the minority class that are similar to the existing samples but with some added noise. One example is the SMOTE (Synthetic Minority Oversampling Technique) method that uses a combination of under sampling the majority class and generating synthetic samples of the minority class.

I generally prefer to duplicate the minority samples to potentially help combat the impact of an imbalanced training set. However, others may disagree with

this approach. I have found that such methods may not always yield improvements and that the impact is often case specific. Such methods may not drastically improve overall accuracy but may specifically improve the performance for the minority class or classes. Again, this is likely case specific and may involve some experimentation.

SMOTE

- Synthetic Minority Oversampling Technique
- Instead of simply duplicating existing minority-class observations, SMOTE creates new synthetic samples by interpolating between existing minority-class samples and their nearest minority-class neighbors in the feature space.
- For a selected minority observation, the algorithm identifies nearby minority observations, chooses one or more neighbors, and generates new points along the line segments connecting them.
- This increases the representation of the minority class while adding some variation to the training data.
- Can help a model learn a broader decision region for underrepresented classes, but it should be applied only to the training data to avoid data leakage.

31

This slide further explains the SMOTE method.

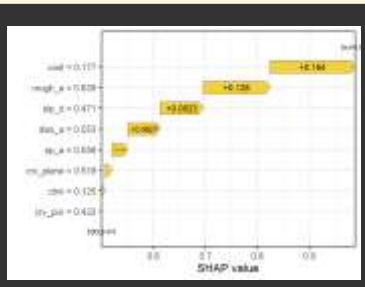
Summary of Key Points: Balancing Training Data

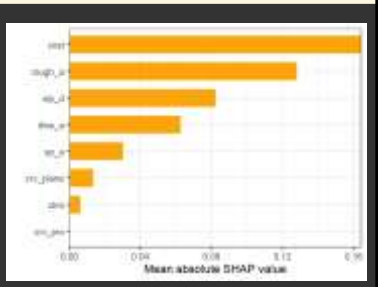
- An imbalance in the number of training samples between classes can have a negative impact on classification accuracy, especially for the minority classes.
- Methods are available to generate a more balanced training set.
- This is a complex problem that is not always easy to combat.

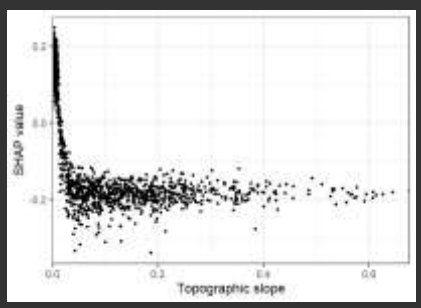
Model Explanations

Shapely

- Shapely Additive exPlanations (SHAP)
- Model-agnostic
- Globally and for individual predictions
- Grounded in cooperative game theory, this method attempts to quantify how the prediction changes when the predictor variable of interest is added to all possible combinations of the other predictor variables in the set







34

Shapley Additive Explanations (SHAP) is a unified framework for interpreting machine learning model predictions, rooted in cooperative game theory, that attributes the contribution of each input feature to a given prediction in a fair and consistent way. The core idea draws from the Shapley value concept, where each feature is treated as a "player" in a coalition game and its contribution is determined by measuring how much the model's output changes when that feature is included versus excluded across all possible subsets of features. Because directly computing this over all feature subsets is computationally expensive, efficient approximation algorithms, such as TreeSHAP for tree-based models, make the approach tractable in practice. The resulting SHAP values for each feature represent the magnitude and direction of that feature's contribution to shifting the model output away from the baseline (expected) prediction, with positive values pushing the prediction higher and negative values pushing it lower.

SHAP values can be used to estimate variable importance both for a single new prediction and on average across the entire dataset, offering complementary perspectives on feature influence. For a specific new observation, the SHAP values directly reveal how each feature contributed to that individual prediction relative to the baseline, making it possible to explain exactly why the model produced a particular output for that case (for instance, identifying that a high value for a given feature drove the prediction substantially upward). For

global variable importance, SHAP values are aggregated across all observations in the dataset, typically by computing the mean absolute SHAP value for each feature, which reflects the average magnitude of each feature's influence on model outputs across the full data distribution. This dual capability makes SHAP particularly powerful, as it bridges local explainability, or understanding a single prediction, and global explainability, or understanding the overall behavior of the model.

Partial dependence plots (PDPs) can be generated using SHAP values to visualize the marginal relationship between a feature and the model output while accounting for feature interactions. In the SHAP-based approach, the SHAP value for a given feature is plotted against that feature's actual observed values across all instances in the dataset, producing what is often called a SHAP dependence plot. Unlike traditional PDPs, which marginalize over all other features by averaging predictions across the data, SHAP dependence plots display the actual contribution of the feature for each individual observation, preserving instance-level variability rather than collapsing it into a single average trend line.

This is the end of this lecture module.

Please return to the West Virginia View Webpage for additional content.

Thanks! Hope you found this useful.