62. Miscellaneous

Other smoothing techniques that I have come across but haven't looked into are: Continuum Removal, Discrete wavelet transform, Triangular (Barlett) Window, Kernel Smoothers, Discrete Wavelet Transforms, and Smoothing Splines.[xiv] [xv]

Time Series Data

63. Date-Time Features

We briefly touched upon this in the example, but this works mainly with time series data. We can engineer a timestamp into other features, such as the hours, minute, second, day of week, season, time converted to Eastern time, business hours or not, public holiday or not, etc. You can also calculate the time difference between two dates, such as the number of hours or months after a public holiday or time between two transactions.

The list is quite extensive and it really depends on the problem you are dealing with.

For example, if your problem is trying to identify fraud at an ATM, the time of day might be important (since there might be a higher chance of fraud happening between 12AM-5AM) but whether or not it's a leap year might not be. Another example might be if you are trying to predict if someone will attend an event, a feature you could construct is the time to event relative to when the person knew about the event (e.g. time of event - time when event was first posted about).[xvi]

64. Lag Features

Lag features are mainly used in time series data. We can introduce data from prior time steps into the current time step. If you are familiar with ARIMA, then this is the same concept. If you are using Python, you can use Pandas' shift() function to achieve this. Below is a picture to demonstrate creating lag features:

Timestamp	Value	Lag_1	Lag_2
1/1/2020	5	NaN	NaN
1/2/2020	4	5	NaN
1/3/2020	3	4	5
1/4/2020	2	3	4
1/5/2020	1	2	3

65. Window Statistics

If we have time-series data, we can generate window statistics, such as the mean temperature for the last two weeks or the spread between the highest and lowest point.

Categorical Data

66. Frequency Encoding

This technique is used for transforming categorical variables into numerical values. It uses the frequency of the categories as a new feature and might help the model understand the relationship between the frequency and the target variable (if one exists).

For example, if we have the following data: [Mac OS, Windows, Mac OS, Linux, Linux, Mac OS]. The frequency of Mac OS is ½, Windows ⅙, and Linux ⅓, a new feature will be created with the frequencies as values: [½, ⅙, ½, ⅓, ⅓, ½], or [0.5, 0.17, 0.5, 0.33, 0.33, 0.5].

67. Integer / Label Encoding

This is a simple way of transforming categorical data into numerical values. You just assign each category an unique integer (e.g. "cat" → 1, "dog"→ 2, "T-Rex" → 3). However, if you do this, you are telling the algorithm that there might exist a natural ordered relationship between the categories but might not make sense if no order exists. An example of an order that makes sense might be tax brackets or placings in a race.

68. One-Hot Encoding

Another technique for transforming categorical data into numerical. You can transform data with *k* categories into a numerical vector of length *k* containing only one 1 and the rest 0s. For example, suppose we have an Animal feature with three categories:

ANIMAL	ANIMAL_CAT	ANIMAL_DOG	ANIMAL_T-REX
CAT	1	0	0
DOG	0	1	0
T-REX	0	0	1
DOG	0	1	0

69. Categorizing Locations

If you have location data, you can consider converting cities to latitude/longitude or adding zip codes to street names.

Text Data

70. Bag-of-Words

In this case, text data is converted into counts of the words (e.g. "the dog jumped over the other dog" → {"dog": 2, "the": 2, "jumped": 1, "over": 1, "other": 1}).

71. Bag of n-Grams

An n-gram is a continuous set of n items from a text, where items can be phenomes, syllables, letters, or words.

For example, let's take the bi-gram or 2-gram of the following string: "the red dog jumped over the other red dog" → {"the red": 1, "red dog": 2, "dog jumped": 1, "jumped over": 1, "over the": 1, "the other": 1, "other red": 1}).

72. Character n-grams

N-grams can be created on words, but also on characters as well.

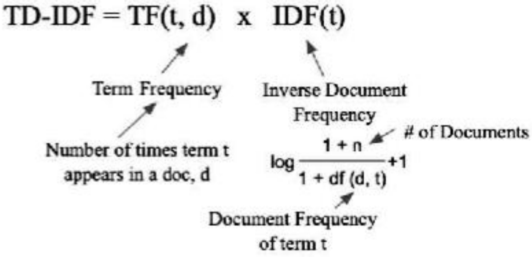
For example, let's take the tri-gram or 3-gram of the following string: "the red dog jumped over the other red dog" → {the, he_, jum, ped, … oth, red, dog}.

73. Parts-of-Speech Tagging

If we are dealing with text data, we can use Python's NLTK package to tag the parts-of-speech of each word (nouns, verbs, adjectives, etc.). Once we have that data, we can create features for the total number of nouns, verbs, and adjectives. Maybe there's a correlation between those who use more verbs to purchase a certain selection of items.

74. Term-Frequency-Inverse Document Frequency (td-idf)

This is similar to bag-of-words but instead of taking the raw counts, each raw count is multiplied by the number of documents in the dataset divided by the number of documents the word appears in.


TD-IDF = TF(t, d) x IDF(t)
Term Frequency
Inverse Document Frequency
Number of times term t appears in a doc, d
1 + n
# of Documents
log
+1
1 + df (d, t)
Document Frequency of term t

75. Frequency-Based Filtering

We can also filter the text data to remove noise, such as common words (which really don't add that much value to the sentence such as "and", "the", or "a") and rare words (such as "Meeseek"). We can rank the words by frequency or how often they appear and then remove those that meet a certain threshold. It might be good to cross-reference the common words with a stopword list, or a list of the most common words in a language.

76. Stemming

Stemming is done by cleaning each word down to its basic linguistic word stem form. For example, {"playing", "player", "plays", "played"} can all be turned into "play".

77. Miscellaneous

There's a lot of potential feature engineering (especially in natural language processing) so we're just going to leave them here: lowercasing, spelling correction, encoding punctuation marks, tokenizing, skipgrams, chopping, lemmatization, document features (number of spaces, tabs, newlines, characters, etc.), Word2Vec/GloVe/Doc2Vec (or any 2Vec-type embedding models), string similarity, entity insertion, reading level of document, dimensionality reduction (reduce text to 50 or 100-dimensional vector), sentiment, and topic models.

Image Data

78. Data Augmentation

Given an image dataset, we can augment the data so the model sees the image from different perspectives. A model that can robustly classify objects even if they are placed in different orientations is said to have the property called invariance: invariance to translation, viewpoint, size, or illumination.

To augment the data, we can take these images and transform the images so the model can learn from these variations. Below are a few data augmentation techniques:

1. **Flip** images horizontally and vertically
2. **Rotate** the images by various degrees
3. **Scale** the image inward or outward
4. **Crop** out random sections of the image (cut the object in half)
5. **Translate** the image along the X or Y-axis
6. **Add Noise** by reducing the quality of the photo
7. **Stretch** by stretching the image in one direction
8. **Shear** by stretching the image in two opposite directions (e.g. top → left, bottom → right)
9. **Elastic Deform** by performing a more flexible shear (think kneading a stress ball)
10. **Grayscale** images by turning pictures to black and white

79. Embeddings

Embeddings are essentially techniques to take non-vectorized data (words, images, graphs) and "embed" it into a vectorized space. Why? Because machine learning models work when the training data is presented as vectors and numbers.

For example, suppose we want to take the word "cat" and replace every instance of the word "cat" in a document with a *n*-dimensional vector. Of course, that can easily be done, but the key is that embeddings should be meaningful- if you query the nearest neighbors of "cat", you should get back the vectorized forms of animals like "dog" or "mouse", but certainly not something completely unrelated like "Statue of Liberty" or "Declaration of Independence".

You can use pretrained embeddings (e.g. Wikipedia2Vec or word2Vec-GoogleNews) or train your own embeddings (which require you to obtain, clean, and process your own dataset).

Geographical Data (Latitude and Longitude Features)

80. Manhattan Distance

The distance between two points is measured along the axes at right angles. Also known as city block distance, since it calculates the absolute differences between coordinates.

81. Haversine Distance

Haversine distance calculates the shortest distance between two points on a sphere given their longitudes and latitudes measured along the surface. Important to know and use in navigation; can use skearn.metrics.pairwise.haversine_distances(*X*, *Y*).

82. Create Clusters

If you have data for pickups and dropoffs, for example, we can create clusters to generate new features like the number of trips going in and out of a certain cluster on a certain day.

83. Distance to Locations or Points of Interest

Building upon the distances, we can bring in additional data by calculating the distance between the given coordinates and potentially important locations (major cities or towns, nearby businesses, etc.).

84. Travel Speed

If given timestamps with location data, you can calculate how fast an ATM user traveled between two ATMs. If the travel speed is impossible (e.g. one transaction took place in California and 10 minutes later in London), it is probably is an indication of fraudulent activity.

Feature Selection

Sometimes we have too many features to deal with, so we need some methods to reduce the number of features down a reasonable size. Additionally, and more importantly, we want to remove features that have no predictive power and are redundant.

85. Forward Feature Selection (using single or groups of features)

We start with zero features and train a model for every feature and compute a metric value on the cross-validation set. We choose the best feature (based on the metric value) and add it to our list of features.

We then train a model with two features (with the one already in the feature list) and choose the best feature to add to the list. We continue this process until we have *n* features (which is determined by you). You can also think of *n* as a hyperparameter that you can tune.

86. Recursive Feature Elimination (using single or groups of features)

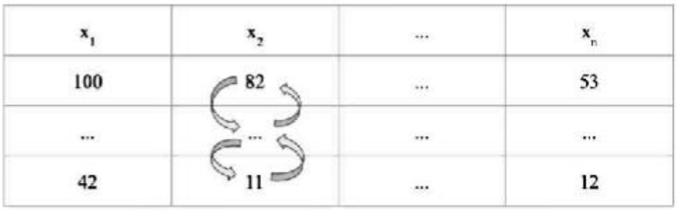
This is just the opposite of forward selection. We start with all the features and train a model. The least significant feature (determined by some p-value or coefficient) is removed and the process repeats until we have *n* features.

89. Permutation Importance

This is another way of determining feature importance. One main difference is that this method is applied after we have trained a model.

Now, we take the validation dataset and take one column and shuffle the values within that one column. Using this dataset with one column shuffled, we make predictions and compare it with the true values to determine performance deterioration. We then undo the shuffled values and repeat this for every feature column.

The main idea is that if we shuffled the more important feature columns, the performance would be much worse than if we shuffled a non-important feature column.



x_1	x_2	...	x_n
100	82	...	53
...	...	...	...
42	11	...	12

89. Correlation Analysis

One technique to reduce features is to use correlation analysis to remove highly correlated features (multicollinearity). Generally, correlated features will not improve your model (or worsen them) but may affect certain models in different ways. For example, correlated features in linear regression may result in solutions that are numerically instable and uninterpretable, since the interpretation of a regression coefficient is that it provides an estimate of the effect of one unit change in an independent variable X_1. However, if X_1 and X_2 are correlated, there's no way of determining the effect of changes in X_1 on Y.

So, to perform feature selection, if we find that two features are highly correlated (say 0.95), we can drop one of the two features.

89. Time Consistency

If we have a time series dataset, we can train a single model using a single feature (or small subset of features) on the first month (or whatever the first time period you define as) of the train dataset and use the model to predict the output variable of the last month (again whatever you define the last as) of the train set.

Compare the training score (first month) and validation score (last month). Ideally, the score should be roughly the same, but if it is lower in the validation, it indicates that the feature is not consistent over time. Remove the feature and see if the removal improves your overall score.

90. Feature Trimming

Introduce a dummy feature with random values into the training set, train a model and calculate feature importance, and then remove features with importance below the dummy feature.[xvii]

91. Unsupervised Feature Selection

There is a class of feature selection methods that are unsupervised. One technique is to select features based on identifying similarity between features and remove the redundancies.[xviii]

92. Boruta

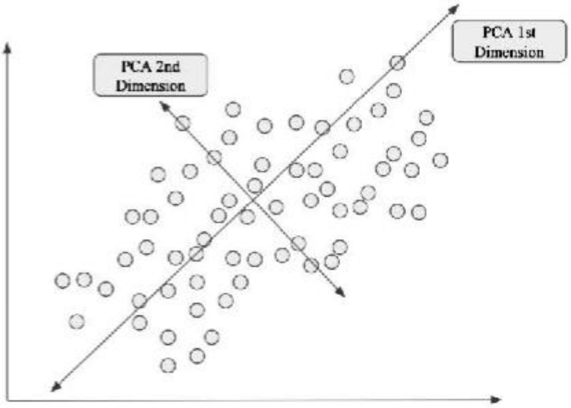
Feature selection idea based on two main ideas:

1) Each feature competes against a randomized version of itself.
    a) A shadow training dataset is created by randomly shuffling each feature. This shadow dataset is attached to the original dataset and the combined dataset is fed to the model to calculate feature importance- a feature is only good if it beats the shadow clone of itself.
2) We then repeat step 1 multiple times until we have a statistically reliable average of what's important or not.[xix]

93. PCA

PCA allows us to summarize a set of correlated features with a smaller set of independent features that collectively explain most of the data in the original set (or explains most of the variability); the idea of variability is important because higher variance along a dimension (or feature) means that more information is contained, which further implies that the dimension is more important.

Essentially, all we are doing is "dropping" the least important features by creating new, independent features in a lower dimensional space by taking the linear combination of the original features. The downside of such an approach is that the resulting independent variables are now less interpretable.



Note that PCA is not a feature selection technique.

Handling Imbalanced Datasets

Sometimes the data classes in training data are extremely imbalanced. For every 100,000 instances of class A, there might only be 1 instance of class B.

94. Appropriate Performance Metric

Using accuracy when your dataset is imbalanced is highly misleading. For example, if your goal is to classify fraudulent transactions, only ~0.5% of transactions are truly fraudulent. So, if you have a model that always predicts not_fraud, you would get accuracy of 99.5%. That might seem great, but in reality, your model is missing 100% of the fraudulent transactions.

Some clarification before diving in:

- True Positive (TP): outcome where model correctly predicts the positive class
- True Negative (TN): outcome where the model correctly predicts the negative class
- False Positive (FP): outcome where the model incorrectly predicts the positive class
- False Negative (FN): outcome where the model incorrectly predicts the negative class

- True Positive Rate (TPR): TP / (TP + FP). Closer to 1, the better.
- True Negative Rate (TNR): TN / (TN + FN). Closer to 1, the better.
- False Positive Rate (FPR): FP / (FP + TN). Closer to 0, the better.
- False Negative Rate (FNR): FN / (FN + TP). Closer to 0, the better.

Some appropriate metrics to combat imbalanced data:

1) Confusion Matrix: table that shows the correct predictions and the types of incorrect predictions
2) Precision: what proportion of positive identifications was actually correct? TP / (TP + FP)
3) Recall: what proportion of actual positives was identified correctly? TP / (TP + FN)
4) F1 Score: harmonic mean of precision and recall; $2 * \frac{precision*recall}{precision+recall}$
5) G-Measure: geometric mean of precision and recall
6) Jaccard Score: size of the intersection divided by the size of the union of two label sets
7) AUC Score: area under the receiver operating characteristic (ROC) curve. The ideal model would have an AUC of 1 (a random algorithm has an AUC of 0.5).

95. Appropriate Algorithm

The choice of algorithm can help you perform better on imbalanced datasets. Tree-based algorithms work well: decision trees, random forest, and XGBoost.

96. Class/Sample Weights or Cost-Sensitive Learning

You can assign weights to minority data points so that they are penalized more when the algorithm classifies them wrong. In other words, we are forcing the algorithm to learn that getting the minority points wrong "costs" more so it pays special attention to them.

97. Oversample the Minority Class

We can split the data into train and test (make sure to split first because test samples might be sampled as train samples, which allows the model to just memorize that data point) and then randomly sample the minority class until we have an equal number between the classes.

In other words, our minority class will have duplicates.

98. Undersample the Majority Class

The opposite of oversampling, so we are removing data from the majority class until we have an equal number between classes. Potential downside of this is that we might lose valuable information in the samples that were removed and we have a smaller dataset to train on.

99. Generate Synthetic Samples (SMOTE)

Using SMOTE, we can synthetically generate samples of the minority class by using a nearest neighbor algorithm. We can think of SMOTE as drawing a line between 2 points in the minority class and then randomly selecting a point along that line. That way, our synthetic minority data points will be similar to the real minority data points.

Section IV: Validation Strategy

Designing a validation strategy is often an overlooked part of training a machine learning model. If done correctly, the proper validation strategy not only helps you understand your model and how it's doing, but also gives you an idea of how it will perform on future, unseen data. There isn't a single validation technique that will work in all scenarios so it is important to understand the problem at-hand and choose an appropriate one.

100. Classic Train-Test Split

The idea is simple; you randomly split your training set into two sets: train and test. Why? Because you want to see how well your model performs on unseen data. The ratio varies but 70%-80% for the train set and 20-30% for the test set is a good starting point.

You train your model on the train set and then generate predictions for the test set to see how well it did. If your error for the test set is much higher than your train set error, then you have probably overfit.

One caveat of this approach is that if you sampled the data with some bias, your entire model will be off. For example, if you were predicting fruits and your train set doesn't have any data points about apples but your test set was full of apples, then your model (and test score) will be biased and perform poorly.

Therefore, it is important to eliminate these biases and make sure the train and test set come from the same distribution.

101. Train-Validation-Test Split

This is just like train-test split but with one extra step. We designate another chunk of the data to a validation set. Here, after training our model, we can tune the hyperparameters of our model. We can also obtain an unbiased evaluation of the model while tuning the model's hyperparameters.

The test set is finally used to estimate a generalization error of the model. However, again we suffer if the distribution of the train, validation, and test sets are different.

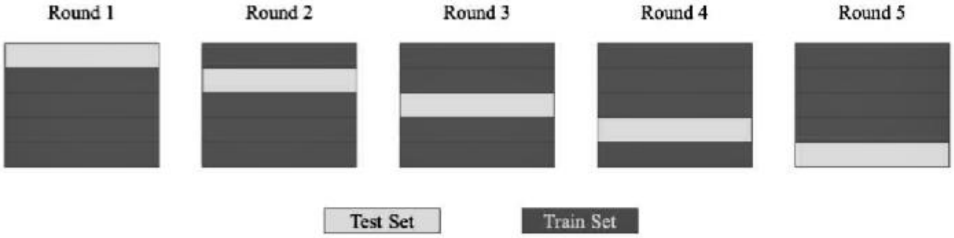
102. *k*-fold Cross-Validation

k-fold is different from train-test split in that instead of taking one split, we take many splits (*k* - 1). Now, we have *k* groups or folds. For each group, leave that group as the test data set and use the remaining groups as a train set. Train a model on the train set and record the score on the test set.

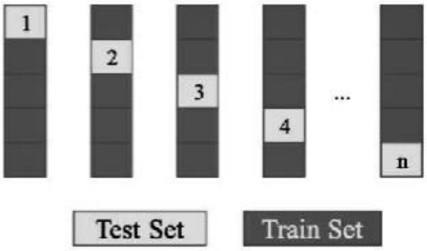
Once we have *k* scores, we can take the average of scores to see how good the model is, or take the variance of the scores to see how stable the scores are.

This validation technique allows each data point to be in the test set once and the train set *k*-1 times. This is a popular method because it gives us more estimates of a model's accuracy, which leads to a less biased estimate and a better idea of how our model performs that a simple train-test split.

There is no right way to choose *k* (the number of folds), but a good rule of thumb is between 5 and 10. At the end of the day, it is a trade-off between computational complexity (too high of a *k* means more folds, which leads to more training time) and accuracy (too small of a *k* brings us closer to train-test split). This is another instance of the bias-variance tradeoff.



Note that leave-one-out-cross-validation (LOOCV) is a special case of k-fold where $k = n$ (the number of points in the dataset). In other words, every iteration we leave one data points out and train the model on n - 1 data points.

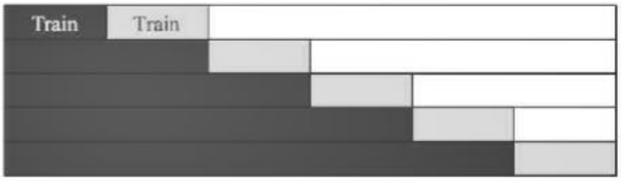


103. Stratified *k*-fold

A variation of *k*-fold where each fold contains around the same percentage of samples of each target class. This way, we can ensure that the distribution between the folds remains approximately the same.

104. Time Series Validation

Time series data is trickier to cross-validate, since when you split your dataset, you want to make sure that the data remains in chronological order. The data that your model trains on should occur before the test data. We can accomplish this using *k*-fold cross validation but ensuring that the training data occurs before the test data.



105. Adversarial Validation

One problem in data science is that the test set you have may be very different from the training set. As a result, it is hard to create an appropriate validation set to test our model's ability to generalize to unseen data.

So, the main idea behind adversarial validation is to train a classifier (logistic regression or random forest) to classify between train and test data. Ideally, if the train and test data are indistinguishable, then the ROC-AUC should be 0.5 (random). In this ideal case, usual validation methods should work *k*-fold cross validation

However, if the train and test sets do differ, we want to choose the data that look most like the test data to be our validation data. So, we run a classifier to get probabilistic predictions for all training data points, see which data points are misclassified as test, and then use the points with the highest probability of being misclassified as validation.

Hyperparameter Tuning

Hyperparameters in machine learning models refer to the parameters which define the model's architecture. For example, the depth or number of trees in a random forest, or the number or neurons/layers a neural network has are all hyperparameters. In other words, they are parameters that are not learned from the data but instead are chosen when we optimize a loss function. We train a model using various hyperparameters, evaluate the results of each iteration, and at the end, choose the model with the best score.

106. Exhaustive Grid Search

Simply build a model for every possible combination all of the hyperparameter values.

107. Random Search

Search over random combinations of all the hyperparameter values by sampling each hyperparameter value from a distribution. This is advantageous over exhaustive search since the run time is much lower but may result in slightly worse performance due to higher variance.

108. Bayesian Optimization

Exhaustive grid search and random search trained the models in isolation and independently. In this case, we use the information from one iteration to inform the next iteration. We use the previous hyperparameters to calculate a posterior expectation of the hyperparameter space and then choose the optimal hyperparameters according to the posterior expectation for the next iteration until we converge to an optimum.[xx]

Final Note

For those who just want to do well in Kaggle competitions, the techniques outlined above should give you a leg up over the majority of the competitors. Even so, there's no substitute to spending the time to go over the winning teams' solutions and their code.

However, doing well in Kaggle shouldn't be the end goal, since modeling is only one component of what data science is. Kaggle competitions ignore the parts where *you* ask the interesting questions, source/clean the data yourself, and write up the results.

To be a well-rounded data scientist (and problem-solver), you should work on developing skills in all parts of the data science lifecycle by continuously investing in life-long learning; new techniques and tools will constantly emerge, so it pays to be up-to-date.

Best of luck.

Works Cited

[i] Molnar, Christoph. "Interpretable machine learning. A Guide for Making Black Box Models Explainable", 2019. https://christophm.github.io/interpretable-ml-book/.

[ii] Michailidis, Marios. "Winning Tips on Machine Learning Competitions by Kazanova, Current Kaggle #3 Tutorials & Notes: Machine Learning." *HackerEarth*, www.hackerearth.com/practice/machine-learning/advanced-techniques/winning-tips-machine-learning-competitions-kazanova-current-kaggle-3/tutorial/.

[iii] Veen, Hendrik van Jacob, Armando Segnini and Le Nguyen The Dat. *Kaggle Ensembling Guide*. 2015. <https://mlwave.com/kafmodeggle-ensembling-guide/>.

[v] Amatriain, Xavier, and Justin Basilico. "Netflix Recommendations: Beyond the 5 Stars (Part 1)." *Medium*, Netflix Technology Blog, 18 Apr. 2017, netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429.

[vi] https://www.kaggle.com/c/ieee-fraud-detection/discussion

[vii] "Microsoft Malware Classification Challenge (BIG 2015)." *Kaggle*, www.kaggle.com/c/malware-classification/discussion/13509.

[viii] "Santander Customer Transaction Prediction." *Kaggle*, www.kaggle.com/c/santander-customer-transaction-prediction/discussion/88939.

[ix] Gutierrez, Daniel. "Ask a Data Scientist: Data Leakage." *InsideBIGDATA*, 26 Nov. 2014, insidebigdata.com/2014/11/26/ask-data-scientist-data-leakage/.

[x] Deotte, Chris. "Feature Engineering Techniques." *Kaggle*, www.kaggle.com/c/ieee-fraud-detection/discussion/108575.

[xi] Mesaoudi, Zakaria EL. "Elo Merchant Category Recommendation." *Kaggle*, www.kaggle.com/c/elo-merchant-category-recommendation/discussion/82127.

[xii] Lee, Dong-Hyun. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks." *Workshop on challenges in representation learning, ICML*. Vol. 3. 2013.

[xiii] Han, Jiawei, et al. *Data Mining: Concepts and Techniques*. Elsevier, 2012.

[xiv] Chan, Carmen. "Smoothing Time Series Data." *Displayr*, 16 May 2019, www.displayr.com/smoothing-time-series-data/.

[xv] Banks, David 2009 Smoothing, lecture notes, Duke University http://www2.stat.duke.edu/~banks/218-lectures.dir/dmlect2.pdf

[xvi] Olariu, Andrei. *Andrei's Web Mining Blog*, 21 Feb. 2013, webmining.olariu.org/event-recommendation-contest-on-kaggle/.

[xvii] Lebedev, Alex. "MLSP 2014 Schizophrenia Classification Challenge." *Kaggle*, www.kaggle.com/c/mlsp-2014-mri/discussion/9854.

[xviii] P. Mitra, C. A. Murthy and S. K. Pal, "Unsupervised feature selection using feature similarity," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 301-312, March 2002, doi: 10.1109/34.990133.

[xix] "Boruta." *Function | R Documentation*, www.rdocumentation.org/packages/Boruta/versions/6.0.0/topics/Boruta.

[xx] Jordan, Jeremy. 15 6 2020. <https://www.jeremyjordan.me/hyperparameter-tuning/>.