Scalable Black-Box Model Explainability
Through Low-Dimensional Visualizations

by Aradhana Sinha

S.B. CS. MIT, 2017

Submitted to the
Department of Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the
Massachusetts Institute of Technology

June 09, 2017

© 2017 Massachusetts Institute of Technology. All rights reserved.

Author:
Department of Electrical Engineering and Computer Science
June 2017

Certified by:
__________________________________________
Thomas Finley, Principal Software Engineer at Microsoft,
Thesis Supervisor
June 2017

Certified by:
__________________________________________
Tomas Palacios, Professor, Thesis Co-Supervisor
June 2017

Accepted by:
__________________________________________
Christopher Terman, Chairman, Masters of Engineering Thesis Committee
Scalable Black-Box Model Explainability Through Low-Dimensional Visualizations
by Aradhana Sinha

Submitted to the Department of Electrical Engineering and Computer Science
On May 12, 2017
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT
Two methods are proposed to provide visual intuitive explanations for how black-box models work. The first is a projection pursuit-based method that seeks to provide data-point specific explanations. The second is a generalized additive model approach that seeks to explain the model on a more holistic level, enabling users to visualize the contributions across all features at once. Both models incorporate visual and interactive elements designed to create an intuitive understanding of both the logic and limits of the model. Both explanation systems are designed to scale well to large datasets with many data points and many features.

Thesis Supervisor: Thomas Finley
Title: Principal Software Engineer, Microsoft

Thesis Co-Supervisor: Tomas Palacios
Title: Professor of EECS, MIT
# Contents

1 Introduction  
1.1 Motivation and Objectives ........................................ 5  
1.2 Improvements on Existing Work .................................. 6  
1.3 Contributions ...................................................... 8  
1.4 Organization of this Paper ....................................... 8  
1.5 Statement of Originality ......................................... 9

2 Related Work  
2.1 Permutation Feature Importance ................................. 10  
2.2 LIME ................................................................. 11  
2.3 Projection Pursuit .................................................. 12  
2.4 Generalized Additive Models .................................... 12

3 Methods  
3.1 Holistic Feature Trends through GAM .......................... 16  
3.2 Example-Specific Intuition via  
    Projection Pursuit .................................................. 18  
3.3 Implementation Improvements ................................... 22  
3.4 Computer Resources ................................................ 23

4 Results  
4.1 Projection Pursuit ................................................. 25  
4.2 Generalized Additive Models .................................... 28
5 Conclusion

5.1 Applications .................................................. 36
5.2 Future Work ..................................................... 37
Chapter 1

Introduction

This chapter describes the motivations behind creating model explanation systems. It then it summarizes the novel contributions that are made in this project. Finally, it offers a brief overview of how the rest of the paper is organized.

1.1 Motivation and Objectives

The mission of Microsoft is to empower. Azure seeks to empower by democratizing advanced analytics, machine learning, and AI capability–by enabling anyone with any background to feed in data and yield meaningful insights.

The wide scope of the target audience presents a challenge. It is not sufficient to merely implement an easy to use machine learning library. It is also essential to deliver insights into what the results mean in an intuitive manner. A normal person of any background should be able to understand what the black-box model predictions mean in context of his or her problem. Users should understand the types of conclusions they can make from their results, and those they cannot. They should understand when overfitting occurs, and when predictions are extrapolated too far. They should know what type of data is required to answer the questions they want to answer.
Restated, it is not sufficient to provide a model that performs with high accuracy. The model created must be interpretable and usable by the user. This problem of creating interpretable, usable models is referred to as the problem of “model explainability.”

Model explainability for a black-box ML system is made more complex by varying computational abilities. Azure charges its customers per computer-hour. This means that calculating the model explanation itself has a monetary cost. This creates a motivation for the explainability system to be computationally light.

1.1.1 Microsoft Azure Requirements

The code-base that underlies Azure and much of the internal machine learning needs at Microsoft needs to be able to deal with large amounts (on the order of terabytes) of data fed into it, as automatically as possible[16][21].

Additionally, models often combine numerical and categorical features. Hence, there needs to be a way to compare both categorical and numerical data in the explainer system[21].

Further, it is desirable to provide a uniformity of user experience across different learners. This both reduces the learning curve for the user, and enables more learners to be able to be added into the underlying machine learning libraries without necessarily having to write a special explainer for each one. The explainer systems, then, should be able to take in the model output files for any of the algorithms currently supported in the code-base, and provide explanations for them.

1.2 Improvements on Existing Work

Most of the explainer systems that are published in papers exist more as proof of concepts, tested on a couple datasets. They do not scale well to large number of data points or a
large number of features.

Previous work on model explainability has taken two primary approaches: a holistic model approach, and an example-specific one. A few of the methods I draw from are covered in more detail in Section 2.

The first approach attempts to summarize how the model works in general. It often involves an ensemble of simpler methods such as small decision trees [18][12][13], or sparse linear models [20][15][19], sometimes accompanied by leave-one-feature-out re-training[19] or re-evaluation [1]. With the exception of the sparse linear models [20], however, these often take up a lot of space (to store the ensembles), and take a long time to re-train [23] or re-evaluate[1]. Sparse linear models, meanwhile, are quick and intuitive, but often sacrifice much accuracy; the resulting model performs not nearly as well as other methods [20]. In this paper, I propose a holistic model approach in Section 3.1 that performs substantially faster than existing approaches, while still achieving comparable accuracy to the best non-interpretable models (such as fast forests and neural nets).

The second approach attempts to only provide explanations for individual data points. Sometimes this is accomplished by highlighting the features with the largest effect on the outcome. This approach is often used in computer vision and deep-learning models [2] [4]. Alternatively, this approach often involves inputting artificial data points in the vicinity of the point to be explained, and learning simpler linear models on these artificial points [14][22]. This paper does not explicitly handle model explainability for computer vision problems. In Section 3.2, however, I propose an example-specific approach. Unlike previous methods [14][22], however, this method does not rely on having either an active model that can return predictions, nor on the use of artificial data. This means that unlike previous methods, it is able to provide intuition for when the user is over-extrapolating, drawing conclusions from areas where there are few training points.
1.3 Contributions

In this project, I propose explainer systems that provide:

1. example-specific decision explanations
2. highlighting of decision boundaries
3. succinct, human interpretable explanations
4. intuitive visualizations
5. flags for over-extrapolation from data

The explainer systems work with any black-box ML system. They do not require prior knowledge about the underlying data distribution, or even an active model (one that can return predictions for user-determined input). Most importantly, these systems work at scale. They terminate in a reasonable amount of time on training sets with millions of instances and hundred thousands of features. The scalability of these methods and their ability to provide intuition on over-extrapolation are the most significant improvements over existing systems.

1.4 Organization of this Paper

Previous work done in the area of model explainability is discussed in Chapter 2. Each method is assessed using the criteria established in section 1.3. Next, section 3 details two novel model explainability systems. The first method detailed in Section 3.1 provides a holistic understanding of how the model operates. The second method detailed in Section 3.2 provides example-specific explanations—describing why a particular data point was classified in its predicted category. The effectiveness of these methods is explored in Chapter 4. Datasets of varying sizes, both in terms of the number of data points and in terms of the number of features, are used to assess the effectiveness of the model
explainability systems. This section also evaluates how some of the speed-improving design choices impact the overall functioning of the model explainability systems. Finally, findings are summarized in Chapter 5. This section also describes current uses for these systems, and discusses future work to be done.

1.5 Statement of Originality

All work presented here is my own, with guidance from my mentors Pete Luferenko and Thomas Finley.
Chapter 2

Related Work

This chapter expands on some of the more promising methods mentioned in section 1.2. Here I cover the model-agnostic explanation systems from which this project draws inspiration. Each method will be assessed against the goals listed in section 1.3.

2.1 Permutation Feature Importance

The Permutation Feature Importance (PFI) approach seeks to determine the most significant features across the entire model[1]. The most significant feature is defined as the one that makes the largest overall contribution to reducing prediction error. An example of a PFI explanation for the Boston house prices dataset[7] is: “Location to Work Hubs, and Number of Rooms.”

To assess the significance of a given feature, PFI permutes the values of that feature-column across all the data points in the dataset. It then re-evaluates the model completely using the new permuted dataset. The feature that results in the greatest loss of model prediction accuracy is said to be the most significant.

By permuting feature values from the preexisting dataset, PFI maintains the experimentally derived distribution of those features. This is a desirable feature, as it enables
PFI to evaluate the significance of each feature, taking into account the natural amount of variation within that feature.

While a relatively simple approach, PFI is often not pragmatic or feasible. Re-evaluating the entire model for each feature takes too much time, especially for higher dimensional problems that have > 15 features.

Additionally, results are not very informative. They are not example specific, they do not highlight decision boundaries, nor do they lend themselves to visual explanations. It becomes unclear how the features listed are significant.

2.2 LIME

This approach seeks to determine the most significant feature that leads to a decision for a given example by identifying local decision boundaries [14]. Unlike PFI, where the most significant feature was the one that contributed the largest error reduction, in the LIME paradigm, the most significant feature is the one that’s closest to a decision boundary. LIME is to a discriminative model what PFI is to a generative one. Whereas PFI was a holistic model approach, LIME is an example-specific one.

LIME assesses a given example, by feeding random points found spatially near the example into the model. These random points are selected using a multi-variable Gaussian centered at the example point, with an arbitrary standard deviation. Based on the classification of the artificially generated points in the vicinity, LIME then attempts to come up with a simple linear decision boundary between these artificial points to highlight the decision rule that most closely affects the classification of the example point.

LIME does achieve my goals of being example-specific, highlighting decision boundaries, and lending itself to intuitive visualizations.

This approach, however, is valid only for classification problems. Additionally, it
requires the use of a live model–one from which the user can feed in inputs and get out predictions on demand. Even more problematically, it does not rely on the training set to pick the artificially generated points–this means that the rules learned are sensitive to the magnitude of the feature numerical values, and that points in the vicinity may not be representative of the true distribution of all data points.

The approach of sampling test points from the area around the example point has a few additional drawbacks. The number of test points that need to be sampled for the same coverage increases multiplicatively with each additional feature dimension. Also, in situations where the point to be explained is not near a decision boundary, this approach does not yield useful information. Nor is LIME able to highlight situations where the point to be explained clearly over-extrapolates from the training data.

2.3 Projection Pursuit

Projection pursuit refers to a host of techniques that attempt to make sense of high-dimensional data, by projecting the data onto a lower-dimensional space, and analyzing the model in that space. LIME is one such technique. This approach can be likened to building a simpler human interpretable model on top of the pre-existing model. This general approach is attractive as lower-dimensional projections are much easier to visualize and intuitively understand.

Selecting which dimensions to project upon, and how to analyze the data in this scenario are relevant sub-problems within this framework.

2.4 Generalized Additive Models

Generalized Additive Models (GAM) are a class of predictors that can be represented as the sum of some unknown function on each of it’s features [8]. Equation 2.1 depicts the
general form of these models where $Y$ is the output, $g$ is a linear function, $\phi_i$ are some distinct function on each of the $d$ features $f_i$ of the input.

$$g(\mathbb{E}(Y)) = \sum_{i=1}^{d} \phi_i(f_i) \tag{2.1}$$

Visualizing each of the $\phi$ functions separately as a plot makes for an immediately intuitive chart. Paired with some representation of data density on the $\phi$ function plots, it becomes easy to visualize where there aren’t enough training points, and thus prevent over-extrapolation.

The choice for the $g$ and $\phi$ functions in equation 2.1 are relevant sub-problems. Typical choices for $g$ include the identity and logit functions. Typical choices for the $\phi$ functions may include scatter-plot smoothing [19], parametric regressions [15], or tree ensembles as $\phi$ functions [12].

In this paper, rather than fit $\phi$ functions to continuous data, the data is immediately discretized into bins. A bin smoother is used as a $\phi$ function. A bin smoother is just a discretized version of scatter-plot smoothing that assigns one constant value per bin. This decision dramatically speeds up the training time, enabling us to use the method on larger more complex datasets. Additionally, discretized histograms enable better enable post-training user editing than a smooth curve would permit. A user that’s only able to modify the lengths of bar histograms is much less likely to over-fit to his or her data than one that can manipulate a continuous curve.

GAM models have a better accuracy than linear regression methods of the form $\mathbb{E}(Y) = \sum_{i=1}^{d} w_i \cdot f_i$ as they have more degrees of freedom. At the same time, they cannot perform better than fully complex models, those of the form $\mathbb{E}(Y) = \Phi(f_0, \ldots, f_d)$ as they do not account for interaction terms between the features. These theoretical bounds are verified by Lou et. al. [12].
2.4.1 Augmentation with FAST Pairwise Interactions

Lou et. al. have also had success approaching the performance of fully complex models by
greedily including selected pairwise interactions that best minimize the residuals of their
GAM models [13]. Keeping with the terminology of that paper, Generalized Additive
Models with Pairwise Interactions will be referred to as GA$^2$M models.

They select these additional pairwise interactions using their FAST algorithm. FAST
first creates equidistant bins for each feature in their dataset. They then create cumula-
tive sum arrays for the targets and frequencies across the binned-training dataset. The
target values are initialized as the residuals after the GAM predictions (without pairwise
interactions).

For example let the dataset contain the following values for feature $A$: $[1, 1, 2, 4, 4, 4, 4]$ with
the corresponding targets, $y$ : $[1, 2, 5, 5, 5, 2, 5]$. Feature $A$ might have four bins with
following cumulative frequency counts: $[2, 3, 3, 7]$, and hence the following cumulative
sum counts: $[3, 8, 8, 25]$. Note that the bins are created using the value of the feature in
the dataset, and the cumulative sum array refers to the cumulative sum of the targets
($y$) across these bins.

After creating these cumulative sum counts, FAST then iterates over all possible
pairs of features. For each feature pair, FAST creates a cumulative sum matrix across
the binned targets. Position $(2, 3)$ in the cumulative sum matrix for features $A$ and $B$,
for example, would contain the sum of the targets of data points where feature $A$ has a
value accounted for in the first 2 bins and has a feature $B$ value accounted for in the first
3 bins. FAST then makes use of the cumulative matrix and cumulative feature arrays, to
pick the best split location.

A split location at position $(i, j)$ divides the two-feature matrix into four quadrants:

1. items that belong in the first $i$ bins of $A$, and the first $j$ bins of $B$.
2. items that belong in the first $i$ bins of $A$, but not in the first $j$ bins of $B$.  

14
3. items that do not belong in the first $i$ bins of $A$, but belong in the first $j$ bins of $B$.

4. items that do not belong in the first $i$ bins of $A$, not in the first $j$ bins of $B$.

An effect value, $\phi$, is assigned to each of the quadrants (analogous to the 1-D bin-smoothers). To most effectively reduce the prediction error, the effect in each quadrant is set to the average of the target values for the items in that quadrant. Given the data structures above, it is easy to use dynamic programming to compute these average target values for the quadrants at each split point in constant time. Let $L^t(i, j) = [a, b, c, d]$ be the look-up table for the sum of targets on the split at $(i, j)$, and let $L^w(i, j) = [a, b, c, d]$ be the look-up table for the sum of the frequencies on the split at $(i, j)$.

The best split location is the one that most reduces the prediction error. Let $q$ represent the et of quadrants created by a split point. Then prediction error can be expressed as $\sum_{k=1}^{N} (y_k - \phi(x_k))^2 = (\sum_{k=1}^{N} y_k^2 - 2\sum_q \phi(q)^2 L^t(q)) + \sum_q \phi(q)^2 L^w(q)$. The only terms in the error function that are affected by the choice of split point are the latter two, and these can be computed in constant time for each choice of split point.

This means that for a given pair of features, the optimal split point can be chosen in $O(b^2)$ time where $b$ represents the number of bins each feature is binned into.

After picking the top few pairwise interactions—those where the 2-D splits most reduce prediction error, the algorithm re-learns the $\phi$ effects to assign to all the quadrants across all the top 2-D splits.

Taken in all together GA$^2$M models take a much longer time to train than do GAM models, for small increases in accuracy[13]. The FAST method, however, does demonstrate the power of binning and using frequencies to more rapidly find significant pairs of features. Binning becomes especially relevant in datasets with a large number of instances.
Chapter 3

Methods

As noted in section 1.2, there are two main use cases for model explainability. The first involves understanding a model holistically: what impact a given feature has on the entire model. The second is the example-specific approach, that seeks to better understand why a given point is classified a certain way.

In this chapter I cover the two explainability systems I have implemented to solve each of these use cases. First the purpose of the method is discussed, followed by implementation details.

Both methods developed emphasize the importance of being fast to come up with the explanations, creating a visual intuitive understanding of the model, and preventing users from over-extrapolating their model.

3.1 Holistic Feature Trends through GAM

3.1.1 Purpose

The first method is a GAM (as described in section 2.4). GAM involves understanding the model holistically—being able to visualize trends in a feature without thinking about
a specific sub-set of examples. Such a system is intended to help:

1. Determine whether feature $X$ has predictive power
2. Characterize how predictions vary across the domain of feature $X$.
3. Be able to assess the impact on accuracy in a version of the model where feature $X$ has no predictive power.

A use case of this method is described in section 4.2.3 regarding COMPAS software, a computer model that affects the sentencing of those indicted with crimes by predicting their risk of recidivism.

### 3.1.2 Implementation

In reference to Equation 2.1, here the $g$ function is kept as the identity function, and the $\phi$ functions are bin smoothers.

In the interest of speed and memory efficiency, the training set feature-values are each binned into one of at most 255 bins by the binning process described in section 3.3.1.

In each iteration, a single split (analogously a two-leaf tree) is learned in parallel on each of the features, with a very small learning rate. The number of iterations is determined by a five-fold cross-validation.

The system outputs a web page that visualizes the bins and their (additive) effects as a histogram for each feature. A screen-shot of this output can be visualized in Figure 4.8. The opacity of each bin represents the density of data in that region. Users have the ability to manually adjust the effects for the histogram bars and view the resulting change in prediction error for an inputted dataset.
3.2 Example-Specific Intuition via Projection Pursuit

3.2.1 Purpose

This method addresses the need to understand why a given example has an associated prediction, and not another in classification problems. In this way its purpose is similar to LIME. Unlike LIME, however, this method can work with static models—models where you are only given the inputs and outputs without the ability to feed in your own inputs. In only relying on the training set and predicted scores, it also avoids need to test multiplicatively more points for each additional feature as LIME does. Finally, unlike LIME this approach does not require the data point to be near a decision boundary to be useful, and prevents over-extrapolation from the training set.

In this second method it is important to understand which features resulted in a given output and not another. It is able to answer succinctly and graphically why a prediction is made, as well as with what confidence that prediction is made.

3.2.2 High-level Implementation

Heuristic projection pursuit methods were employed. These are described in section 3.2.4. Given a large dataset, the program does the following, in each iteration.

1. Randomly selects sets of $k$ features for some small $k$
2. Selects a random sub-set of the data.
3. For each set of $k$ features, runs PFI or FAST on the $k$ features for the sampled sub-set of the data.[13].
4. Projects data points on the most promising $k$ features, and attempts to come up with a simple explanation
5. Removes data points that are classified correctly, and whose neighbors are classified correctly by the simple explanation (within some margin of error, and with some minimum number of neighbors). Saves regions of correctness.

6. Visualizes explanation.

7. If not explanation is arrived upon after three iterations (that is no points are removed in step 5 for three iterations), the margin of error is increased.

The algorithm stops once there are too few data points left.

When attempting to explain a specific point, we go through each iteration in order. At each iteration, we check to see whether this data point falls within the region of correctness. We use this region of correctness and the chained simple explanations to explain why a point was classified a certain way. An example of an explanation may be of the form, “Since the flight departed between 11pm and 1am on a Saturday it would have been likely to arrive late, but since the flight belonged to Alaska and left from Seattle, it was on time”. Here the specific flight is the point being explained. The simple explanation relies on two layers of explanations. In the first layer, the point falls into a region from 11pm to 1am on Saturday where flights are overwhelmingly late. It, however, has been classified as being on time, and is not removed in step 5 of the first iteration. In the second layer it falls into the category of Alaska Airlines flights from Seattle which are overwhelmingly on time.

3.2.3 Ranges and Boundaries in Explanations

The method outlined above works in a manner almost the opposite of LIME. Whereas LIME attempts to find decision boundaries by scattering points in regions of uncertain behavior, this method attempts to corral regions where the points are overwhelmingly correctly explained and of the same class. If a decision boundary in the projected space is close by, it is visualized as in LIME. Like LIME, it can provide the range of a certain
feature that is significant, citing for example not just the time of day feature, but specifically 11pm -1am. Whereas LIME creates computation cost every time a point is asked to be explained, this method front-loads the computation, and makes feature explanation a $O$(number of layers) process, which in practice is pretty close to constant.

### 3.2.4 Explanations Sought Heuristically

The objective of the method is to provide intuition for what the model is doing, in a quick effective manner. Hence doing PFIs for randomly selected sets of $k$ features on a subset of the data enables the program to check several possibilities rapidly. It does not matter whether the program gets the best possible set of $k$ features, only that something significant is selected. By not looking for the best possible set of $k$ features and instead focusing on only removing points from areas of overwhelming certainty, the program saves a considerable amount of time in comparison to other projection pursuit approaches [22].

Instead of using the best projection to explain all the points, the program saves multiple “regions of correctness” per projection. Each point considered explained (and hence removed in step 5) within a “region of correctness” must be 1. classified correctly by the rule learned on the projection, and 2. 95% of it’s neighbors must be classified correctly. This neighbor correct classification requirement starts at 95%, and goes down for every three iterations where no points have been considered explained and removed (step 7).

Calculating the nearest neighbors is itself a heuristic process, that relies on k-d trees divided at the split points determined by the binning method outlined in section 3.3.1. These heuristics enables the program to explain points more quickly, without sacrificing much accuracy.

Finally, when $k = 2$, the FAST method described by Lou et. al. [13] may also be used. This method involves putting the features into equal sized bins and determining which single split on each pair of features most reduces prediction error (as described
in section 2.4.1). Alternatively, the binning approach outlined in section 3.3.1 may be used in conjunction with the rest of the FAST method. Let \( n \) be the number of data-points, and \( d \) be the number of features in a dataset. The run-time of this process then becomes a \( O(n) \) to pre-process, and a constant time look-up for each feature pair. Total preprocessing time then once takes \( O(nd^2) \) time.

### 3.2.5 Avoids Over-extrapolation

As discussed earlier, a point is not removed and considered explained in step 5 until both it and its surrounding neighbors are classified correctly by the simple explanation in the projection space. By thus requiring a certain number of the neighbors of each point to also be classified correctly, the system prevents explanations from over-extrapolating from the training set.

Moreover, the results of the iteration are visualized by plotting the training data alongside the point being explained. By representing a simple rule and where the training data falls, the visualization makes it easier for the end user to avoid over-extrapolation. Restated, they are less likely to make conclusions about a data point that is far away from the visualized training set data.

### 3.2.6 Pre-computing iterations can give rapid on-demand explanations

A problem that example-specific explainers often run into, is that a custom explanation per data point can be expensive to compute (e.g. LIME). In this paradigm, it is possible to run the algorithm in advance, and then only rapidly check through regions of correctness when attempting to explain a point. This one-time computation is ideal.
3.2.7 Replicability of explanations vs. Interpretability

Running the algorithm many times may produce different region of correctness explanations. Key features and feature combinations, however, tend to be preserved in the earlier iterations.

This is not a weakness of the method, as the shorter explanations do not of course capture the entire complexity of the model, but do make the explanations human-interpretable. Coming up with an exhaustive more deterministic explanations for each point would take much more time, and ultimately perhaps not be more human interpretable.

3.3 Implementation Improvements

3.3.1 Binning

To speed up the training process and reduce memory required, the values for each feature in a dataset are processed into binned values. Each feature can have at most 255 bins over a finite domain.

Categorical variables with greater than 255 values are hashed into bit vectors prior to being binned, and can be put into bins by value.

Continuous variables are binned by the dynamic programming k-means algorithm outlined in [24] on $O(n \log n)$ time. This method was chosen as it performs much faster than heuristic k-means for larger $k$ values.

3.3.2 Flocks

Each feature or set of related features are stored in data-structures called flocks. These flocks are designed to reduce storage space, and speed-up training.
The flocks do not store the true value of the feature but rather its associated bin value (as determined in section 3.3.1. Each flock may include a single feature or multiple related features. There exist two different types of flocks. These are enumerated below.

**Sparse:** Rather than store the dense array of values, they store only the non-zero values in a value array, and the lengths of the gaps between these values in another array. For the multi-feature version of this flock, each location in the value-array may have up to some fixed number $n$ non-null entries; these flocks will be referred to as $n$-hot sparse flocks.

**Dense:** For data columns that do not have many null values, a dense vector is produced that stores all the values. Multi-feature dense flocks are used for features with near identical binned values.

The multiple-feature flocks are often invoked on categorical variables. For example, let there be a categorical variable for color, that can have the possible values “red”, “blue”, or “green”. This categorical variable would be transformed in a three-length feature vector (so $<0,1,0>$ for blue, or $<0,0,1>$ for green). The binning process would create only two bins for each of the three places in the feature vector. This in turn would be wrapped in the a 1-hot repetitive array. If, however, the original features allowed an input vector to have “red” and/or “green” and/or “blue” instead of only one of the above, the ultimate flock would probably result in a 3-hot sparse flock.

These data structures may only be cursored over in the forward direction, and are meant to be used with binned values.

### 3.4 Computer Resources

All the experiments presented in this paper were run on an HP Z240, with an Intel Xeon E3 processor, 8GB RAM, and 1TB 7200 rpm SATA HDD.
Chapter 4

Results

In this section, I assess the two systems implemented. First the performance of the projection pursuit method is assessed on datasets of varying sizes, and both high and low dimensionality. The PFI and FAST projection seeking heuristics are examined across datasets with varying numbers of features. Next, visualizations are presented as intuitive explanations that incorporate the information captured by the projection pursuit method.

Then the proposed GAM approach is assessed. This approach is promising compared to the version proposed by Lou. et. al. [12] in terms of speed, accuracy, and space requirements. The impact of flocks and bins in speeding up the training process is demonstrated. Next, the impact of learning rate is discussed—despite an overall good performance, GAM destabilizes at higher learning rates. Finally, static visualizations are presented for the results of this explanation method.
4.1 Projection Pursuit

4.1.1 Projection Selection

Three methods of selecting projections were proposed in section 3.2.4, namely PFI on the entire dataset, PFI on a random sample from the dataset, and FAST (when the dimensionality of the projection is 2).

First the performance of PFI versus that of PFI on a random sample from the dataset is explored in terms of their time to completion and average depth of explanation. Time to completion is defined as the time it takes for all points in the training set to be placed into an explainable region. The average depth of explanation refers to the average number of layers of explanation across the data points in the training set. Ideal human-interpretable explanations are short and to the point, with preferably three or fewer layers. Results visualized in Figure 4.1 demonstrate the effects of running PFI from a random sample from the dataset. This approach works to substantially speed up the training process if the initial dataset is large enough (permitting a representative sample to be chosen), and the number of features is not terribly high in proportion to the number of samples.

Second, PFI approaches are compared to FAST. Two binning approaches were used with FAST: first the equal sized bin approach put used by Lou et. al.[13], and then using an variable-length binning system that clustered by value as described in section 3.3.1. Again the metrics used for comparison are time to completion and average length of explanations. FAST did not terminate in a reasonable amount of time for datasets with a high number of features (like the 20 topic news classification[17] dataset or the arrhythmia dataset[6]). This is not surprising since the time complexity for FAST depends on the square of the number of features. Nevertheless FAST did show speed-ups for datasets with a high number of instances, and low number of features, as is demonstrated in Figure 4.2. The variable-length value-based binning described in section 3.3.1 outperforms the equal sized binning system.
Figure 4.1: The graphs above compare training time and average length of explanations for various sized samples during PFI projection selection (with projections into 2-D space). In the headers, $n$ refers to the number of samples, and $d$ the dimensionality of the features. 20 Topic News classification\,[17] represents a high $n$, high $d$ dataset. This dataset also had the advantage of there being multiple sparse features (word counts) that predicted topic selection (the outcome) well without interacting with other features, and sampling methods excelled here—reducing training time without significantly impacting the length of the explanations. The low $n$, high $d$ dataset involving Arrhythmia \,[6] did not perform well under any PFI selection, though lower sampling rates performed markedly worse. The high $n$, low $d$ dataset involving Census Income \,[9] performed well under sampling, increasing speed of training without impacting explanation length. Finally, the low $n$, low $d$ dataset Boston Housing \,[7], did not substantially benefit from sampling methods.
Figure 4.2: The graphs above depict the time to completion and length of explanation under the optimal PFI projection method (as determined in Figure 4.1), FAST with equal-sized bins, and FAST with variable-length bins. The variable length bins take a little longer to compute than the equal-sized bins, and yield slightly more succinct explanations. On datasets with a small number of instances and dimensions such as the housing dataset [7], FAST does not improve speed. On datasets with a larger number of instances and low-dimensionality, FAST does provide substantial speed-ups as demonstrated by the Census Income dataset [9] above.
4.1.2 Visualizations

The aim of visualizing the projections are to provide intuition about three relevant states: the explainer’s predictions, the model we are attempting to explain, and (when available) ground truth values.

Between the model and the ground truth, we have two types of information—whether the prediction was correct, and the confidence with which the prediction was made.

Between the explainer and the model, we again have two types of information—the location of the simple explanation decision rule, and the location of the training data points within each region of correctness.

Examples of some visualizations are depicted in Figures 4.3 and 4.4. Figure 4.3 provides intuition for why the model is frequently wrong, by visualizing a rule that the model uses. Figure 4.4 demonstrates how these visualizations may help a user avoid over-extrapolating their models.

4.2 Generalized Additive Models

4.2.1 Much Faster and More Compact than Previous GAMs

Figures 4.5 and 4.6 demonstrate that the $\phi$ function proposed in this paper was significantly faster, and took much less training space. Both Split-based methods achieved an AUC of 97.1% and both tree ensembles achieved an AUC of 97.4%; the split method achieved similar accuracies to previous GAM algorithms.

In these images, the label Tree Ensemble refers to the method of Lou. et. al. [12], who proposed using tree ensembles as their $\phi$ function over each feature. The Split method refers to the method proposed in this paper, where only a single two-leaf split is learned on each feature in parallel. The label FB refers to the speed-ups implemented
Figure 4.3: The figure above represents the second iteration from a dataset on soil types. **Model → Truth:** Points in brown are classified as one soil type by the model, and those in blue are classified as another type. The points plotted represent the training data projected into this space. If the shape of the point is a circle, the model correctly predicted the ground truth. If the shape of the point is a star, the model was wrong in it’s prediction of the ground truth. The degree of shading within each point determines how certain the model was that it had an accurate prediction. **Explainer → Model:** The black line pictured above is the line the explainer drew as a simple explanation of the model behavior in this projection. The color of the background squares represents the classification made by the explainer. The shade of the background represents how certain the explainer was that it correctly captured the model behavior. Regions of dark background indicate that more points in the model were present here that were classified correctly. This is a useful indicator since scatter points may layer, and their densities may be misleading when plotted. **Interpretation:** The image above indicates that the model classifies soil type based on how close it is to the water. The model is often not correct. The visualization enables us to see why the model predictions are frequently wrong.
Figure 4.4: Instructions on how to interpret the image are outlined in the earlier part of Figure 4.3. This image comes from the Iris dataset. The model correctly classifies all points, and the explainer correctly explains the model behavior. This image also enables us to see how the system can more easily highlight outliers. Recall that the transparency of the background color is inversely related to how sure the explainer is at that location—a function of how many training points are in the area. If our user sought to explain why the model was classifying a flower with an 8 in Feature 0 and a 2 in Feature 4 as being blue, the explainer would detect that there aren’t many training point at that location, and plot this graph demonstrating that there is not enough information for the model to reasonably draw conclusions for that point.
Figure 4.5: This graph depicts the additional space required for training for each of the $\phi$ functions (labels described in section 4.2.1). This metric does not include the space required for the entire dataset, only the additional space required by the trainer (including any data it may choose to cache). It becomes apparent that binning saves space enormously. Rather than manipulate the values individually the program is able to only store their bin indicies and manipulate those instead. Splits also takes significantly less space since it only needs to store the split points for the bin indicies, as opposed to needing to store the partitioned sets at each split in the tree ensemble.

through the use of flocks and variable-length binning as described in section 3.3. Note that regular Splits method also employs the variable length binning discussed in section 3.3.1, but does not employ flocks. In order to fairly compare the different methods, all of them were implemented in the same code-base and relied on the same data cursoring and reading methods.

The Bag of Words dataset[11] was used for these experiments. 300,000 NY Times Articles and 300,000 PubMed articles were selected. The various $\phi$ functions described in the previous paragraph were used to classify articles into one of the above two categories. This dataset was chosen for it’s large size.

4.2.2 Sensitive to Learning Rate Parameter

Given that all features learn a split simultaneously, the Generalized Additive Models (GAM) are very sensitive to the parameters provided for their learning rate. A large learning rate can result in the model not converging as depicted in Figure 4.7. An overly
Figure 4.6: The figure above depicts the duration of the training in minutes. The inset graph contains the pre-processing time in seconds. This is the time needed to create bins, and flocks. It is apparent that although binning and creating flocks take slightly more pre-processing time, they save a considerable amount of training time. The most costly part of the training process is merely iterating over the values. Flocks and bins greatly speed up this process by compressing the number of values that need to be iterated over in each feature. This is especially useful in bag of words datasets like the one pictured above, where the vast majority of features are null for a given data point.

small learning rate, however, can require many iterations and take a long time to train. Cross-validation on the learning rate is recommended.

4.2.3 Case Study: COMPAS

The use of this method is demonstrated on COMPAS, software that determines the probability that an individual accused of a crime, will commit another crime [3]. Since this software has the power to affect sentencing of people within the legal system, it becomes important that the software not be affected by age, gender, or race in any way. The question here then is not whether the system’s predictive accuracy is equivalent in defendants of different demographics, but rather whether these demographics impact risk classification.

COMPAS takes in a person’s entire arrest record, age, gender, and then asks 156 questions about their personal life unrelated to the crimes they have been involved with. It does not take ethnicity as an input[3].
Figure 4.7: This graph represents the results from various GAMs all trained on the same COMPAS dataset[10]. The x axis represents the learning rate used, the y-axis represents the average error of the resulting model (on a log scale), and the size of the circle signifies the number of iterations it was run for. Regardless of the number of iterations an experiment was run, higher learning rates tend to lead to higher variance in the outcome, which could destabilize the learning.

GAM was used to replicate the risk predictions made by COMPAS using the dataset compiled by ProPublica [10]. This dataset incorporated the entire arrest record, age, gender, and race, but did not include the personal questions asked. Some of the resulting charts are depicted in Figure 4.8. A web-page version of the visualizations was also created that lets you input a dataset and a model, and then drag around the bin heights, and see the effect on the prediction error in real time.

Figure 4.8 makes it easy to visualize trends. Older people tend to be labeled as lower risks. Increased time in prison corresponds to increased risk until about a period of 25 years. Those who have served sentences longer than 25 years are classified as lower risk. Finally, though race is not taken in as an explicit input, race has substantial predictive power over the score. The histogram immediately calls into question whether risk assessments might be biased unfavorably against African Americans and Asian people. The Asian ethnicity bar is very faint, however, and only represents two data points. There is probably not enough evidence to evaluate bias there. The African-American ethnicity
Figure 4.8: The charts above are some of the ones generated while analyzing COMPAS results through GAM. The darker the bin, the more instances fall into that bin (proportional to the bin width). The y-values are points on the COMPAS Risk of Recidivism Assessment with higher numbers indicating higher risk. Effects are additive.
bar, however, does contain many instances, and is a trend worth investigating further.
ProPublica did find evidence of bias in COMPAS against African Americans upon testing [10].
Chapter 5

Conclusion

In this report two methods were proposed as explainer systems for black-box models. The first projection pursuit method is the only example-specific black-box explanation system that scales well to large datasets—those involving over a hundred thousand features, and over a hundred thousand instances. The second GAM approach enables easy visualization of how features impact the outcome of the entire model, and enables manual editing of the model post-training.

Both accomplish the goal of intuitively preventing users over-extrapolating their models by representing the density of the training set. Both are also much faster than previous versions proposed. Neither LIME nor PFI would finish running the 20 News Classification problem in a reasonable amount of time. The GAM approach outlined here performs several times faster than the tree method proposed by Lou et. al, and takes up significantly less space.

5.1 Applications

The GAM system is already a part of the ML libraries that underlie Azure at Microsoft. It has also been used by the people at Bing to help them understand the effects of their
UI modifications on their click metrics. A major airline may use GAM to help set their seating prices.

The ability to view relevant projection pursuit charts, however, is not shipped.

5.2 Future Work

5.2.1 Increase Use of the Projection Pursuit Approach

The projection pursuit model should be made more aesthetically pleasing and perhaps optionally less information dense. There were complaints that the charts were too difficult to interpret.

Next steps might involve integrating these visual charts or text descriptions with the existing web page for the GAM system.

5.2.2 Gradient Adjusting Methods for GAM Approach

The GAM approach could be made much faster if larger learning rates were used. These larger learning rates, however, increase instability of the algorithm when features are trained in parallel.

It would be worth experimenting to see whether a self-adjusting gradient method similar to Adagrad would perform better. It might be worth testing a scheme where if an iteration of learning splits on all features does not result in lower error overall, the learning rate would halve.
5.2.3 Incorporate Feature Interactions into GAM

This paper greatly valued the ability to rapidly train on large datasets, to be able to provide model explainability for Azure. Next steps would involve relaxing the speed criteria in favor of slightly more accurate models. It might be worth attempting to implement a heuristic version of FAST. While the original algorithm proposed in Lou et al [13] does not be able to scale to the datasets with hundreds of thousands of features, it is possible that some of the heuristics used in projection pursuit could be used to identify important pairwise interactions.
Bibliography


Acknowledgements

Thank you Pete Luferenko, Thomas Finley, and Misha Bilenko for enabling me to conduct this project at Microsoft, and for your guidance throughout the process. Thank you Prof. Tomas Palacios for advising me, and making this project possible. Finally, thank you Alyssa Smithh and Allan Ko for pointing me to interesting articles and the COMPAS dataset.