A Structural Risk Minimization Approach to Feature Engineering using a Memetic Genetic Program

Ryan Gryder & Mark Kon

December 29, 2020

Abstract

There are arguments for the future of machine learning not residing in building better models, but in finding the best features for those models. In fact, finding good features should improve the performance of *any* model or classifier. If the feature set is fixed, there is some threshold of performance that will not be breached by the standard collection of ML models, unless a completely new feature is created that in a sense 'expands' the possibilities of the final form of the model function. Here we mathematically formalize the feature engineering problem by framing it within Vapnik's structural risk minimization (SRM). The SRM trade-off of prediction error and complexity is an attractive viewpoint for framing the feature engineering problem as well, since there is an intuitive notion of wanting new features that are not too wildly complicated in form. Framing FE within SRM give it some theoretical foundation, justifies a complexity-penalized search for engineered features in the first place that will increase model performance, and places the large number of existing specific FE methods within a unifying statistical learning-based framework. We also frame the entire problem of finding features + model as a twice-applied SRM. Considering the practical side of implementing feature engineering, many search procedures are 'global' in nature and suffer from missing nearby optimal features. The main feature engineering approach of tree-based Genetic Programming, an evolutionary algorithm, unfortunately suffers from this. The main difficulty with injecting a local search step in FE is the space is not well-defined and there is not an appropriate distance metric to base the local search on. In this work, we propose a local search step within tree-based GP for FE that searches locally in terms of tree representations using the tree edit distance metric.

1 Introduction

Feature engineering and its strengths. Feature engineering is the general process of taking an original set of features for a particular prediction/classification problem and transforming them into a new feature set that is in some sense 'better'. We refer to

feature engineering broadly as including both feature selection (subset of original features) and feature construction (functions of original features). Feature engineering is very useful for dimensionality reduction [SOURCE] and when the original features possess little discriminatory information about the response [SOURCE]. When nonlinear feature engineering is paired with a linear predictive model or vice-versa, training accuracy is improved in most cases [SOURCE].

Drawback to feature engineering: too many specific methods. However, there are two main drawbacks to feature engineering. One is that due to the large amount of specific methods, it is hard to decide which to use in a given application, and each method is limited in the types of features it can create. By definition, any feature selection procedure is limited by only selecting a subset of the original features. As for feature construction: PCA can only create linear functions of the original feature [SOURCE]; autoencoders limit the types of engineered features by the choice of activation function [SOURCE]; kernel PCA limits the type of nonlinear features by the choice of kernel [SOURCE]. The rigidity of the methods forces you to commit to certain types of features. If you want to consider and compare multiple types of engineered features, the only option is to use multiple methods and compare the feature sets by fitting models on each set which can by costly. There is one feature engineering method that does allow flexibility in terms of the types of feature functions considered: Genetic Programming. We will return to this method shortly, but it unfortunately still suffers from a separate drawback of feature engineering.

Another drawback: overfitting and generalizibility. The second drawback is that feature engineering is prone to overfit models particularly when the engineered features involve many terms [SOURCE] or the procedure uses training and testing data, which is a common mistake [SOURCE]. When one wants to consider generalizibility of a model fit on engineered features, it is only after the features are found and the model is fit that one usually investigates prediction on unseen test data. Overfitting is only really considered in reference to the model rather than the engineered features, and there is not yet a procedure to take into account the possibility of over-dependence on training data inside of the feature engineering process itself. There is no mathematical framework for the generalizability of feature engineering as an optimization problem, and the only existing methods to ensure engineered features generalize well to test data is to place a hard limit on the 'size' of a feature in terms of the number of terms [SOURCE].

SRM for feature engineering. Structural risk minimization - as an extension for empirical risk minimization - is a mathematical framework for the generalizability of learning models and prevention of overfitting [SOURCE]. Under certain conditions, a learning model's test error is bounded by a combination of training error and complexity. In practice, multiple kinds of models are compared by looking at their trade-off between training error and model complexity. If we can establish an SRM-like framework for feature engineering, than both of the previously drawbacks can be mitigated. SRM has the specific benefits of formalizing generalizibility of models and comparing multiple kinds of models; a successful SRM-like procedure for feature engineering will do the same.

The SRM-like criteria. What would such an SRM-like procedure for feature engineering look like? We propose a criteria for finding the best set of engineered features that improves the performance of *any* model fit on those features with respect to SRM. That is,

our engineered feature criteria reduces training error and complexity of a model fit on the features and improves the generalizibility of the model. Our criteria for an optimal set of engineered features is the simultaneous maximization of the mutual information between the new feature set and response and minimization of the complexity of the new feature set. The former condition decreases training error from a model fit on the new features, and the latter decreases the complexity of the model fit on the new features.

VC dimension as feature complexity. While information-based feature engineering is not new [SOURCE], considering the 'complexity' of engineered features is new aside from a hard limit on their 'size'. When quantifying the complexity of a particular model, it is usually in terms of the parameter estimates, e.g., ridge, lasso, elastic net. [SOURCE] [INSERT wigglyness norm] A parameter-less measure of complexity that can be applied to engineered feature functions is VC dimension, which is a measure on how many points the function can 'shatter' [SOURCE]. VC dimension results in a natural hierarchy of the complexity of different function classes: [INSERT hierarchy]. VC dimension was the traditional measure of the complexity of a learning model in classical SRM. In this work, we prove a nice result relating the VC dimension of an engineered feature set to the VC dimension of a model fit on the new feature set.

Genetic programming as the search. We also consider the practical side of finding optimal engineered features according to our criteria. A common and popular search method is tree-based Genetic Programming, which is an evolution algorithm that evolves populations of engineered feature functions represented as symbolic expression trees [SOURCE]. Nodes in the trees can be operations, constants, and the original input features; this allows for great flexibility in the types of feature functions discoverable. We describe how a GP procedure can incorporate our optimal feature criteria.

Injecting a local search. A weakness of GP and an open problem for feature engineering is the need for a local search. Evolution algorithms in general suffer from being global search procedures that can miss nearby local optima [SOURCE]. It has been proposed that a local search could enhance the results of feature engineering procedures [SOURCE]. While this has found success in feature selection [SOURCE], a local search step for feature construction procedures has yet to be implemented; the difficulty lies in quantifying the distance between functions of the original features. In this work, we propose a local search step within GP for feature engineering that uses a metric for the distance between tree structures. Numerical results are presented on EEG data, and our memetic SRM-like feature engineering procedure is competitive with the recent benchmark results.

2 Structural Risk Minimization of Feature Engineering

2.1 Feature engineering as function composition

Feature engineering expands the set of model functions. Given a fixed set of features $\mathbf{x} = (x_1, x_2, \cdots, x_p)$ and response y, it is a known phenomenon that many widely used machine learning models will predict the response with similar levels of accuracy. (SOURCES) A particular model such as cubic spline or neural net might perform the 'best', but this margin of success compared to the set of models is usually small. That is, there is some

'threshold' of model accuracy that will not be breached by the handful of standard models unless a very particular problem-specific model is used or, alternatively, new features are considered (SOURCES). For example, Guo et al. (2011) used a genetic program to engineer features for EEG data classification. [SOURCE] On the original untransformed features, classifiers were unable to breach an accuracy of 67% for classification of normal vs seizure-free vs seizure EEG signals. However, after their genetic program determined that $\frac{x_1}{x_2}$ was a 'good' potential new feature, Guo achieved a tremendous boost in classification accuracy to 94%.

Can we explain this phenomenon mathematically? Without augmenting the feature set $\mathbf{x}$, we think of all models as functions $f(\mathbf{x})$ that one would typical consider in a data prediction scenario as forming a set, $\mathcal{F}$ of possible model functions. If the response y is continuous, this $\mathcal{F}$ would most likely contain linear functions (regression), polynomial functions (regression and spline), piecewise constant functions (decision tree), etc.. However, $\mathcal{F}$ most likely does not contain any $f : X \to Y$ with a component involving $\frac{x_1}{x_2}$. Such model functions do not exist within the set $\mathcal{F}$, but by using $\frac{x_1}{x_2}$ as a new feature, we have expanded the set of model functions to a new set $\mathcal{H}$ which contains linear functions, polynomial functions, piecewise constant functions, etc., *composed* with functions like $\frac{x_1}{x_2}$. We think of feature engineering as expanding the set of possible model functions through function composition. Traditionally, an engineered feature set $\mathbf{x}^* = g(\mathbf{x})$ is viewed as a change of variables or change in feature space. The set $\mathcal{F}$ of possible model functions takes as input a new variable $\mathbf{x}^*$, but the function forms themselves, i.e., linear functions, polynomial functions, etc., remain the same. On the other hand, we adopt the viewpoint that engineered features $g(\mathbf{x})$ viewed as functions themselves expands the set of possible model functions to a new collection $\mathcal{H}$ that now contains functions h where $h = f \circ g$. Again referencing the previous example, we think of the significant boost in model performance from using new feature $\frac{x_1}{x_2}$ a result of the novel model function $h = f(g(\mathbf{x})) = f(\frac{x_1}{x_2})$ formed by the function composition. Suppose f is a linear function. Traditional feature engineering would label new feature $\frac{x_1}{x_2}$ as $\mathbf{x}^*$, and the model $f(\mathbf{x}^*)$ is still a linear function. However, if we view $\frac{x_1}{x_2}$ as is, then the resulting model function $f(\frac{x_1}{x_2})$ is no longer a linear function!

Our point of view is that in a feature engineering context, the final or complete model function is the result of the function composition of engineered feature function g with a 'standard' machine learning model f. This framework is intuitive for explaining why feature engineering is so powerful, e.g., linear functions f of engineered feature functions $g(\mathbf{x}) = \frac{x_1}{x_2}$ expand the possibilities for resulting model functions mapping data $\mathbf{x}$ to response y. Using our established notation, we call g the engineered feature function, f the model function, and $h = f \circ g$ the *complete model function*.

2.2 Structural Risk Minimization

Review of classic SRM. We would like to further formalize the feature engineering problem by including it in the paradigm of statistical learning theory, namely structural risk minimization (SRM). As we see later, viewing feature engineering from the point of view of function composition allows for this formulation. In general terms, SRM is the

idea that a particular learning model's prediction error on unseen test data is bounded by prediction error on training data plus a quantity related to its complexity, i.e., the bias-variance tradeoff. SRM provides the mathematical theory for why this bound holds, the conditions for it to hold, and specifics for how tight this bound is for certain problems. In practice, if one wishes to choose a model from a set according to SRM, the best model in terms of performance on future unseen data simultaneously minimizes training error and model complexity.

Here we briefly review classical SRM a la Vapnik [SOURCE]. Suppose the training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n)$ of n observations is drawn from joint probability distribution $P(X, Y)$ on random variables X and Y. Denote the particular class of considered model functions f as $\mathcal{F}$. If $\mathcal{X}$ is the range of X and $\mathcal{Y}$ is the range of Y, then $f : \mathcal{X} \to \mathcal{Y}$. We consider $\mathcal{X} = \mathbb{R}^p$ and $\mathcal{Y} = \mathbb{R}$ or $\{0, 1\}$, i.e., the regression and classification settings. Denote $L(\cdot, \cdot)$ as the loss function. Define the risk $R(f)$ and empirical risk $R_{emp}(f)$ of model f as the following:

$$R(f) = \int L(Y, f(X)) dP(X, Y) \tag{1}$$

$$R_{\text{emp}}(f) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i,)) \tag{2}$$

The true optimal f is one that minimizes risk $R(f)$ since this is the average loss over the entire distribution $P(X, Y)$. But since the training data is all that is available, one finds f that minimizes the empirical risk $R_{emp}(f)$, i.e., the average training loss. This is known as Empirical Risk Minimization (ERM). Vapnik was concerned with the consistency of ERM. That is, if $f_0 \in \mathcal{F}$ minimizes R_{emp}, does this minimum empirical risk converge in probability to the minimum risk R as $n \to \infty$? Is the asymptotic rate of convergence fast and does it hold for any probability distribution $P(X, Y)$ generating the data? Vapnik's main result is that if the loss function $0 \leq L(Y, f(X)) \leq A$ is bounded over all $f \in \mathcal{F}$, then ERM is consistent and rapidly converging for any probability distribution as long as the *VC dimension* of the losses $L(Y, f(X))$, $f \in \mathcal{F}$ is finite. We define VC dimension below for a set of indicator functions and a set of bounded real-valued functions generally denoted as $k(z)$, $k \in \mathcal{K}$ that don't necessarily need to be proper loss functions.

Definition. VC dimension of indicator functions. The set of indicator functions $k(z), k \in \mathcal{K}$ *shatters* a set of m points/vectors $z_1, z_2, \cdots, z_m$ if they can be separated in all 2^m possible ways by functions in $\mathcal{K}$. The *VC dimension of the set of indicator functions* is the maximum number m of points that can be shattered by the set of functions. If for any positive integer n, there exists a set of n points that can be shattered by $k(z), k \in \mathcal{K}$, then the VC dimension is infinity.

Definition. VC dimension of real-valued functions. Let $A \leq f(z) \leq B$, $f \in \mathcal{F}$ be a set of bounded real-valued functions. Consider the set of indicator functions $\mathbb{I}[f(z) - c]$, $f \in \mathcal{F}$ where $\mathbb{I}$ is the step function and $A \leq c \leq B$. The *VC dimension of the set of real-valued functions* $f(z)$, $f \in \mathcal{F}$ is defined as the VC dimension of the set of indicator functions.

The VC dimension measures the complexity of a set of functions by looking at the number of points it can perfectly classify; it intuitively measures the capacity of a set of model functions to overfit training data. The VC dimension relates empirical risk

(training error) to risk (test error) via the following inequality. Given bounded loss functions $0 \leq L(Y, f(X)) \leq A$, $f \in \mathcal{F}$, with probability at least $1 - \eta$, the following holds for all $f \in \mathcal{F}$:

$$R(f) \leq R_{emp}(f) + \frac{A\epsilon}{2}\left(1 + \sqrt{1 + \frac{4R_{emp}(f)}{A\epsilon}}\right) \tag{3}$$

where $\epsilon = 4\frac{m\left(\ln\frac{2n}{m}+1\right)-\ln\eta}{n}$, n is the size of the training data, and m is the VC dimension of losses $L(Y, f(X))$, $f \in \mathcal{F}$.

We say that a model f generalizes well if it keeps risk $R(f)$ small. This means that is minimizes loss over the entire probability distribution which the data comes from, i.e., it can perform well on unseen testing data as long as it comes from the same distribution as the training data. Based on the above inequality, a model will generalize will if it simultaneously minimizes error on training data and VC dimension complexity; the inevitable trade-off between the two is the principle behind structural risk minimization. In practice, SRM amounts to comparing multiple sets of models $\mathcal{F}_k$ each with a VC dimension m_k. In a classification setting, these $\mathcal{F}_k$ can be a logistic regression model, k-nearest neighbor, support vector machine, etc. The optimal choice of f is one that minimizes the right-hand side of the above inequality given its error on the training data and VC dimension of the set of functions it belongs to. This f could be a particular parameter-valued logistic regression model for example.

2.3 Feature Engineering in SRM

Insert the function composition idea into SRM. To insert feature engineering into the SRM framework, we consider function classes formed by the function composition of engineered feature functions with model functions. If $\mathcal{G}$ is a particular class of engineered feature functions $g : \mathbb{R}^p \to \mathbb{R}^q$ mapping p original features to q new features and $\mathcal{F}$ is particular class of model functions $f : \mathbb{R}^q \to Y$, their composition defines a class of complete model functions $\mathcal{H} = \{f \circ g | f \in \mathcal{F}, g \in \mathcal{G}\}$. If the VC dimension m of the function composition class $\mathcal{H}$ is finite and $L(y, f(g(\mathbf{x})))$ is totally bounded over $f \circ g \in \mathcal{H}$, then we still have:

$$R(f \circ g) \leq R_{\text{emp}}(f \circ g) + m \cdot \epsilon \tag{4}$$

In this setting, SRM amounts to comparing function classes $\mathcal{H}_{ij}$ with VC dimensions m_{ij}. Here, $\mathcal{H}_{ij} = \{f \circ g | f \in \mathcal{F}_i, g \in \mathcal{G}_j\}$ where there is a collection of considered classes of model functions $\mathcal{F}_i$ composed with a collection of classes of engineered feature functions $\mathcal{G}_j$.

Our layered SRM procedure. The function composition makes it difficult to directly find the optimal $f \circ g$ that minimizes empirical risk and VC dimension simultaneously. The most natural procedure is to find the optimal engineered feature function g first according to some separate criteria, then find the optimal model function f when composed with this

specific g using SRM. The second step can be accomplished by a typical SRM procedure as long as the resulting complete model function classes from the composition with g have finite VC dimensions and totally bounded losses. The first step for finding optimal g requires careful consideration: our proposed criteria will be elaborate on later. For the moment, our detailed procedure for finding optimal $f \circ g$ is the following:

1. Among considered classes of engineered feature functions g, find the optimal g_0 according to some criteria (elaborated later).
2. Given considered function classes $\mathcal{F}_i$ for model function f, ensure that function composition classes $\mathcal{H}_i = \{f \circ g_0 | f \in \mathcal{F}_i\}$ for the complete model function satisfy the assumptions for SRM:
   (a) VC dimension of $\mathcal{H}_i$ is finite for all i.
   (b) $L(y, f(g_0(\mathbf{x})))$ is totally bounded over $f \circ g_0 \in \mathcal{H}_i$ for all i.
3. Perform SRM to find optimal $f_0 \circ g_0$.

Criteria for engineered feature functions. Now we explain our proposed method for finding a set of optimal engineered features. The ultimate goal of this procedure is to find the optimal complete model function $f_0 \circ g_0$ that simultaneously minimizes empirical risk and VC dimension. As pointed out earlier, we cannot hope to perform a usual SRM to accomplish this directly, so the best we can do is to use a criteria for finding g_0 that enhances the results when using a typical SRM to find optimal f_0 composed with this g_0. That is, the optimization criteria for g_0 should account for the ultimate SRM goal of simultaneously minimizing $R_{emp}(f \circ g_0)$ and VC dimension of $\mathcal{H}_i = \{f \circ g_0 | f \in \mathcal{F}_i\}$ among our considered classes of model functions $\mathcal{F}_i$. This will get us close to finding the optimal $f_0 \circ g_0$ overall after SRM is performed to find f_0 given g_0. In simple terms, we seek an engineered feature function criteria that will reduce the training error when a model function is fit on the new features as well as keep the VC dimension of the complete model function small. With this in mind, our criteria for the optimal engineered feature function g is the following: simultaneous maximization of the mutual information between random variables $g(X)$ and Y and minimization of the VC dimension of the class of functions which g belongs to. We discuss each of these components in turn.

Definition. Let X and Y be two continuous random variables with ranges $\mathcal{X}$ and $\mathcal{Y}$, respectively. Denote their joint probability density $P_{(X,Y)}$ and marginal densities P_X and P_Y. Then the *mutual information* between X and Y is defined as

$$I(X, Y) = \int_{\mathcal{Y}} \int_{\mathcal{X}} P_{(X,Y)}(x, y) \log \frac{P_{(X,Y)}(x, y)}{P_X(x) P_Y(y)} dx dy \tag{5}$$

Note that in the classification setting, Y is a discrete random variable with a probability mass function P_Y over its two possible values. In this case, the outer integral is replaced with a sum of two terms for each of the values of Y.

Mutual information. The motivation is that if $I(g(X), Y)$ is large, then the engineered feature set $g(\mathbf{x})$ where training data $\mathbf{x}$ is considered as a realization of random variable

X greatly reduces the uncertainty about response y. Therefore, a model fit on these informative engineered features should have a reduced training error compared to other possible engineered features. This result can be made rigorous in terms of Bayes' error in the classification setting, which is the smallest possible error of any classifier [SOURCE]. The following lemma relates the bounds on Bayes' error when predicting y from $\mathbf{x}$ to the mutual information between random variables Y and X.

Lemma 1. Consider random variables X and Y where Y is binary-valued. Define B_e as Bayes' error when classifying n realizations of Y from n realizations of X, i.e., n is the number of observations. If $I(X, Y)$ is the mutual information between X and Y and $H(Y)$ is the entropy of Y, then the following inequality holds:

$$\frac{1}{\log N}(H(Y) - I(X, Y) - 1) \leq B_e \leq \frac{1}{2}(H(Y) - I(X, Y)) \tag{6}$$

Proof: Each bound is a known named result. [SOURCE]

This lemma implies that if we search for engineered features with maximal mutual information to the response, then this will reduce the bounds on Bayes' error on the training data, which is the best classification performance achievable. However, we do need to consider the computation of mutual information, which requires knowledge of the probability distributions of X,Y, and $g(X)$ for considered g. Since the information is often unavailable, we need to use density estimation when applying this feature engineering procedure on actual data. We explain our density estimation procedure in the Numerical Results section.

VC dimension of function composition. The second component of our criteria for the optimal engineered feature function is that it belongs to a function class with minimal VC dimension. The motivation is that a small VC dimension of the engineered feature function class that optimal g_0 belongs to should also lead to a small VC dimension of the considered complete model function classes $\mathcal{H}_i = \{f \circ g_0 | f \in \mathcal{F}_i\}$ as a result of the function composition. Before we introduce and prove a lemma validating this claim, we must clarify what we mean by the VC dimension of a class $\mathcal{G}$ of engineered feature functions g. As noted previously, we consider feature functions $g : \mathbb{R}^p \to \mathbb{R}^q$ returning a set of q new engineered features. However, VC dimension is only defined for classes of scalar functions, so we can only define the VC dimension of the function class $\mathcal{G}_i$ of each individual engineered feature $g_i : \mathbb{R}^p \to \mathbb{R}$ where $g(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}), \cdots, g_q(\mathbf{x}))$ and $\mathcal{G} = \mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_q$. Our criteria for an optimal engineered feature function g based on VC dimension is that the sum of the VC dimensions of the individual components, i.e., the function class of each individual scalar engineered feature, is small. The motivation for wanting a minimal $\sum_{i=1}^{q} VC_{dim}(\mathcal{G}_i)$ is that this will reduce the upper bound on the VC dimension of the class of complete model functions $f \circ g$ where $g \in \mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_q$. We prove this in the following lemma.

Lemma 2. Let $\mathcal{F}$ be a class of functions from $\mathbb{R}^q \to \mathbb{R}$ and $\mathcal{G}_i$ a class of functions from

$\mathbb{R}^p \to \mathbb{R}$ where $i = 1, \cdots, q$. Then their VC dimensions have the following relationship:

$$VC_{dim}(\mathcal{F} \circ (\mathcal{G}_1 \text{ X } \mathcal{G}_2 \text{ X } \cdots \text{ X } \mathcal{G}_q)) \leq c \cdot [VC_{dim}(\mathcal{F}) + \sum_{i=1}^{q} VC_{dim}(\mathcal{G}_i)] \tag{7}$$

Proof: I think I have done this.

Lagrangian problem for finding optimal engineered features. When formulating our criteria for a a set of optimal engineered features, we will consider a collection of r classes of scalar functions $\{\mathcal{G}_k\}_{k=1}^r$ from which the q new engineered feature functions will be drawn from. From this collection of r function classes, we seek the q function classes $\mathcal{G}_1, \mathcal{G}_2, \cdots, \cdots, \mathcal{G}_q$ that simultaneously minimize their sum of VC dimensions and contain q functions that together (their Cartesian product) maximize their mutual information with the response. This can be formulated as the following Langrangian double maximization problem:

$$\max_{\mathcal{G}_i \in \{\mathcal{G}_k\}_{k=1}^r} \quad \max_{g \in \mathcal{G}_1 \text{ X } \mathcal{G}_2 \text{ X } \cdots \text{ X } \mathcal{G}_q} I(g(\mathbf{x}), y) - \lambda \cdot \sum_{i=1}^{q} VC_{dim}(\mathcal{G}_i) \tag{8}$$

Each term in the above maximization for finding the optimal engineered features is associated with a component from the SRM procedure for finding the optimal model function. Maximizing the mutual information between the engineered features and response will reduce the training error of a model fit on those engineered features; minimizing the sum of VC dimensions of the individual engineered features will reduce the upper bound on the VC dimension of the model function composed with the engineered features. The result of the double optimization is (close to) the overall optimal complete model function, i.e., the optimal function composition of a model function with an engineered feature function.

We have to consider the practical difficulties of solving such an optimization problem which are caused by needing to search for a combination of function classes as well as search for a q-dimensional output function. There is also the additional difficulty of computing the mutual information between a one-dimensional and multi-dimensional random variable [SOURCES]. For practical purposes, we consider a simpler alternative optimization problem of finding each of the q scalar output engineered feature functions one at a time. Among the r classes of scalar functions $\{\mathcal{G}_k\}_{k=1}^r$, first we find the single class $\mathcal{G}$ and and function $g \in \mathcal{G}$ that simultaneously maximizes its mutual information with the response and minimizes its VC dimension. This scalar function will be the first engineered feature. Setting this engineered feature aside, we repeat the process for the second engineered feature and so on. We are finding the top q solutions to the following optimization problem:

$$\max_{\mathcal{G} \in \{\mathcal{G}_k\}_{k=1}^r} \quad \max_{g \in \mathcal{G}} I(g(\mathbf{x}), y) - \lambda \cdot VC_{dim}(\mathcal{G}) \tag{9}$$

This optimization problem is easier to solve due to reduced dimensionality of the search space and the computation of mutual information between one-dimensional random variables. This formulation is also attractive in that it makes our search for an optimal engineered feature look very similar to a traditional SRM; the only difference is that mutual information replaces empirical risk.

2.4 Engineered feature search space: symbolic expressions

Here we discuss the search space for the engineered feature functions. Rather than specifying the considered function classes $\mathcal{G}_i$ directly, e.g., linear functions of one variable, polynomials of degree two, etc., we define the larger set from which all of the engineered feature functions will be considered.

The space of all possible functions of existing p features $\mathbf{x} = (x_1, x_2, \cdots, x_p)$ without any restrictions is unfathomably large. However, we want the problem formulation to include as many possibilities of function 'types' as the researcher desires: linear, nonlinear, etc. Borrowing notation from symbolic regression [source], we represent possible functions of initial features as symbolic expressions where the 'symbols' are the initial features, constants (if desired), and functions or operators such as multiplication ('*'), addition ('+'), etc. The set of symbols, $\mathbb{P}$ or primitive set, is a set of basic operations of different arity such as $\{+, -, *, /\}$ denoted O for operator set and a set of input variables of zero arity such as $\{x_1, \cdots, x_p\}$ denoted T for terminal set. Then $\mathbb{P} = O \cup T$ where T can include constants. The search space for the new engineered features is all symbolic expressions $g \in \mathbb{S}$ where $\mathbb{S}$ is the syntactic space defined by $\mathbb{P}$. These g are the same engineered feature functions we've been referring to. The class of functions $\mathcal{G}_i$ that a particular g belongs to will only be considered during the process of actually solving the Langrangian. That is, if a candidate g is considered, say a linear function of k variables, then its function class $\mathcal{G}_i$ is all linear functions of k variables and the VC dimension of this $\mathcal{G}_i$ will be calculated accordingly for the purposes of considering this g as a solution to the Lagrangian.

The main reason for this representation of the space of engineered feature functions is that it lends itself to a particular kind of search method that is the most common for feature engineering, which we elaborate on in the next section. It also allows us the researcher to have simple and direct control over the types of engineered features considered in terms of what they look like algebraically.

3 Search method: Tree-based Genetic Programming

In this section, we discuss a common search algorithm for feature engineering when the search space is formulated as symbolic expressions over an operator set: genetic programming. Here we provide an overview of genetic programming as a particular type of evolution algorithm, review its application to feature engineering and how it can accommodate our specific optimization formulation, and introduce a local search step into the algorithm.

3.1 Evolutionary Algorithms and Genetic Programming

Review Evolution Algorithms. An evolutionary algorithm (EA) is a metaheuristic optimization algorithm where the search for an optimal solution is done in a biology and evolution-inspired manner. The general idea is to 'evolve' a population of solutions until the 'fittest' emerges; the most well-performing or fit solutions are chosen as parents for the next generation in a tournament style, and offspring solutions are formed when a parent solution 'mutates' or two parent solutions 'crossover'. [source] The idea is that well-performing solutions are combined (crossover) and/or altered (mutation) to hopefully get close to the optimal solution. The Darwinian concept of survival of the fittest is applied to the search for an optimal solution to an optimization problem where the objective function is the fitness. Evolution algorithms are global search procedures as the evolution operators lead to significant 'jumps' within the search space, i.e., offspring solutions can be quite different from their parent solutions in terms of form and distance if there is such a notion in the particular problem. This global nature of EA's is seen as an advantage since they rarely get stuck in a basin of a suboptimal local minima. [source] EA's have a rich history of success in a variety of learning problems. [source] Below is the general procedure for an evolution algorithm:

1. Define fitness (objective) function. Set parameters: population size, number of generations, tournament size, probability of crossover, probability of mutation.
2. Randomly generate initial population.
3. Evaluate fitness of current population.
4. Use tournament selection to select parents of next generation.
5. Generate next generation by performing genetic operators (mutation, crossover) on parents.
6. Evaluate fitness of next generation (the offspring).
7. If stopping criteria is satisfied, end. If not, return to step 4.

Review genetic programming and existing applications to FE. There are many subclasses of evolution algorithms where the distinctions are made based on the specific models that are evolved, representation of solutions, fitness function, evolution operators used, etc. A common evolution algorithm for feature engineering is genetic programming (GP). The distinguishing feature of GP is that the solutions to the problem, i.e., the individuals in the population, are represented as trees. Initially proposed by Kyoza [source], it was originally used for solving the symbolic regression problem. Drawing from our previous notation, suppose one is searching for the best engineered feature as a symbolic expression involving terminal set $T = \{x_1, x_2, x_3, 0.5\}$ and operator set $O = \{+, -, *, /\}$. Then candidate feature $x_1 * (x_2 * 0.5 - x_3)$ can be represented by the following tree:

Within the tree, terminal nodes take on elements in the terminal set T and internal nodes takes on elements in the operator set O in order for the tree to produce an adequate

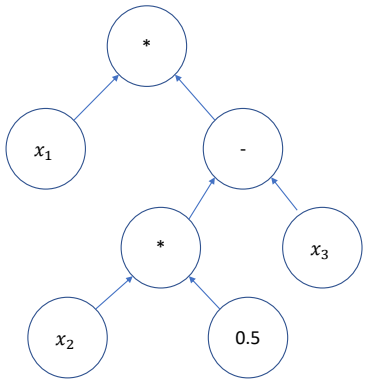

Figure 1: Caption

symbolic expression output. A GP algorithm takes the most fit trees and applies the typical genetic operators to form offspring trees; the genetic operators augment trees by changing their node values and edge connection topology.

Genetic programming has been used for feature engineering in a variety of applications [source]. Fitness measures, i.e., the objective function, can be generally divided into wrappers and filters. Wrappers evaluate the fitness of features by fitting a learning model on the features while filters look at the intrinsic characteristics of the features and how they relate to the response, e.g, mutual information and Fisher criterion [source]. Overfitting and feature complexity is usually addressed by setting a limit on tree length or computing AIC, BIC, etc. [source]. To our knowledge, VC dimension has yet to be used as a complexity measure in feature engineering. Additionally, an open problem is injecting a local search step into the feature engineering framework. In Section 3.2, we expand on how a local search step within a feature engineering framework is desirable and propose a particular local search method inspired by similar efforts in symbolic regression. This local search operator is compared to the typical global search operators of crossover and mutation.

3.2 Global and Local Search Operators

Global search operators. Within evolution algorithms, the most common evolution operators are crossover and mutation. To form the next generation of candidate solutions for an optimization problem, offspring solutions are formed by current solutions with high fitness, i.e., parents, and performing mutation and/or crossover on them. Crossover refers to the general procedure of taking two parent solutions and somehow combining them to form a new offspring, while mutation refers to an augmentation of a single parent solution. These operators are motivated by the biological source of genetic differences

between parents and offspring in a population of biological organisms. The specifics of crossover and mutation are dependent on the specific evolution algorithm. In the genetic programming setting, the operators work on the level of the tree structure of the candidate solutions. Crossover replaces a subtree of a parent with a subtree of an additional parent; mutation takes a node within a parent tree and replaces the subtree with a newly generated random one. [source]

It is not difficult to see why these operators are denoted as global operators. Offspring trees have the potential to represent a symbolic expression that is vastly different from their parent(s) both in form and fitness. That is, these global operators can make large sudden 'jumps' within the search space. [source] While this is an advantage of the algorithm from the point of view of getting stuck in suboptimal local minima, it can be a disadvantage if a parent solution is close to an optimal solution. As a pitfall of genetic programming and evolutionary algorithms in general, the global nature of these search algorithms has led to the development of *hybrid or memetic evolution algorithms* that insert a local search step into the general procedure above.

Local search operator. Hybrid/memetic evolution algorithms with a local search step have been successful in some applications [source]. The general idea is that before a parent solution undergoes the global evolution operator of mutation or crossover to produce an offspring for the next generation, one searches locally around the parent for a potentially more fit neighbor as a replacement. The procedure for a hybrid evolution algorithm is as follows with the local search additions in bold:

1. Define fitness (objective) function. Set parameters: population size, number of generations, tournament size, probability of crossover, probability of mutation, **local search neighborhood size**.
2. Randomly generate initial population.
3. Evaluate fitness of current population.
4. Use tournament selection to select parents of next generation.
5. **Replace each parent with most fit nearby solution if such exists.**
6. Generate next generation by performing genetic operators (mutation, crossover) on parents.
7. Evaluate fitness of next generation (the offspring).
8. If stopping criteria is satisfied, end. If not, return to step 4.

In terms of feature engineering, local search steps have been applied only to the case of feature subset selection. Particularly useful when the number of original features is very large, EA's for feature selection represent a candidate feature subset as a binary vector, i.e., $[0, 1, 1, 1, 0, ..., 0,]$ representing the inclusion/exclusion of original features. A local search around a candidate feature subset takes the binary vector and considers changing only 1 bit for example. This has been demonstrated successfully in [source]. If a local search

step can be successful for feature selection, then why not feature extraction, i.e., general feature engineering? This is an open problem recently posed by [source].

Tree edit distance metric for nearby features. The difficulty in performing a local search in feature engineering lies in quantifying 'nearby' features with a valid notion of distance. A natural choice in quantifying the distance between symbolically expression features is to use their tree representation. If there is a metric for calculating the distance between symbolic expression trees, then that can guide the local search. Such a metric is known as *tree edit distance* [source].

Definition. Tree edit distance. Given two symbolic expressions $g_i, g_j \in \mathbb{S}$ where $\mathbb{S}$ is the syntactic space defined by symbol set $\mathbb{P}$, then the *tree edit distance* between g_i and g_j, denoted as $\|g_i - g_j\|_{\mathcal{T}}$, is the minimum number of tree edits to transform g_i into g_j. Tree edits are node value changes, edge deletions, and edge insertions.

We hope that a local search based on metric $\mathcal{T}$ for high-quality nearby features mimics what a traditional gradient procedure would do. We cannot hope to calculate an actual gradient of the objective function, i.e., fitness, with respect to symbolically expressed functions g_i within our formulation. A practical solution is that when a candidate well-performing feature function is found, we perform a very small number of tree edits, i.e., look locally, and see if we can find a more fit engineered feature. That is, we move a small distance in the metric space of symbolic expression trees in the direction of maximum change in fitness, i.e., the 'gradient'.

Local search as a point mutation. When looking back at the global evolution operators of crossover and mutation within GP, it's clear that the operations make quite significant changes to the tree structure of a given parent tree both in terms of edge connections and node values. By comparison, the simplest and least disruptive changes to a parent tree structure would be changing a single node value or inserting/deleting a single edge with the former being the least disruptive. Our proposed local search step is changing the value of a single node in a given tree, which we call a *point mutation* to contrast with the usual subtree mutation global search operator. This local search operator can be represented with respect to the tree edit distance metric $\mathcal{T}$ as the following: given candidate parent symbolic expression g_p, replace g_p with most fit g_{p^*} where $\|g_p - g_{p^*}\|_{\mathcal{T}} = \epsilon$ and tree edits are only node value changes. In our case, we set $\epsilon = 1$. That is, before a candidate parent undergoes crossover and/or mutation, we consider replacing the parent with a nearby neighbor that differs in the node value of one node.

There is a computational consideration in that if a candidate tree consists of a nodes and the primitive set has b number of symbols, then the point mutation local search requires checking the fitness of $(b-1)a$ amount of nearby symbolic expression trees. This can become quite large for long trees and a large number of symbols, and we consider this challenge when applying our method to data in the following section.

3.3 Specifics of our Memetic GP FE procedure

When applying a memetic GP with our feature engineering criterion, the fitness of a symbolically-expressed engineered feature g is exactly our objective function in Eq. (9): $I(g(\mathbf{x}), y) - \lambda \cdot VC_{dim}(\mathcal{G})$ where $\mathcal{G}$ is the function set that g belongs to. Since the function sets are not determined beforehand but rather a result of the considered g generated

by the evolutionary algorithm, calculating the VC dimension of the set g belongs to is not straightforward. If g is a long algebraic expression involving a variety of arithmetic operations, it is not clear what the larger function set even is. This problem has been considered by [source] when using GP for symbolic regression. The authors demonstrate that when algebraic expressions are expressed as symbolic expression trees, an estimate for their VC dimension is the number of nodes that are not '+' or '-', which captures the degree of nonlinearity in the function. We adopt this VC dimension estimation in our feature engineering procedure.

As for computing the mutual information between an engineered feature function of the training data $g(\mathbf{x})$ and the response y, we assume the probability distribution generating the data is unknown, and so it requires estimation. We use kernel density estimation with a Gaussian kernel to estimate the $P_{g(X)}, P_Y, P_{(X,Y)}$ based on the training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. The values of parameters for the Gaussian kernel density estimation as well as the Lagrangian multiplier λ in the fitness function are tuned during the application to an actual data set.

4 Results

Summary of section. In this section, we present results demonstrating the effectiveness of our memetic GP feature engineering procedure in terms of finding features with higher mutual information with the response than those found by a non-memetic GP feature engineering procedure as well as principal component analysis (PCA). We also show how our proposed feature engineering criterion ultimately leads to increased prediction accuracy on testing data when a classifier is fit using the engineered features, and we again make a comparison to non-memetic GP, PCA, as well as the case where no feature engineering is done and all original features are used. Test set accuracy is used to compare feature engineering methods since our proposed criterion is driven by Structural Risk Minimization and the ultimate goal of increased generalizability.

The data. We apply our proposed feature engineered methodology to electroencephalogram (EEG) wave data collected from the DREAMS project. [source] EEG waves are used to diagnose sleep disorders such as insomnia and sleep apnea, determine the presence of epilepsy and seizures, and identify the visible waveforms of sleep spindles and K-complexes. This particular data set is concerned with the identification of sleep spindles; abnormalities in their form can indicate neuropathologies or sleep disorders. Sleep spindles are of great interest in the medical community because they have a multitude of theoretical and clinical implications in understanding brain activity during sleep and the development of disorders. [source]

Since visual inspection is typically used to identify these structures, which can be prone to human error and biases, there is a need for learning models to automatically analyze EEG data and identify sleep spindles. However, this is made difficult by the large number of features of the signals which is where dimensionality reduction via feature engineering techniques comes into play. [source] As stated by Ivert, the construction of a reduced set of new features as functions of the original features is preferable to choosing a subset of the original features in this context; much information is gained about the

relationships between the original variables when the engineered feature functions are viewed as 'rules'. [source]

Previous efforts on this data and comparison benchmarks. Using GP-based feature engineering in EEG signal data was first successfully done by Guo in the context of detecting seizures. [source] Here, we follow the recent benchmark work of Miranda in using GP-based feature engineering for the more difficult problem of identifying the presence of sleep spindles, which is a binary classification problem. Miranda's GP procedure used area under the ROC curve (AUC) of different classifiers to measure the fitness of engineered feature sets, i.e., a wrapper type evolution algorithm, and only penalized complexity by imposing a hard tree depth limit of 10 nodes, i.e., 10 symbols in the algebraic expression of an engineered feature. [source] We hope to improve on the results of Miranda by using mutual information as fitness, imposing a complexity penalty of VC dimension, and injecting a local search step into the engineered feature search.

Parameter settings and algorithm specifics. As for specific parameter settings within our GP-feature engineering procedure for this EEG data set, our population size of 1000 engineered features is evolved over 20 generations. Tournament selection with a size of 20 is used to select individuals for evolution and continuation to the next generation. Individuals, the engineered feature trees, experience evolution via the crossover and subtree mutation operators with a probability of 0.85 and 0.15, respectively. A point mutation local search is performed around each individual that is selected for the next generation: for each single node in a candidate tree, we consider replacing its symbol with a random one from the primitive set with the same arity. Not every point mutation is considered due to a significantly increased computation time. Out of all of these trees that are 1 tree edit distance away from the original, the most fit continues to the next generation (which may be the original feature tree). Our set of operators is $O = \{+, -, *, /, \ln, \sqrt{}\}$ to mimic Miranda. Note that '/', 'ln', and '$\sqrt{}$' are *protected* operators to prevent division by zero, natural logarithm of zero, and the square root of a negative number. The algorithm settings are summarized in the following Table.

Algorithm parameter	**Set value**
Population size	1000
Number of generations	20
Tournament size	20
Crossover probability	0.85
Subtree mutation probability	0.15
Point mutation type	random replacement
Operators	$+, -, *, /, \ln, \sqrt{}$

Experiment specifics. The data set of 75 attributes was randomly split into a training and test set using 70% and 30% of the signal samples, respectively. Two different specialists classified each sample as positive or negative depending on the presence of sleep spindles. A training signal sample is labelled positive if both specialists agree on the presence of sleep spindles while a test signal sample is labelled positive if either specialist identified sleep spindles. Additionally, the training data was randomly balanced while the test data was not resulting in a difficult generalizability problem. This training/test split is

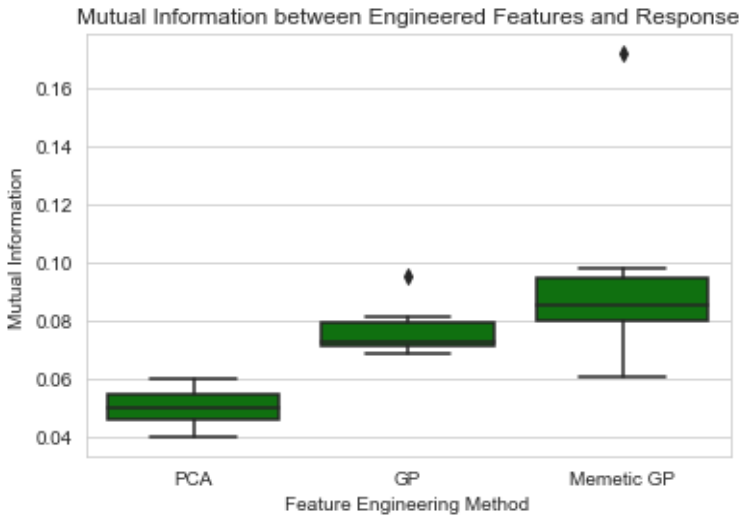

Figure 2: Mutual information

performed 10 times and results are averaged for each experiment. The four experiments are the following: all 75 original features, 10 PCA features, 10 GP features using our criterion, and 10 memetic GP features using our criterion with a local search. The feature sets are fed into five classifiers: support vector machine (SVM), Naive Bayes (NB), K-nearest neighbor (K-nn), decision tree (DT), and multi-layer perceptron (MLP). [insert parameters of each]

Success in maximizing mutual information. First, we compare PCA vs GP vs memetic GP in their ability to find engineered features with maximal mutual information with the binary response. The boxplots are constructed using the average mutual information for the 10 engineered features from each feature engineering method. The repetition of the train/test split results in the boxplot.

Despite displaying higher variability in the mutual information of engineered features, the local search step does indeed allow the memetic GP search procedure to find engineered features with higher mutual information that are missed by a typical non-memetic GP search. It is worth noting that even a GP without a local search that employs our feature criterion is still able to out-perform PCA in the mutual information of features.

Success in generalizability. Next, we compare the performance of five different classifiers using engineered feature sets from PCA, GP, memetic GP, and the original 75 features. We wish to assess generalizability, so performance is measured using prediction accuracy on the test data set from the repeated training/test splits.

The most notable result from these boxplots is that the memetic GP feature engineering procedure with our proposed criterion leads to high classification accuracy on test data that is consistent across all classifiers. For the other feature engineering procedures, including non-memetic GP, poor performance from one or two classifiers is evident. We also note that when our feature engineering criterion is used within GP, increased test accuracy is achieved by NB and DT classifiers, and this performance boost is further enhanced by a local search.

Benchmark comparison. Lastly, we compare our feature engineering criterion to the

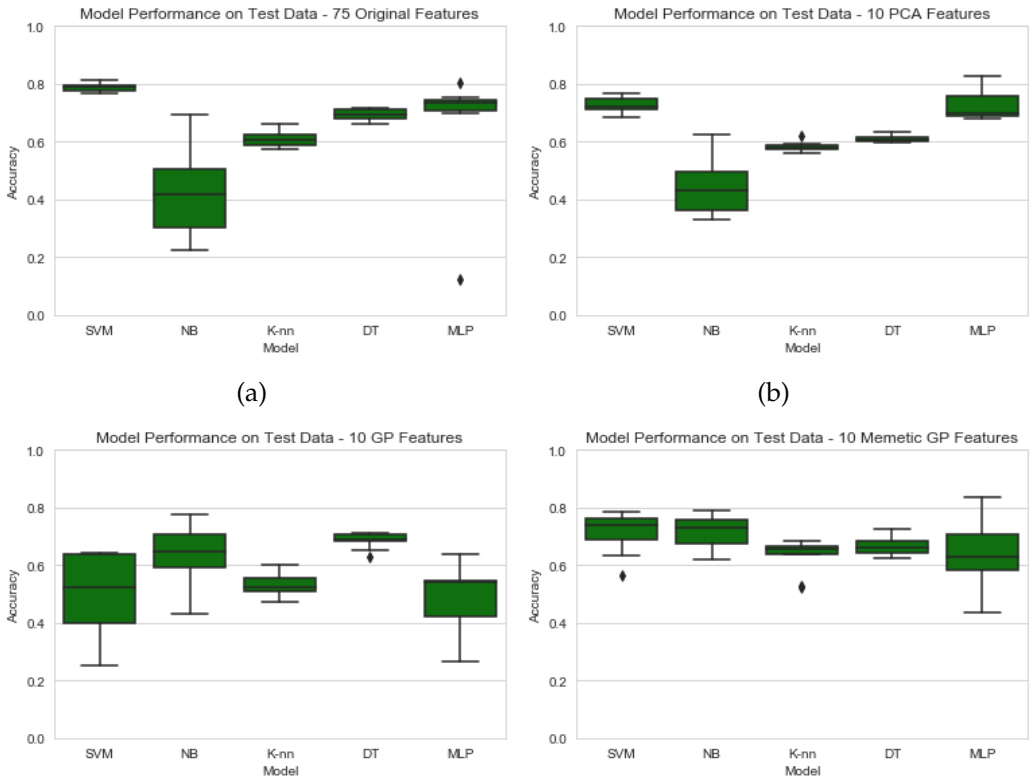

Figure 3: Test data accuracy

benchmark works of [sources]. Explain briefly the feature engineering methods of each source. Results are presented from the performance of the best Naive Bayes classifier on engineered feature test data sets.

Reference	Recall	Specificity	Precision	f1 score
Lachner-Piza et al., 2018	0.65	0.98	0.38	0.48
Tsanas and Clifford, 2015	0.76	0.92	0.33	0.46
Zhuang et al., 2016	0.51	0.99	0.70	0.59
Miranda et al., 2019	0.75	0.98	0.35	0.48
Non-memetic GP	0.64	0.93	0.67	0.76
Memetic GP	0.62	0.94	0.75	0.83

Our proposed feature engineering criterion yields comparable model performance to that of the included authors regardless of whether there is a local search step or not, and this holds for all metrics presented here. The non-memetic GP procedure when compared to previous works yields the best and second best f1 score and precision, respectively. This is evidence of the effectiveness of our specific criterion - mutual information and VC dimension - in finding features that result in good model performance even without a local search. With the exception of recall, including a local search does improve model performance. The memetic GP with our criterion yields the highest overall precision and

f1 score compared to the authors mentioned, which suggests that our feature engineering method is preferred when such metrics are of prime importance.