AutoFE: Efficient and Robust Automated Feature Engineering

by

Hyunjoon Song

B.S., Massachusetts Institute of Technology (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 2018

Author .....
Department of Electrical Engineering and Computer Science
May 25, 2018

Certified by .....
Samuel Madden
Professor of Electrical Engineering and Computer Science
Thesis Supervisor
May 25, 2018

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

AutoFE: Efficient and Robust Automated Feature Engineering

by

Hyunjoon Song

Submitted to the Department of Electrical Engineering and Computer Science
on May 25, 2018, in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Feature engineering is the key to building highly successful machine learning models. We present AutoFE, a system designed to automate feature engineering. AutoFE generates a large set of new interpretable features by combining information in the original features. Given an augmented dataset, it discovers a set of features that significantly improves the performance of any traditional classification using an evolutionary algorithm. We demonstrate the effectiveness and robustness of our approach by conducting an extensive evaluation on 8 datasets and 5 different classification algorithms. We show that AutoFE can achieve an average improvement in predictive performance of 25.24% for all classification algorithms over their baseline performance obtained with the original features.

Thesis Supervisor: Samuel Madden
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I would like to give thanks to my family and friends who have supported me throughout my MIT career. Without their unconditional support, I definitely would not have been able to get to where I am today.

I am also deeply grateful to Professor Samuel Madden for providing me the opportunity to explore research in machine learning systems. His guidance and support were nothing short of extraordinary and helped me get through difficult situations. Without his generosity and kindness, this thesis would not have been possible.

Lastly, I would like to thank Manasi Vartak for providing me the original inspiration to work on automated feature engineering and helping me get started on this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents the AutoFE system. In this introductory chapter, we explain the motivation for building a system like AutoFE in the context of the current state of machine learning. Then, we provide a list of contributions and a high-level outline of this thesis.

1.1 Motivation

Over the last five years, advances in machine learning have led to breakthroughs in fields such as computer vision and speech recognition. Image recognition systems have become better than humans at classifying objects in images [34]. Speech recognition systems have beaten teams of humans at transcribing spoken language [39]. These impressive feats have sparked a large interest among businesses, governments, and organizations in using machine learning to tackle their own problems in data science.

Data science involves extracting insights, knowledge, and predictive models from data. A typical data science pipeline consists of four steps: problem formulation, data acquisition, feature engineering, and machine learning model selection & tuning. It starts with formulating the predictive problem of interest, usually based on the needs of the individual or the organization. Then, it involves acquiring and storing data relevant to the problem, usually in relational tables. Next, it deals with synthesizing new features from collected data by intuiting what might predict the outcome. Lastly,

it involves selecting a machine learning model and tuning its configuration to one that performs the best.

Feature engineering is one of the most important steps and time-consuming steps. It is very important because the efficacy of many learning algorithms relies heavily on input features; therefore, the performances of the same machine learning model with the same configuration with a set of even slightly different features can vary significantly [11]. Also, it is very time-consuming because it requires intuition and domain-specific knowledge of the data to create useful features. For example, the winners of the Groupo Bimbo Inventory Prediction competition reported that 95% of their time was spent on feature engineering and only 5% on modeling [38].

Automation of feature engineering can bring significant benefits to both individuals and organizations. It allows data scientists to focus more on other steps of the pipeline and iterate through multiple pipelines more efficiently. It is often unclear whether to spend more time and effort engineering new features, select a different machine learning model, or even acquire more relevant data. However, automated feature engineering can remove a lot of the uncertainty and speed up the decision process by providing an idea of how much value can be extracted from the current dataset and model. Also, it would address the problem of shortage of data scientists for businesses and governments by obviating the necessity of intuition and domain-specific knowledge of the data to create useful features. Using automated feature engineering, non-experts can not only extract value from their data quickly but also build a machine learning model that performs comparable to or better than models built by experts.

Hence, the goal of this thesis is to build a system that automatically generates a large set of new interpretable features and selects a set of useful features that optimizes the performance of a classification algorithm. As an example, we consider the California Housing dataset that contains information from the 1990 U.S. Census data [1]. It contains a binary class label, which indicates whether the median house value is above 200K or not, and eight features: median income, housing median age, total rooms, total bedrooms, population, households, latitude, and longitude.

One possibly useful feature to create is the difference between total bedrooms and total rooms, which provides total non-bedrooms such as bathrooms, living rooms, and kitchens. Another possibly useful feature is the product of median income and population, which provides approximately total income in a neighborhood. A more complex possibly useful feature is a relative median income based on the discretization of housing median age. It can be generated as follows:

1. Partition the housing median age values into bins: [10,20], (20,30], etc.
2. Group all instances based on their housing median age assignment.
3. For instances of each group, find the average of median income of all members of the group.

AutoFE applies a set of operators to different combinations of the original features and generates a large set of new features that contains not only these features but also other possibly useful features that can be overlooked. Since there can be many redundant features and irrelevant features among the generated ones, AutoFE considers randomly chosen subsets of features and iterates to find the best subset that optimizes the performance of any classification algorithm.

1.2 Contributions

The main contributions of this thesis include the following:

1. We describe a simple and powerful interface that enables automatic generation of interpretable features using commonly used feature operators.
2. We present an implementation of a modified genetic algorithm that finds the best set of features that optimizes the performance of any classification algorithm.
3. We describe the implementation and evaluation of the AutoFE system against publicly available datasets and a prior system shown to have good performance.

1.3 Outline

This rest of this thesis is organized as follows. Chapter 2 discusses related work in the field and previous work upon which this thesis is built. Chapter 3 describes the design and implementation of the AutoFE system. Chapter 4 presents microbenchmarks and end-to-end benchmarks with comparisons to related work on a variety of datasets. Chapter 5 discusses possible improvements and future work. Chapter 6 concludes with a summary of the previous chapters and the contributions made by this thesis.

Chapter 2

Background

In this chapter, we formally define the problem AutoFE solves and describe prior related work addressing similar problems.

2.1 Problem Definition

We assume a relational dataset that consists of many instances, where each instance contains a set of features and a label. Our goal is to generate a large set of new features using the original features along with new, candidate features derived from these features. Hopefully, this will allow a classification algorithm to find a function that better approximates the true underlying function that maps input to label than it would just using the original features. More formally, we have the following definition.

Definition 1 (AutoFE problem). Let F_{orig} be $\{f_1,\ldots,f_n\}$ where f_i denotes the name of an original feature and F_{new} be $\{f_1^{new},\ldots,f_m^{new}\}$ where f_i^{new} denotes the name of a new feature derived from original features. Let F_{cand} be $F_{orig} \cup F_{new}$ and $F_{select} \subset F_{cand}$. For $i = 1,\ldots,p+q$, let x_i denote a feature vector with values for features in F_{select} and $y_i \in Y$ the corresponding target value where $Y = \{0,1\}$. Given a training dataset $D_{train} = \{(x_1, y_1), \ldots, (x_p, y_p)\}$ and the feature vectors $x_{p+1}, \ldots, x_{p+q}$ of a test dataset $D_{test} = \{(x_{p+1}, y_{p+1}), \ldots, (x_{p+q}, y_{p+q})\}$ drawn from the same underlying data distribution, as well as a classification algorithm $C(\cdot,\cdot)$ and an evaluation metric $E(\cdot,\cdot)$, the AutoFE problem is to automat-

ically find F_{select} that maximizes $E(\,C(\,D_{train}, \{x_{p+1}, \ldots, x_{p+q}\})\,, \{y_{p+1}, \ldots, y_{p+q}\})$. The classification algorithm $C(\,\{(\,x_i, y_i)\,, \ldots, (\,x_j, y_j)\,\}, \{x_u, \ldots, x_v\})$ produces predictions $\{\hat{y_u}, \ldots, \hat{y_v}\}$ after learning from $\{(\,x_i, y_i)\,, \ldots, (\,x_j, y_j)\,\}$. The evaluation metric $E(\{\hat{y_i}, \ldots, \hat{y_j}\}, \{y_i, \ldots, y_j\})$ measures the performance of predictions against target values.

We want to select a subset of features from a set of candidate features that maximizes the performance of a function that is learned by a classification algorithm from training data when evaluated on test data.

2.2 Related Work

In this section, we explore prior work done making machine learning more accessible in many different directions.

2.2.1 Machine Learning Toolbox

In the past, machine learning researchers have traditionally used MATLAB or R packages when developing machine learning applications. However, they were sometimes difficult and slow to use and lacked proper documentation and consistency across the API. This gave rise to many toolboxes such as scikit-learn [29] and KNIME [7] for small datasets and GraphLab [27] and MLlib [28] for large datasets. These provide state-of-the-art implementations of many well known machine learning algorithms with extensive documentation and API consistency in general-purpose high-level languages such as Python. These toolboxes lowered the barrier to use machine learning for non-experts in the software and hardware industries as well as in fields outside of computer science such as biology or physics. However, the barrier for non-experts to achieve state-of-the-art performance for their own problems still remains high because the efficacy of many machine learning algorithms relies heavily on input features [11]. In AutoFE, we focus on building a system to automate the process of engineering useful features for any dataset to allow non-experts to achieve the best possible performance with any algorithm effortlessly.

2.2.2 Pipeline Optimization

Machine learning practitioners build complex, multi-stage pipelines involving dimensionality reduction, data transformations, and training supervised learning models to achieve high accuracy [35]. However, these pipelines are often pieced together with libraries and packages in an ad-hoc manner in practice. Further, they lack scalability as data volume and package complexity increase. Frameworks such as KeystoneML [36], MLbase [19], TensorFlow [6], and Apache MxNet [24] were introduced to address this problem by automatically constructing scalable, efficient pipelines with node-level optimization and end-to-end optimization. They have shown to provide significant improvements in runtime for various workloads, making them an attractive choice for machine learning researchers without a strong background in distributed systems and low-level primitives. While these frameworks are very useful for building efficient machine learning pipelines, the predictive performance of machine learning models still largely depends on the knowledge and expertise of the practitioner to use existing techniques to create useful features and select and tune the best performing model. The ultimate goal of AutoFE is to eliminate knowledge and expertise required to build state-of-the-art models for any dataset. In this paper, we focus on automatically generating and selecting a set of useful features to maximize the performance of any traditional classification algorithm for a variety of datasets.

2.2.3 Architecture Search

Over the years, state-of-the-art performances for datasets such as CIFAR-10 and ImageNet have been produced by carefully hand crafted deep neural network architectures such as AlexNet [20], ResNet [15], and PyramidNet [14]. This has fueled an effort to discover model architectures automatically, a field known as architecture search. In the last few years, reinforcement learning was used to produce image classification models competitive with state-of-the-art [41, 9, 42]. This year, an alternative approach based on evolution was shown by Google Research to perform considerably better than reinforcement learning at early search stages and produce state-of-the-art

deep neural networks [33, 31]. In AutoFE, we focus on feature engineering, which precedes architecture search in a machine learning pipeline, to extract more useful information from raw data that can help classification algorithms perform better. Automated feature engineering used in conjunction with architecture search has the potential of producing better results than results presented in this thesis and is an area of future work for AutoFE. Also, we build on the success of evolution to effectively navigate through a large search space and find great solutions by using a genetic algorithm to find a set of features that maximizes the performance of any traditional classification algorithm.

2.2.4 Data Science Machine

One of the very few works closely related to AutoFE is the Data Science Machine (DSM) [17], which also attempts to automate feature engineering for relational datasets. It creates an entity graph from multiple relational tables and performs automatic SQL query generation to join tables along different paths of the entity graph. Then, it exhaustively enumerates all possible features that can be construted by applying a predefined set of operators to joined tables. It reduces the size of the augmented feature set to a more manageable size using truncated SVD and selects features based on their f-value with respect to their target value. Although AutoFE and DSM share similar goals, their approaches differ in several ways.

One difference in our approaches is that we put more emphasis on relational datasets with a single table instead of datasets with multiple tables because single data tables are much more commonly found in publicly available datasets on the OpenML directory [4] and the UCI Machine Learning Repository [5]. Also, dealing with multiple tables complicates the problem further because it brings additional challenges such as joining tables and generating useful features using entity relations. Therefore, we focus on effectively automating feature engineering for relational datasets with a single table and demonstrating the effectiveness of our system on a wide variety of datasets.

Another difference is that we do not exhaustively enumerate all possible features

in feature generation. Exhaustive feature generation results in high time and memory cost, which can take days to complete and can slow down the development process. In AutoFE, we implement a randomized approach to select features and apply a set of operators to generate candidate features, the maximum number of which can be configured by the user. We demonstrate that our system provides good results on a variety of datasets in just a few hours with this approach.

Another difference is that we do not rely on a statistical measure such as f-value to select features. Statistical measures are often considered to be poor indicators of feature importance [40]. Our approach addresses the problem by considering many random subsets of candidate features and selecting a subset of useful features that maximizes the performance of any traditional classification algorithm.

A final difference is that we prioritize interpretability. The DSM framework utilizes truncated SVD to reduce the exhaustive feature set to a manageable size. However, it is not possible to interpret the product of truncated SVD in any way, other than the fact that it represents an incomprehensible approximation of the original data. In AutoFE, we focus on making new features that are easy to make sense out of by applying a set of defined operators to the original features. For example, we apply the multiplication operator to the median income feature and the population size feature and generate a new feature that is the product of median income and population size, which provides approximately total income in a neighborhood.

2.2.5 ExploreKit

ExploreKit [18] is the most closely related work to AutoFE that attempts to automate feature engineering. It operates on relational datasets with a single table and exhaustively enumerates all possible features using a predefined set of operators. Then, it employs a machine learning based strategy for feature selection that considers characteristics of the dataset that may affect the likelihood of features being effective and sorts them based on their likelihood.

There are several similarities between its approach and our approach for feature generation. Like ExploreKit, AutoFE operates on relational datasets with a single

table for reasons described earlier. Also, AutoFE shares a similar set of operators used to generate candidate features, which includes basic operators such as normalization, discretization, multiplication, and group-by-then-mean that are most commonly used in feature engineering.

Our approach for feature selection differs, however. ExploreKit relies on statistical tests to gather characteristics of the dataset and the candidate features and uses them to select features that seem most likely to be effective. While the approach is novel, its effectiveness is unclear. Statistical tests are notoriously poor indicators of feature importance [40]. Also, it is unknown whether dataset characteristics can actually help estimate feature importance. Lastly, using a classification algorithm to select features to improve the performance of another classification algorithm adds unnecessary complexity and uncertainty to the problem. In AutoFE, we build on the success of evolution in architecture search and rely on a genetic algorithm to find a set of useful features among a large set of candidate features. We found that AutoFE performs considerably better than ExploreKit for all datasets we have considered.

Chapter 3

AutoFE Architecture

In this chapter, we provide an overview of the AutoFE architecture at a high level. Then, we describe the main components of the system in detail — feature generation and feature selection.

3.1 Architecture Overview

The architecture of AutoFE is composed of a feature generator, a splitter, a distributed system of feature selectors, and an evaluator (Figure 3-1). Given a dataset, we use a feature generator to generate a large set of new interpretable features and augment the dataset. Then, we use a splitter to split the augmented dataset into training data, validation data, and test data. With training data and validation data, we run feature selection in a distributed manner. After we obtain the best set of features, we use an evaluator that trains a classification algorithm with the feature set on training data and evaluates its predictive performance on test data.

3.2 Feature Generation

The goal of this phase is to generate a large set of new features using the current feature set.

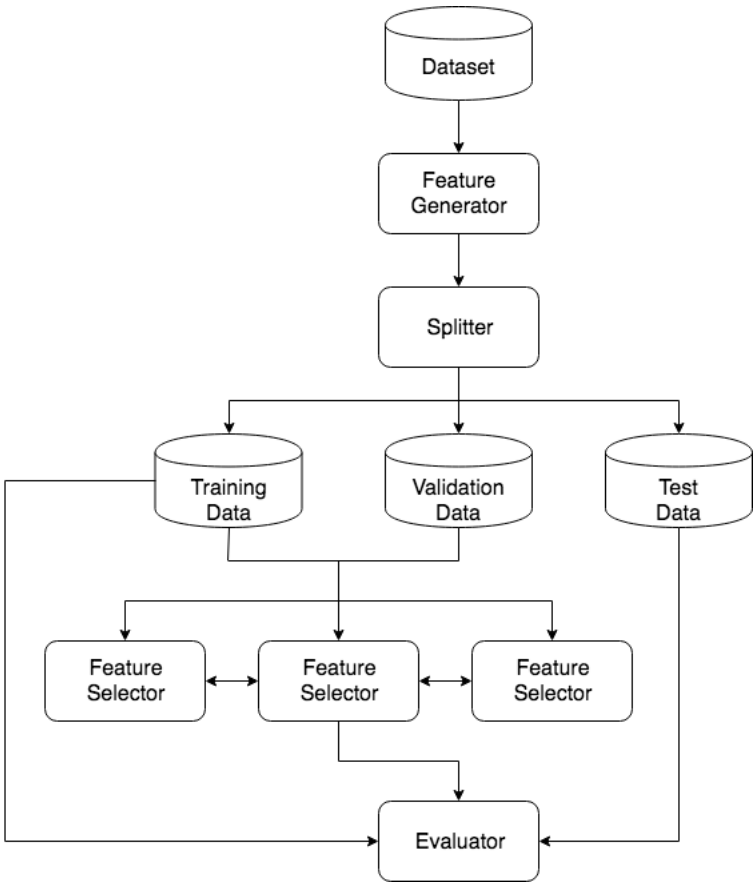


Figure 3-1: AutoFE Architecture Overview

3.2.1 Operators

We use operators to transform one or more features into a new feature. The three types of operators are unary, binary, and group-by-then.

Unary Operators

Unary operators are applied on a single feature to create a new feature. Each operator belongs to one of three sub-groups.

Discretizers are used to convert continuous features into discrete ones. Discretization is necessary in many popular classification algorithms such as Decision Trees and Naive Bayes and has been shown to contribute to performance [12]. Also, it provides us with transformations of continuous features that can be utilized by group-by-then operators. We have implemented the EqualRange discretization for

numeric features. For example, suppose a numeric feature has a minimum value of 0 and a maximum value of 20 and the number of groups is 10. The EqualRange method assigns each group a bucket of size 2, i.e., the first group would contain values in [0,2], the second group would contain (2,4], and so on.

Normalizers are used to fit the scale of continuous features to specific distributions. Normalization has been shown to improve the performance of multiple machine learning algorithms [13]. We have implemented normalization to the range [0,1]. For example, suppose X is a feature column. After normalization, X' equals to $(X - X_{min})/(X_{max} - X_{min})$.

Categorizers are used to convert nominal features into numeric ones. Categorization provides us with more numeric features that can be used by other operators. We have implemented categorization that randomly assigns a unique value to every unique nominal value for every nominal feature. For example, suppose the set of values in a nominal feature is {'Caucasian', 'African American', 'Asian'}. The values would be assigned values of 1, 2, and 3 respectively.

Binary Operators

Binary operators are applied on a pair of features. We have implemented the four basic arithmetic operators: addition, subtraction, multiplication, division. Since addition and multiplication are commutative, we do not add if a column with the operands in reverse order is already added.

Group-by-then Operators

Group-by-then operators are applied on a pair of features to create a new feature. We have implemented five operators: GroupByThenMin, GroupByThenMax, GroupByThenAvg, GroupByThenStdev, and GroupByThenCount. These operators implement the SQL-based operations with the same name. For example, consider the table shown in Table 3.1. We first discretize age using equal range and then take averages of instances in every discretized age group.

Age	BMI	GroupAge	GroupByAgeThenAvgBMI
12	25	[12,14]	29
23	20	(22,24]	20
25	30	(24,26]	31
26	32	(24,26]	31
13	33	[12,14]	29
32	28	(30,32]	26
31	24	(30,32]	26
20	25	(18,20]	21.5
19	18	(18,20]	21.5
21	29	(20,22]	29

Table 3.1: Example of Using the Group-by-then Operator in Feature Generation

3.2.2 Implementation

We generate new features by applying the operators in the following order.

1. We apply categorizers on all nominal features of the initial feature set F_i and create $F_{c,i}$, which contains all numeric features and categorized versions of all nominal features in F_i.

2. We apply discretizers on all numeric features of the initial feature set F_i and create $F_{d,i}$, which contains discretized versions of all numeric features in F_i.

3. We apply binary operators and group-by-then operators and create $F_{a,i}$ by doing the following until the size of $F_{c,i} \cup F_{d,i} \cup F_{a,i}$ is less than a user-configurable constant.

   (a) We randomly choose to apply binary operators or group-by-then operators and randomly select any two features from $F_{c,i}$.

   (b) If we choose binary operators, we apply binary operators on the two features and add generated features to $F_{a,i}$.

   (c) If we choose group-by-then operators and at least one of the two features is originally numeric, we apply group-by-then operators on them and add generated features to $F_{a,i}$.

4. We apply normalizers on all features in $F_{c,i} \cup F_{d,i} \cup F_{a,i}$ and create $F_{n,i}$, which contains normalized versions of all features in the augmented feature set.

We parallelize all operations using MapReduce to make them scale roughly with the number of cores on the running machine. We also set the size of the candidate feature set to be 1000 by default simply due to time constraints. The size can be configured to any number by the user and recommended to be larger with powerful compute resources.

We take the randomized approach to selection of features to apply operators rather than rely on statistical measures because they are often considered to be poor indicators of feature importance [40]. Also, even with a strong indicator of feature importance, it is unclear whether applying operators to relevant features produces useful features. The randomized approach has shown to perform very well in empirical studies described in Chapter 4.

We select a pair of features on which to apply binary operators and group-by-then operators specifically from $F_{c,i}$, in order to preserve interpretability. We avoid generating additional new features using new features from $F_{a,i}$ because it is very difficult to interpret a feature synthesized from new features. For example, suppose we choose a GroupByThenMean feature and a GroupByThenCount feature and apply a GroupByThenStdev operator to them. The new feature is a relative GroupByThenMean measurement based on the discretization of the GroupByThenCount values, which is very hard to interpret.

3.3 Feature Selection

The goal of this phase is to select a set of useful features from the augmented set of features.

3.3.1 Genetic Algorithm

Feature selection is a difficult task mainly due to a large search space, as there are 2^n subsets of features where n is the number of features. An exhaustive search for the best subset of a given feature set is infeasible in most situations. In the past, a variety of search methods such as complete search, greedy search, random search, and heuristic search has been applied to feature selection. However, most existing feature selection methods still suffer from stagnating in local optima and/or high computational cost [25]. Therefore, feature selection poses the problem of finding a balance between "exploration" of uncharted territory and "exploitation" of current selections.

The tradeoff between exploration and exploitation has been most thoroughly studied through the multi-armed bandit problem. It is a problem in which a fixed set of resources must be allocated between competing choices / arms in a way that maximizes their expected gain. The most widely-used algorithms for the multi-armed bandit problem are Upper Confidence Bound, Thompson Sampling, and Successive Rejects. Although they differ in their approaches, they all provide strong theoretical guarantees and reliably maximize the cumulative sum of rewards over multiple rounds.

A recent work that focused on the conversion rate optimization problem, which is a special case of the multi-armed bandit problem, proposed a genetic algorithm with a slight variant that performed reliably better than all of the widely-used multi-armed bandit algorithms [30]. In a traditional genetic algorithm, the population in every generation consists of the elite candidates and their offspring. However, in the proposed algorithm, the elite candidates in every generation are separately collected, and the population only consists of their offspring. Then, after all generations, the algorithm evaluates the performances of the collected elite candidates and returns the best candidate. Since all of the candidates in every generation are new, the evolutionary process can effectively explore more regions in the search space and identify an arm that is most likely to be the true best arm.

Like conversion rate optimization, feature selection is a special case of the multi-armed bandit problem. Each arm is a subset of features, and the reward associated with each arm is the performance measurement of a classification algorithm after training with the subset of features. We want to identify the best arm that yields the highest reward. Therefore, we build upon the success of this past work on genetic algorithms and present a method that addresses the feature selection problem, the details of which will be explained by the following.

3.3.2 Classification Algorithm

We use traditional classification algorithms that are widely used in practice, which are Logistic Regression, Naive Bayes, Decision Trees, Multi-layer Perceptron, and Random Forest. We chose them to demonstrate that automated feature engineering can significantly improve the performance of the traditional methods we use today. Using the default configuration, we train the algorithms with a subset of features on training data and evaluate its predictions on validation data against a metric. Then, with the subset of features that produced the best result on validation data, we train the algorithms and present their performances on test data.

3.3.3 Evaluation Metric

The evaluation metric we use to evaluate the performance of a classification algorithm is the area under the receiving operating characteristic curve (AUC), which is a widely used metric for classification algorithms in practice. The curve is created by plotting the true positive rate against the false positive rate at various threshold levels. Using normalized units, the area under the curve is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. It has been shown to be statistically consistent and more discriminating than other metrics [16]. Therefore, we rely on AUC to evaluate the predictive performance of trained models in the feature selection algorithm.

3.3.4 Implementation

Algorithm 1 Feature Selection

```
1: procedure SelectFeatures
2:     Initialize elite pool E as empty
3:     for i=1 to G do
4:         for j=1 to N do
5:             Train classifier with feature subset F_select_j on training data
6:             Evaluate the performance of classifier on validation data
7:             Set fitness for feature subset F_select_j as AUC on validation data
8:         end for
9:         Copy the best C_e percentile feature subsets of current generation i to elite
           pool E
10:        while Size of elite pool E is greater than N do
11:            Remove the worst feature subset from elite pool E
12:        end while
13:        Initialize parent pool A as the best C_p percentile feature subsets of current
           generation i
14:        Initialize offspring pool O as empty
15:        while Size of offspring pool O is less than N do
16:            Pick two parent feature subsets from parent pool A using k-way deter-
               ministic tournament selection
17:            Generate two offspring feature subsets by performing uniform crossover
               between two parents
18:            Mutate two offspring feature subsets, each bit is altered with probabil-
               ity C_m
19:            if offspring feature subsets are previously unseen then
20:                Add offspring feature subsets to offspring pool O
21:            end if
22:        end while
23:        Set next generation i + 1 as offspring pool O
24:    end for
25:    Return the best feature subset in elite pool E
26: end procedure
```

We apply the modified genetic algorithm described above to find the subset of features that allows a classification algorithm to produce the best AUC. Each genome represents a subset of features. We use binary encoding to make each genome a string of bits. Each bit corresponds to a selection of a feature in its index in the list representation of the feature set. An example of a binary encoding of a subset of feature is shown below. The fitness of each genome is the AUC of the classification

algorithm on validation data after training the algorithm with the subset of features on training data.

$$
\begin{aligned}
F_{cand} &= \left[f^1_{cand}, f^2_{cand}, f^3_{cand}, f^4_{cand}, f^5_{cand}\right] \\
F_{select} &= \left[f^1_{cand}, f^3_{cand}, f^4_{cand}\right] \\
S_{F_{select}} &= \text{"10110"}
\end{aligned}
$$

We use k-way deterministic tournament selection as a method of choosing parent genomes for recombination. The method involves randomly picking k genomes from the population and choosing the top two genomes. An alternative method is fitness proportionate selection, which involves assigning each genome the probability of getting chosen by its fitness divided by the sum of fitnesses of all genomes and choosing two genomes based on the assigned probabilities. However, it is known to perform poorly in practice due to its stochastic noise [8] as demonstrated later. Therefore, we implement k-way deterministic tournament selection for our recombination method in the genetic algorithm.

We first initialize the parameters of the algorithm. G is the number of generations, which is set to 10 by default, based on diminishing returns with more generations observed in our experiments. N is the number of feature subsets / size of population in each generation, which is set to 10 times the size of the feature set by default, based on a rule of thumb suggested in evolutionary algorithms [37]. C_e is the percentage of the feature subsets that we consider as elite, which is set to 20 by default, based on exploratory behavior as demonstrated later. C_p is the percentage of the feature subsets that we consider as eligible to be parent, which is set to 20 by default, based on exploratory behavior as demonstrated later. C_m is the probability that each bit in the binary encoded string of the genome will be flipped, which is set to 0.08 by default, based on a rule of thumb recommended in evolutionary algorithms [21]. k is the size of tournament used in the k-way deterministic tournament selection for

recombination. We set k to be $\frac{N*C_p}{2}$, which is half the size of the parent pool, by default because it needs to be large enough to include feature subsets with strong fitness but small enough to not include best performing feature subsets every time.

Algorithm 1 presents the pseudocode of the algorithm. Before we begin evolution, we initialize an empty list for elite pool E in line 2. For each generation i, we do the following. For each subset of features F_{select_j} in the generation, we train a classifier with the feature set on training data, evaluate its performance on validation data, and set the fitness of the feature subset as the AUC in lines 5-7. Then, we sort the feature subsets by their fitness and copy the feature subsets and their fitness that are in the top C_e percentile to the elite pool E in line 9. If the size of the elite pool is greater than the size of the generation N, we repeatedly eliminate a feature subset with the worst fitness until the size of the elite pool is not greater in lines 10-12. Then, we initialize an empty list for parent pool A and add the feature subsets and their fitness that are in the top C_p percentile to it in line 13. We also initialize an empty list for offspring pool O in line 14. Until the size of the offspring pool is less than the size of the generation N, we do the following. We first pick two parent feature subsets from the parent pool A using a k-way deterministic tournament selection where k equals to half the size of the parent pool in line 16. Using the two parent feature subsets, we perform a single-point uniform crossover to produce two offspring feature subsets in line 17. We then apply mutation to the offspring feature subsets with probability C_m in line 18. If the offspring feature subsets are previously unseen, we add them to offspring pool O in line 20. After we fill up the offspring pool O, we set the next generation to be the offspring pool and proceed the iteration in line 23. After we finish evolution, we return the feature subset with the highest fitness in the elite pool E in line 25.

The algorithm is run in a distributed manner with a centralized controller. A single master machine coordinates the steps of the algorithm among a group of worker machines using remote procedure calls. For each generation, the master sets the list of feature subsets to consider on every worker machine using SetSamplesRPC. After setting the list of feature subsets, the master initiates the trainings and evaluations of

the classification algorithm with different feature sets using RunTrainingRPC. Then, while it is waiting for all workers to finish, it continuously polls for statuses using GetStatusRPC. When a worker is finished, the master retrieves all its evaluations using GetEvalsRPC.

Chapter 4

Evaluation

In this chapter, we present the evaluations of the AutoFE system against a variety of classification datasets. We show a microbenchmarking of different parameters of the system and an end-to-end benchmarking of the system to make comparisons of different classification algorithms and prior work.

4.1 Experimental Setup

We evaluated AutoFE on 8 supervised binary classification datasets with large variety in number of instances, number of features, and feature type composition. All datasets are available on the OpenML directory, the Kaggle platform, and the UCI Machine Learning repository [4, 2, 5]. Their characteristics are presented in Table 4.1. They were partitioned randomly into training, validation, and test data, with 64% assigned to the first, 16% assigned to the second, and 20% assigned to the third. We use the area under the receiver operating characteristic curve (AUC) as the evaluation metric and calculate it using a trained classifier's predictions on test data.

We use five types of classification algorithms in the evaluation: Logistic Regression, Naive Bayes, Decision Trees, Multi-layer Perceptron, and Random Forest. For all algorithms, we used the implementation included in version 0.19 of the scikit-learn library with the default parameters [29].

We conducted our experiments on a 8-core machine with 16GB of RAM for feature

Name	Instances #	Features #	Numeric Features %
Indian Liver	585	10	90%
Diabetic Ret.	1,151	19	100%
Seismic Bumps	2,584	18	77.78%
Bank Marketing	45,211	16	43.75%
Adult	48,842	14	42.86%
Churn	50,000	230	83.48%
Santander	76,020	370	100%
Vehicle	98,528	100	100%

Table 4.1: Dataset Characteristics

generation and a cluster of 40 2-core machines with 2GB of RAM for feature selection.

4.2 Microbenchmark

In this section, we demonstrate the effect of different values for the number of generations and the elite and parent percentages and different recombination strategies in our genetic algorithm.

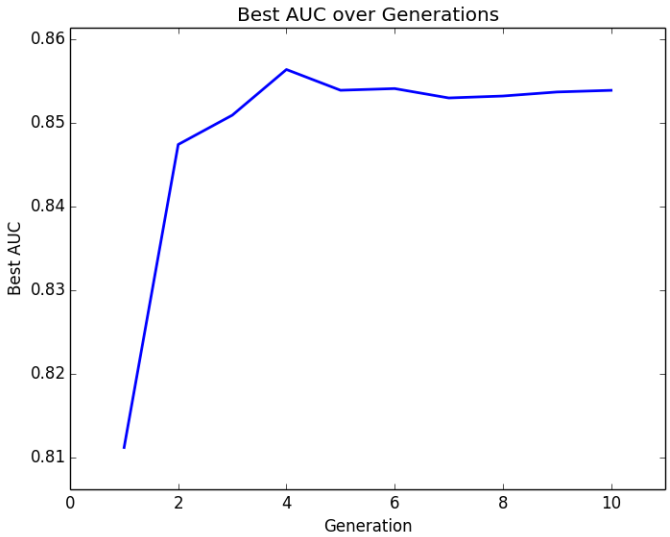


Figure 4-1: Best AUC over Generations

Figure 4-1 presents the best AUCs achieved in every generation from 1 to 10 with population size N of 10,000, elite percentage C_e of 20, parent percentage C_p of 20,

mutation probability C_m of 0.08, and recombination strategy of a k-way tournament selection where k equals $\frac{N*C_p}{2}$ when training a Multi-layer Perceptron classifier on the Adult dataset. The results show that our genetic algorithm substantially increases the best AUC achieved by the classifier over generations. Also, the improvement in performance begins to converge after four generations, which indicates that the population became homogeneous and would have not made much progress past ten generations. The minimum number of generations needed to reach convergence varies a lot from dataset to dataset but we found that a maximum of 10 generations was sufficient for the improvement to converge for all datasets considered in our evaluations. It is worth noting that finding an optimal number of generations would significantly reduce the unnecessary high computation cost of genetic algorithms after convergence but is a very difficult problem that can be an area of research on its own.

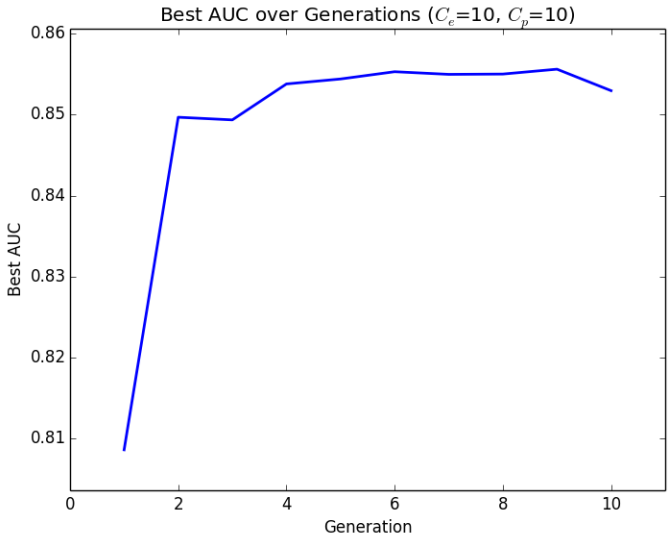


Figure 4-2: Best AUC over Generations for $C_e = 10$ and $C_p = 10$

Figure 4-2, 4-3, and 4-4 show the best AUCs achieved with different values for elite and parent percentages with generation number G of 10, population size N of 10,000, mutation probability C_m of 0.08, and recombination strategy of a k-way tournament selection where k equals $\frac{N*C_p}{2}$ when training a Multi-layer Perceptron classifier on the Adult dataset. The results demonstrate that larger values of C_e and C_p lead to more exploratory behavior at a cost of performance at early stage. Exploratory behavior

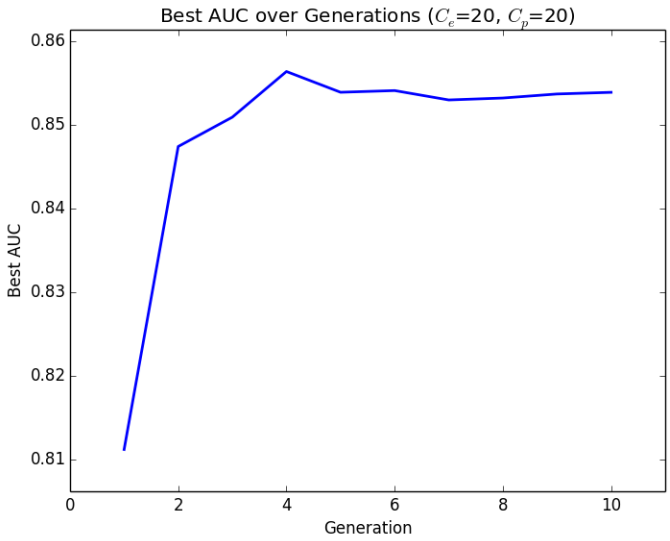


Figure 4-3: Best AUC over Generations for $C_e = 20$ and $C_p = 20$

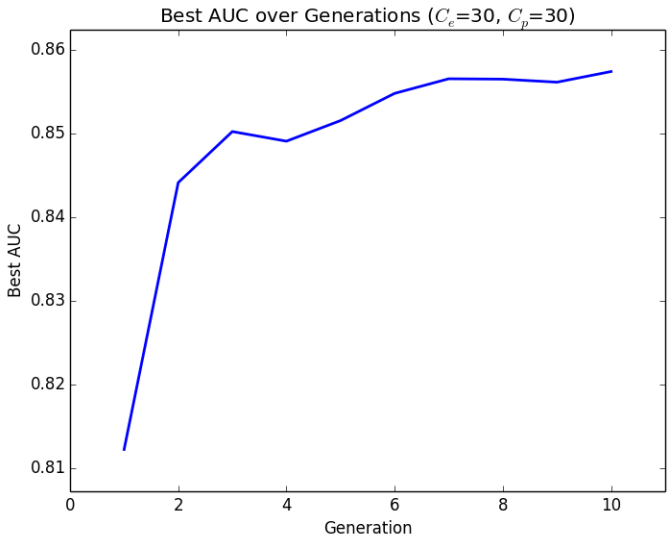


Figure 4-4: Best AUC over Generations for $C_e = 30$ and $C_p = 30$

can aid in escaping local optima and moving closer to the global optimum but can also prevent the evolutionary process from converging. We found that the elite and parent percentage of 20 was sufficient for our genetic algorithm to converge after 10 generations and produce good results for all datasets in consideration. Recent research suggests that an adaptive scheduling of crossover probability, mutation probability, and elite and parent percentages based on the diversity of population can produce

better results than a fixed value [22]. We plan on incorporating this feature in the next iteration of the algorithm.

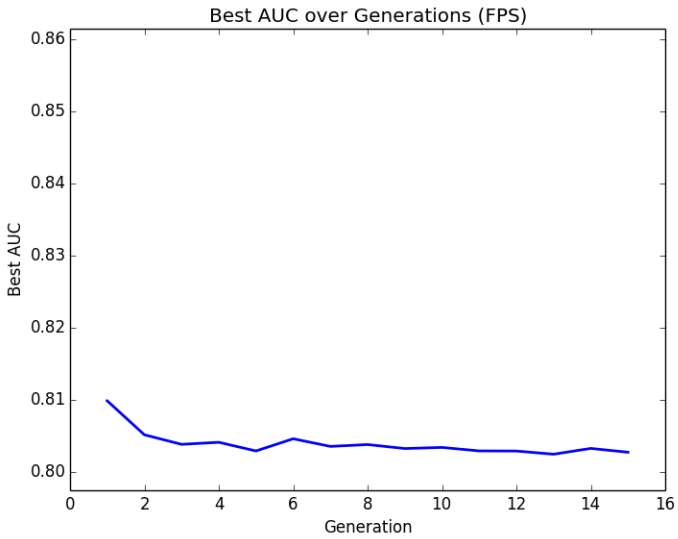


Figure 4-5: Best AUC over Generations for Fitness Proportionate Selection

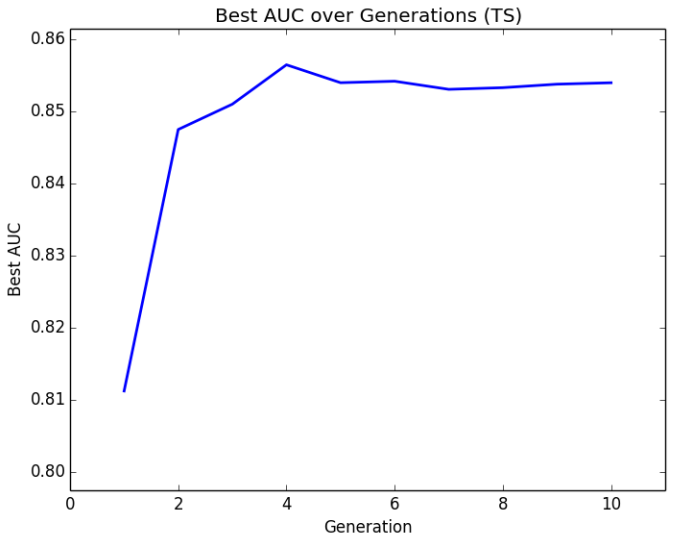


Figure 4-6: Best AUC over Generations for Tournament Selection

Figure 4-5 and 4-6 show the best AUCs achieved with different recombination strategies with generation number G of 10, population size N of 10,000, elite percentage C_e of 20, parent percentage C_p of 20, and mutation probability C_m of 0.08 when training a Multi-layer Perceptron classifier on the Adult dataset. The recombination

strategy illustrated in Figure 4-6 is k-way deterministc tournament selection described in Section 3.3.4. The results show that tournament selection considerably increases the best AUC achieved by the classifier over generations while fitness proportionate selection does not improve performance at all. This can be explained by the fact that fitness proportion selection is known to perform poorly in practice due to its stochastic noise [8]. We have not yet tried different values of k for the tournament selection strategy; however, we found that the value of $\frac{N*C_p}{2}$ for k was sufficient for our genetic algorithm to produce good results for all datasets.

4.3 End-to-end Benchmark

In this section, we illustrate the overall performance of the AutoFE system and make comparisons of different classification algorithms and prior work.

4.3.1 Overall Performance

To illustrate the overall performance of the system, we evaluated AutoFE against three datasets: Adult, Santander, and Churn. All datasets have at least 50,000 instances and at least 100 features. We chose them specifically because a large number of instances reduces the chance of overfitting with a classification algorithm [11]. We demonstrate that the AutoFE system can automatically perform feature engineering and significantly improve the performance of all classification algorithms across the three datasets.

Table 4-2, 4-3, and 4-4 show the performance of five different types of classification algorithms for all three datasets. The improvement is the increase in AUC in percentage from the baseline result, which is obtained by training a classification algorithm on training data with the original features and evaluating its performance on test data. Figure A-1, A-2, and A-3 show the corresponding receiver operating characteristic curves for all five different types of classification algorithms. AutoFE significantly improved the performance of all classifiers, achieving an average improvement of 35.86% for Logistic Regression, 19.98% for Naive Bayes, 21.83% for Decision

Trees, 41.17% for Multi-layer Perceptron, and 7.36% for Random Forest.

We notice that the improvement in performance for Random Forest is not as large as that of other algorithms. This observation can be explained by the nature of the Random Forest algorithm. In short, Random Forest operates by constructing a multitude of decision trees using random subsets of features and merging them together to provide an accurate and stable prediction. Therefore, it intrinsically performs some level of feature selection. However, we show that even with this feature, Random Forest can still benefit from AutoFE.

We also notice that there is no single best classification algorithm for all three datasets as we expected. We found that Decision Trees performed the best for the Adult dataset, Multi-layer Perceptron for the Santander dataset, and Logistic Regression for the Churn dataset. Since results vary significantly from dataset to dataset, we designed the AutoFE system to make it easy for users to plug in different classification algorithms and explore results on their own. We plan on implementing an interactive user interface and automated model selection and hyperparameter tuning to make the process even easier for users as future work.

For each dataset, we found the AUC produced by the best machine learning model that won its respective competition on Kaggle [2] and KDD Cup [3]. Table 4.5 presents the comparison of performance between the best models and AutoFE for all three datasets. The listed AUCs for AutoFE represent the best AUC of all five classification algorithms used by the system. AutoFE performed only slightly worse than competition winning models with an average difference of 4.35%. It is difficult to pinpoint what caused this difference because winners in these competitions usually do not publish their solutions. We suspect that they took advantage of more advanced classification algorithms such as Gradient Boosting [10] and automatic hyperparameter optimization of models such as Hyperband [23]. Also, it is most likely that winners are experienced data scientists who spent a significant amount of effort in synthesizing new features using more sophisticated operators. With additions of advanced classification algorithms, automatic hyperparameter optimization, and sophisticated feature operators, AutoFE can potentially compete with state-of-the-art.

Type	Baseline AUC	AutoFE AUC	Improvement
Logistic Regression	0.61	0.90	+46.49%
Naive Bayes	0.82	0.87	+6.25%
Decision Trees	0.75	0.90	+20.09%
Multi-layer Perceptron	0.54	0.88	+63.36%
Random Forest	0.88	0.88	+0.32%

Table 4.2: Performance of Classification Algorithms for the Adult Dataset

Type	Baseline AUC	AutoFE AUC	Improvement
Logistic Regression	0.59	0.79	+32.53%
Naive Bayes	0.51	0.76	+48.76%
Decision Trees	0.55	0.70	+28.40%
Multi-layer Perceptron	0.66	0.81	+22.11%
Random Forest	0.69	0.76	+10.87%

Table 4.3: Performance of Classification Algorithms for the Santander Dataset

Type	Baseline AUC	AutoFE AUC	Improvement
Logistic Regression	0.55	0.71	+28.57%
Naive Bayes	0.58	0.61	+4.94%
Decision Trees	0.53	0.62	+16.99%
Multi-layer Perceptron	0.51	0.70	+38.04%
Random Forest	0.58	0.65	+10.88%

Table 4.4: Performance of Classification Algorithms for the Churn Dataset

Name	Best AUC	AutoFE AUC	Improvement
Adult	0.95	0.90	-5.26%
Santander	0.82	0.81	-1.22%
Churn	0.76	0.71	-6.58%

Table 4.5: Comparison of Performance Between Competition Winning Models and AutoFE

4.3.2 Prior Work Comparison

To demonstrate the effectiveness of AutoFE, we compare the performance of AutoFE relative to a prior system called ExploreKit [18] on five datasets: Indian Liver, Diabetic Retinopathy Debrecen, Seismic Bumps, Bank Marketing, and Vehicle. We chose them specifically because many other datasets that ExploreKit used are designed for multi-class classification tasks, which AutoFE does not currently support. However, we can easily implement functionality to handle multi-class classification by transforming multi-class classification problems into multiple binary classification problems using the one-against-all strategy [26]. We plan on adding this functionality and doing a more comprehensive comparison as future work.

Table 4-5 shows the comparison of performance of two systems for all five datasets. The listed AUCs represent the best AUC of different classification algorithms used by both systems. Error reduction is an evaluation metric used by ExploreKit to compare the final result to the baseline result and is calculated using the following formula.

$$\frac{(1 - AUC_i) - (1 - AUC_f)}{(1 - AUC_i)}$$

We substitute the AutoFE AUC for AUC_f and the ExploreKit AUC for AUC_i and calculate error reduction to compare the performance of AutoFE to that of ExploreKit. AutoFE performed considerably better than ExploreKit for all five datasets, achieving an average error reduction of 16.26%. The improvement can be primarily explained by the differences in our feature selection approach. We do not rely on any kind of statistical measure to infer feature importance because they are notoriously poor indicators [40]. Also, we do not use a classification algorithm to estimate the likelihood of features being effective for another classification algorithm because it adds unnecessary complexity and uncertainty to the problem. In AutoFE, we instead rely on a genetic algorithm to find a set of useful features among a large set of candidate features.

Name	ExploreKit AUC	AutoFE AUC	Error Reduction
Indian Liver	0.80	0.80	+0.00%
Diabetic Ret.	0.81	0.82	+5.26%
Seismic Bumps	0.62	0.74	+31.58%
Bank Marketing	0.88	0.92	+33.33%
Vehicle	0.91	0.92	+11.11%

Table 4.6: Comparison of Performance Between ExploreKit and AutoFE

The runtime of AutoFE heavily depends on the number of features and the number of instances. For example, it takes less than an hour for AutoFE to process the Indian Liver dataset, which has 585 instances and 10 features, while it takes around five hours to process the Vehicle dataset, which has 98,528 instances and 100 features. For all five datasets, ExploreKit reports that the allowed runtime was three days, which is significantly longer than the maximum runtime of AutoFE. We demonstrate that AutoFE not only produces signficantly better results than ExploreKit on average but also in less than 6.94% of the runtime.

Chapter 5

Future Work

In this chapter, we explore the possible future directions of AutoFE. While we believe that the AutoFE system is already very useful, there are four possible future areas of improvement.

5.1 Multi-relational Data

One future area of work for AutoFE is to support multi-relational data. Currently, AutoFE can perform feature engineering only on single relational tables. Multi-relational data would bring additional challenges in generating useful features across multiple tables but would greatly increase the flexibility of AutoFE by enabling it to support a wide range of datasets. Many recent data science competitions such as KDD Cup [3] and Kaggle [2] provide their participants multi-relational data; therefore, adding support for it can add a lot of value for users potentially interested in using the system.

5.2 Supported Data Types

Another future area of work for AutoFE is to support a variety of datatypes such as date time and location coordinates. AutoFE currently only supports integers, floats, and text. Adding support for different datatypes requires additional functionality

to recognize the datatype and discretize accordingly. For example, date time values could be discretized using bucketization so that a date or time range is identified by a unique value, and any value that falls in the range is assigned that value. Also, location coordinates could be discretized using crosses so that the set of coordinates is represented as a rectangular grid where each cell of fixed size is identified by a unique value, and any coordinate that falls within the cell is assigned that value. Since there is a wide variety of datatypes across different datasets, adding more datatype support will greatly increase the flexibility of AutoFE.

5.3 User Interface

Other future area of work for AutoFE is to implement a user interface that allows users to easily define their own operators and also their own machine learning model. Currently, AutoFE utilizes commonly used basic operators such as multiplication and group-by-mean and traditional classification algorithms such as Logistic Regression and Naive Bayes. It needs to expose ways for users to pass in their custom operators defined in lambda functions and their custom classification algorithms in their own implementations and also ways for them to selectively choose a set of operators and algorithms for AutoFE to work with. The system was originally designed and implemented with this goal in mind so it is expected to not be difficult to make this addition. Allowing users to interact and guide the system enables the augmentation of human intelligence with machine intelligence, which we believe can push the boundaries of success in data analytics.

5.4 Automated Machine Learning

Last future area of work for AutoFE is to automate not only feature engineering but also model selection and hyperparameter optimization. There has been a lot of work in both areas recently, which has shown a lot of success using evolutionary algorithms [32]. Currently, a genome in the genetic algorithm of AutoFE represents a selection

of features in a binary string. It could further represent a selection of a machine learning model and a selection of its hyperparameters in a binary string or a different data structure. Then, the genetic algorithm would work to find a model with a set of hyperparameters and a set of features that performs the best. Adding in this functionality would bring additional challenges in scalability but would significantly increase the performance of AutoFE to an extent comparable to or better than state-of-the-art systems.

Chapter 6

Conclusion

This thesis set out to build an end-to-end system for doing data science with relational data. In doing so, we made the following main contributions:

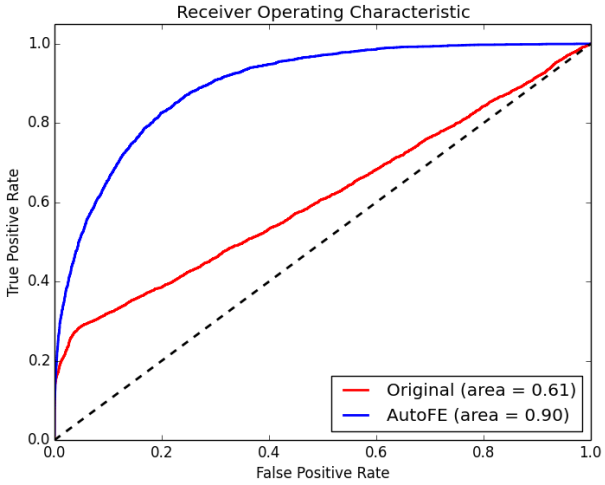
1. Generating useful interpretable features through structured operations
2. Selecting the best performing set of features for a given classification algorithm
3. Benchmarking the system to demonstrate its performance

We described the architecture of AutoFE as well as provided insight into our key architectural design decisions. We showed the implementations of feature generation and feature selection in detail. We also demonstrated that AutoFE significantly improves the predictive performance of different classification algorithms across all datasets that were considered by an average of 25.24% in AUC and that our models performed only slightly worse than competition winning models with an average difference of 4.35% in AUC. Further, we showed that AutoFE performed considerably better than a prior system called ExploreKit, achieving an average of 16.26% in error reduction in just less than 6.94% of the runtime.

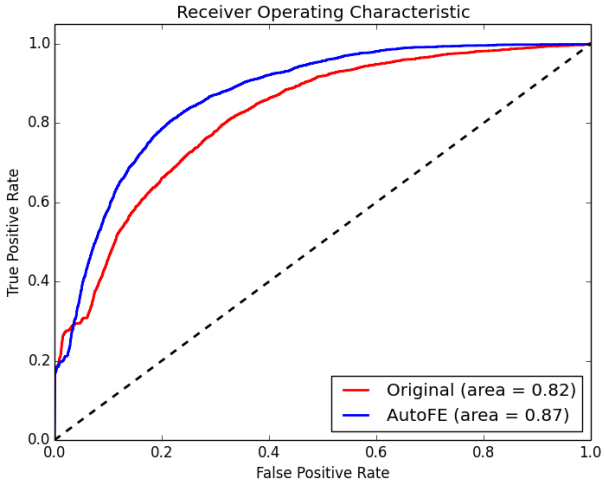
We hope that the AutoFE system will be useful for tackling the growing number and breadth of data science problems in the years to come.

Appendix A

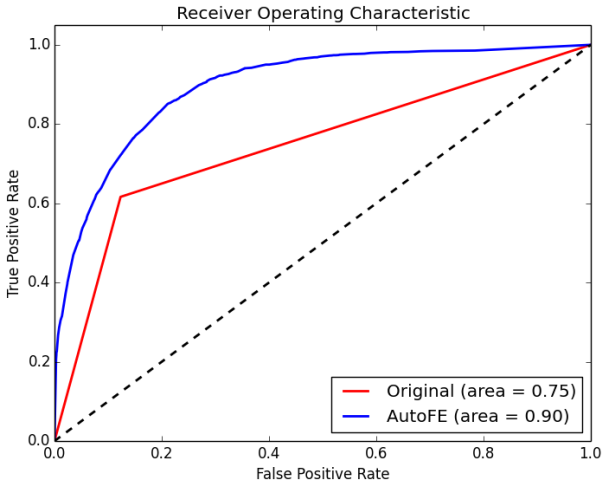
Figures



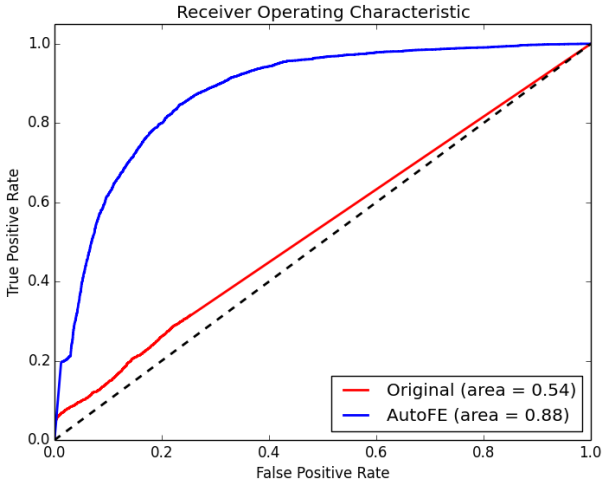
(a) Logistic Regression



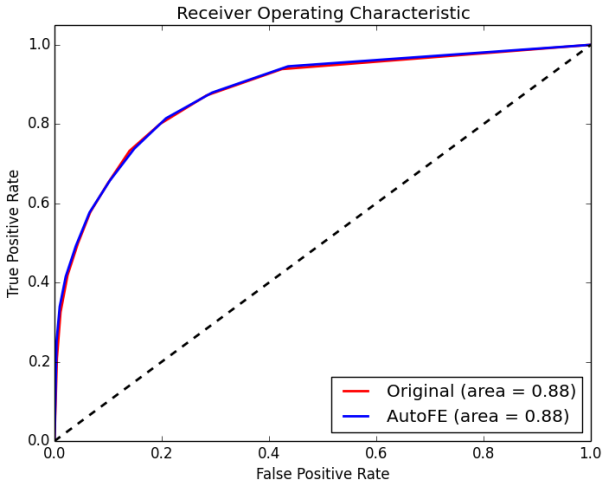
(b) Naive Bayes



(c) Decision Trees

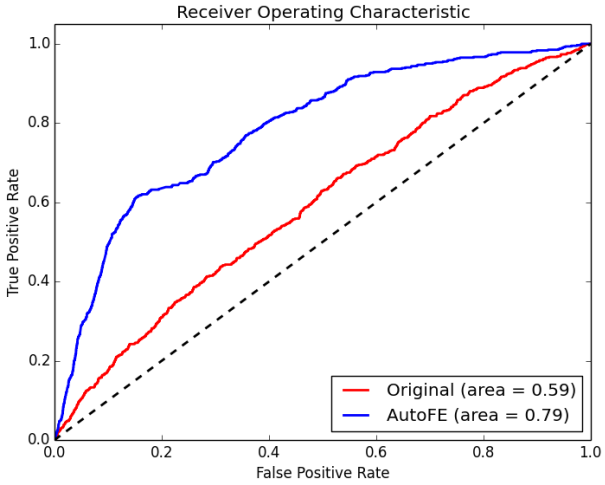


(d) Multi-layer Perceptron

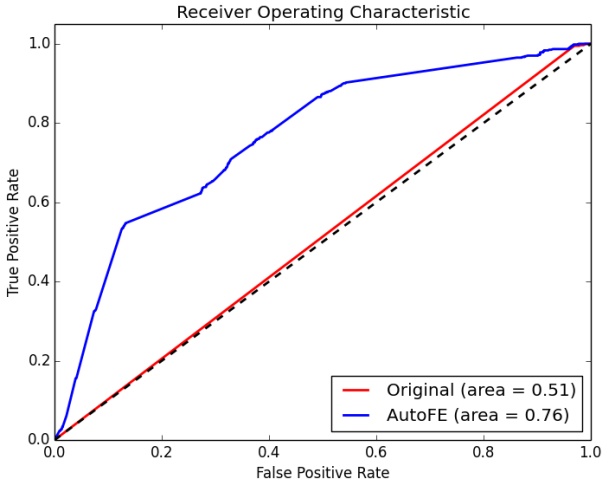


(e) Random Forest

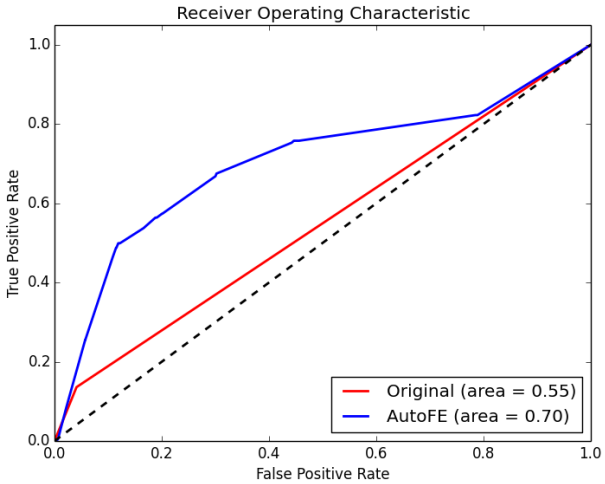
Figure A-1: Receiver Operating Characteristic Curves for the Adult Dataset



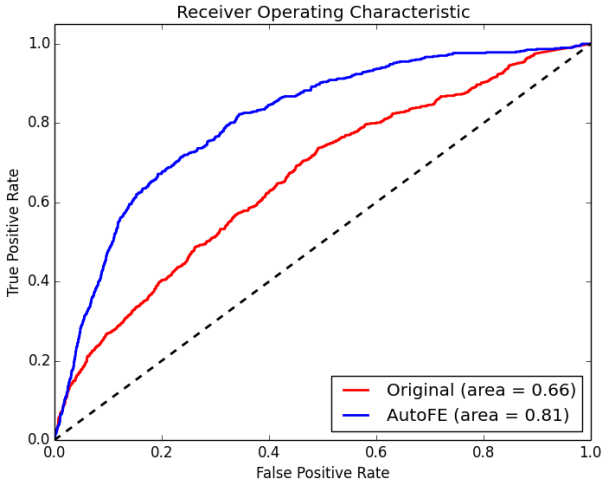
(a) Logistic Regression



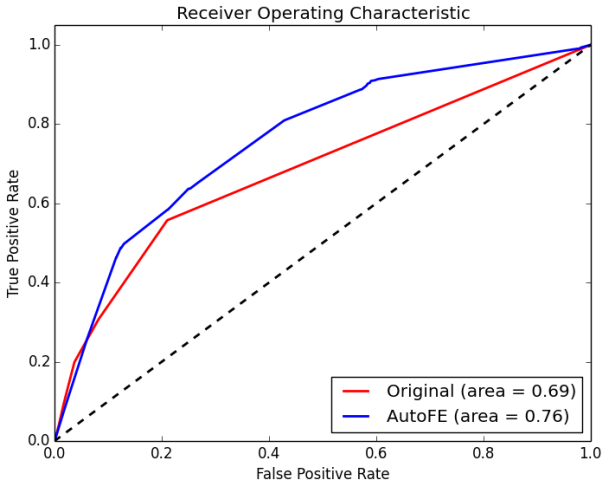
(b) Naive Bayes



(c) Decision Trees

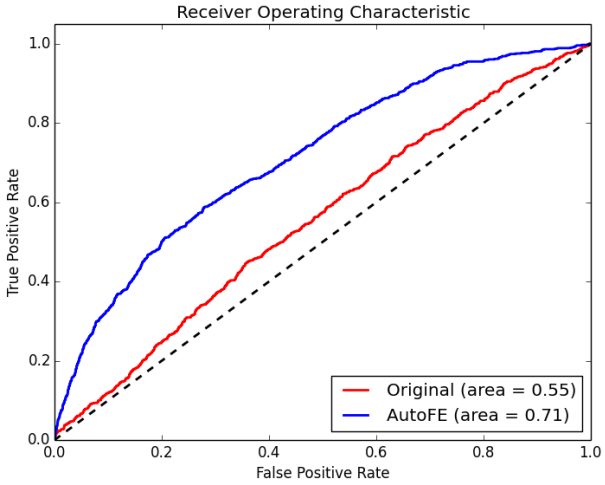


(d) Multi-layer Perceptron

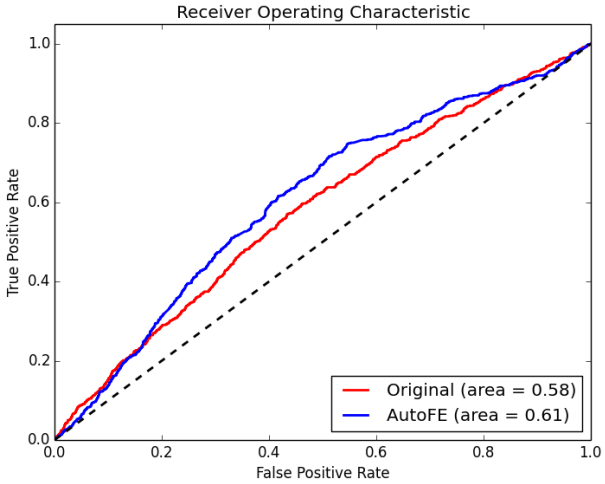


(e) Random Forest

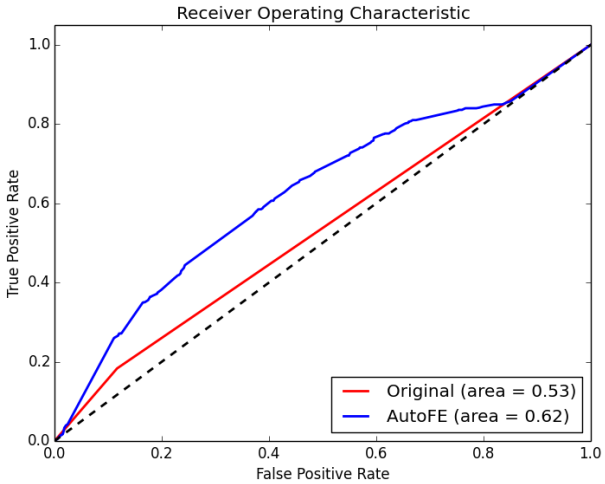
Figure A-2: Receiver Operating Characteristic Curves for the Santander Dataset



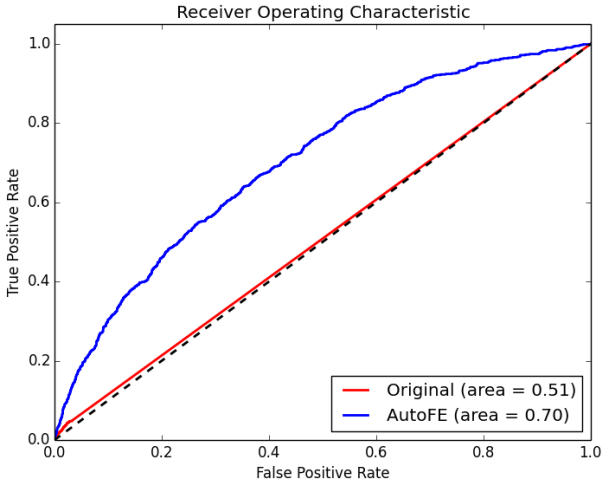
(a) Logistic Regression



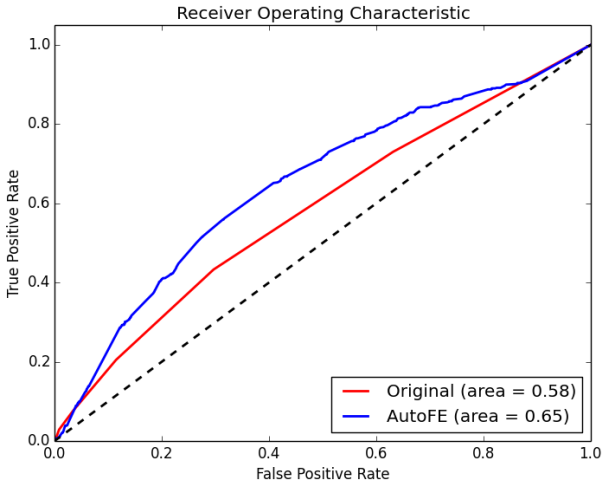
(b) Naive Bayes



(c) Decision Trees



(d) Multi-layer Perceptron



(e) Random Forest

Figure A-3: Receiver Operating Characteristic Curves for the Churn Dataset

Bibliography

[1] California Housing. https://www.dcc.fc.up.pt/ ltorgo/Regression/cal_housing.html/.

[2] Kaggle. https://www.kaggle.com/.

[3] KDD Cup. http://www.kdd.org/kdd-cup/.

[4] OpenML. https://www.openml.org/.

[5] UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/index.php/.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, et al. Large-scale machine learning on heterogeneous systems. 2015.

[7] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, et al. KNIME: The Konstanz information miner. *Data analysis, machine learning and applications*, page 319âĂŞ326, 2008.

[8] T. Blickle and L. Thiele. A Comparison of Selection Schemes Used in Evolutionary Algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.

[9] B. Bowen, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv:1611.02167*, 2016.

[10] T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. *arXiv:1603.02754*, 2016.

[11] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[12] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Proc. Twelfth International Conference on Machine Learning*, pages 194–202, July 1995.

[13] P. A. Estévez, M. Tesmer, C. A. Perez, and J. M. Zurada. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2), 2009.

[14] D. Han, J. Kim, and J. Kim. Deep pyramidal residual networks. *arXiv:1610.02915v4*, 2016.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[16] J. Huang and C. X. Ling. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3), 2005.

[17] M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. *Data Science and Advanced Analytics (DSAA)*, 2015.

[18] G. Katz, E. C. R. Shin, and D. Song. Explorekit: Automatic feature generation and selection. *IEEE International Conference on Data Mining*, 2016.

[19] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, et al. Mlbase: A distributed machine-learning system. *CIDR*, 2013.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[21] M. Kuhn, T. Severin, and H. Salzwedel. Variable Mutation Rate at Genetic Algorithms : Introduction of Chromosome Fitness in Connection with Multi - Chromosome Representation. *International Journal of Computer Applications*, 72(17), 2013.

[22] A. Laoufi, S. Hadjeri, and A. Hazzab. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms for power economic dispatch. *International Journal of Applied Engineering Research*, 1(3):393–408, 2006.

[23] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *arXiv:1603.06560*, 2016.

[24] M. Li, D. G. Anderson, J. W. Park, A. Smola, et al. Scaling distributed machine learning with the parameter server. *OSDI*, 2014.

[25] Y. Liu, G. Wang, H. Chen, H. Dong, X. Zhu, and S. Wang. An Improved Particle Swarm Optimization for Feature Selection. *Journal of Bionic Engineering*, 8(2):191–200, June 2011.

[26] Y. Liu and Y. F. Zheng. One-against-all multi-class SVM classification using reliability measure. In *Proc. IEEE International Joint Conference on Neural Networks*, volume 2, pages 849–854, 2005.

[27] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, et al. Distributed graphlab: a framework for machine learning and data mining in the cloud. *PVLDB*, 5(8):716–727, 2012.

[28] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, et al. Mllib: Machine learning in apache spark. *arXiv:1505.06807*, 2015.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikitlearn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[30] X. Qiu and R. Miikkulainen. Enhancing evolutionary optimization in uncertain environments by allocating evaluations via multi-armed bandit algorithms. *arXiv:1803.03737v2*, 2018.

[31] E. Real, A.Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *arXiv:1802.01548*, 2018.

[32] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-Scale Evolution of Image Classifiers. In *Proc. Thirty-fourth International Conference on Machine Learning*, 2017.

[33] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. *ICML*, 2017.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3), 2015.

[35] J. Sanchez, F. Perronnin, T. Mensink, , and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222âĂŞ245, 2013.

[36] E. R. Sparks, S. Venkataraman, T. Kaftan, M. Franklin, and B. Recht. Keystoneml: Optimizing pipelines for large-scale advanced analytics. *arXiv:1610.09451*, 2016.

[37] R. Storn. On the usage of differential evolution for function optimization. In *Proc. North American Fuzzy Information Processing*, pages 519–523, June 1996.

[38] Kaggle Team. Grupo Bimbo Inventory Demand, Winners' Interview: Clustifier & Alex & Andrey. http://blog.kaggle.com/2016/09/27/grupo-bimbo-inventory-demand-winners-interviewclustifier-alex-andrey/, 2016.

[39] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig. Achieving Human Parity in Conversational Speech Recognition. *arXiv:1610.05256*, 2017.

[40] Y. Zhang, S. Li, T. Wang, and Z. Zhang. Divergence-based feature selection for separate classes. *Neurocomputing*, 101:32–42, 2013.

[41] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learnings. *arXiv:1611.01578*, 2016.

[42] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv:1707.07012*, 2017.