ENHANCING DEEP LEARNING MODELS FOR CAMPAIGN PARTICIPATION PREDICTION

A Dissertation

by

Demet Ayvaz

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in the
Department of Computer Science

Özyeğin University
September 2019

ENHANCING DEEP LEARNING MODELS FOR CAMPAIGN PARTICIPATION PREDICTION

Advisor: Assoc. Prof. Murat Şensoy
Co-Advisor: Asst. Prof. Gonca Gürsun

Approved by:

Associate Professor Murat Şensoy,
Advisor
Department of Computer Science
Özyeğin University

Assistant Professor Furkan Kıraç
Department of Computer Science
Özyeğin University

Associate Professor Ali Fuat Alkaya
Department of Computer Science
Engineering
Marmara University

Assistant Professor Boray Tek
Department of Computer Engineering
Isik University

Date Approved: 31 July 2019

Professor M. Tolga Akçura
Department of Business
Administration
Özyeğin University

ABSTRACT

Companies engage with their customers in order to establish a long-term relationship. Targeting the right audience with the right product is crucial for providing better services to customers, increasing their loyalty to the company, and gaining high profit. Therefore, companies make huge investments to build campaign management systems, which are mostly rule-based and highly depend on business insight and human expertise. In the last decade, recommendation systems usually use modeling techniques such as deep learning to understand and predict the interests of customers. Classic deep neural networks are good at learning hidden relations within data (generalization); however, they have limited capability for memorization.

Wide & Deep network model, which is originally proposed for Google Play App. recommendation, deals with this problem by combining *Wide & Deep* network models in a joint network. However, this model requires domain expert knowledge and manually crafted features to benefit from memorization. In this thesis, we advocate using *Wide & Deep* network models for campaign participation prediction, particularly in the area of telecommunication. To deal with the aforementioned issue with that model, this thesis introduces the idea of using decision trees for automatic creation of combinatorial features (cross-product transformations of existing features) instead of demanding them from human experts. A set of comprehensive experiments on campaign participation data from a leading GSM provider has been conducted. The results have shown that automatically crafted features make a significant increase in the accuracy and outperform *Deep* and *Wide & Deep* models with manually crafted features.

Furthermore, since a limited number of access to the customers is allowed, making

well-targeted offers that are likely to be acceptable by the customers plays a crucial role. Therefore, an effective campaign participation prediction require to avoid false-positive predictions. Accordingly, we extended our research towards classification uncertainty to build network models that can predict whether or not they will fail. Consequently, we adopt evidential deep learning models to capture the uncertainty in prediction. Our experimental evaluation regarding prediction uncertainty has shown that the proposed approach is more confident for correct predictions while it is more uncertain for inaccurate predictions.

ÖZETÇE

Firmalar müşterileri ile uzun vadeli ilişkiler geliştirebilmek adına iletişim kurarlar. Doğru kitleyi doğru ürünle hedeflemek, müşterilere daha iyi hizmet sunarak şirkete olan bağlılıklarını artırma ve dolayısıyla şirket gelirlerini artırma noktasında kritik önem taşır. Bu nedenle şirketler, çoğunlukla kural bazlı çalışan ve insan uzmanlığına dayalı kampanya yönetim sistemleri oluşturmak için büyük yatırımlar yapmaktadır. Son on yılda öneri sistemleri genellikle müşterilerin beklentilerini anlamak ve tahminlemek amacıyla derin öğrenme gibi modelleme tekniklerini de kullanmaktadır. Klasik derin sinir ağları veri içindeki gizli ilişkileri öğrenmede oldukça başarılıdır (genelleme) ancak hatırlama noktasında daha sınırlı yetkinlikleri vardır.

Aslen Google Play uygulaması için tasarlanan *Wide* & *Deep* öğrenme modeli *Wide* ve *Deep* ağ modellerini tek bir model içerisinde birleştirerek bu soruna çözüm sunmaktadır. Bununla birlikte, bu model, uzman bilgisine ve uzmanlar tarafından birden fazla verinin bir araya getirilmesi ile hazırlanmış girdilere ihtiyaç duyar. Bu tezde, özellikle telekomünikasyon alanında, kampanya katılımı tahmini için *Wide* & *Deep* ağ modellerini kullanmayı öneriyoruz. Tez kapsamında, modellerde uzman bilgisine duyulan ihtiyacı ortadan kaldırmak amacıyla, bağımsız değişkenler arasındaki çapraz ilişkileri otomatik olarak oluşturmak için karar ağaclarını kullanan bir yöntem geliştirilmiş ve GSM müşterilerinin kampanya katılımını içeren gerçek veri setleri üzerinde kapsamlı bir değerlendirme yapılmıstır. Yapılan karşılaştırma sonucunda önerilen yöntem ile *Wide* & *Deep* model performansının önemli oranda artırılabildiği ve *Deep*, *Wide* & *Deep* modellerden daha iyi sonuçlar elde edildiği gözlemlenmiştir.

Ayrıca, kampanya sistemlerinde sadece sınırlı sayıda müşteri erişime izin verildiğinden, kabul edilme ihtimali en yüksek olan tekliflerin müşteriye sunulması kritik

önem taşımaktadır. Dolayısıyla, kampanya katılımının başarıyla tahminlenebilmesi hatalı tahminlerden kaçınmayı da gerektirmektedir. Bu nedenle başarısız olup olmayacaklarını tahmin edebilecek ağ modelleri oluşturmak amacıyla araştırmamızı sınıflandırma belirsizliği konusunda genişleterek kanıta dayalı derin öğrenme modellerini çalışmamıza dahil ettik. Sonuçlar, önerilen modelin doğru sınıflandırılmış öğeler üzerinde en düşük belirsizlik değerine sahip olduğunu, yanlış sınıflandırdıgı öğeler üzerinde ise daha yüksek belirsiklikle karar vermiş olduğunu göstermektedir.

ACKNOWLEDGEMENTS

First of all, I would like to thank Associate Professor Murat Şensoy and Assistant Professor Reyhan Aydoğan for their support, guidance, and motivation throughout the thesis work and especially giving time and helping me constantly. I also want to thank them for their contribution to my education during my graduate years.

I thank Professor M. Tolga Akçura, Assistant Professor Furkan Kıraç, Associate Professor Ali Fuat Alkaya, Assistant Professor Boray Tek for participating in my thesis jury and giving me feedback.

I also want to thank my friends Assistant Professor Reyhan Aydoğan and İbrahim Ersin Öziş for their support, motivation and the unique friendship that we shared throughout these years.

Finally, I should express my gratefulness to my family, especially to my mother Ayşe Semra Ayvaz and my father Bilal Bekir Ayvaz for their love, support, self-sacrifice, and endless-thrust in me. Special thanks to my first teacher, my father, for his guidance and support throughout my life. Without him, none of these would be possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Understanding the needs and preferences of customers can create new opportunities for companies and enable them to establish long-term relations with their customers. Using the gathered knowledge over time by means of their interactions with their customers, companies can make well-targeted recommendations and personalized offers (e.g., providing some discounts for particular items) with the aim of increasing their sales.

Companies, especially telecommunication companies, interact with their customers via a variety of access channels such as email, SMS, MMS, Web forms, and so on. During explicit and implicit interactions with their customers, they have the opportunity to monitor and analyze the behavior of their customers; consequently, well-targeted offers can be made. For instance, by detecting the location of its customers, a telecommunication company may infer that they are abroad; accordingly, it can suggest roaming offers. For this purpose, Complex Event Processing tools [7] are mostly used. Briefly, such tools rely on a vast amount of offline and streaming data and make offers based on predefined business rules. However, the process of selecting which offer to be made to the current customer is not personalized. That is, the same offers are made to all customers satisfying the same rules irrespective of their preferences/taste, although they might have different interests; thus, prefer different offers.

Furthermore, rules and regulations limit the number of access per customer; therefore, it is crucial to make offers more likely to be acceptable by the customers – which

we refer to as "campaign participation prediction problem". It requires understanding and learning preferences of customers over time. Consequently, companies can make well-targeted recommendations with a high acceptance rate.

This thesis formulates this problem as a classification problem which aims to predict whether or not the customer is likely to accept the offer to be made by the campaign management team of the company. Recent studies have demonstrated that deep learning models achieve substantial performances in the field of recommendation systems [8]. On the one hand, their ability to find the hidden relations among attributes (i.e., features) makes them strong candidates in finding what properties of offers make them acceptable for the customers. They are also good at capturing general patterns in the given training samples. However, they may fail in *memorizing the data* (i.e., "learning the frequent co-occurrence of items or features and exploiting the correlation available in the historical data") [2]. For instance, the model may misclassify an example, which is already in the training set. On the other hand, recommendation systems can benefit from memorization.

To deal with aforementioned memorization issue, Cheng *et al.* propose *Wide & Deep* Learning for Recommender Systems, where a deep neural network is used to extract hidden features automatically from the data for generalization while manually crafted features are fed into the wide network (i.e., single-layer neural network) to increase memorization capability of the joint model. Here, *manually crafted feature* (i.e., cross feature) denotes the combination of some existing features that are more meaningful when used together. Those features are added by a domain expert to the input features. For instance, a domain expert in a telecommunication company may consider using "age" and "income" together more meaningful rather than processing them separately.

However, one important limitation of this approach is the necessity of expert knowledge to craft combinatorial features for the wide part manually. This thesis

proposes to use a decision tree to create those cross features instead of eliciting them from a domain expert while adopting *Wide & Deep* Learning models in recommendation systems for particularly enhancing campaign participation prediction. The motivation for using decision trees is based on the fact that they rely on selecting the most informative features and construct rules consisting of several features such as "if x and y then". Accordingly, the combination of features in the constructed rules correspond to the cross features, and we suggest replacing the hand-crafted features with them in the *Wide & Deep* model.

To evaluate the performance of the proposed approach, we focus on a telecommunication use case where GSM operators aim to recommend offers targeting their customers' communication needs and preferences. Since GSM companies usually have an excessive number of customers with diverse backgrounds, preferences, and behavioral characteristics, it is already a challenge to learn what kind of offers/campaign that can be acceptable by each customer. Our evaluation results on this use case show that the proposed approach using decision trees outperforms *Deep* and *Wide &* Deep Neural Networks with handcrafted features.

Furthermore, while aiming to make the correct predictions, we avoid false-positive classification – classifying an offer as a potentially acceptable offer that is in fact not acceptable for the customer. In this domain, false-positive prediction affects the campaign management process negatively due to the limited number of access to customers available. Therefore, it is important to know how certain we are about our prediction. Accordingly, we incorporate prediction uncertainty to our model based on the methodology described by Şensoy *et al.* [9]. Associated uncertainty may let the decision-maker avoid an offer, which has a high risk of being rejected by the customer. Our experimental results show that the proposed approach with uncertainty extension assigns high confidence for true positive predictions while low confidence for false-positive predictions. When decision-maker considers the offers

with only high confidence, its performance in terms of acceptance rate is 99 percent for campaign participation use case.

Our main contribution in this thesis can be summarized as follows:

- To best of our knowledge, it is the first study applying deep learning in the problem of campaign participation prediction for GSM customers.
- Using decision trees for hand-crafted features enable us to alleviate the necessity of feature engineering and domain knowledge expertise required for *Wide & Deep* model. The idea of automatically deriving cross features for this model is novel. In an experimental setup, we show that the proposed approach significantly outperforms existing approaches.
- We incorporate the classification uncertainty to our model by adopting the methodology proposed by Şensoy *et al.* [9]. Considering the uncertainty enabled us to avoid offers that are likely to be rejected by the customers.

The rest of this thesis is organized as follows: Section 2 provides background information on related areas such decision tree classification, deep network architectures, *Wide & Deep* network model [2], which is the base model for this work and other feature generation techniques which aims to eliminate the need for expert knowledge. Section 3 provides details of the proposed decision tree-based feature creation method and describes our methodology in solving the campaign participation prediction task, including data collection and uncertainty prediction details. Section 4 evaluates our approach and provides results of our experiments on campaign participation datasets. Appendix A provides details and results on the adult income dataset, while Appendix B includes our results on the Criteo sample dataset. Lastly, Section 5 concludes the report with an overview of contributions.

CHAPTER II

BACKGROUND

In this chapter, we overview the background information on deep learning in general as well as more specific topics such as wide and deep models, feature generation techniques, which aim to eliminate the need for expert knowledge, and decision tree classifiers.

2.1 Deep Learning

Deep Learning, also known as deep neural learning or deep neural networks, is a subset of Machine learning which is based on artificial neural networks. The main idea behind artificial neural networks is to mimic the behavior of the human brain as it learns or make deductions out of data. The basic building block of artificial neural networks is artificial neurons, which take input from external sources, make calculations with some internal parameters (such as weight and bias) and produce output. Artificial neurons are also known as node (or unit) or perceptron. The fundamentals of artificial neural networks are discussed in References [10, 11].

The deep term here is a technical term and refers to the number of layers. A shallow neural network has at most one hidden layer while a deep neural network has more than one. Additional layers enhance the capability of the neural network to learn features of data and also the feature hierarchy. Simple features in one layer recombine from one layer to the next and forms more complex features.

In a recent survey of deep learning, Alom *et al.* lists advantages of deep learning as generalization, scalability, robustness and universal learning capability [1].

1. **Universal Learning:** The DL approach is sometimes called universal learning

because it can be applied to almost any application domain.

2. **Robust:** Deep learning approaches do not require the precisely designed feature. Instead, optimal features are automatically learned for the task at hand. As a result, the robustness of natural variations of the input data is achieved.

3. **Generalization:** The same DL approach can be used in different applications or with different data types.

4. **Scalability:** The DL approach is highly scalable. Microsoft invented a deep network known as ResNet [12]. This network contains 1202 layers and is often implemented at a supercomputing scale.

Based on the data available and the learning problem at hand, deep learning approaches can be categorized into three main learning models, which are supervised, semi-supervised, or partially supervised, and unsupervised. There is also the deep reinforcement learning approaches, which are often discussed under the broader scope of semi-supervised and unsupervised approaches.

Supervised learning is a learning approach that uses labeled data. Every sample in the dataset has a related label associated with it so that learning models can calculate the exact loss for each decision. The models can update model parameters to minimize the loss and learn a mapping function from input to output. Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) are some examples of network architectures frequently used in supervised deep learning.

The semi-supervised learning approach uses partially labeled data. As in the case of medical images like CT scans or MRIs, a radiologist can go through some of the images and label them as tumors, and other deceases. Labeling all of the images is not possible since it takes a long time for most manual labeling cases. Semi-supervised deep learning models are good at assigning labels and increasing their accuracy on

unlabelled data in comparison to unsupervised models. Deep Reinforcement Learning (DRL) can be considered as an example of semi-supervised learning techniques.

In unsupervised learning, the training dataset is a collection of samples without a label or correct answer. The goal of an unsupervised learning model is to learn the internal representation or important features to discover unknown relationships or structures within the input data. Clustering, dimensionality reduction techniques, and generative models can be listed under this category. There are also members of the deep learning family that are used for clustering or dimensionality reduction such as Autoencoders, Restricted Boltzmann Machines (RBM), and Generative Adversarial Networks (GAN).

There are a number of Deep Network Architectures, which are shortly described in the following sections.

2.1.1 Convolutional Neural Network (CNN)

Convolutional Neural Networks are the popular choice for different computer vision tasks such as image recognition. This network structure was first proposed by Fukushima in 1988 [13]. Due to limits of computation hardware, it is not widely used until LeCun *et al.* [14] applied a gradient-based learning algorithm to CNNs and obtained successful results for the handwritten digit classification problem.

The overall architecture of a convolutional neural network classifier consists of 2 main parts, which are (i) feature extractor and (ii) the last layer, i.e., logistic classification layer. The future extractor part consists of a combination of convolution and max-pooling layers. In the feature extraction layers, each layer of the network receives the output from its immediate previous layer as its input and passes its output as the input to the next layer. Higher-level features are derived from features propagated from lower-level layers. As the features propagate to the highest layer or level, the dimensions of features are reduced depending on the size of the kernel for

the convolutional and max-pooling operations respectively. The output of the last layer of the CNN is used as the input to a fully connected network which is called the classification layer. Figure 1 is the conceptual diagram of a convolutional neural network [1].

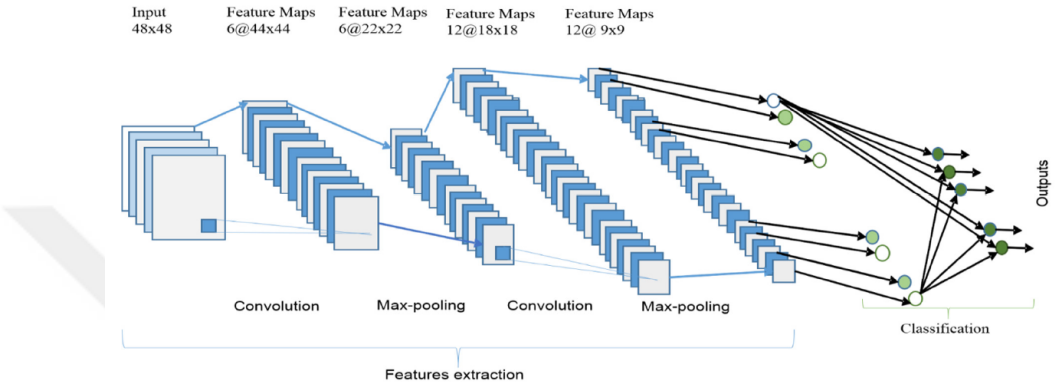

Figure 1: The conceptual diagram of Convolutional Neural Network [1]

2.1.2 Recurrent Neural Network (RNN)

Recurrent Neural Networks are unique due to their ability to remember or persist information. Traditional network models do not have a memory since they operate on fixed-size input, output vectors, and they have a fixed number of layers or steps in their network structure. In other words, they do not include loops. So traditional networks as CNNs or DNNs make their decisions only based on their current input. While recurrent neural networks are good at predicting what is coming next, due to their ability to operate over a sequence of data through time. Their ability to remember important things about the input they received enables them to be very precise in predicting the next value in the sequence. This is the reason why they are the preferred model for sequential data like time series, speech, text, financial data, audio, video etc. An example of RNN use-cases is language modeling in which RNNs are used for language understanding. RNN structure can hierarchically capture the sequential nature of the text and predict the set of words or sentences based on

previous ones. Figure 2 is conceptual diagram of an RNN network [1].

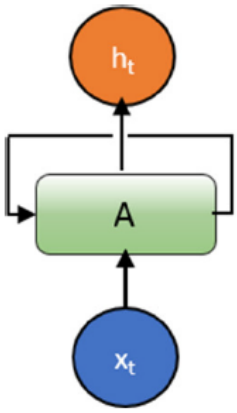

Figure 2: The conceptual diagram of Recurrent Neural Network [1]

2.1.3 Long Short-Term Memory (LSTM)

LSTM network architecture is an extension to Recurrent Neural Networks and enhances their ability to remember for longer sequences. Therefore they are well suited for problems in which important experiences have a long-time period in between. LSTMs can remember their memory for a long time since they use an enhanced version of memory on which they can perform read, write, and delete operations. Their memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information. In an LSTM, you have three gates: input, forget, and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget gate) or to let it impact the output at the current time step (output gate). Figure 3 shows a conceptual diagram of an LSTM [1].

2.1.4 Gated Recurrent Units(GRU)

GRU is simpler than LSTM but more effective in terms of computation cost. GRU replaces forget and input Gates of LSTM with a single update gate and merges the

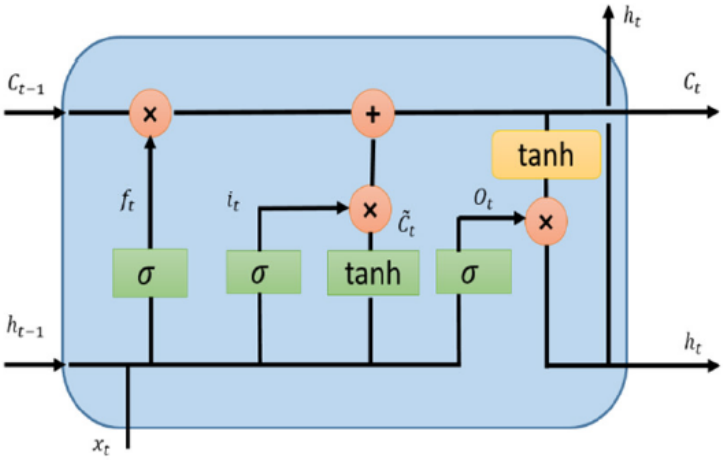

Figure 3: The conceptual diagram of Long Short-Term Memory network [1]

cell state. Even though the GRU requires fewer network parameters, which makes the model faster and less prone to overfitting. On the other hand, LSTM provides better performance if you have enough data and computational power [15]. Figure 4 shows a conceptual diagram of a GRU [1].

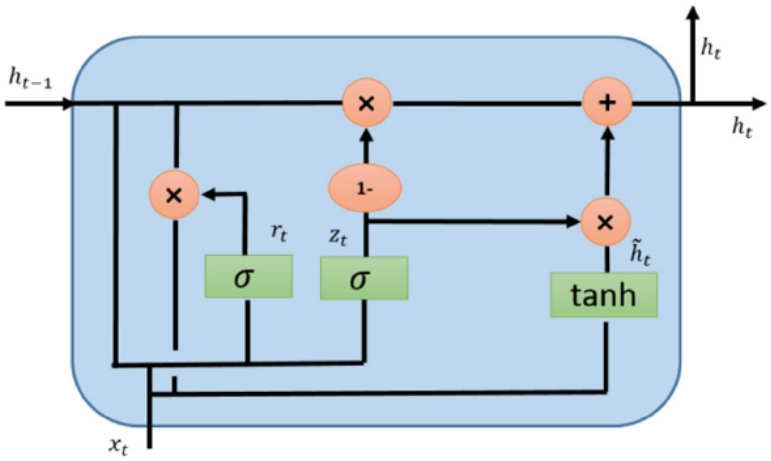

Figure 4: The conceptual diagram of Gated Recurrent Units [1]

2.1.5 Auto-Encoder (AE)

Auto-encoders are a specific type of neural network where the input is equal to the output. AE networks have efficient encoding and decoding capabilities and used for unsupervised feature learning. The main objective of an auto-encoder is to learn and represent input data, typically for data dimensionality reduction and compression. An auto-encoder is a deep network with two main components: *encoder* and *decoder*. During the encoding phase, the input is mapped into a lower-dimensional space, which is also called the latent space. Encoding can be repeated with multiple layers until the desired dimensional space is reached. In the decoding phase, original futures are regenerated from encoded ones. The conceptual diagram of auto-encoder with encoding and decoding phases is shown in Figure 5 [1].

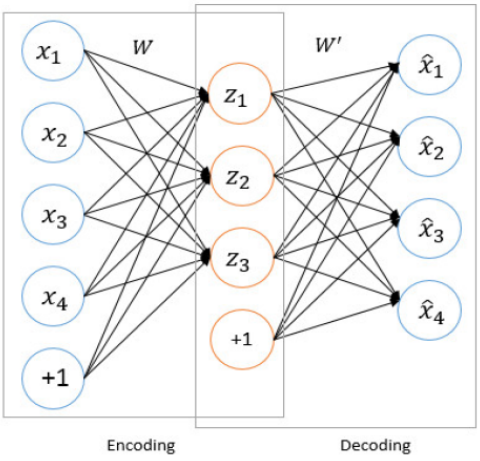

Figure 5: The conceptual diagram of Auto-Encoders [1]

2.1.6 Deep Belief Network (DBN)

A deep belief network (DBN) is a sophisticated type of unsupervised machine learning model. In general, Deep Belief Networks consists of smaller restricted Boltzmann machines (RBMs) or Autoencoders stacked on top of one another. In this network, architecture connections only exist between layers and not within the layers. The

stacked RBMs provides a system that can be trained greedily and extract a deep hierarchical representation of training data. Deep belief nets have been used for generating and recognizing images, video sequences, and motion-capture data. If the number of units in the highest layer is small, deep belief nets perform a non-linear dimensionality reduction.

2.1.7 Generative Adversarial Network (GAN)

Generative Adversarial Networks are first proposed by Ian Goodfellow in 2014 [16]. GAN network architecture is very promising due to its ability to generate new samples. Basically, the proposed model is capable of learning any probability distribution and sample from the learned distribution. The network architecture consists of two networks competing against each other in a zero-sum game. In the case of an image generation problem, the generator synthesizes samples from Gaussian noise while the discriminator classifies each sample as a true sample from the training set or fake one generated by the generator. The training process continues until the generator's samples become close to the original samples in the training set. GAN is in the family of unsupervised learning algorithms. Figure 6 is a conceptual diagram for a generative adversarial network [1].

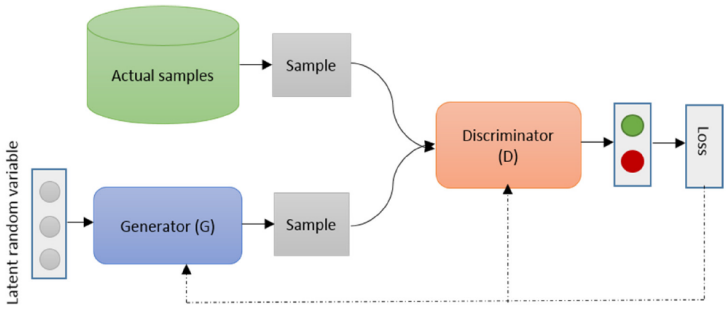

Figure 6: The conceptual diagram of Generative Adversarial Networks [1]

2.2 *Wide & Deep Learning Models*

Wide & Deep shown in Figure 7 is initially introduced for App recommendation in Google play [2]. It is inspired by human learning process, which is a complicated process including both *memorization* and *generalization*.

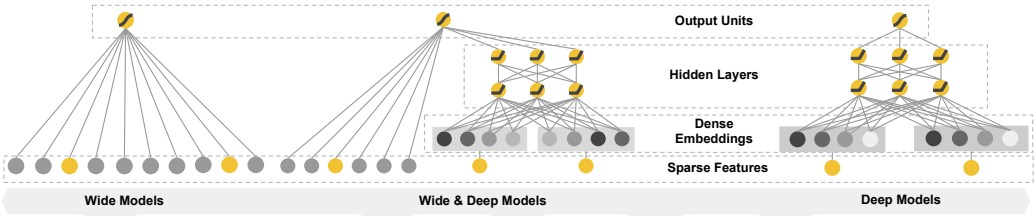

Figure 7: Wide, Deep and Wide & Deep network models [2]

Memorization refers to the learning from previously seen examples, and generalization refers to applying previous knowledge to understand unseen examples. The key idea behind *Wide & Deep* Neural Networks is to combine the benefits of both memorization and generalization in one model. *Wide & Deep* jointly trains a wide linear model (for memorization) alongside a deep neural network (for generalization), and combines the strengths of both.

2.2.1 Model Architecture

The wide model is a single layer perceptron, which has the capability of catching the direct features from historical data. Deep model is a multilayer perceptron, which catches the generalization by producing more general and abstract representations. The model is proved to be useful for large scale regression and classification problems with sparse inputs such as recommender systems, search, and ranking problems.

Input for wide model is a manually crafted combination of raw features and transformed features (cross product transformation of raw features to capture correlation between them). The input vector is formally represented as $\{x, \phi(x)\}$. $\{\phi(x)\}$ represents the cross product transformations of the original features x. Using this input, the output of the wide model is formulated as $y = W_{wide}^{T}\{x, \phi(x)\} + b$. Each layer

of deep model computes $a^{(l+1)} = f(W^{(l)}a^{(l)} + b^{(l)})$ where f is the activation function (i.e., ReLU); and $W^{(l)}, a^{(l)}, b^{(l)}$ are models weights, activations, and bias for the l^{th} layer. Combining these two models, the prediction of *Wide & Deep* model for binary classification is

$$P(Y = 1|x) = \sigma(W_{wide}^{T}[x, \phi(x)] + W_{deep}^{T}a^{(l_f)} + b) \tag{1}$$

where σ is the sigmoid function, and the symbols W_{wide}^{T}, W_{deep}^{T}, and b are weights of the wide and deep models, and the bias, respectively.

2.2.2 Feature Columns

Feature columns are intermediary transformers between estimator models and raw data. The set of future columns are rich enough to make it possible between a wide range of row features to data formats that models can use. We list types of different columns as follows:

1. Numeric Column: Numeric Columns are used to feed numerical attributes into learning models. The numeric column also has a shape attribute which makes it possible to feed numerical vectors or matrixes into the model.

2. Bucketized Column: Bucketized column is an extension to the numeric column and is used to discretize a numeric column. In other words, a bucketized column is a way to convert a numerical column into a categorical one. An example can be a numerical column which holds the year a house is built. With a bucketized column it is possible to set boundaries like $< 1960, >= 1960$ *but* $< 1980, >= 1980$ *but* $< 2000, >= 2000$ and place each year into one of these buckets. The resulting feature will be four digits one-hot encoding of year values. The advantage is that the model can learn four weights instead of one and has a better understanding of data.

3. Categorical Identity Column: Categorical Identity Column is a special version of Bucketized column. For this column type, each bucket represents an integer instead of intervals. The output is again a one-hot encoding vector.

4. Categorical Vocabulary Column: Categorical Vocabulary Column is used to feed string attributes into the model. Categorical vocabulary columns provide a good way to represent strings as a one-hot vector. Estimator API supports two types of categorical vocabulary columns based on how the set of strings are introduced to the model which are categorical_column_with_vocabulary_list and categorical_column_with_vocabulary_file namely.

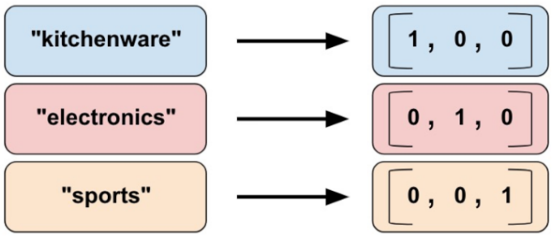

Figure 8: A mapping from string values to vocabulary columns [3]

5. Hashed Column: The hashed column is used to map attributes with a large number of possible values. The raw values are passed through a hash function and forced to map a smaller set of values limited by hash_bucket_size. Figure 9 shows a representation of this mapping procedure [2]

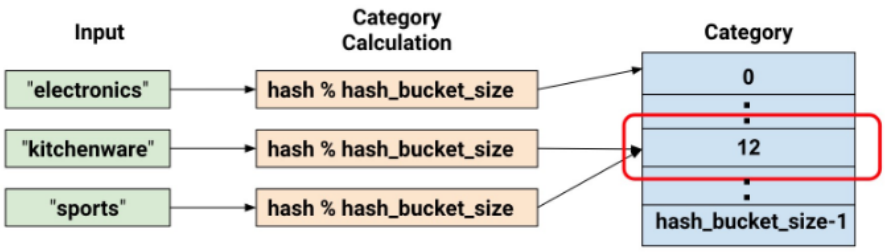

Figure 9: A representation of data with hash buckets [3]

6. Crossed Column: The crossed column is a way of feeding combinatorial features into the model. To create cross features, corresponding values of selected features are combined and passed through a hash function. The resulting values are mapped to a smaller set of values limited by hash_bucket_size.

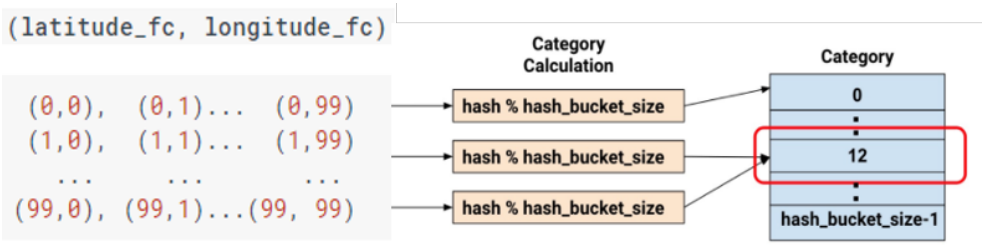

Figure 10: A representation of data with crossed column [3]

7. Indicator Column: Indicator columns never work on raw input but takes other categorical columns as input and covert to one-hot encoding representation. So that the resulting features can be fed as input to deep modols

8. Embedding Column: Indicator columns are used when the set of possible values is small. Embedding columns on the other hand is used when the data has high variation and using one-hot encoding is not efficient.

   Instead of representing the data as a one-hot vector of many dimensions, an embedding column represents that data as a lower-dimensional, ordinary vector in which each cell can contain any number, not just 0 or 1.

2.3 Feature Generation Methods

In this thesis, we explore methods to improve the performance of *Wide* & Deep learning models by eliminating the need for expert knowledge. There are some other methods in the literature that are specialized to find cross features without feature engineering. The two outstanding examples are DeepFM [4] and Deep Crossing[5].

2.3.1 DeepFM: A Factorization-Machine based Neural Network

DeepFM [4], which is short for Deep Factorization Machine, is an end-to-end model, which seamlessly integrates Factorization Machine (FM) and Multi-layer Perceptron (MLP). It is designed to work with highly sparse data as in Click Through Rate (CTR) prediction problem. CTR prediction is basically the problem of predicting whether the user will click to an online advertisement or not.

For CTR prediction models, it is important to learn underlying relations within the data. The key requirement is effectively modeling feature interactions. Some interactions like "male teenagers like shooting games and RPG games" are easy to understand since they can be observed by human experts. This observation is an order-3 relation between age, gender, and product category. However, there are many other feature interactions hidden in the data and impossible to be observed by experts. DeepFM is one of the deep learning models dealing with feature detection problem.

DeepFM architecture is an alternative to *Wide & Deep* models from Google [2] since it combines wide and deep models in one architecture. The main difference is that the DeepFM model uses the power of factorization machines to learn low-order interactions, so input to both deep part and the wide part are the same, which is the raw input. In comparison to *Wide & Deep* model, DeepFM does not require costly feature engineering. It replaces the wide component with a neural interpretation of the factorization machine. Figure 11 [4] illustrates the structure of DeepFM.

It can model the high-order feature interactions via deep neural network and low-order interactions via a factorization machine. The factorization machine utilizes addition and inner product operations to capture the linear and pairwise interactions between features.

As in Figure 11, the architecture consists of two components, Deep component, and FM component. Each input feature i has a scalar weight w_i and a latent vector V_i associated with it. The scalar w_i is used to weigh its order-1 importance while

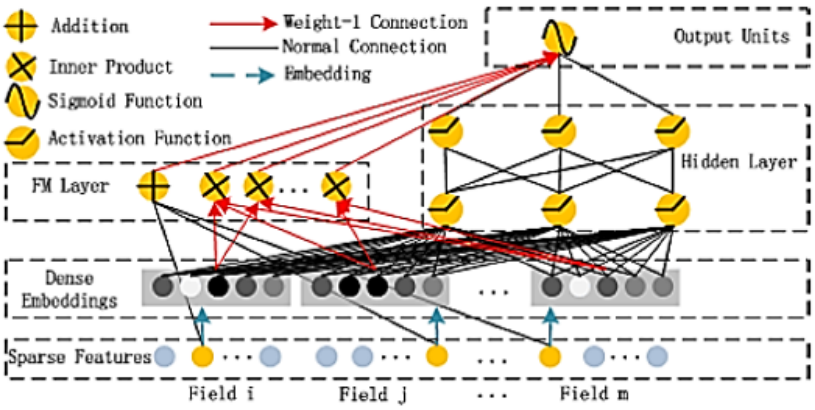

Figure 11: Wide & Deep architecture of DeepFM [4]

latent vector V_i is used to measure its impact of interactions with other features. The latent vector V_i is fed to both wide and deep components to learn order-2 relations and higher-order relations, respectively.

Each input feature is either categorical or numerical. Each categorical feature is fed into model as one-hot encoding while each numerical field is fed as itself or one hot-encoding after discretization.

All parameters, including w_i, V_i, and the network weighs and bias parameters are trained jointly for the combined prediction model:

$$y = sigmoid(y_{FM} + y_{DNN}) \tag{2}$$

The wide part of the model is a factorization machine, which is proposed by Rendle [17] to learn feature interactions for the recommendation. The FM component learns both order-1 and order-2 relations within the data. Order-2 relations are modeled as the inner product of latent vectors.

Figure12 [4] shows the architecture for FM Part of deepFM model. As Figure12 shows, the output of FM is the summation of an *Addition* unit and a number of *Inner Product* units:

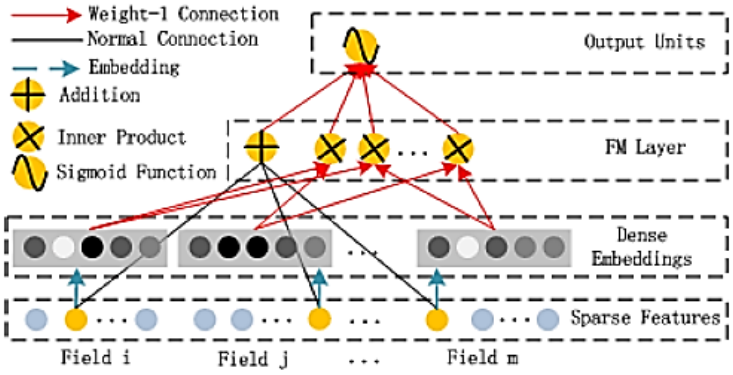

Figure 12: The architecture of FM [4]

$$y_{FM} = \langle w, x \rangle + \sum_{j1=1}^{d} \sum_{j2=j1+1}^{d} \langle V_i, V_j \rangle x_{j1}.x_{j2} \tag{3}$$

The Addition unit $\langle w, x \rangle$ reflects the importance of order-1 features, and the Inner Product units represent the impact of order-2 feature interactions.

The deep component is a feed-forward neural network that is used to learn higher-order relations. The latent vectors in FM are used to create embedding features in deep part by compressing original features. Even if the dimensionality of all input features is different, the size of all embedding features is equal.

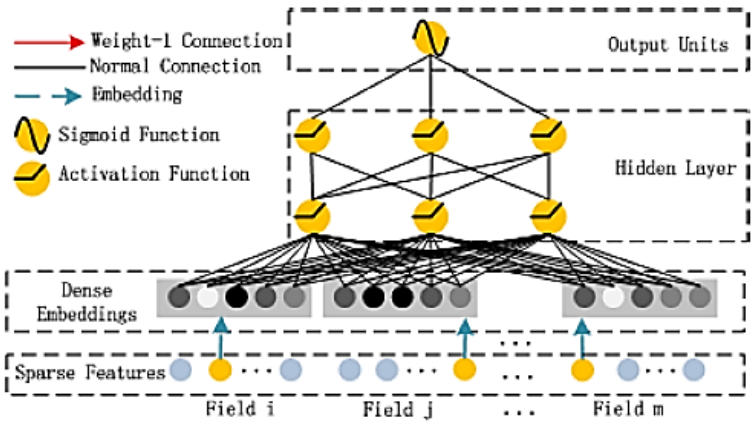

Figure 13: The architecture of DNN [4]

2.3.2 Deep Crossing

Deep Crossing is a model proposed by Shan et al. [5] as a solution to the problem of prediction with highly sparse heterogeneous inputs without manual feature generation. The original method is discussed as a solution to the sponsored search problem but can be generalized to other web-scale problems. Sponsored search is the problem of showing the most relevant advertisements which match the user intends the best. Intend here depends on the user's search query.

The proposed method has been used with a variety of raw (i.e., minimal processing, no combining/crossing features into more complex representations) dense, sparse, and text features to produce a model that provides superior performance to a state of the art model which used manual tuning. To motivate this, consider the problem that Deep Crossing seeks to solve: training in the presence of heterogeneous (especially textual) inputs.

Deep Crossing can be conceptually thought of as four separate operations combined into one. The model has four types of layers which are the Embedding, the Stacking, the Residual Unit, and the Scoring Layers namely. Figure 14 shows model architecture for deep crossing [5]. The objctive function is log loss. Log Loss is defined as below:

$$logloss = -\frac{1}{N}\sum_{i}^{N}(y_i log(p_i) + (1 - y_i) log(1 - p_i)) \tag{4}$$

where N is the number of samples, i is the index for the training samples, y_i is the actual label per sample and p_i is the output of scoring layer.

Since Deep Crossing treats each input individually, the embedding step of Deep Crossing is only concerned with high-dimensional inputs. Inputs that are low dimensional are given an identity embedding and passed directly through to the stacking layer. Inputs that are high dimensional (most notably text, however other high dimensional inputs are also considered here) are converted into low dimensionality via an embedding layer. Embedding is applied to per individual feature if the feature

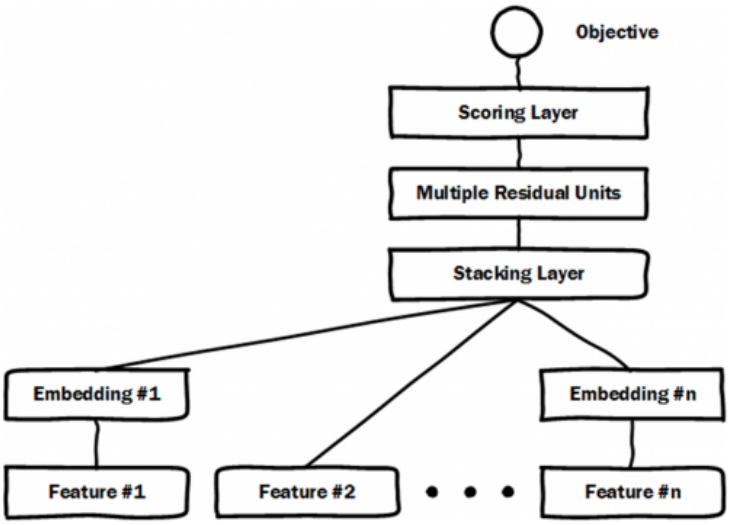

Figure 14: Deep Crossing model architecture [5]

is high dimensional or sparse. The embedding layer consists of single-layer neural networks with Relu activations. The size embedding layer has a significant impact on the resulting model size.

The stacking step is a pseudo-step in Deep Crossing. Its purpose is to combine the results of the embedding layer for each input into a single vector that can be used in the remainder of the network.

Residual units are powerful building blocks in neural networks. As the name suggests, the goal of a residual unit is to learn the residual between the (given) input and the (learned) output. This is accomplished by adding the input to the output of the last layer before the activation function. In this way, the identity transformation is explicitly encoded (by adding the input), and thus the layers must only learn the difference between the input and the "true" output.

2.4 Recommender Systems

Recommender systems are information filtering systems whose goal is to filter out irrelevant data and present information on items and products that are likely to

be of interest to the reader. Recommender systems are widely being used by e-commerce sites to improve user experience by predicting the preferences of the user and presenting the most relevant products. Many example applications can be listed as recommendations in web search, books, movies, music, restaurants, food, apparel, vehicles, targeted advertisements, medicines, news, potential customers for companies and many more.

A broad classification of recommender system classifies it into three categories, namely, Collaborative filtering, Content-based filtering, and Hybrid filtering. Figure 15 gives an outline of the classification [6]

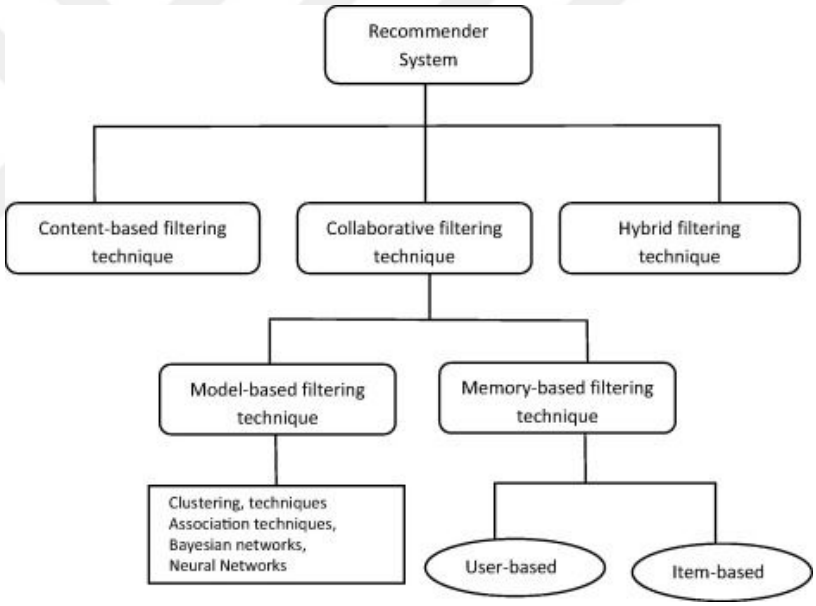

Figure 15: Classification of recommender system approaches [6]

2.4.1 Collaborative Filtering

Collaborative filtering is a method of making automatic predictions about the interests of a user by collecting preferences or ranking information from many users.

Collaborative filtering approaches can be grouped into two main categories: memory-based and model-based. Memory-based approaches make heavy use of user database to find customers with similar tastes. These methods use some explicit indicators

such as ranks given by the users or implicit information such as items bought by the user, page visits, and so on. The memory-based system can be either an item-item or a user-user system.

The item-item collaborative filtering systems focus on relations between items that are bought together. If items A, B, and C usually enter in the shopping cart together, the system recommends product C to users who added or bought item A and B.

In user-user collaborative systems, the focus shifts from item to the user, and the system detects similarities between purchase behaviors and rankings of users. If users A, B, and C have similar rankings or buying history, an item bought by A and B can be found interesting by user C.

Model-based collaborative filtering methods use a user database to learn a model that can predict user preferences. Deep learning models and clustering models also fall into this category.

2.4.2 Content Based Filtering

Content-based filtering methods focus on the preferences of a single user and model recommendation task as a user-centered classification problem. The system uses discretized, pre-tagged attributes of items and users which are item profiles and user profiles namely. In these systems, user profiles have the same attributes as item profiles and indicate user's preferences. The system decides based on the similarity between the user profile and item profiles.

2.4.3 Hybrid Filtering

Both content-based filtering and collaborative filtering have their limitations and strengths. Hybrid filtering methods combine the advantages of both approaches and can avoid their limitations. Collaborative filtering and content-based filtering can be combined by building two systems separately and later combining their results or

by adding content-based capabilities into the collaborative filtering system and vice versa.

Netflix movie recommender system is an example of such hybrid systems. The system makes recommendations based on the searching and watching behavior of similar users and also recommends movies that share common characteristics with the movies which are rated highly by the user.

2.5 Decision Tree Classification

In this thesis, we use decision tree models as base models to create cross features automatically. This section gives some background information on decision tree classification.

Decision trees are one of the most commonly used supervised predictive modeling approaches in machine learning. The idea is to use a tree-structured model to go from observations (branches in the nodes) about a sample to conclusions about the item's target value (labels in the leaves).

A tree structure starts with a root node. Each internal node denotes a test on an attribute, and each branch denotes an outcome of the test, and each leaf node holds a class label.

2.5.1 Decision Tree Learning Algorithms

Decision tree learning is the process of constructing a decision tree from labeled data samples. There are different algorithms proposed to construct decision trees. Some of the decision tree learning algorithms are listed below.

- ID3 (Iterative Dichotomiser 3)
- C4.5 (an extension of ID3)
- CART

- Chi-square automatic interaction detection (CHAID)

ID3 algorithm, which is developed by J. R. Quinlan [18], is the core algorithm for building decision trees. This algorithm employs a top-down, greedy search through the space of possible branches. ID3 uses Entropy and Information Gain to construct a decision tree.

C4.5 [19] builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. C4.5 made a number of improvements to ID3 such as handling both continuous and discrete attributes, handling training data with missing attribute values and pruning trees after creation

Chi-square Automatic Interaction Detector is proposed by G. V. Kass [20]. The algorithm stops sub-tree creation if not statistically significant by the chi-square test.

CART stands for Classification and Regression Trees is proposed by Breiman [21]. The algorithm uses the Gini index to measure the quality of each split.

2.5.2 Decision Tree Metrics

Decision tree learning algorithms usually work top-down and at each step find the variable which splits the data best. They measure homogeneity on subsets to decide the best split, and they use different homogeneity metrics to decide. To calculate the average homogeneity of a split, the metrics are applied to each subset, and the resulting values are combined. Some of the metrics used to measure homogeneity are listed below:

- Gini impurity
- Information gain
- Chi-Square

- Variance reduction

2.5.3 Gini Index

Used by the CART (classification and regression tree) algorithm for classification trees.

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified. It means an attribute with a lower Gini index should be preferred. Gini index performs only binary splits and CART (Classification and Regression Tree) uses the Gini method to create binary splits.

The Formula for the calculation of the of the Gini Index is given below:

$$GiniIndex = 1 - \sum_{i}^{c} p_i^2 \tag{5}$$

where c is the number of classes and p_i is the probability that an arbitrary sample belongs to class i

2.5.4 Entropy and Information Gain

Information gain is used by the ID3 , C4.5 and C5.0 tree-generation algorithms and it is based on the concept of entropy from information theory.

Entropy is a measure to calculate the homogeneity or impurity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is equally divided then its entropy becomes $log_2(c)$. The formula for entropy is as follows:

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{6}$$

where c is the number of classes and p_i is the probability that an arbitrary sample belongs to class i

The information gain is based on the amount of decrease in entropy after the data is split based on an attribute. The learning methodology is to find the attribute that

returns the highest information gain. The information gain is calculated as follows:

$$\begin{gathered} Information\ Gain = entropy(parent) - [average\ entropy(children)] \\ Gain(S, A) = Entropy(S) - \sum_{v \in values(A)} \frac{\|S_v\|}{\|S\|}\ Entropy(S_v) \end{gathered} \tag{7}$$

Where A is the selected attribute, v is one the possible values of A and S_v denotes set of all samples with value v on their A attribute.

2.5.5 Chi-Square

It is a measure to find out the statistical significance between the differences of sub-nodes and parent node. It is measured as the sum of squares of standardized differences between observed and expected frequencies of the target variable. The chi-square algorithm can make binary or higher splits. Higher Chi-Square values imply higher the statistical significance of differences between sub-node and Parent node. The formula for chi-square is as follows

$$X_c^2 = \sum_i \frac{(O_i - E_i)^2}{E_i} \tag{8}$$

Where E_i is the expected value before split based on the parent node and O_i is the observed value after the split.

2.5.6 Variance reduction

Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:

$$Variance = \frac{\sum(X - \bar{X})}{n} \tag{9}$$

Where $\bar{X}$ is the mean of the values, X is the actual, and n is the number of values.

Introduced in CART, [4] variance reduction is often employed in cases where the target variable is continuous (regression tree).

CHAPTER III

ENHANCING DEEP MODELS FOR CAMPAIGN PARTICIPATION PREDICTION

In order to predict whether customers are likely to accept the given offers, we propose the idea of using *Wide & Deep network models* with cross features due to their ability to combine *generalization* and *memorization* in one model. Furthermore, we propose to adopt decision trees for cross features required for that model instead of generating them manually. This chapter describes our methodology elaborately and also the evidential deep learning concept and its use in the campaign participation prediction task for quantifying prediction uncertainty.

3.1 Methodology for Campaign Participation Prediction

This section describes our general methodology to train and test models with the proposed cross feature creation method to predict the campaign participation behavior of customers. Figure 16 illustrates the steps taken in the process of campaign participation prediction. The first phase (i.e., task 1 in the figure) of our work is to collect data from a variety of data sources such as Campaign Management, Billing, and Real-Time Marketing systems. After collecting data, the second phase (i.e., task 2–5) is preprocessing the data for prediction task described as follows:

- Parsing offer messages automatically by using a set of regular expressions to obtain offer attributes (i.e., raw features of offers).
- Constructing the dataset for deep learning models.
- Balancing dataset using undersampling [22] so that the resulting datasets are

not biased towards any decision.

- Employing the proposed method in order to create cross features to be used by the wide model. The details of cross feature generation methodology have been explained in Section 3.3.

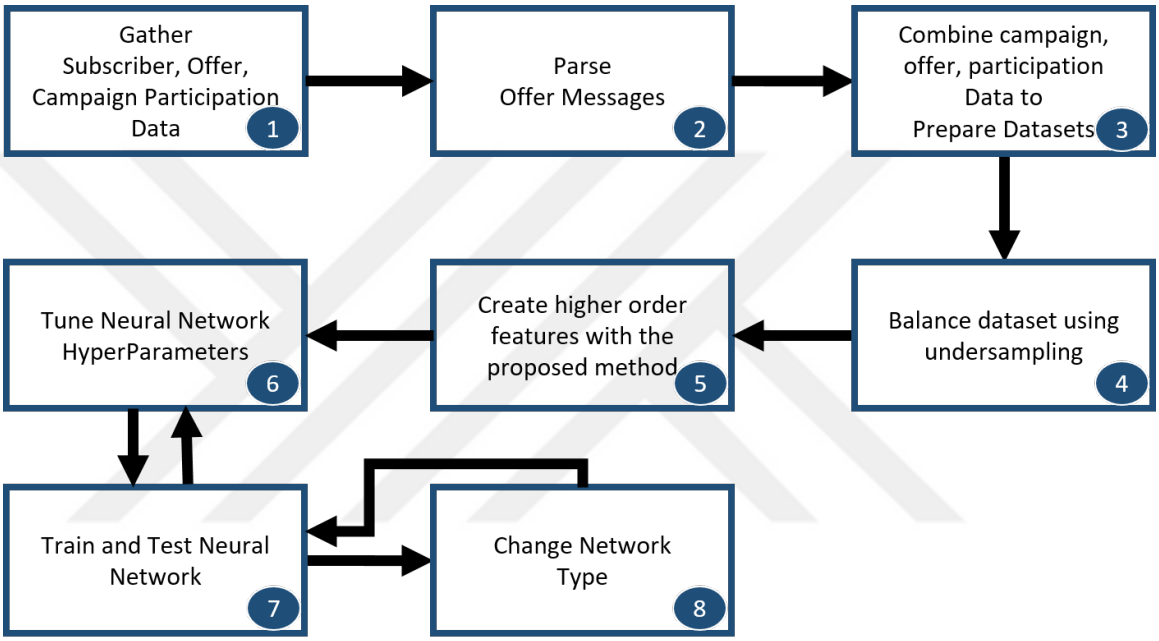

Figure 16: Overview of the methodology

The last phase (i.e., task 6–8) is the learning process in which the model for predicting campaign participation behavior of customers is built. Here, we construct *Wide & Deep* network models, which takes customer and offer attributes as inputs and outputs its prediction in terms of a probability distribution over the possible responses of the customer: *accept* or *reject*. It is worth noting that this model requires cross features provided by domain experts as input features for the wide network. For the case of campaign participation prediction, expert knowledge might be insufficient since it is not straightforward to capture the relations from the given data. To overcome this problem, we design and implement a decision tree based method, which automatically constructs interpretable cross features (i.e., task 5).

As usual, the learning process involves tuning hyper-parameters (i.e., task 6) such as the number of layers, the number of neurons in each layer, learning rate, regularization rate are selected. After finalizing the most suitable parameter set, the model is trained and tested with each dataset separately (task 7). For evaluation purposes, we train and test the Deep Network and *Wide & Deep* Networks with different feature sets (i.e., manually crafted cross features and automatically crafted cross features) so that we can observe the effectiveness of automatically crafting features in comparison to manually crafted cross features (task 8).

Apart from making accurate prediction, this thesis also focuses on the confidence of learners in order to prevent the consequences of false positive in campaign participation prediction. Therefore, we also extended *Deep* and *Wide & Deep* models in Estimator API with the methodology described by Şensoy *et al.* [9] and created *Evidential Deep*, *Evidential Wide & Deep* models in order to prevent failure cases and make more confident decisions. The details of these models and uncertainty concept are shared in Section 3.5

3.2 Data Collection and Preprocessing

In this thesis, we focus on the process and enhancement of campaign management for leading telecommunication companies. Such companies serve millions of customers with voice, data, TV and value-added consumer and enterprise services on mobile and traditional networks. This chapter describes how we collect and preprocess data obtained from a leading telecommunication service provider.

3.2.1 Campaign Management System for Telecommunication

We collected campaign participation data from Campaign Management, Billing, and Real-Time Marketing systems as described before. This section includes an overview of the campaign management and marketing platform of the selected provider. Figure 17 shows an overview of the campaign management system, which provides the

required participation dataset.

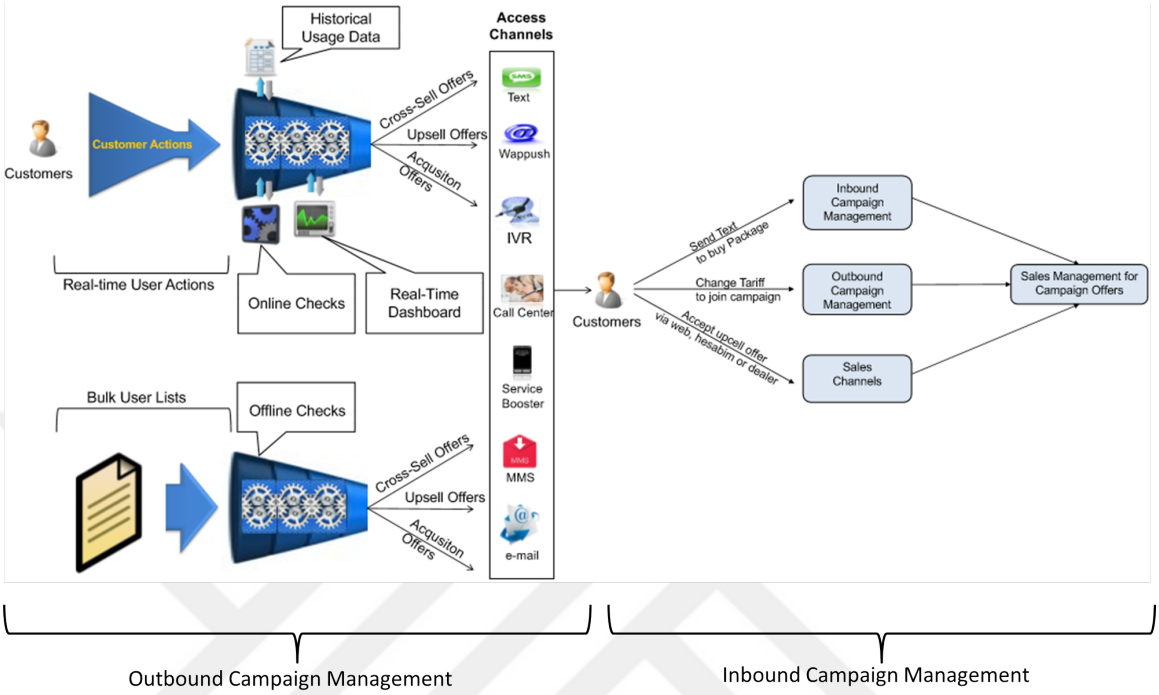

Figure 17: High-level architecture of campaign management system

The system described in Figure 17 consists of two main sub-systems: *outbound* and *inbound* campaign management systems. Outbound campaign management refers to any kind of marketing where a company initiates the conversation and sends messages out to an audience while inbound campaign management is the part where the conversation is initiated by the customer. Campaign management in total is a combination and coordination of these two systems. In this architecture, outbound marketing is responsible for making offers and sending advertisement messages to customers so that customers initiate another conversation to buy the advertised offers via inbound channels such as dealers, web site or applications.

The outbound campaign management system also consists of two sub-systems:*real-time* and *bulk* (or offline) campaign management systems. The real-time marketing system executes some scenarios (i.e., set of rules), listen for specific events (e.g., change of tariff), and executes some predefined eligibility conditions to detect target

customers for related campaigns before accessing customers with predefined methods. For instance, when a customer exceeds the data limit in his/her package, the system may make an offer to the customer to increase his/her data limit in exchange for an additional payment. The offer is decided and delivered to the customer based on the rules and instructions defined in the related campaign.

The offline system, on the other hand, takes bulk customer lists as input and apply some predefined offline checks and access the customers via specified channels, e.g., through call center representatives. Each list in this system is a set of target customers selected for a specific offer and an input for a predefined campaign strategy, which includes checks and access channels to deliver the offer to the target customer. To find the best match among customers and offers, these lists are composed by taking into account information about customers and offers, such as offer price, max sales limit, access limitations (daily, weekly and monthly allowed access counts per customer), and customer permissions. The company uses those data in order to maximize the revenue of the company under given constraints and limitations.

Offers and campaigns are put forwarded to customers via one of the access channels (text, email, web, etc.). Note that an offer is basically a recommendation to customers to increase both product or service sales and customer satisfaction. In order to participate in a particular campaign or accept an offer, they may send a text message to a specific short number (Inbound channel) or take a predefined action to attend the related campaign (outbound channel). Besides inbound and outbound channels, they may also participate via one of many sales channels such as dealers.

Whenever a customer is accessed via one of the access channels, this is called *campaign extended response*. If the user accepts the offers via inbound, outbound or one of other sales channels, this is called *campaign accepted response*. The percentage of the *campaign accepted response* to the *campaign extended response* defines the *success ratio* of the related campaign in targeting.

3.2.2 Campaign Participation Dataset

We performed a detailed analysis of campaign participation data of the GSM service provider and observed that the acceptance rates of offers differ significantly. Some offers have acceptance rates as high as 10% while some other offers are accepted only by 1% of the advertised customers. To make a fair evaluation and in-depth analysis of machine learning algorithms, we divided our offer dataset into three subsets based on the acceptance rates of the offers as follows:

1. **Rarely accepted offers:** This dataset contains data from offers whose acceptance rate is lower than 2%.

2. **Moderately accepted offers:** This dataset contains data from offers whose acceptance rate is between 2% and 7%.

3. **Highly accepted offers:** This dataset contains data from offers whose acceptance rate is higher than 7%.

The main intuition for the given percentages above relies on our data analysis. Our analysis has shown that only 10% of all offers lay below 2% acceptance rate and 10% of the offers lay above 7% acceptance rate. Thus, we defined the interval between 2% and 7% as the expected acceptance rate range for offers. An offer has a high likelihood of having an acceptance rate between those rates. Our goal here is to examine campaigns with similar acceptance rates together and to compare them with other groups based on acceptance behavior.

We picked 25 offers from each aforementioned category. An example offer sent to the customers as a text message is shown below:

"Get 500 minutes and 500MB only for 21 TL/month. You can join this campaign by texting 'MONTHLY 500MIN' to 1111 and start to benefit from this special offer"

These offers cannot be processed directly by machine learning algorithms. Therefore, we need to vectorize them to represent the content of the offers in a way that these algorithms can process. For this purpose, we selected offer attributes listed in Table 1 and implemented an algorithm to extract that information from the offer text using a number of regular expressions. More complicated methods such as using word embeddings with recurrent neural networks or self-attention mechanisms can be used to vectorize this data [23]. However, it requires significantly more offer samples; and more importantly, these methods are not in the scope of this work, and we preferred to keep the vectorization part simple to focus on evaluating the performance of the different models with respect to our main contributions.

Every offer in our dataset has a price and validity duration. These features can be thought of as mandatory features. All the other features extracted out of offer text are optional, and their existence depends on the offer type. For instance, an offer promoting a specific package has DATA, SMS, VOICE attributes, while an offer promoting a digital application has MEMBERSHIP, APP_DATA, or DISCOUNT attributes. Once, the offers are vectorized using these attributes; we can use various machine learning approaches to process the data.

After gathering and processing the data regarding offers/campaigns, we gathered the offers which are made to the customers and their responses accordingly. The response can be explicit, i.e., the customer *accepts* it, or implicit the customer does not do any action in response, meaning an implicit *reject*. Attributes of response data are listed in Table 2

The response data includes the campaign participation information of the customers and is originally imbalanced since the number of customers accepting each offer is much smaller than the number of customers not accepting it. To obtain balanced datasets, for each offer category, we randomly sampled 60000 records in a way that half of them are accepted while half of them are rejected. The response dataset

Table 1: Offer attributes

PRICE (TL)	Sales Price
DURATION(DAYS)	Amount of time during which offer can be used
DATA (MB)	Data allowance
DATA NIGHT (MB)	Data allowance, which can only be used at night time
VOICE (MINS)	Voice amount in minutes
SMS (#)	Sms Amount
DOUBLE_OFFER (BOOL)	Offer which duplicates bought credits
DOUBLE_LIMIT	Min. Amount of credit to be bought to activate doubling offer
INTRENATIONAL ROAMING	data/sms/voice in offer can be used in foreign countries or not
FREE_DATA (MB)	Data allowance given as a gift with an offer
FREE_VOICE (MINS)	Minutes given as a gift with an offer
FREE_SMS (#)	Minutes given as a gift with an offer
CANCELATION (BOOL)	Offer message includes how to cancel info or not
DISCOUNT (BOOL)	Applies promotion on sales price or invoice
ADVANCE (TL)	Allows to use extra credit (negative balance)
EXCLUSIVE (BOOL)	Offer message includes "just for you" text
MEMBERSHIP (BOOL)	Offer gives membership to services and applications
APP_DATA (MB)	Data allowance that can only be used with a specific application
OFFER (BOOL)	Offer requires a specific application to get activated and recommends the application
WEEKEND ONLY (BOOL)	data/sms/voice in offer can only be used at the weekend

is also the core dataset that connects offer data and subscriber data (i.e., customer). Each response record includes the offer_id which uniquely identifies offer made to customer and subscriber_id which uniquely identifies the subscriber who received the

Table 2: Response attributes

RESPONSE_HISTORY_ID	Unique id of response in Response collection System
SUBSCRIBER_ID	Unique id of subscriber in Response collection System
RESPONSE_TYPE	Type of campaign (Offline or Real-time)
RESPONSE_DATE (DATE)	Date of response
RELATED_OFFER_ID	Unique id of offer
RELATED_RESPONSE_ID	Unique id of response in Campaign Management System
RELATED_CAMPAIGN_ID	Unique id of campaign

offer. Figure 18 shows the relation between response, subscriber, and offer data.

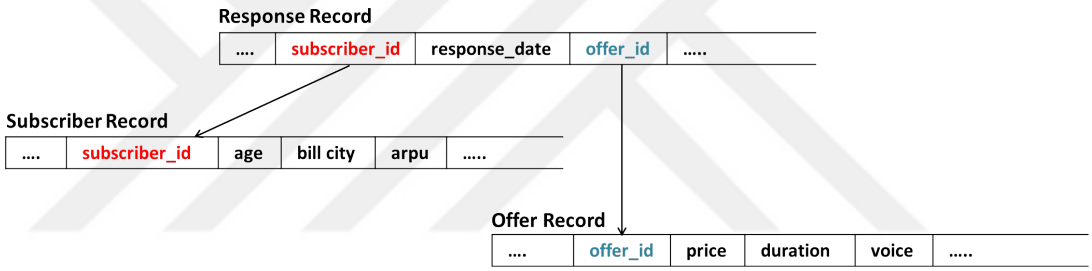

Figure 18: Relations among subscriber, offer and response data

Subscriber dataset basically includes features of those customers which have at least one response in our response dataset. These features include personal information such as age, gender, bill city, tenure as well as information regarding their usage habits such as average revenue per user (ARPU), voice usage, data usage, and so on. There are 146 attributes in the subscriber dataset. Some example attributes are listed in Table 3. The complete list of 146 attributes is not shared in this thesis due to confidentiality limitations. In the prediction process, we use those attributes to train the underlying models.

The output of our data collection process is three different campaign participation datasets. Each of these datasets contains customer usage behavior, demographic information, offer information, and users' campaign response (accepted and extended

Table 3: Examples of subscriber attributes

AGE	Subscriber age
AVG_DATA_USAGE_M3	Avg. Data usage for last three months
AVG_DATA_USAGE_M6	Avg. Data usage for last six months
GENDER	Subscriber gender
AVG_VOICE_USAGE_M3	Avg. voice usage for last three months
AVG_VOICE_USAGE_M6	Avg. voice usage for last six months
BILL_CITY	Billing city
TENURE	Subscription age in months
AVG_INVOICE_M3	Avg. invoice for last three months

responses) for a different acceptance rate interval.

User preferences and privileges are also taken into account during the data collection process; only the data of users who granted data processing permission is included in datasets.

3.3 Decision Tree Based Cross Feature Generation

The success of machine learning models strongly depends on the input features. Even though deep learning models are strong candidates to find hidden relations among attributes, they may fail in *memorizing the data* (i.e., "learning the frequent co-occurrence of items or features and exploiting the correlation available in the historical data") [2]. *Wide & Deep* learning models deal with this problem by combining *Wide & Deep* networks in a single model. However, they need manually crafted features requiring domain expertise knowledge.

campaign participation dataset, whose details are shared in Section 3.2.2, is a highly complex dataset with around 170 features and expert knowledge on this dataset is nearly absent. Even though human experts have information on the meaning of each feature in the dataset, they are unable to detect relations among features. This

is the main challenge we encountered while building efficient deep learning models for campaign participation prediction. To cope with this issue, we propose to detect cross relations automatically by using decision trees. Cross features are basically cross-product transformations of the original raw features and are used to represent these cross relations among features. The idea is to combine raw features so that the resulting feature is more informative than a feature itself. A more detailed explanation of cross features can be found in Section 2.2.2. This section includes details of the method for generating those features automatically.

The proposed method uses decision tree models to create cross features which requires also constructing the categorical features for continuous features automatically. Decision trees are good at finding the most important/informative features or feature sets and also finding boundaries for continuous features. For example, if the data has a continuous value such as age, the decision tree algorithm can find some boundaries such as (e.g., < 20, $20<x<50$, >50) for classification rules. Those boundaries can be used for the generation of categorical features (i.e., discretization of continuous values) effectively. Furthermore, rules generated by decision trees can be used to construct cross features. In other words, the branching structure in the decision tree can help us to find out the hidden relations among the features. This property makes decision tree a good candidate for generating cross features.

The proposed feature creation process consists of two sub-processes: (i) *tree traversal* and (ii) *feature generation*. The *tree traversal* process is basically the pre-processing step of the proposed method. The output of *tree traversal* process is a list of paths in the decision tree, which we will name as *path_list*, and a map of continuous features with their discretized boundaries, which we will name as *features_map* for the rest of this section. The output of this phase will be used in *feature generation* step. Figure 19 is the flow diagram for the *tree traversal* process.

The first step of the flow diagram in Figure 19 is to build a decision tree for the

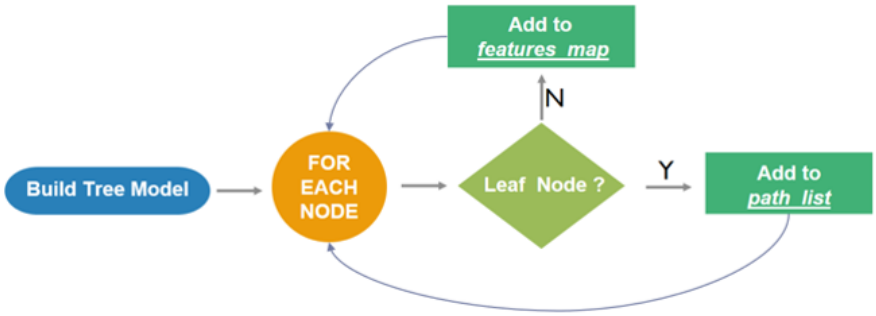

Figure 19: Flow diagram for the *tree traversal* processes

given dataset. Figure 20 is a conceptual tree model, which we will use as an example for the rest of this section to describe our approach. In this example, the sample dataset involves five features: $f1, f2, f3, f4, f5$ and we have three possible classes: A, B, and C. In the given representation, circle nodes denote leaf nodes capturing the associated class label. Note that the class label is determined by the majority rule on the dataset bounded by the path. For example, when the path is "$f1 < 50$ and $f2 < 100$ and $f1 > 15$", the example is classified as "A". The rectangular nodes represent branching conditions such as "$f1 < 50$".

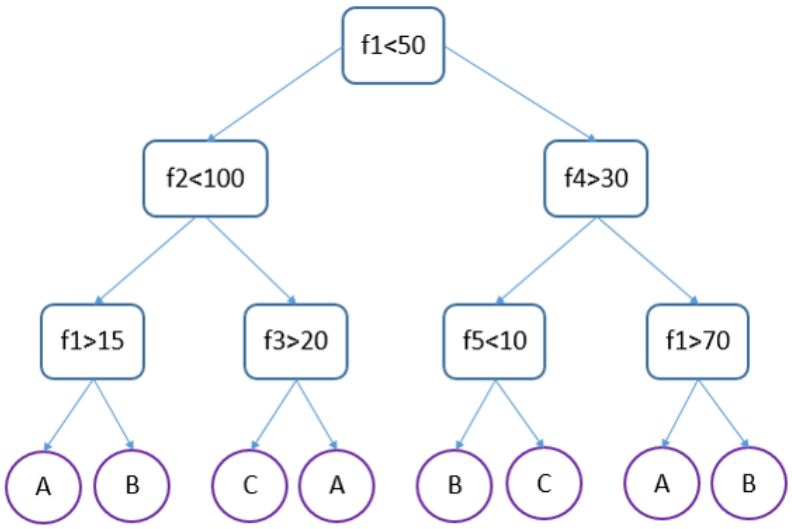

Figure 20: Sample tree model for feature generation

After building the decision tree, pre-order traversal is applied to visit every node in the tree. For each node visited, it is checked to see whether it is a test node with a branching condition or a leaf node with a class label. If the node is a test node then it is added to *features_map*; otherwise, it is added to *path_list*. In other words, when it reaches the leaf node, it considers the pre-ordered sequence of visited nodes as a path and accordingly adds it to the path list.

Adding a node to *features_map* does not mean adding the actual node but it means updating the *features_map* content according to branching condition on the node. The keys of *features_map* are the feature names such as $f1, f2$ and the values of *features_map* are the list of decision values such as $[15, 50, 70], [100]$ respectively. While adding a node to *features_map*, first we check if the map already has the current feature key. If it includes the key, we get the corresponding list and add the decision value to the list; otherwise, we add a new list with the feature name as key and add value to the newly created value list. As an example, the *features_map* for the decision three model in Figure 20 is as follows:

$$features\_map = \big\{f1 : [15, 50, 70], f2 : [100], f3 : [20], f4 : [30], f5 : [10]\big\}$$

As mentioned before, if the visited node is not a test node, which means it is a leaf node with a target class label; then the pre-order visit sequence of the nodes up to that node is added to *path_list*. Let's assume the case of our previous example with the decision tree model in Figure 20. The pre-order traversal of the whole tree is a linear representation of the tree and equal to: $\{f1, f2, f1, ClassA, ClassB, f3, ClassC, ClassA, f4, f5, classB, ClassC, f1, classA, classB\}$. A similar tree linearization method is used before to apply LSTM-based networks on tree-structured data [24], while here we are using it to extract new features of the data.

The pre-order traversal first visits branching nodes $f1 < 50$, $f2 > 100$, $f1 > 15$ in this order and then visits a leaf node, which is $Class = A$. Once $Class = A$ node is visited, pre-order visit sequence $f1, f2, f1$ is added to path list. After $Class = A$ is

visited, $Class = B$ leaf node is visited but this visit does not add a new path to the list, since $f1, f2, f1$ is already in the list. The pre-order traversal continues visiting $f3$, and then again a leaf node which is $Class = C$ is visited. Once $Class = C$ node is visited, the pre-order visit sequence $f1, f2, f1, f3$ is added to path list. The process continues to visit every node in the tree and adds every new path to the *path_list*. The resulting *path_list* for the sample tree is as follows:

$$path\_list = [[f1, f2, f1]$$
$$[f1, f2, f1, f3],$$
$$[f1, f2, f1, f3, f4, f5],$$
$$[f1, f2, f1, f3, f4, f5, f1]]$$

The outputs of *tree traversal* process, which are *features_map* and *path_list*, are inputs of the *feature generation* process. The goal of *feature generation* process is to generate categorical features and cross features that can directly be used by *Wide & Deep* models. Figure 21 is the flow diagram for the *tree traversal* process.

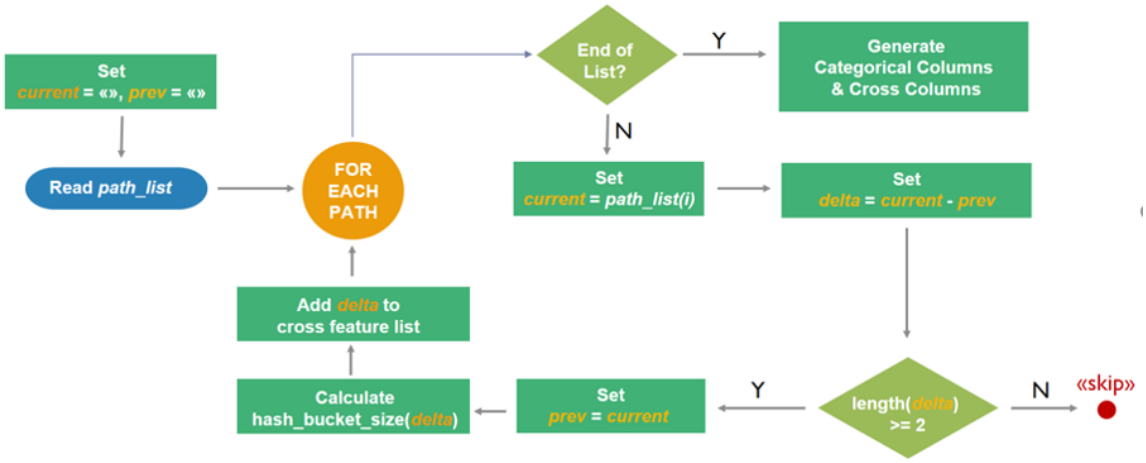

Figure 21: Flow diagram for automatic feature generation

The first step of the flow diagram in Figure 21 is to read *path_list*. The *feature generation* process iterates over each path in the path list in order. During these iterations, the difference of each path from the previous one is found in a way similar

to the delta encoding method. Delta encoding is a way of storing or representing sequential data in the form of differences, and it is also known as data differencing. In order to transform *path_list* into delta encoding format, we keep track of two variables, *current* and *prev* for the current path and previous path respectively. The difference of the *current* from the *prev* is called as *delta*. After finding delta, we check whether or not the length of the delta is greater than one. If the difference between two items is a single node, this delta value is skipped; otherwise, the delta value is added to the cross features list. Each *delta* value is a set of features, which will later be used to create cross features.

As an example, if we take the sample path list above as input, the delta encoding for this path list will be $\left[[f1,f2,f1],[f3],[f4,f5],[f1]\right]$. Since the length of the second and fourth delta items in the list is one, they are ignored during calculation. The resulting feature sets for cross feature generation is $\left[[f1,f2,f1],\ [f4,f5]\right]$. Since we use one-hot encoding in our learning methods, size of the possible values for each cross feature may play a crucial role. The designer may avoid large sized cross features due to the performance of learning algorithms. Therefore, in our system, we can determine the maximum value size for cross features by setting a hash_bucket_size and accordingly cross features are created for each feature set in the list.

To create a informative cross feature, we also need to set a reasonable hash_bucket_size for each *delta* (i.e., a set of features used for cross feature generation). Selecting small hash_bucket_size values results in high amounts of data collisions (i.e., different values resulting the same bucket) thus, data loss while picking very large values results in highly sparse and inefficient input features. We determine the hash_bucket_size by multiplying the number of possible values for each feature as shown in in Figure 22, which presents the pseudo-code for the whole process for automatic creation of cross features with decision tree models. More detailed explanation of pseudo-code is as follows:

```
1.  x_data = input_feature List
2.  y_data = input_labels
3.  num_of_levels = Number of levels for decision tree model to be used

4.  dtree = TRAIN_DECISION_TREE_MODEL(X_data, Y_data, num_of_levels)
5.  #{dtree_feature_values_map = {a= [19.5, 29.5], b=[1.0,2.5...] .....}}
6.  #{dtree_paths=['a,b,c', 'a,b,c,d,e', ......]}
7.  dtree_paths, dtree_feature_values_map = DEPTH_FIRST_TRAVERSAL(dtree)

8.  cross_features = []
9.  upper_bucket_limit = 100000
10. FOR path IN dtree_paths LOOP
11.     current = path.split(',')
12.     cross_feature = diff(current, features)

        hash_bucket_size = 1
13.     FOR feature in cross_feature LOOP
14.         hash_bucket_size = hash_buchet_size * (LEN(dtree_feature_values_map.get(feature))+1)
15.     END LOOP

16.     IF hash_bucket_size > upper_bucket_limit THEN
17.         sub_feature_list, sub_feature_sizes = SPLIT_CROSS_FEATURES(cross_feature, limit=1000)
18.         cross_features.appendAll(sub_feature_list)
19.     ELSE
20.         cross_features.append(cross_feature)
21.     END IF
22.     features.append(cross_feature.getItems())
23. END LOOP
24. numerical, bucketized, cross, indicator = creatorColumns(cross_features,
                                                  dtree_feature_values_map)
```

Figure 22: Pseudo-code for automatic feature generation

- **Lines 1–2:** Pre-processing and initialization of training data and model parameters such as the number of levels for the decision tree model.

- **Line 3:** Decision tree model training

- **Line 6:** Generation of *features_map* and *path_list* with depth_first_traversal. The details for the creation of *features_map* and *path_list* are explained previously in this section.

- **Lines 7–9:** Initialization of temporary variables such as cross_features, prev, current and delta and upper limit for hash bucket size.

- **Lines 10–12:** Detection of unique, non-repeating paths with a methodology like delta encoding of the decision tree paths. At this step, every decision path is

subtracted from the next path in the list so that only the newly discovered paths added to the cross feature list and the repeating paths within cross features are eliminated.

- **Lines 13–15:** For each cross feature generated from each path of the decision tree, hash bucket size is calculated by multiplying the number of possible values for each feature in this path.

- **Lines 16–18:** If hash bucket size value is very large (e.g., over 100000), the list of features is split into smaller feature sets as long as they are separable and the list contains at least four features. The rule of thumb while splitting cross features is to split them in such a way that hash bucket size for each feature set is less than a reasonable value such as a ten thousand or less. For our datasets, we applied this extra split operation for one or two cross features, and the experimental results have shown that cross features with very large hash bucket size give worse results in terms of efficiency in comparison to the ones with smaller hash bucket size.

- **Lines 19–21:** If hash bucket size value is within allowed limits, the cross feature is added to the list with the calculated size value.

- **Line 24:** At this step, the calculated output is used to create feature columns that can be used by the neural network models in the estimator API. This is the point where the proposed method integrates with the neural network model.

Figure 23 is another example of decision tree models, which are being used to create cross features on campaign participation dataset. Note that this example is simplified for demonstration purposes. Figure 24 shows a sample representation of *features_map* with some example values created based on the decision tree model in Figure 23. Let's consider *SUM_REFILL_M1* and *PRICE_TL* features. According

to the decision tree in Figure 23, raw feature *SUM_REFILL_M1* is used in five test nodes and each test node checks with a different boundary value such as *1, 2.5, 5.5, 46 and 57.5* while $PRICE\_TL$ feature is used in two nodes and these nodes make checks on values *19.5 and 29.5*. As a result, the resulting *features_map* includes SUM_REFILL_M1 and $PRICE\_TL$ as keys and the values are the corresponding value lists for these features: $[1, 2.5, 5.5, 46, 57.5]$ and $[19.5, 29.5]$.

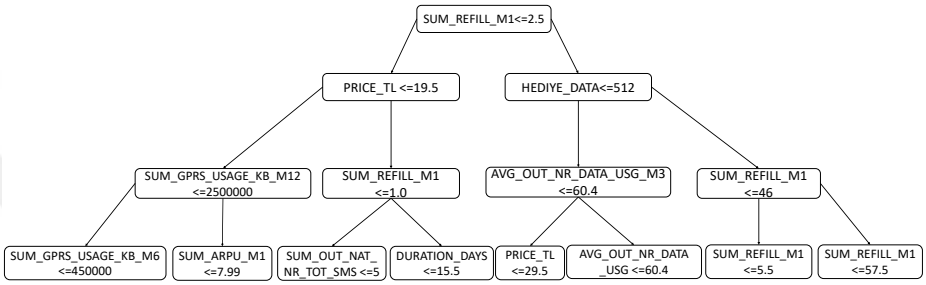

Figure 23: Decision tree model example

```
{ SUM_REFILL_M1: [1, 2.5, 5.5, 46, 57.5],
  PRICE_TL: [19.5, 29.5],
  HEDIYE_DATA: [512]
  ...
}
```

Figure 24: Sample representation for feature map

Sample representation of *path_list* with three initial sample paths is shown in Figure 25. If we transform this *path_list* into delta encoding format, we end up with the differencing list $\big[[SUM\_REFILL\_M1, PRICE\_TL, SUM\_GPRS\_USAGE\_KB\_M12, SUM\_GPRS\_USAGE\_KB\_M6], [SUM\_ARPU\_M1], [SUM\_REFILL\_M1, SUM\_OUT\_NAT\_NR\_TOT\_SMS]\big]$ which is also shown with red color in Figure 25.

The final output of the proposed method is a set of categorical features and cross features, as shown in Figure 26. As seen from this figure, the first path $[SUM\_REFILL\_M1, PRICE\_TL, SUM\_GPRS\_USAGE\_KB\_M12, SUM\_GPRS\_USAGE\_KB\_M6]$ is considered as a cross feature while $SUMARPU\_M1$ is taken as a categorical feature.

SUM_REFILL_M1, PRICE_TL, SUM_GPRS_USAGE_KB_M12, SUM_GPRS_USAGE_KB_M6

SUM_REFILL_M1, PRICE_TL, SUM_GPRS_USAGE_KB_M12, SUM_GPRS_USAGE_KB_M6, SUM_ARPU_M1

SUM_REFILL_M1, PRICE_TL, SUM_GPRS_USAGE_KB_M12, SUM_GPRS_USAGE_KB_M6, SUM_ARPU_M1, SUM_REFILL_M1, SUM_OUT_NAT_NR_TOT_SMS

Figure 25: Sample representation for path list

Categorical Feature Examples	PRICE_TL_CAT = tf.feature_column.bucketized_column(PRICE_TL, boundaries=[19.5, 29.5])
	SUM_REFILL_AMT_M1_CAT = tf.feature_column.bucketized_column(SUM_REFILL_AMT_ M1, boundaries=[1.0, 2.5, 5.5, 46.0, 57.5])
Cross Feature Example	tf.feature_column.crossed_column([SUM_REFILL_AMT_M1_CAT,PRICE_TL_CAT, SUM_GPRS_USAGE_KB_M12_CAT, SUM_GPRS_USAGE_KB_M6_CAT], hash_bucket_size=162)

Figure 26: Sample categorical & cross features

3.4 Networks with Automatically Generated Features

This section explains how we combine the proposed method with *Wide & Deep* network models to eliminate the need for manually crafted features. *Wide & Deep* network models are joint networks consisting of a wide part and a deep part. Original network model feeds categorical features and manually crafted cross features as input to wide part and feeds numerical features and categorical features to deep part. In other words, categorical features are used in cross features and fed into the wide part of the network to increase memorization capabilities. They are also fed into the deep network to learn hidden relations and to increase the generalization capability of the network. As a result, generating efficient categorical features is as important as generating efficient cross features.

The outputs of the cross feature generation process, namely numeric_column (i.e.

a single numeric feature), bucketized_column, crossed_column and indicator _column (i.e., a single categorical feature), are used by the neural network. For each key value in *features_map*, we create categorical features (bucketized_column) based on the value lists in *features_map*. For example, consider our previous example. For feature f_1, the boundary list $[15, 50, 70]$ splits numerical column f_1 into 4 mutually exclusive areas which are $f_1 \leq 15$, $15 < f_1 \leq 50$, $50 < f_1 \leq 70$ and $70 < f_1$. The resulting feature is a one-hot encoded 4 digit categorical column, which can take one of four possible values. The model learns four weights instead of one for this specific example. Consequently, we enhance learning capability from data by increasing the number of learned weights per feature. The categorical features for the sample decision tree in Figure 20 is listed as follows.

$$categorical\_f_1 = tf.feature\_column.bucketized\_column(f_1, boundaries = [15, 50, 70])$$

$$categorical\_f_2 = tf.feature\_column.bucketized\_column(f_2, boundaries = [100])$$

$$categorical\_f_3 = tf.feature\_column.bucketized\_column(f_3, boundaries = [20])$$

$$categorical\_f_4 = tf.feature\_column.bucketized\_column(f_4, boundaries = [30])$$

$$categorical\_f_5 = tf.feature\_column.bucketized\_column(f_5, boundaries = [10])$$

These newly created categorical features (bucketized_column) are used with original categorical features to create cross features (crossed_column). The generated code for cross features are used within the model without any modification. The cross feature list for the sample decision tree in Figure 20 includes two values which are $[f_1, f_2, f_1]$ and $[f_4, f_5]$. As mentioned before cross features are cross product transformations of the categorical features. While generating cross features we use corresponding categorical features instead of numerical ones. Cross features generated for

the sample decision tree in Figure 20 is listed as follows.

$$cross\_feature1 = tf.feature\_column.crossed\_column(\\ [categorical\_f_1, categorical\_f_2, categorical\_f_1],\\ hash\_bucket\_size = [32])$$

$$cross\_feature2 = tf.feature\_column.crossed\_column(\\ [categorical\_f_4, categorical\_f_5],\\ hash\_bucket\_size = [4])$$

The *hash_bucket_size* values 32 and 4 are calculated based on the methodology described in pseudo-code in Figure 22. We multiply possible number of values for each categorical feature in the generated cross feature. As an example, *cross_feature1* consists of three categorical features which are *categorical_f_1, categorical_f_2, categorical_f_1* and the *hash_bucket_size* is the multiplication of maximum possible values for each feature which is $4 \times 2 \times 4 = 32$.

The categorical and cross features generated by the proposed method are added to the original feature set in campaign participation data.

3.5 Uncertainty-aware Predictions

In order to prevent false positives in predictions and to make more reliable decisions, uncertainty-aware network models are used for this work instead of standard network models. The section provides details on uncertainty concept and their extension for evidential deep learning models.

3.5.1 Uncertainty and the Theory of Evidence

Dealing with uncertainty is a fundamental issue for AI. There are many potential sources of uncertainty that AI systems must be able to cope with; the reason may be imperfect domain knowledge, imperfect case data (errors in sensor data), and so on.

Several alternative logics that take uncertainty and ignorance into consideration have been proposed and successfully applied to practical problems.

The theory of belief functions also referred to as evidence theory or Dempster-Shafer theory (DST) [25], which is a general framework for reasoning with uncertainty. Rather than computing probabilities of propositions, it computes probabilities that evidence supports the propositions. This measure of belief is called a belief function, written $Bel(X)$. A binomial opinion applies to a single proposition and can be represented as a Beta distribution. A multinomial opinion applies to a collection of propositions and can be represented as a Dirichlet distribution. Through the correspondence between opinions and Dirichlet distributions, Subjective Logic [26] provides an algebra for these functions. If there are K possible outcomes or class labels for a sample, Subjective Logic assumes K mutually exclusive singletons and assigns a belief mass to each of them and expresses "I do not know" as an opinion for the truth over possible states. K mass values and an uncertainty value for "I do not know" are all non-negative and sum up to one as represented in Equation 10. In other words, the sum of all belief masses for the assignment of a sample to K categories in classification tasks does not necessarily add up to one, due to the existence of a non-zero uncertainty.

$$u + \sum_{k=1}^{K} b_k = 1 \tag{10}$$

A belief mass b_k for a singleton k (i.e., corresponding class) is computed using the evidence. Belief, uncertainty and the Dirichlet strength (S) are calculated in Equation 11 where the evidence e_k for a singleton is the amount of support for the classification of a sample; belief mass b_k for a singleton is directly proportional to the evidence collected for class k; sum of all evidences plus number of categories (K) is the *Dirichlet Strength* and uncertainty is inversely proportional to it. Note that according to formulation, if all evidences are zero, the belief for each singleton

becomes zero as well and uncertainty becomes 1. The reverse interpretation is also possible. If evidence for at least one class is very large, Dirichlet strength gets very large and uncertainty approaches to zero.

$$b_k = \frac{e_k}{S}, \qquad u = \frac{K}{S}, \qquad S = \sum_{k=1}^{K}(e_k + 1) \tag{11}$$

3.5.2 Evidential Deep Learning (EDL)

The Dempster–Shafer Theory of Evidence (DST) assigns belief masses to the set of exclusive possible states, e.g., categories in classification tasks, [25] so that "I do not know" can be mapped as an opinion for the truth over possible states. Subjective Logic (SL) formalizes DSTs notion of belief assignments as a Dirichlet Distribution [4]. Sensoy *et al.* indicates that a neural network is also capable of forming opinions for classification tasks as Dirichlet distributions [9]. The proposed model deals with uncertainty estimation problem and approaches the problem from a theory of evidence perspective [25, 26]. The standard output of a classification problem is logits, which are fed to softmax function to calculate a categorical distribution for the sample. On the other hand, EDL converts the logits into evidence for each category using a non-negative activation function. By doing so, it creates a Dirichlet distribution over possible softmax outputs, instead of a single point estimation of the categorical distribution for the sample.

Evidential neural networks for classification are very similar to classical neural networks. The model differs from the original model with its output layer and the loss function. In these models, the softmax layer is replaced with a non-negative activation layer such as *relu*, *exponent*, or *softplus* to create non-negative output. Then, the output of the network model for each target class is taken as evidence for that class. The evidence e_k derived for the k^{th} class for a given sample is, then, computed as $e_k = relu(logits_k)$, where $logits_k$ is the output of k^{th} neuron in the output layer of the neural network for the sample. The derived evidence is used to calculate

parameters of a Dirichlet distribution, which is indeed a probability distribution over all possible categorical distributions for labels. While classical softmax output of a neural network is a point estimation of a categorical distribution, the derived Dirichlet distribution is its prior.

Let y_i be the ground-truth for observation x_i in one-hot encoding format, $\boldsymbol{e}_i$ be the output evidence vector of the network for the same sample x_i and $\boldsymbol{\alpha}_i$ be the parameter vector of the Dirichlet density on the predictors, i.e., $\boldsymbol{\alpha}_i = \boldsymbol{e}_i + \mathbf{1}$. y_i encodes the ground-truth for observation x_i with $y_{ij} = 1$ and $y_{ik} = 0$ for all $k \neq j$. Then, the means of the corresponding Dirichlet distribution α_{ij}/S_i is taken as an estimate of the probability that the sample belongs to the j^{th} class.

The loss function used in the original paper depends on the sum of squares loss and calculated with the following formula, where Γ is gamma function, ψ is digamma function, and λ_t is annealing parameter.

$$loss_i = \underbrace{\sum_{j=1}^{K}(y_{ij} - \alpha_{ij}/S_i)^2}_{\text{data fit}} + \underbrace{\frac{\alpha_{ij}(S_i - \alpha_{ij})}{S_i^2(S_i+1)}}_{\text{Dirichlet variance}} + \lambda_t \underbrace{log\left(\frac{\Gamma(\sum_{k=1}^{K}\tilde{\alpha}_{ik})}{\Gamma(K)\prod_{k=1}^{K}\Gamma(\tilde{\alpha}_{ik})}\right) + \sum_{k=1}^{K}(\tilde{\alpha}_{ik}-1)\left[\psi(\tilde{\alpha}_{ik}) + \psi(\sum_{k=1}^{K}\tilde{\alpha}_{ik})\right]}_{\text{KL divergence with the uniform Dirichlet distribution}} \quad (12)$$

where t is the current training epoch and λ_t is the annealing coefficient and calculated according to equation $\lambda_t = min(1.0, t/10) \in [0, 1]$. In this formula, $\tilde{\alpha}_i$ represents the Dirichlet parameters after removal of evidence for the true label of the sample and calculated with equation $\tilde{\boldsymbol{\alpha}}_{\boldsymbol{i}} = \boldsymbol{y_i} + (1 - \boldsymbol{y_i}) \odot \boldsymbol{\alpha_i}$.

The first term in Equation 12 aims to minimize the prediction error; the second term aims to increase the certainty by decreasing the variance, and the last term minimizes the evidence for the wrong labels. By combining these three terms, the loss function aims to achieve the joint goals of minimizing the error and the variance

while being uncertain in its wrong predictions.

The loss function is optimized to generate more evidence for the correct class labels for each sample and avoid misleading evidence. The loss also tends to shrink the variance of its predictions on the training set by increasing evidence, but only when the generated evidence leads to a better data fit.

During training, the proposed model may discover patterns in the data and generate evidence for specific class labels based on these patterns to minimize the overall loss. The model assigns evidence to each class as long as the evidence assigned to the correct class is higher than the evidence for other classes. However, the loss tends to reduce evidence to zero for incorrectly classified samples through the Kullback-Leibler (KL) divergence term in the loss function. That is, the KL divergence term regularizes predictive distribution by penalizing those divergences from the "I do not know" state that do not contribute to data fit.

To prevent the consequences of false positive in campaign participation prediction, we extended *Deep* and *Wide & Deep* models in Estimator API with the methodology described by Şensoy *et al.* [9] and created *Evidential Deep*, *Evidential Wide & Deep* models.

Original *Deep* and *Wide & Deep* models in estimator API uses cross-entropy loss function and *sigmoid* function at their final layer for binary classification. In other words, they have a single output in their final layer and generate a single value between 0 and 1 for binary classification. For multi-class classification tasks, on the other hand, they have multiple outputs in the output layer, i.e., one output for each class label. Thus, the first step of building uncertainty-aware models is to change the last layer so that it will give two outputs, one for each Dirichlet parameter. Then, instead of using a sigmoid function to predict class probabilities, we convert logits (i.e., the output of the network) to evidence using *relu* activation functions, so that the output of *Evidential Deep* and *Evidential Wide & Deep* models is an evidence

vector instead of a point estimate of categorical distribution.

The third step of our adaptation is to replace cross-entropy loss function in the Tensorflow's estimator API with the loss function described above [9]. With the change of loss function, the resulting model optimizes the α parameters of a Dirichlet distribution and outputs prediction uncertainty besides predicted class values. For each observed sample i uncertainty is calculated with the following formula as in [9]:

$$u_i = \frac{K}{\sum_{k=1}^{K} \alpha_{ik}} \tag{13}$$

where K is the number of classes.

CHAPTER IV

EXPERIMENTS AND RESULTS

This section evaluates and compares our approach with other deep learning models and provides results of our experiments on campaign participation datasets. We also performed experiments on other datasets such as adult income (See Appendix A for details) and Criteo datasets (See Appendix B for details) to see the effectiveness of the proposed method on different problems. Details for additional experiments are shared in Appendix A and B.

The following network models are trained and tested to evaluate the performance of the proposed feature generation method.

- **Deep:** Deep model is the base model for our performance evaluation.
- **Wide & Deep:** Wide & Deep model is the original model proposed by Cheng *et al.* This model uses manually crafted cross features from human experts.
- **Wide & Deep with Decision Tree based columns:** *Wide & Deep Model with Decision Tree based columns* is our proposed model and uses automatically created cross and categorical features which are generated by the proposed cross feature generation method.
- **Wide & Deep with Decision Tree based columns + manually crafted columns:** This model is a modification over *Wide & Deep with Decision Tree based columns* model and uses both automatically created features which are generated by the proposed feature generation method and also the manually crafted ones. We include this model in our experiments to see whether there is still a need for expert knowledge besides automatically crafted features.

- **DeepFM:** Deep Factorization Machines is a model proposed by for CTR prediction by Guo *et al.* This model also eliminates the need for expert knowledge. We include this model in our experiments to compare the proposed model with an existing model dealing with the same problem.

For each experiment, we define a parameter set including the model parameters such as the number of levels, learning rate, and type of activation functions and use the same settings for all models to make a fair evaluation. The parameters for each experiment are shared in the corresponding experimental setup section.

Tensorflow's estimator API supports some performance metrics such as AUC (area under curve) and accuracy but does not support metrics to measure uncertainty such as evidence. We also added metrics listed below to evaluate our models in terms of uncertainty and generated evidence.

- **Mean evidence:** Average evidence measured on all samples
- **Mean evidence fail:** Average evidence measured on incorrectly classified samples.
- **Mean evidence success:** Average evidence measured on correctly classified samples.
- **Mean uncertainty:** Average uncertainty measured on all samples
- **Mean uncertainty fail:** Average uncertainty measured on incorrectly classified samples.
- **Mean uncertainty success:** Average uncertainty measured on correctly classified samples.

4.1 Campaign Participation Dataset

4.1.1 Experimental Setup for Campaign Participation Datasets

To evaluate effectiveness of our cross feature generation methodology, we ran two set of experiments on campaign participation datasets.

1. **Random Split:** For this set of experiments, we split our datasets into training and test datasets randomly to see the effectiveness of created features when both test and training data include samples of all offers.

2. **Mutually Exclusive:** For this second set of experiments we selected some offers as test offers for each dataset and split their data as test dataset so that training data does not include any sample test offers. The goal of the second experimental setup is to see the effectiveness of created features on a set of unseen, possibly out-of-distribution offers.

For both experiments, we applied 1/3 ratio between test and training data.

We used accuracy as an evaluation metric and evaluated seven different models in total. The results indicate that the proposed cross feature generation methodology improves the performance of *Wide & Deep* models and produce more efficient results when compared to models with manually crafted features.

We also measured evidence and uncertainty values of predictions to check whether the model decides based on evidence or randomly in lack of evidence. We measured mean uncertainty values over all successful and unsuccessful predictions separately to obtain the threshold levels for the models in which models are confident.

To find a common parameter setting that gives the best performance on all models, we performed a careful parameter evaluation step. The resulting parameter settings are listed below:

1. Network Structure: 100-75-50-25-10

2. Learning rate: 0.01

3. L1 regularization rate: 0.01

4. dropout rate: 0.05

5. Activation function: relu

6. Optimizer: adam

For the campaign participation dataset, decision tree models with $70 - 75$ % accuracy are used to find fieldsets which are more distinctive when used together. The decision trees used for all six campaign participation experiments are built based on this rule. Trees with $70 - 75$ % accuracy and seven levels are used to create cross features.

4.1.2 Results on Randomly Selected Campaign Participation Datasets

The efficiency of different models with different set of input attributes are evaluated on randomly split campaign participation datasets of GSM customers. The results are shown in Table 4

Table 4: Accuracy values of different models on random split datasets

Model	Less Than 2%	Between 2% and 7%	Over 7%
Decision Tree	0.75	0.7	0.7
Random Forest	0.75	0.73	0.7
Deep	0.74	0.91	0.88
Wide & Deep (with manually crafted columns)	0.74	0.92	0.90
Wide & Deep with Decision Tree based columns	**0.792**	**0.981**	**0.971**
Wide & Deep with Decision Tree based columns + manually crafted columns	0.785	0.98	0.96
DeepFM	0.71	0.68	0.67

One observation out of this data is that none of the models achieves above 80% accuracy for the least participated dataset which is "Less Than 2%" namely. This may

be due to customer diversity in this dataset. Those campaigns with low acceptance rates are the ones that are not commonly accepted by their target audience but accepted by a group of customers with different characteristics. It may be more difficult to find common characteristics or patterns within data for this dataset.

The results also indicate that *Wide & Deep* network with manually crafted features improves the performance of *Deep* model by 1 or 2% while the proposed model with automatically created features improves performance from 5% on least participated dataset up to 9% on most participated dataset. Note that *Wide & Deep model with Decision Tree based columns + manually crafted colums* model shows similar performance to proposed *Wide & Deep model with automatically crafted features* for all datasets. This result indicates that using expert knowledge besides automatically created features is not needed.

DeepFM [4] seems to be the least successful model in terms of accuracy when compared to other *Wide & Deep* models. For rarely accepted (less than 2%) dataset, it has relatively better accuracy when compared to the other two datasets. This may be due to the dense nature of campaign participation datasets. As sparsity in data increases, the performance of deepFM increases, since it is built on the sparsity assumption in the data. Again *Wide & Deep* model with automatically created features has better performance on all datasets, which proves the success of automatically created features.

Figure 27, Figure 28 and Figure 29 show change of accuracy per epoch for compared neural network models on highly accepted (Over 7%), moderately accepted (between 2% and 7%) and rarely accepted (less than 2%) datasets, respectively. The models with automatically created features seem to have a steeper learning curve in initial steps when compared to other models. One possible reason is that starting with more discriminative or ready to use features helps those models to learn faster than others in initial steps.

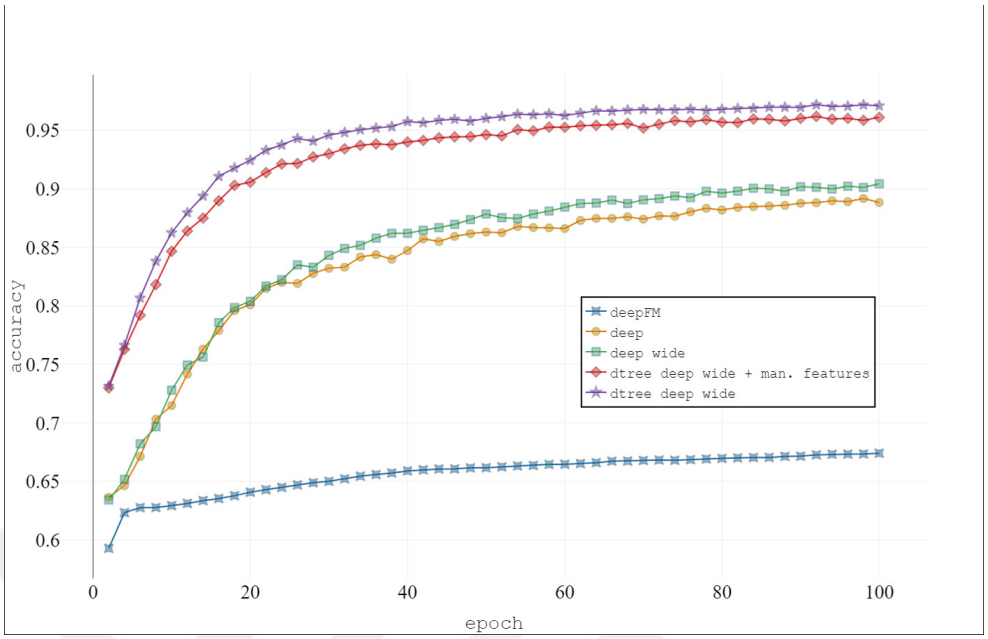

Figure 27: Accuracy comparison on highly accepted (Over 7%) dataset

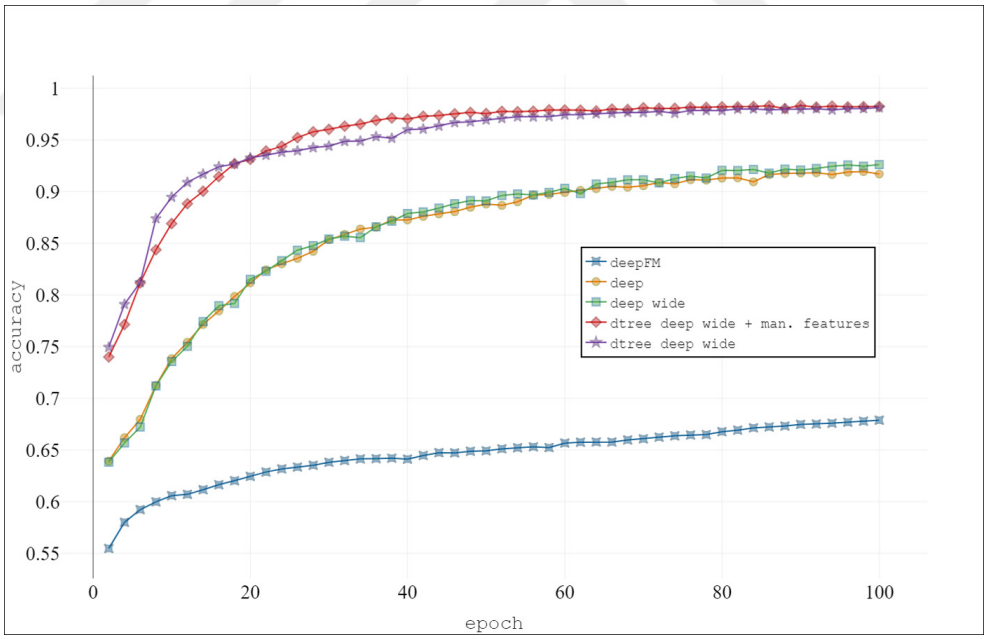

Figure 28: Accuracy comparison on moderately accepted (Between 2% and 7%) dataset

4.1.3 Results on Mutually Exclusive Campaign Participation Datasets

In the previous section, we use data for training and testing from the same set of offers. However, in this section, we train our model on data from one set of offers and

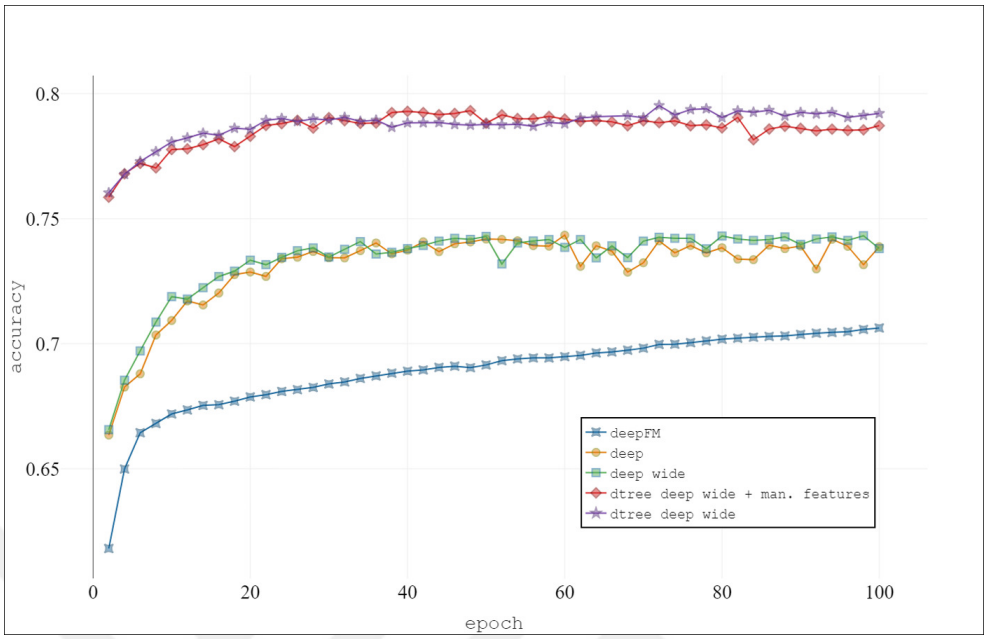

Figure 29: Accuracy comparison on rarely accepted (Less Than 2%) dataset

tested it on data from another set of offers. Hence, we measure how well our model generalize on completely different and possibly diverse offers. For each dataset, seven offers (33 % of offers) selected to create test set and the remaining offers are used to create training set. The offers are selected in such a way that accept and reject ratio of each dataset remains around 50 % so that the test set is not biased towards a decision. Models with different set of features are trained and tested on all datasets and results are listed in Table 5

Table 5: Accuracy values of deep models on mutually exclusive datasets

Model	Less Than 2%	Between 2% and 7%	Over 7%
Deep	0.63	0.70	0.69
Wide & Deep	0.67	0.70	0.71
Wide & Deep with Decision Tree based columns	**0.70**	**0.85**	**0.86**
Wide & Deep with Decision Tree based columns + manually crafted columns	0.7	0.85	0.84
DeepFM	0.62	0.63	0.59

The results indicate that all models perform around 10 % less in terms of accuracy

when compared to the previous experiment. This may be due to the reduced amount of similarity between the training and test samples. Since offer sets in training and test datasets are mutually exclusive, learning from previously seen patterns is less likely on this second set of experiments. The results also indicate that manually crafted features increase model performance around 1 or 2 % while automatically created features are more successful and improve performance of models between 7 % and 15 %. DeepFM again seems to have less accuracy when compared to *Wide & Deep* models on campaign participation datasets.

4.1.4 Uncertainty Results on Random Split Campaign Datasets

In this set of experiments, we evaluated the performance of different models in terms of confidence and uncertainty. We also performed threshold analysis to examine the change of accuracy for different models if model rejects to predict for samples above a varying uncertainty threshold. The results indicate that *Wide & Deep model with automatically crafted features* makes better assignment and use of uncertainty measures when compared to others. As a result, it is possible to obtain significant improvement in accuracy by using a reasonable uncertainty threshold.

Table 6 lists evidence and uncertainty values measured on all random split datasets with proposed *Wide & Deep model with automatically crafted features*. The results indicate that on all datasets neural network model assigns much higher evidence to successfully classified samples in comparison to incorrectly classified ones. As a result, average uncertainty on failed samples is much higher than average uncertainty on correct classifications. In other words, for all datasets, the model is also able to predict whether it will fail or not.

In addition to evaluating proposed *Wide & Deep* model on all 3 datasets, we also evaluated performance of different models such as *Deep*, *Wide & Deep* and *Wide &*

Table 6: Accuracy and uncertainty results on campaign participation datasets for Wide & Deep model with automatically crafted features

	$LessThan2\%$	$Between2\%and7\%$	$Over7\%$
accuracy	0.77	0.967	0.945
evidence fail	14.87	34.14	24.30
evidence	72.20	190.27	92.81
evidence success	90.26	195.46	96.79
uncertainty fail	0.491	0.43	0.387
uncertainty	0.30	0.06	0.093
uncertainty success	0.24	0.05	0.085

Deep model with automatically crafted features on the same dataset which is "Moderately Accepted Offers" dataset (Acceptance rate between 2% and 7%). Table 7 lists evidence and uncertainty values measured for different models on the same campaign participation dataset. The results indicate that the proposed model generates more evidence when compared to other models on the same dataset and has the least uncertainty value on correctly classified samples.

Table 7: Accuracy and uncertainty results on random split moderately accepted offers dataset (acceptance rate between 2% and 7%)

	$Deep$	$Wide\&Deep$	$DtreeBasedWide\&Deep$
accuracy	0.89	0.91	0.967
evidence fail	5.73	10.67	34.14
evidence	43.10	55.71	190.27
evidence success	47.95	60.11	195.46
uncertainty fail	0.65	0.59	0.43
uncertainty	0.25	0.20	0.06
uncertainty success	0.20	0.16	0.05

Figure 30 and Figure 31 display evidence and uncertainty metrics measured during execution of *Wide & Deep model with automatically crafted features*. Figure 30 plots change of evidence for correctly classified and incorrectly classified samples separately while Figure 31 shows change of uncertainty on correctly classified and incorrectly

classified samples besides the change of accuracy on all test samples. Figure 30 indicates that the neural network generates much more evidence for correctly classified samples, and evidence tends to increase throughout the run while it generates much less evidence for incorrectly classified ones. As a result, it has a very low uncertainty on correctly classified samples which is around 0.05. Uncertainty measured for incorrectly classified samples is very high, nearly around 0.43. In other words, the network is able to predict whether it will classify correctly for a sample or not based on evidence.

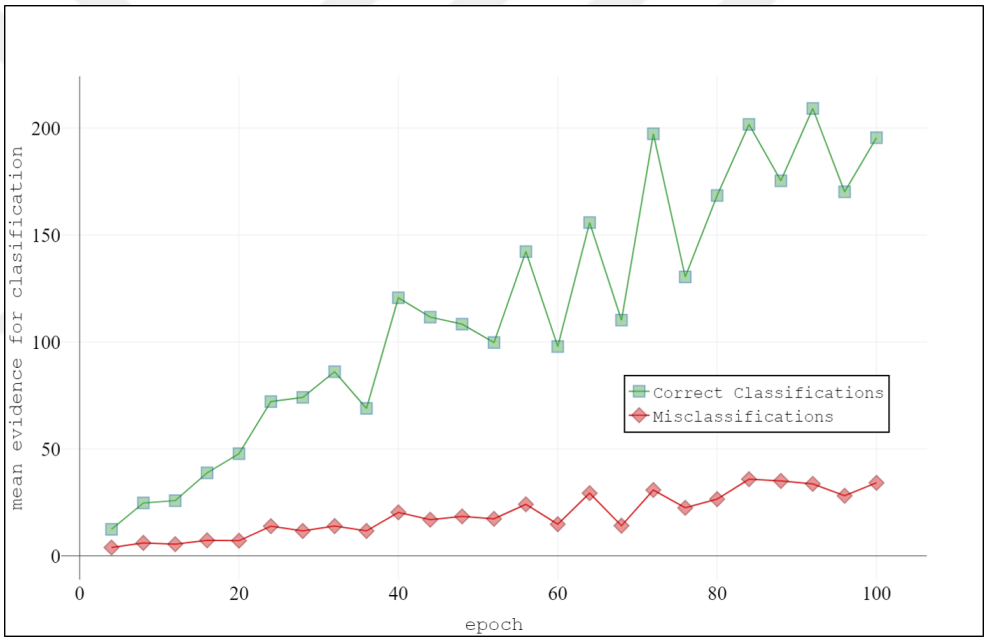

Figure 30: The change of evidence per epoch on campaign participation dataset for Wide & Deep model with automatically crafted features

Figure 32 and Figure 33 display evidence and uncertainty for *Wide & Deep* model while Figure 34, Figure 35 shows results for Deep Model. The results support the findings we observed on Figure 30 and Figure 31 for *Wide & Deep model with automatically crafted features*. Both models generate high evidence for correctly classified samples and fails when evidence is low. Evidence has a tendency to increase for correctly classified samples between successive epochs while evidence for incorrectly

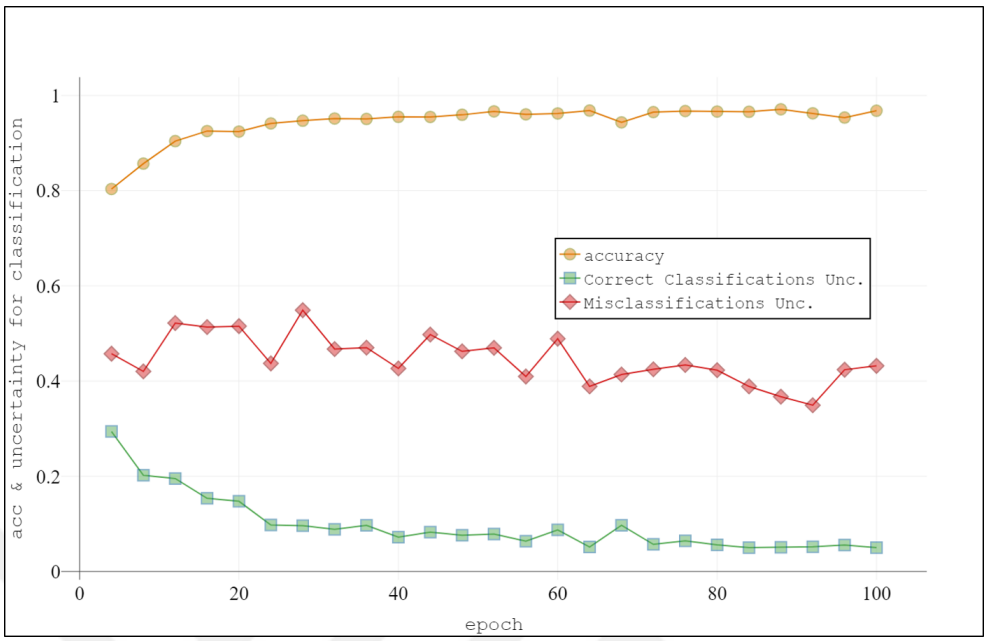

Figure 31: The change of accuracy and uncertainty on correctly classified samples and misclassifications on campaign participation dataset for Wide & Deep model with automatically crafted features

classified samples remain low. As a result all models have low average uncertainty on correctly classified samples and high average uncertainty on incorrectly classified samples.

Figure 36 plots how test accuracy changes if the model rejects making predictions above a varying threshold. According to Figure 36, model accuracy increases and reaches nearly to 100% as uncertainty threshold decreases. For each threshold value, we calculate accuracy only using the test samples whose predictive uncertainty is less than the threshold. In other words, we try to measure if the neural network also predicts when it fails by assigning high uncertainty to its wrong predictions and this information can be used to increase model accuracy on a subset by preventing failure cases.

Labels on Figure 36 are the percentage of samples the network model is willing to predict for each uncertainty threshold. The results indicate that by setting a

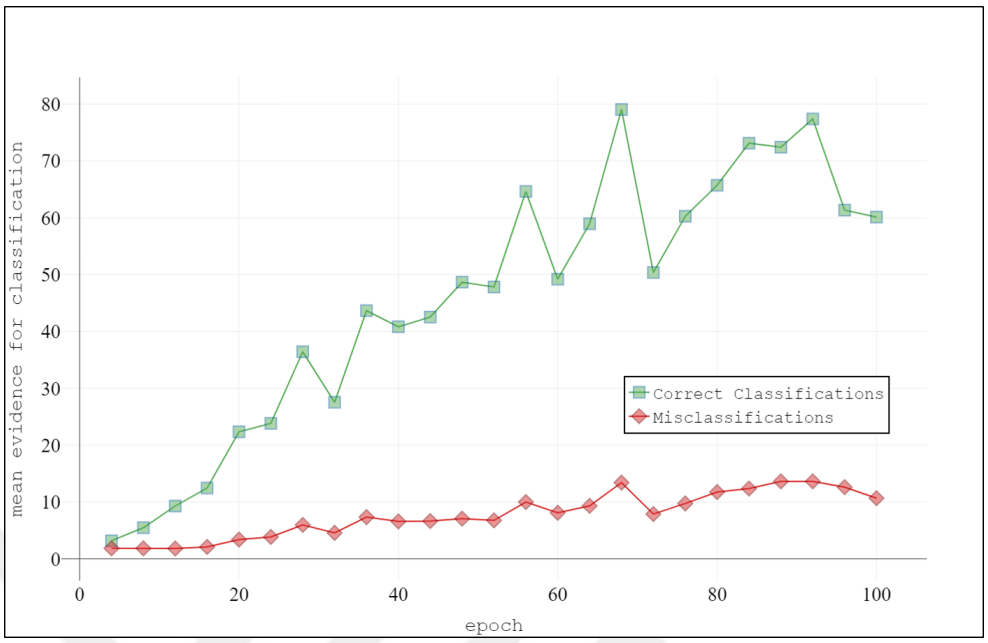

Figure 32: The change of evidence per epoch on campaign participation dataset for Wide & Deep model

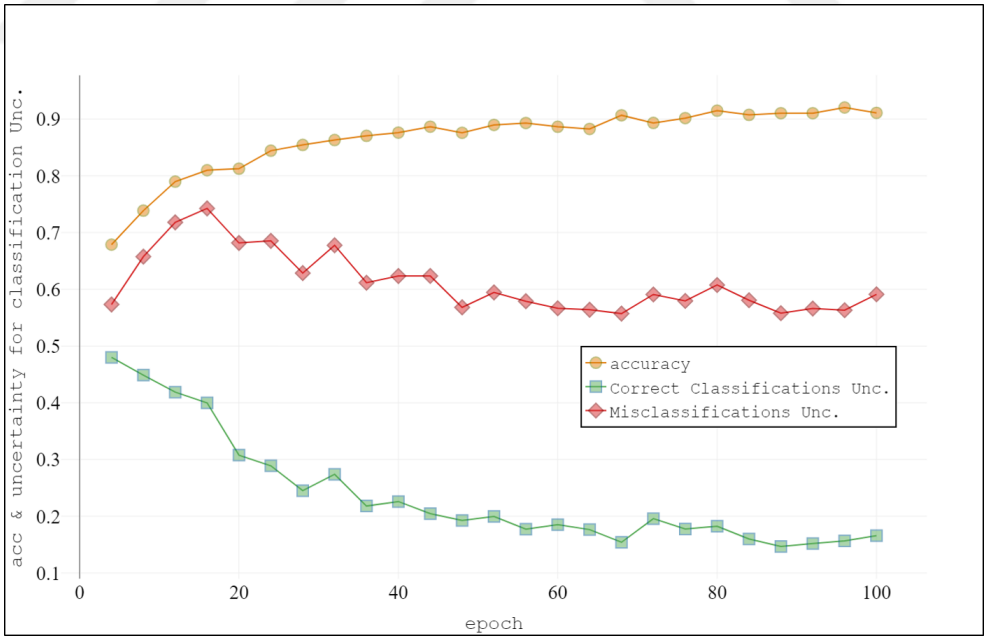

Figure 33: The change of accuracy and uncertainty on correctly classified samples and misclassifications on campaign participation dataset for Wide & Deep model

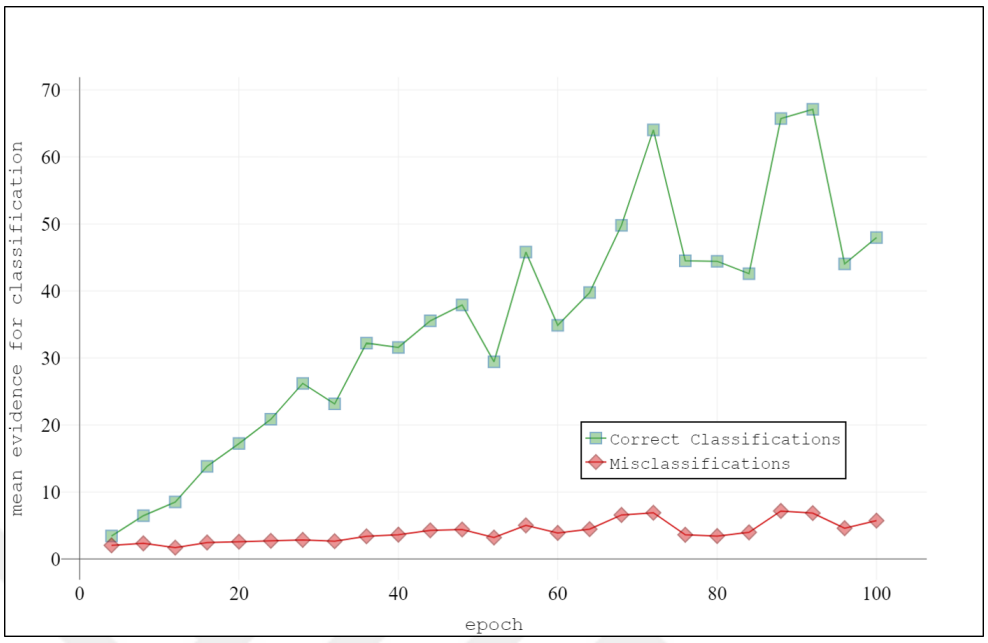

Figure 34: The change of evidence per epoch on campaign participation dataset for Deep model

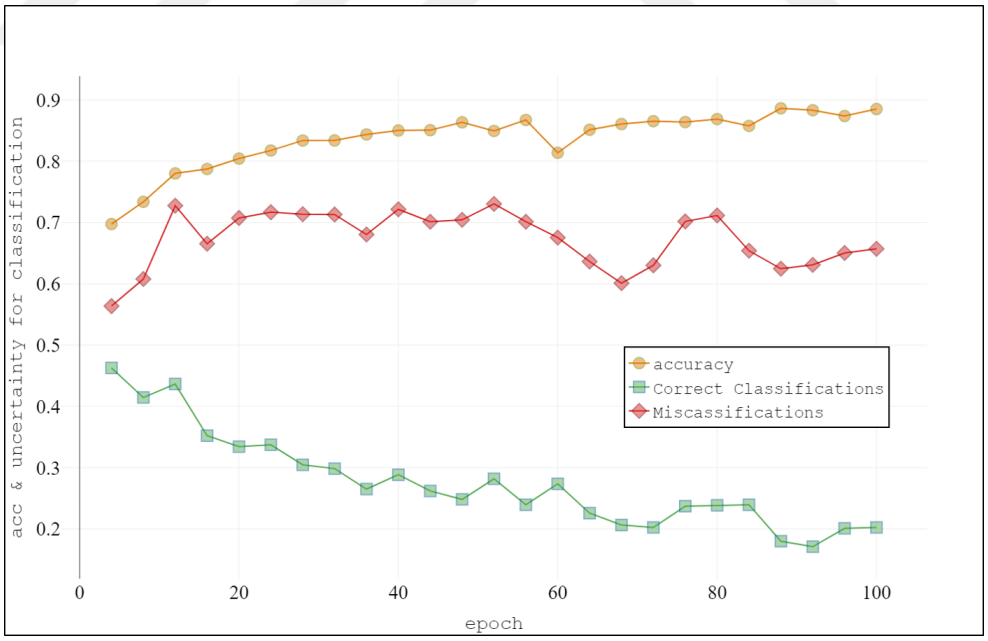

Figure 35: The change of accuracy and uncertainty on correctly classified samples and misclassifications on campaign participation dataset for Deep model

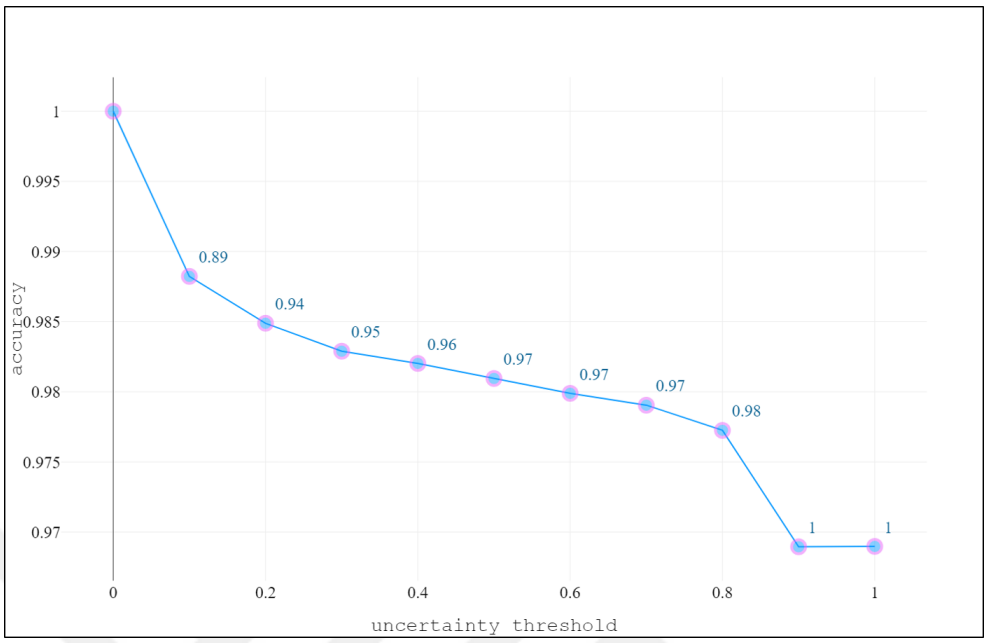

Figure 36: The change of accuracy with respect to uncertainty threshold on campaign participation dataset for Wide & Deep model with automatically crafted features

threshold of 0.1, the model will make predictions for 89% of all test samples, but nearly all predictions will be correct with around 99% accuracy.

Figure 37 and Figure 38 show change of accuracy with a varying uncertainty threshold for *Wide & Deep* and *Deep* models, respectively. The results indicates that *Wide & Deep model with automatically crafted features* is more accurate than other models for any threshold point and covers more test samples for each threshold.

4.1.5 Uncertainty Results on Mutually Exclusive Campaign Datasets

In this section, we repeated the experiments we have done on random split campaign participation dataset in Section 4.1.4 with mutually exclusive offers. In these experiments, we evaluated performance of different models in terms of evidence and uncertainty and performed some threshold analysis to examine the change of accuracy for different models if model rejects to predict for samples above a varying uncertainty threshold. The results support our findings on the random split dataset and indicate

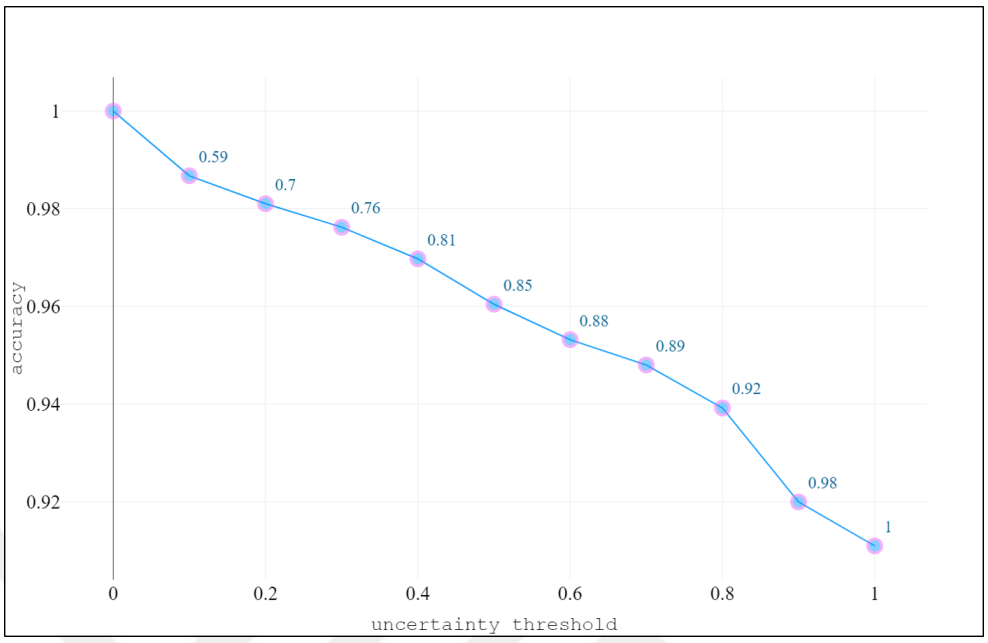

Figure 37: The change of accuracy with respect to uncertainty threshold on campaign participation dataset for Wide & Deep model

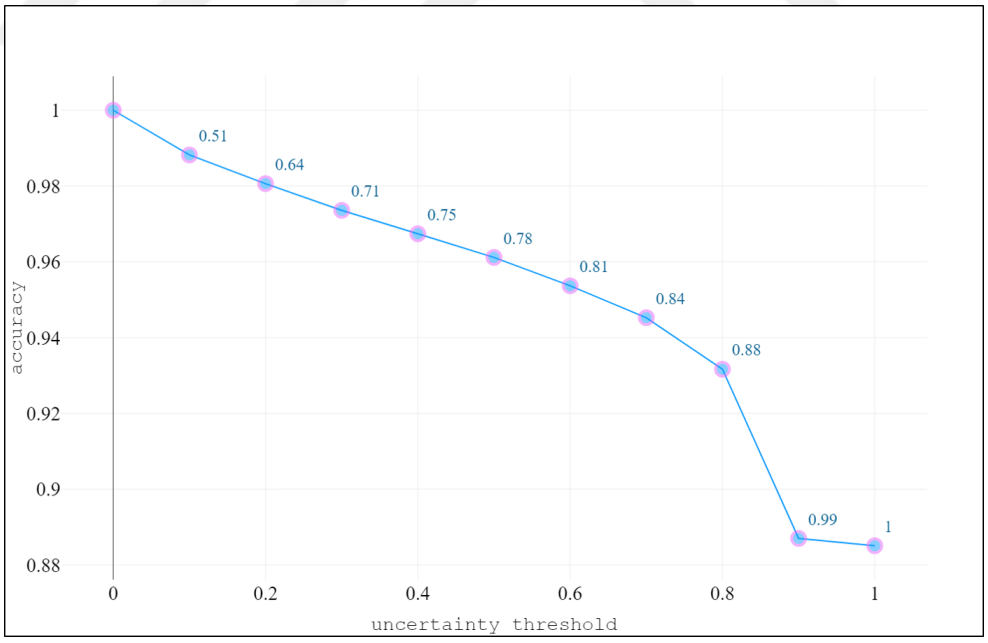

Figure 38: The change of accuracy with respect to uncertainty threshold on campaign participation dataset for Deep model

that *Wide & Deep model with automatically crafted features* makes better assignment of predictive uncertainty when compared to others on the mutually exclusive dataset. As a result, it is possible to obtain significant improvement in accuracy by using a reasonable uncertainty threshold.

We trained and tested the proposed model, which is *Wide & Deep model with automatically crafted features*, on all mutually-exclusive campaign participation datasets and list the evidence and uncertainty results in Table 8. The results indicate that the neural network model generates much higher evidence for correctly classified samples in comparison to incorrectly classified ones. As a consequence, all models assign high uncertainty to their failed classifications while they assign very low uncertainty to correct classifications. In other words, for all datasets, the model is also able to predict whether it will fail or not.

Table 8: Accuracy and uncertainty results on mutually exclusive campaign participation datasets for Wide & Deep model with automatically crafted features

	$LessThan2\%$	$Between2\%and7\%$	$Over7\%$
accuracy	0.743	0.839	0.848
evidence fail	9.67	31.30	24.57
evidence	35.77	102.22	64.88
evidence success	43.37	130.81	72.06
uncertainty fail	0.51	0.34	0.27
uncertainty	0.33	0.15	0.12
uncertainty success	0.27	0.12	0.1

As in previous experiment, we also evaluated performance of different models such as *Deep*, *Wide & Deep* and *Wide & Deep model with automatically crafted features* on the same dataset which is "Mutually exclusive Moderately Accepted Offers" dataset (Acceptance rate between 2% and 7%). Table 9 lists evidence and uncertainty values measured for different models on the same campaign participation dataset. The results indicate that the proposed model generates the highest evidence on the same

data and has the least uncertainty on correctly classified samples.

Table 9: Accuracy and uncertainty results on random split moderately accepted offers dataset (acceptance rate between 2% and 7%)

	Deep	*Wide&Deep*	*DtreeBasedWide&Deep*
accuracy	0.71	0.73	0.839
evidence fail	20.86	24.3	31.30
evidence	35.85	59.9	102.22
evidence success	41.98	73.15	130.81
uncertainty fail	0.45	0.36	0.34
uncertainty	0.34	0.24	0.15
uncertainty success	0.29	0.19	0.12

Figure 39 and Figure 40 display evidence and uncertainty metrics measured during the execution of *Wide & Deep model with automatically crafted features*. Figure 39 plots change of evidence for correctly classified and incorrectly classified samples separately while Figure 40 shows change of uncertainty on correctly classified and incorrectly classified samples besides the change of accuracy on all test samples. Figure 39 indicates that the neural network generates much more evidence for correctly classified samples, and evidence tends to increase throughout the run while it generates much less evidence for incorrectly classified ones. As a result, it has a relatively low uncertainty on correctly classified samples which is around 0.12. Uncertainty measured for incorrectly classified samples is much higher when compared to correctly classified samples and around 0.34. In other words, the network is able to predict whether it will classify correctly for a sample or not based on evidence.

Figure 41 and Figure 42 displays evidence and uncertainty metrics measured during *Wide & Deep* model execution while Figure 43, Figure 44 shows same metrics for *Deep* model. The results support the findings we observed on Figure 39 and Figure 40 for *Wide & Deep model with automatically crafted features*. Both models generate high evidence for correctly classified samples and fails when evidence is low.

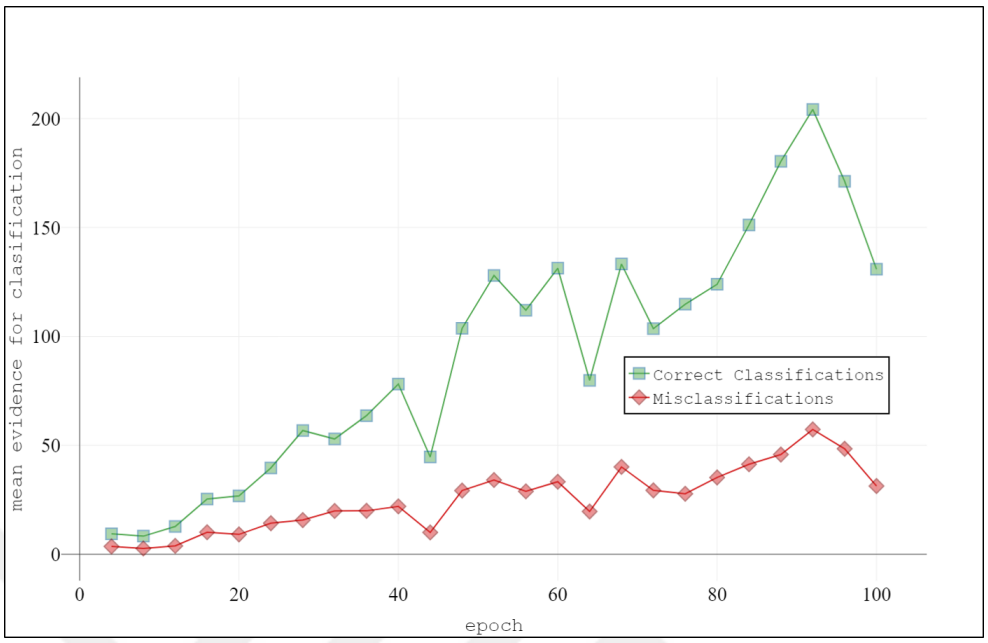

Figure 39: The change of evidence per epoch on mutually exclusive campaign participation dataset for Wide & Deep model with automatically crafted features

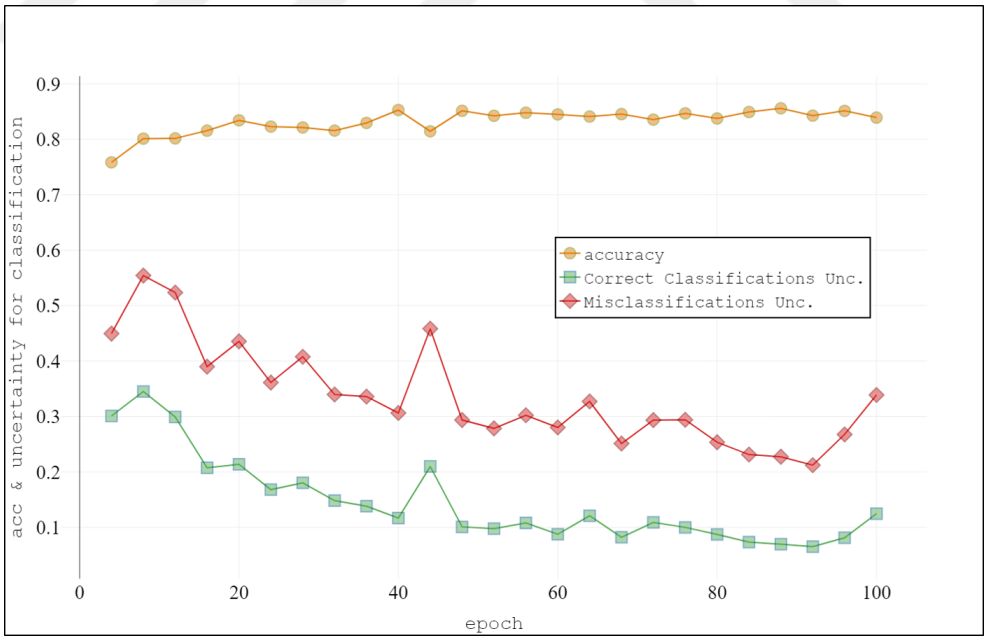

Figure 40: The change of accuracy and uncertainty on correctly classified samples and misclassifications on mutually exclusive campaign participation dataset for Wide & Deep model with automatically crafted features

Evidence has a tendency to increase for correctly classified samples between successive epochs while evidence for incorrectly classified samples remain low. As a result all models have low average uncertainty on correctly classified samples and high average uncertainty on incorrectly classified samples.

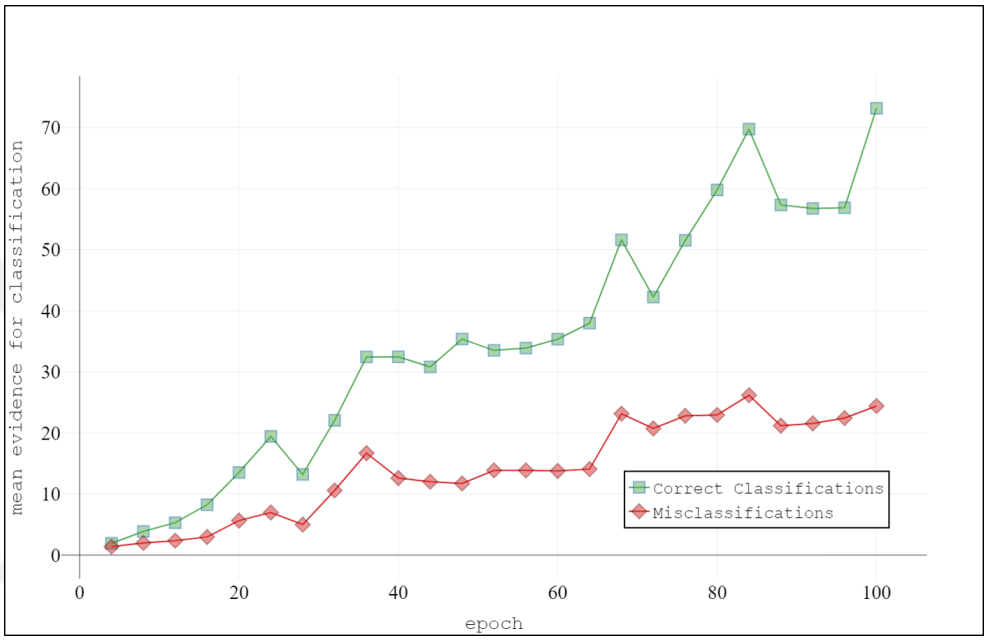

Figure 41: The change of evidence per epoch on campaign participation dataset for Wide & Deep model

Figure 45 plots how test accuracy changes if the model rejects making predictions above a varying threshold. According to Figure 45 model accuracy increases and reaches over 0.92 as uncertainty threshold decreases. For each threshold value, the neural network rejects to predict for a set of samples with the assumption that samples above a predefined uncertainty threshold will be classified incorrectly. In other words, the neural network also predicts when it fails by assigning high uncertainty to its wrong predictions, and this information can be used to increase model accuracy on a subset by preventing failure cases.

Labels on Figure 45 are the percentage of samples the network model is willing to predict for each uncertainty threshold. The results indicate that by setting a

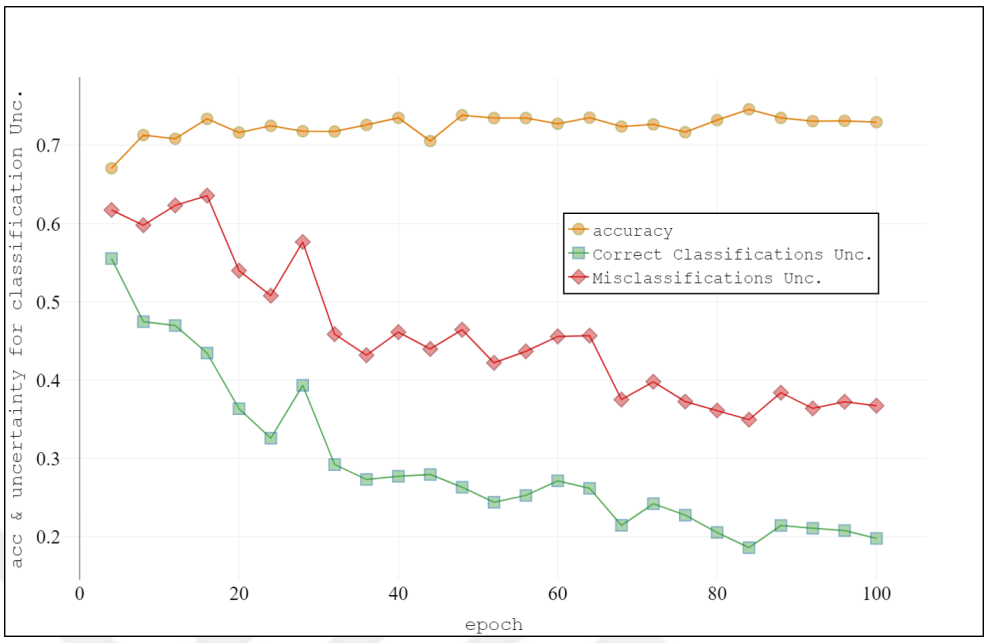

Figure 42: The change of accuracy and uncertainty on correctly classified samples and misclassifications on campaign participation dataset for Wide & Deep model

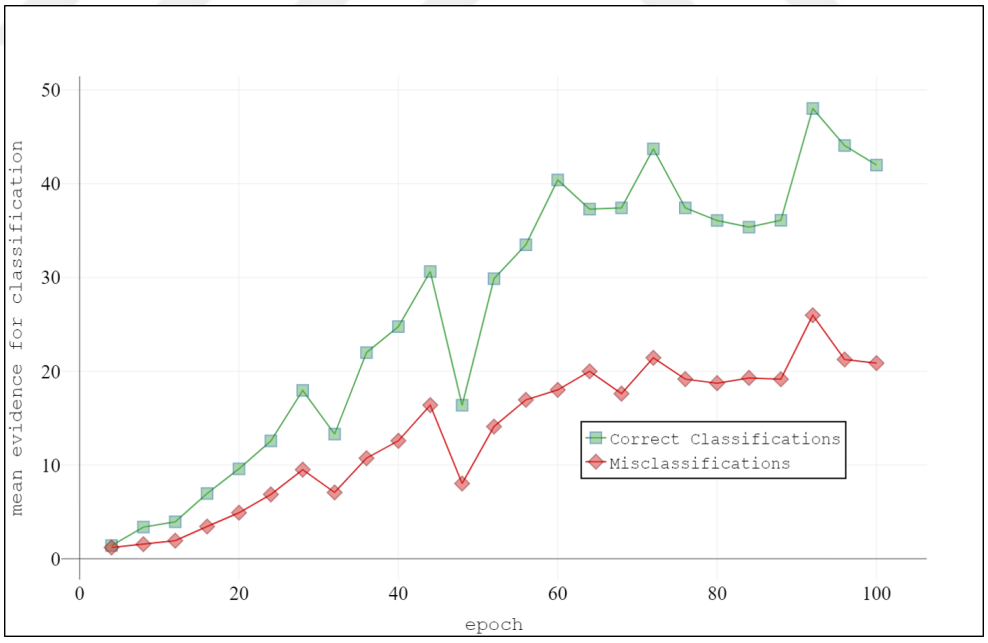

Figure 43: The change of evidence per epoch on mutually exclusive campaign participation dataset for Deep model

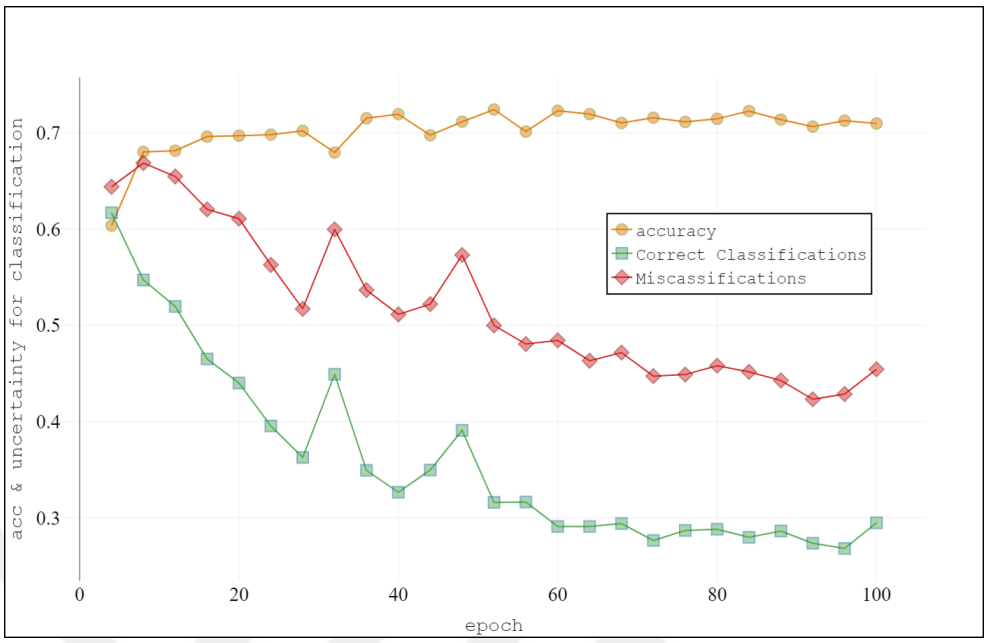

Figure 44: The change of accuracy and uncertainty on correctly classified samples and misclassifications on mutually exclusive campaign participation dataset for Deep model

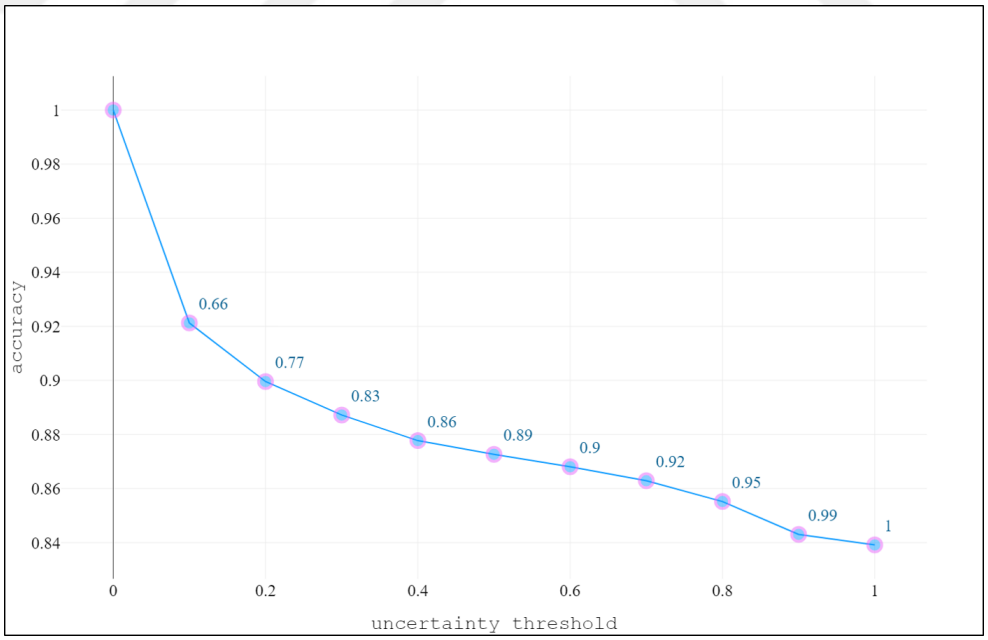

Figure 45: The change of accuracy with respect to uncertainty threshold on mutually exclusive campaign participation dataset for Wide & Deep model with automatically crafted features

threshold of 0.2, the model will make predictions for 77% of all samples with nearly 90% accuracy.

Figure 46 and Figure 47 show change of accuracy with a varying uncertainty threshold for *Wide & Deep* and *Deep* models respectively. The results indicates that *Wide & Deep model with automatically crafted features* is more accurate than other models for any threshold point and covers more test samples for each threshold.

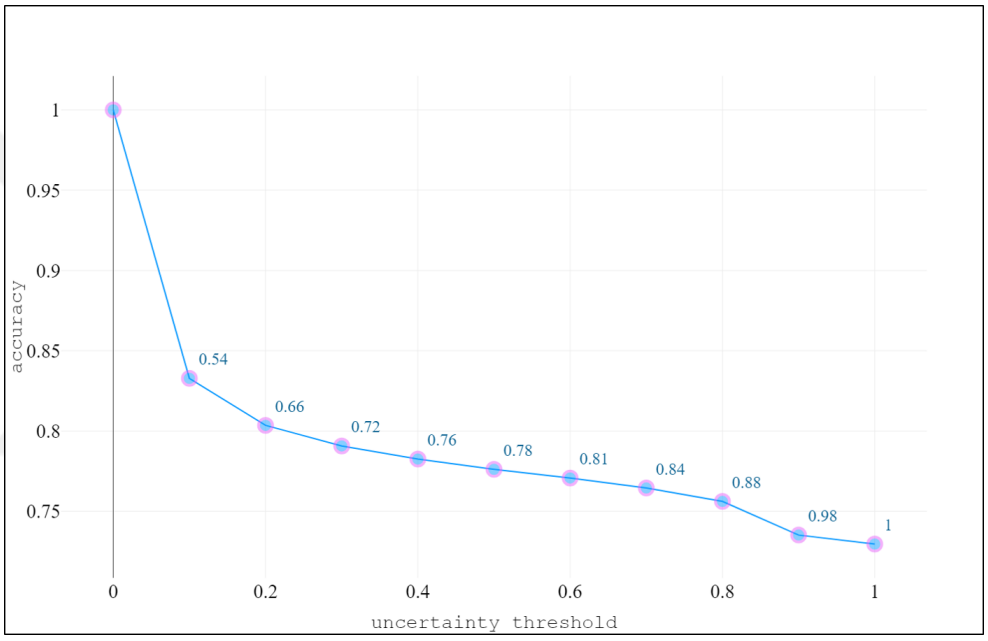

Figure 46: The change of accuracy with respect to uncertainty threshold on mutually exclusive campaign participation dataset for Wide & Deep model

4.1.6 Training and Evaluation Time Comparison

Figure 48 shows average evaluation time per item measured during test executions for each model. The results indicate that all models are time efficient enough to be used in real time. DeepFM has the least evaluation time and it is nearly 10 times faster than *Wide & Deep* models. The maximum measured average value belongs to *Wide & Deep model with automatically crafted features + man. crafted features* and it is around 1.5 milliseconds. The overall picture shows that there is not a difference

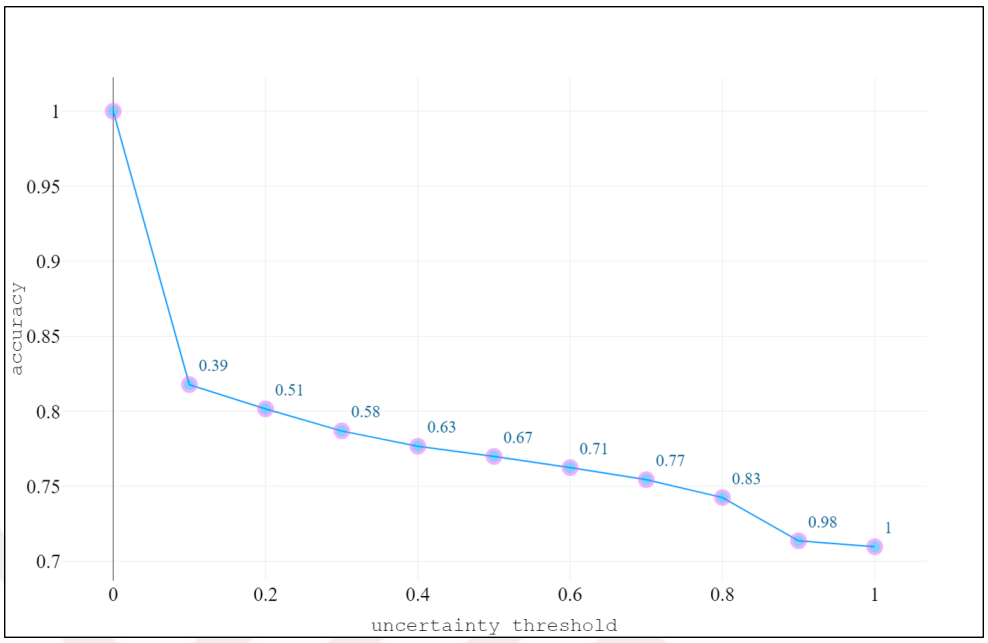

Figure 47: The change of accuracy with respect to uncertainty threshold on mutually exclusive campaign participation dataset for Deep model

or trend among different datasets, model evaluation time only increases as number of features used by the model increases.

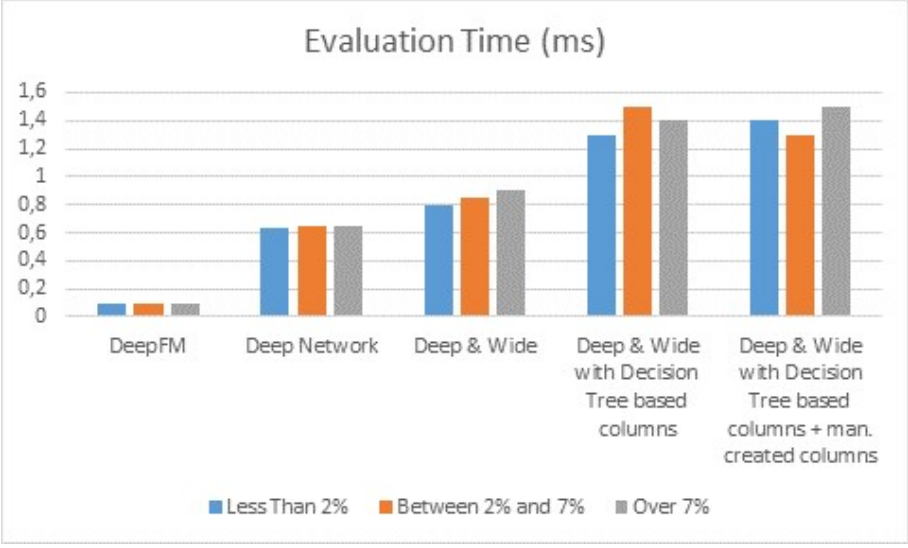

Figure 48: Evaluation time per item

Figure 49 shows training time measured for each model during experiments. Similar to evaluation time comparison, training time also increases as the number of

features used by the model increases.

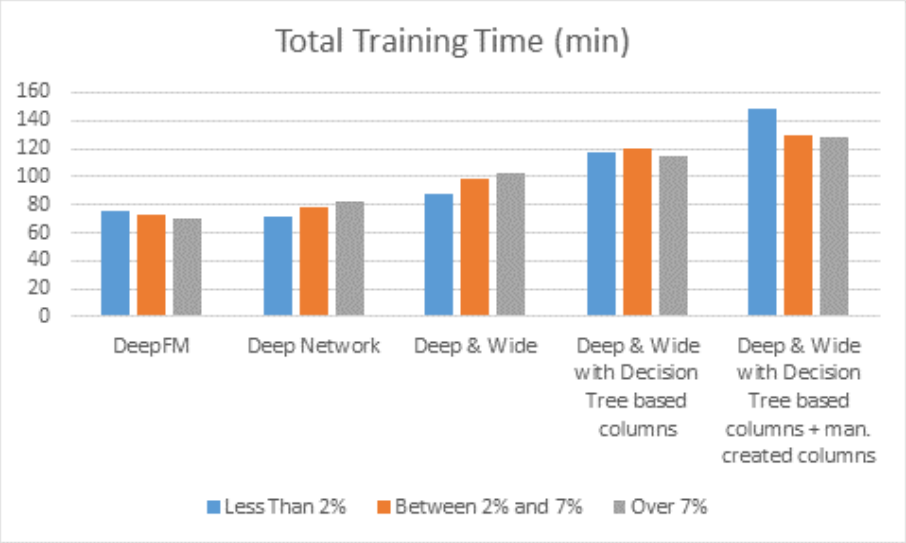

Figure 49: Total training time for each model

CHAPTER V

DISCUSSION AND CONCLUSIONS

In this dissertation, we applied deep learning to the campaign participation prediction problem of GSM customers for the first time and used *Wide & Deep* learning models to predict the acceptability of a given offer by the customers. Those models require domain expert knowledge and well-designed combinatorial features, namely cross features in order to have effective memorization capabilities. We propose a decision tree based method for generating those features automatically and consequently alleviate the necessity of feature engineering.

To evaluate the performance of the proposed method, we used three different campaign participation dataset with different user acceptance rates and performed both in-distribution (i.e., random split) and out-of-distribution (i.e., mutually exclusive) queries to predict campaign participation behavior of GSM customers. We also evaluated the proposed method on different datasets such as adult income and Criteo sample datasets. The details of this evaluation study are provided in Appendix A and Appendix B. The results on all datasets showed that the proposed approach significantly outperforms existing approaches in all of these settings.

Marketing is one of the most challenging areas when it comes to adapt to the changing needs of customers. Companies continuously update their product portfolio and make different campaigns, including new products and new offerings to address the changing needs of customers. For this reason, the neural network models which we build for campaign participation prediction problem are not specific to a static set of offers and can make predictions for any offer which means previously unseen offers are also in the scope of these models. The models take customer attributes and offer

attributes as input and predict whether the customer will accept the offer or not. We performed experiments on out-of-distribution campaign participation datasets and verified that the neural network models are capable of making predictions for previously unseen offers. Even if the models with automatically crafted features acquire reasonable accuracy on previously unseen offers, there is still a need to detect whether the models decide with evidence or in lack of evidence. As long as previously unseen offers are considered, there is always a possibility that the learning models predict one of the possible outputs randomly since they cannot say "I do not know". That is one of the reasons which makes uncertainty-aware networks necessary for solving campaign participation prediction problem.

Another important aspect of campaign participation prediction is the cost of failures (i.e., false positives). In such an environment, access to customers is limited by the nature of the environment or regulations and rules. More importantly, there is the risk of customer dissatisfaction if too many irrelevant offer messages are sent to customers. Therefore, companies aim to target the right audience with the right offer by keeping customer access at a minimum. A neural network model, which can measure its risk in making prediction, can prevent failure cases by not making decisions in lack of evidence. Therefore, this thesis advocates using uncertainty-aware networks (i.e., evidential deep neural networks) described by Şensoy *et al.* [9] for campaign participation prediction.

Using uncertainty-aware network models, we evaluated different neural network models in terms of evidence and uncertainty and performed some threshold analysis by querying if the models reject to predict for the given samples above a varying uncertainty threshold. The results indicate that *Wide & Deep models with automatically crafted features* are more confident in their correct classifications since they are able to generate more evidence out of the same data. Consequently, it is possible to obtain significant improvement in accuracy by using a reasonable uncertainty threshold.

Those models can be used to make predictions for unseen offers as far as uncertainty is concerned.

As future work, we can use identity-preserved transformations [27] to enable our models to generate offers and advertisements that will produce a positive response when sent to the customers or GSM users. The idea is basically keeping model parameters such as weights, and bias values and model output fixed and using a Gradient Descent Optimizer to shift input vector in the latent space so that it generates target output value, which is ACCEPT in our problem. Identity preserving transformation is one way of generating input. Other candidate methods for input generation such as VAE or GAN networks can also be used, but the important requirement here is having accurate network models with an awareness of uncertainty.

APPENDIX A

ADULT INCOME DATASET

Adult dataset, also known as Census income dataset, is one of the widely used datasets in machine learning to evaluate the performance of classification models. It was originally extracted by Berry Baker from 1994 Census database and publicly available in the UCI machine learning repository.

The dataset consists of 48,842 records, each of which has a binary label indicating whether the yearly income of an individual is over 50K or less than 50K. 76% of the records have $\leq$50K label while 24% has >50K label. Each record also includes 8 categorical and 6 continuous attributes, 14 attributes in total. Attribute details and statistics are listed in Table 10

The dataset is shared in 2 files containing the test set and training set records. The training set consists of 32,561 records, while the test set consists of 16,281 records.

A.1 Methodology

This section describes the methodology to train and test the resulting model on the income dataset. Figure 50 shows the steps taken in this work. Income dataset is a publicly available dataset on the UCI machine learning repository. As a first step, we get the test and training data fro the repository in CSV format.

Second, we use the proposed method to create cross features to be used by the proposed model. Besides *Wide & Deep model with automatically crafted features*, we also trained and tested *Wide & Deep model with manually crafted features* for comparison. For this experiment, we selected attributes like age, education, occupation, workclass, which are known to affect an individual's income and created cross features

Table 10: Income dataset attributes

AttributeName	*Type*	*Values*
work-class	categorical	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked
education	categorical	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
marital-status	categorical	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
occupation	categorical	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
relationship	categorical	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	categorical	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black
sex	categorical	Female, Male
native-country	categorical	54 countries
age	number	min:17, max:90, mean:38.6
fnlwgt	number	min:12285, max:1490400
education-num	number	min:1, max:16
capital-gain	number	min:0, max:99999
capital-loss	number	min:0, max:4356
hours-per-week	number	min:1, max:99

by combining 2 or 3 of those features randomly. We ran a few experiments and select the best set of cross features that gives the highest accuracy to build a *Wide & Deep*

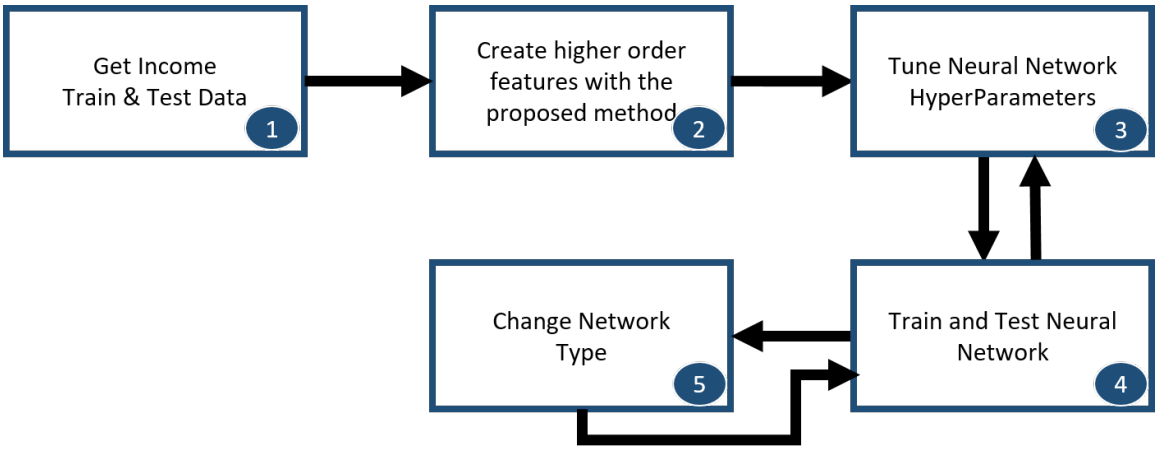

Figure 50: Overview of the methodology for income dataset

model with manually crafted features for the experimentation purpose.

As third step, hyper-parameters such as number of layers, number of neurons in each layer, learning rate, regularization rate, are selected.

Fourth and Fifth steps are model training and testing steps. Different architectures with different set of features such as *Deep*, *Wide & Deep with manually crafted features*, *Wide & Deep with automatically crafted features* are trained and tested to see effectiveness of cross features with respect to models which does not use automatically generated features.

A.1.1 Experimental Setup for Income Datasets

For this set of experiments, we used the original income test and training data, which are already shared in separate data files.

We used accuracy,evidence and uncertainty as an evaluation metrics and evaluated 3 models which are *Deep*, *Wide & Deep* and *Wide & Deep model with automatically crafted features*.

To find a common parameter setting that gives the best performance on all models, we performed a careful parameter evaluation step. The resulting parameter settings are listed below:

1. Network Structure: 100-50-25-10

2. Learning rate: 0.005

3. Droppout Rate: 0.05

4. Activation function: relu

5. Optimizer: adam

For income dataset, trees with four or more levels achieve over 80% accuracy. So a tree structure with four levels and 82% accuracy is used to create cross features automatically.

A.2 Results on Adult Income Dataset

In this set of experiments, we trained and tested 3 network models which are *Deep*, *Wide & Deep* and proposed *Wide & Deep model with automatically crafted features* on income dataset. With this experiment our goal is to see if the proposed cross feature generation methodology improves model accuracy on different datasets.

We use classification accuracy as a comparison metric for this part. The accuracy values obtained with each model are listed in Table 11. Table 11 also includes mean evidence and uncertainty values for each network model.

Table 11: Accuracy and uncertainty results on income dataset

	Deep	*Wide&Deep*	*DtreeBasedWide&Deep*
accuracy	0.845	0.848	0.862
evidence fail	4.96	4.54	3.43
evidence	32.51	33.94	36.43
success evidence	37.57	39.16	42.12
uncertainty fail	0.60	0.62	0.57
uncertainty	0.31	0.33	0.28
uncertainty success	0.26	0.26	0.22

The highest accuracy value observed in this experiment is around 86% and belongs to *Wide & Deep model with automatically crafted features* while the lowest accuracy observed is around 84% and belongs to *Deep* model. *Wide & Deep* model, on the other hand, performs slightly better than the Deep model, which shows adding expert knowledge and wide component also improves the performance.

Even though the proposed model improves the accuracy of *Deep* model, the amount of improvement is less than the improvement we observed on other datasets, which are campaign participation and Criteo datasets. The overall results show that the effectiveness of the proposed method increases as data complexity increases.

Figure 51 shows the change of accuracy for all three models during 100 epochs. In Figure 51, it is seen that all three models have similar accuracy change trends. All models show a rapid increase in accuracy for the first 4 to 12 epochs and slightly improves afterward. Proposed decision tree based cross feature generation methodology seems to have higher accuracy throughout the run except for a few initial epochs.

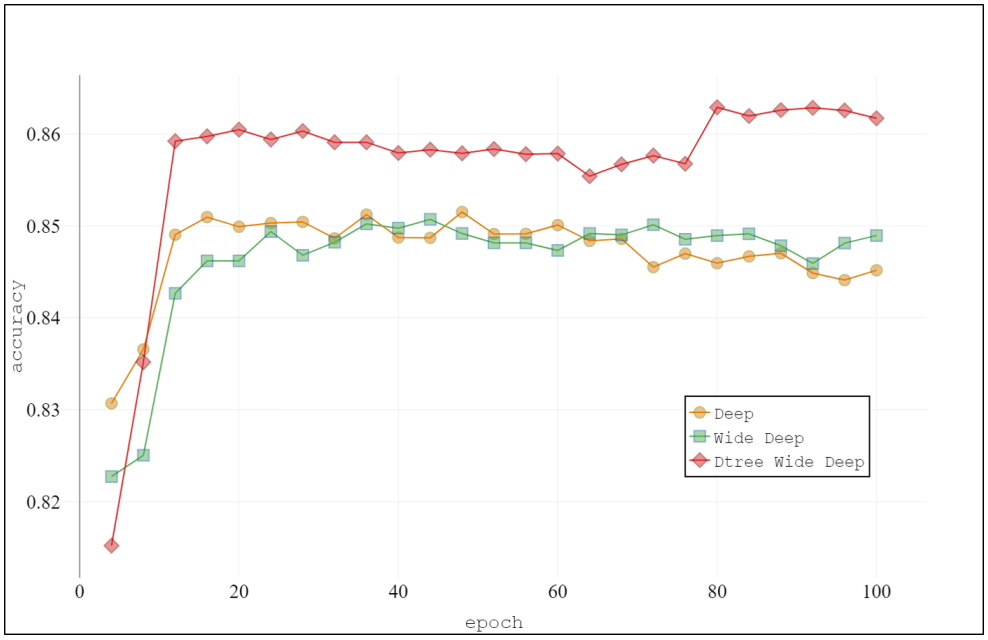

Figure 51: The change of accuracy per epoch for deep, Wide & Deep and Wide & Deep with automatically crafted features

Besides deep models, we trained and tested logistic regression and decision tree models to see the performance of other models on income dataset. We trained decision tree models from 2 levels up to 17 levels. The highest accuracy obtained with these decision tree models is around 85%. This accuracy is achieved by a tree model with 8 levels. The accuracy of the logistic regression model is around 84%. For this dataset, the performance of simpler models such as Logistic Regression and Decision Trees are close to the more complex models, such as neural network models, possibly due to the low dimensionality of the dataset.

Figure 52 and Figure 53 display evidence and uncertainty metrics measured during the execution of *Wide & Deep model with automatically crafted features*. Figure 52 plots change of evidence for correctly classified and incorrectly classified samples separately and shows that the neural network generates much more evidence for correctly classified samples and evidence tends to increase throughout the run while it generates much less evidence for incorrectly classified ones. Figure 53 plots change of uncertainty on correctly classified and incorrectly classified samples besides the change of accuracy on all test samples. Figure 53 shows that the model has a very low uncertainty on correctly classified samples which is around 0.2, and uncertainty measured for incorrectly classified samples is very high at nearly around 0.6.

Figure 54 plots how test accuracy changes if the model rejects making predictions above a varying threshold. Similar to the previous cases, model accuracy increases and reaches nearly to 100% as the uncertainty threshold decreases. Labels on Figure 54 are the percentage of samples the network model is willing to predict for each uncertainty threshold. The results indicate that by setting a threshold of 0.4, the model can classify 73% of the samples with an accuracy of 94%. By setting a threshold of 0.1, the model will only make predictions for 43% of all samples, but nearly all predictions will be correct with 99% accuracy.

Figure 55 and Figure 56 show change of accuracy with a varying uncertainty

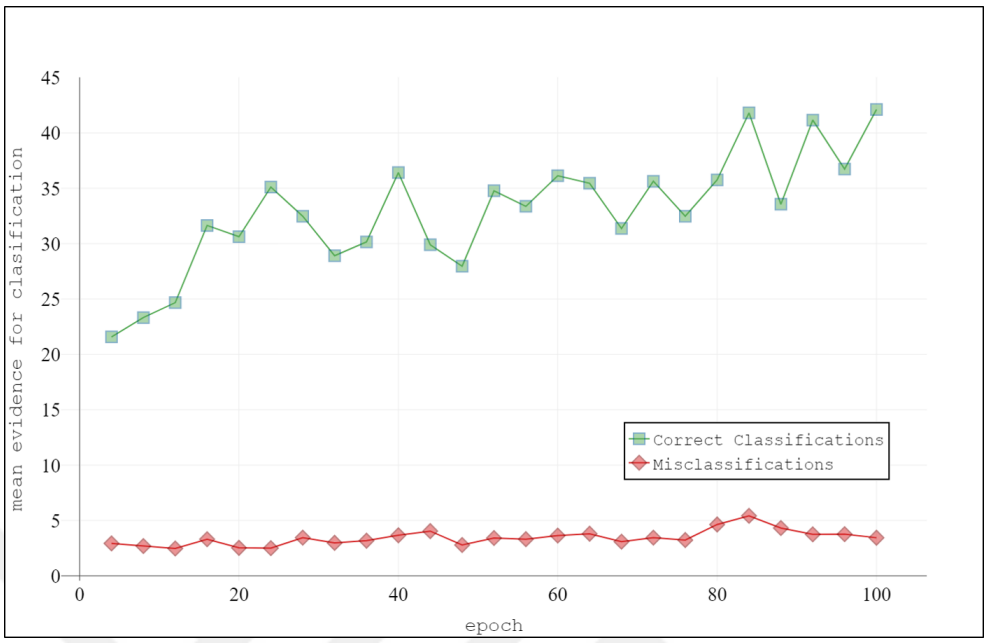

Figure 52: The change of evidence per epoch on Income dataset for Wide & Deep model with automatically crafted features

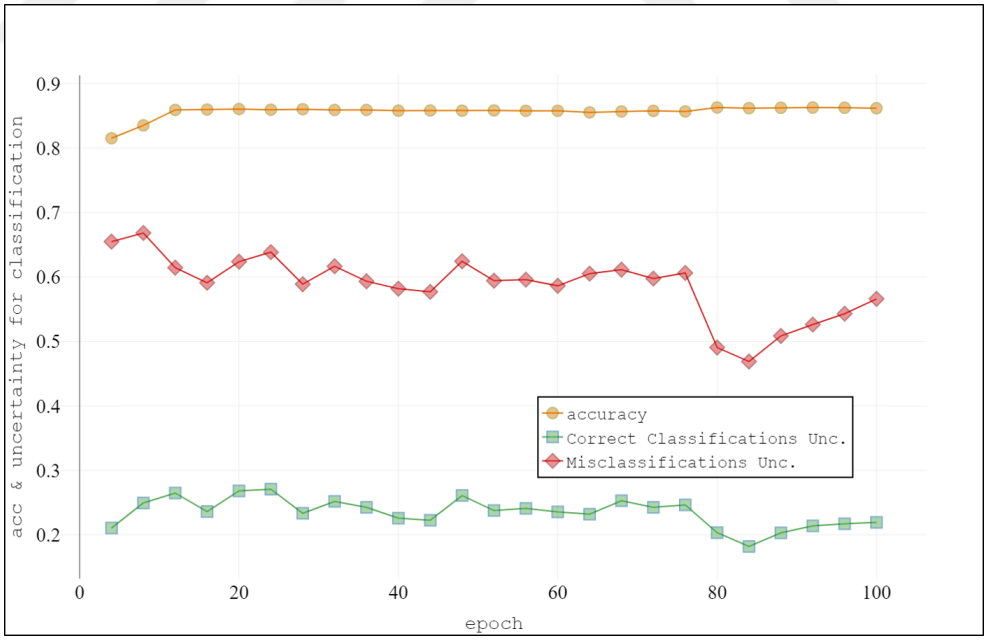

Figure 53: The change of accuracy and uncertainty on correctly classified samples and misclassifications on Income dataset for Wide & Deep model with automatically crafted features

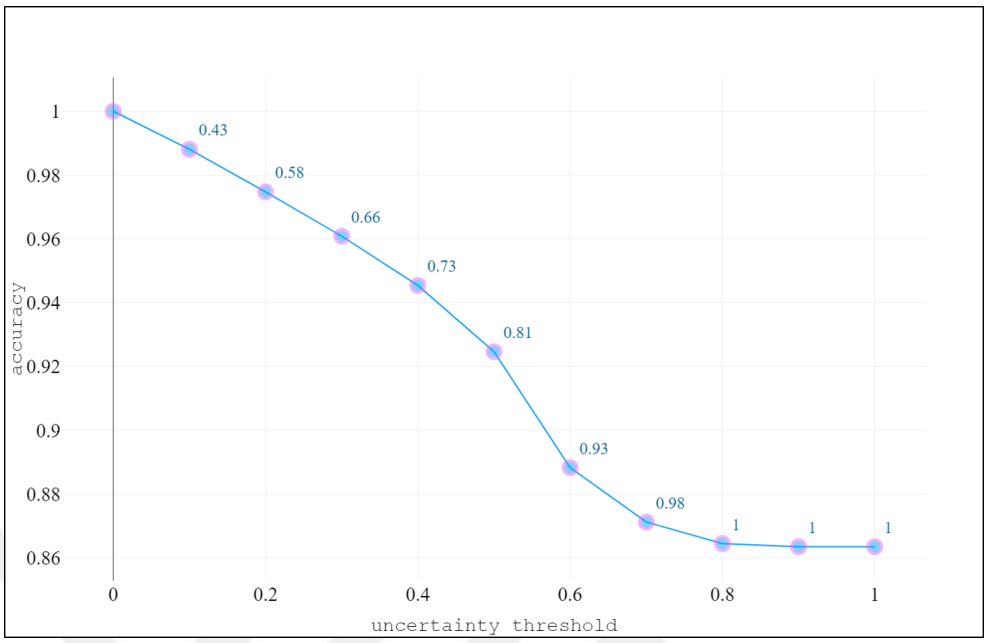

Figure 54: The change of accuracy with respect to uncertainty threshold on income dataset for Wide & Deep model with automatically crafted features

threshold for *Wide & Deep* and *Deep* models respectively. The results indicates that *Wide & Deep model with automatically crafted features* is more accurate than other models for any threshold point and covers more test samples for each threshold.

A.3 Training and Evaluation Time Comparison

Figure 57 shows both training time and average evaluation time per item measured during test and training executions. According to Figure 57, Deep model has the least evaluation time and it is nearly 2 times faster than *Wide & Deep* models. The maximum measured average evaluation time belongs to *Wide & Deep model with automatically crafted features* and it is around 0.47 milliseconds. The results indicate that all models are time efficient enough to be used in real-time. The overall picture shows that model evaluation and training time only increases as the number of features used by the model increases.

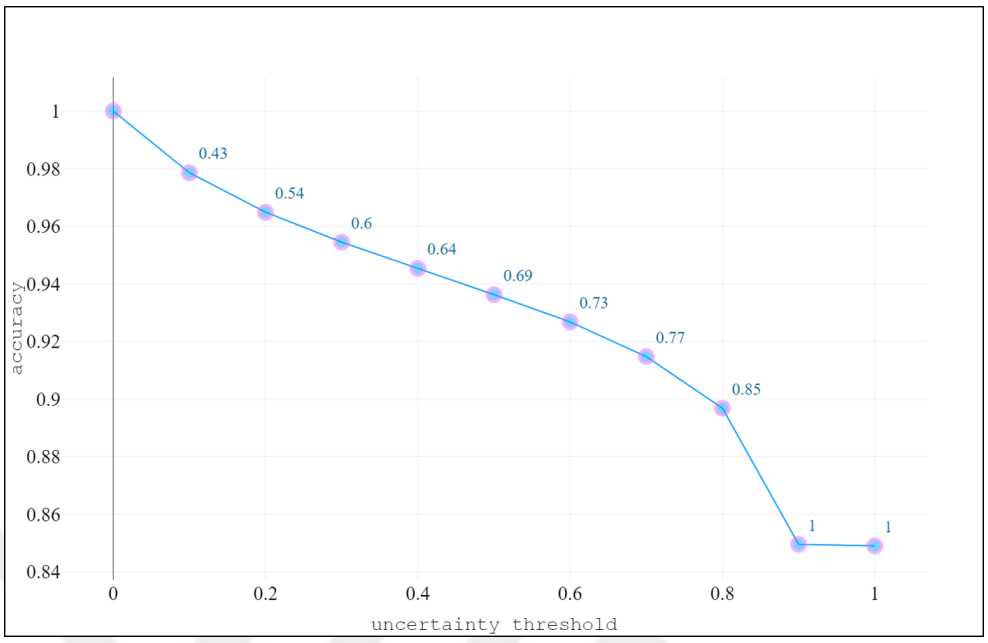

Figure 55: The Change of accuracy with respect to uncertainty threshold on Income dataset for Wide & Deep model

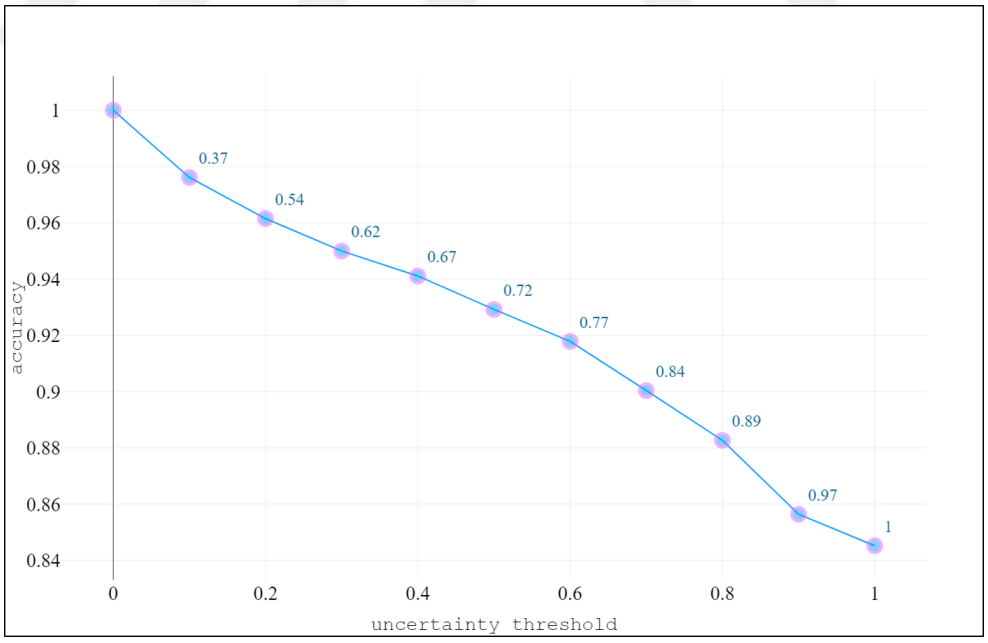

Figure 56: The change of accuracy with respect to uncertainty threshold on Income dataset for Deep model

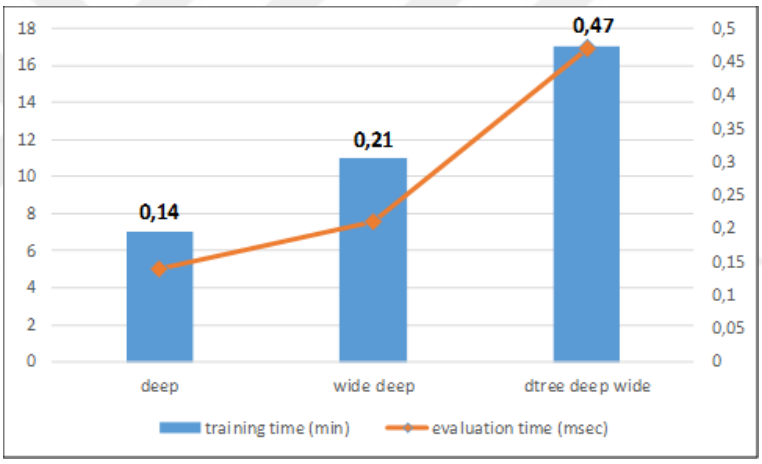

Figure 57: Evaluation time per item and total training time per model on income dataset

APPENDIX B

CRITEO DATASET

Internet advertising is a multi-billion dollar business and one of the challenging problems in the area of machine learning. CRITEO dataset is a public click-through rate (CTR) prediction dataset, which includes click records of 45 million users during 24 days period. We selected this dataset to evaluate the performance of the proposed method on a relatively sparser dataset, which has different dataset characteristics when compared to income and campaign participation datasets.

Each record in Criteo dataset includes a LABEL column which indicates whether the user clicked the advertisement or not and 39 attribute columns. The dataset consists of 13 numerical, 26 categorical columns and a label column. All attribute columns are anonymized. Numerical columns are named from $num1$ through $num13$ based on order while categorical columns are all hashed and named like $char1$ through $char26$. Since all columns are hashed and anonymized, it is even harder or impossible to create cross features based on expert knowledge for this dataset.

Since original dataset includes 4.3 billion click records and it is not possible to process this data with current processing power we have, we performed a sampling on the dataset and selected 60000 accept and 60000 reject click records, which do not have any unknown values and performed our experiments on this subset dataset.

B.1 Methodology

This section describes the methodology to train and test the resulting model on Criteo sample dataset. Figure 58 shows the steps taken in this work. Criteo dataset is a publicly available dataset in UCI machine learning repository. First we get the

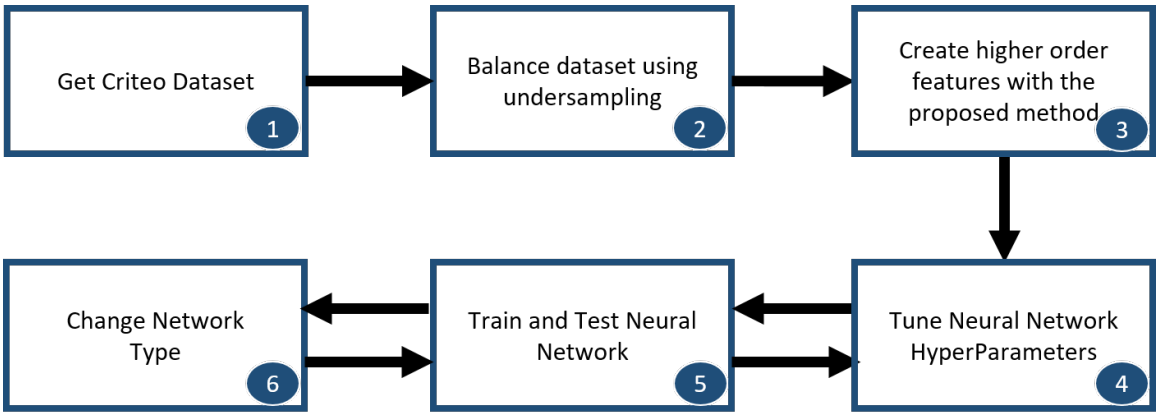

Figure 58: Overview of the methodology for Criteo dataset

data from the repository in csv format. The original data includes click records of 45 million users during 24 consecutive days which results billions of records. Since processing the original dataset would require high amounts of processing power, as a second step we sampled 120000 records from the original data. In order to keep data unbiased we keep accept and reject record ratios equal, 60000 from each type of record is selected.

Third, we use the proposed method to create cross features to be used by the proposed model. As in our previous experiments we also created cross features manually to build a *Wide & Deep model with manually crafted features* for comparison purposes. For this dataset, any kind of expert knowledge is absent since data is totally anonymized. In the dataset, categorical fields are named as char1 through char23 while numerical fields are named from num1 through num13. All categorical fields are also hashed so that it is impossible to create cross features based on human experts. To create cross features for *Wide & Deep* model, we selected features randomly and created cross features by combining 2 or 3 of those features randomly. We ran a few experiments and select the best set of cross features that gives the highest accuracy to build a *Wide & Deep* model with manually crafted features for the experimentation purpose.

As the fourth step, hyper-parameters such as number of layers, number of neurons

in each layer, learning rate, and regularization coefficient are selected. After finding the most efficient parameter set, the models are trained and tested with the Criteo dataset.

B.1.1 Experimental Setup for Criteo Datasets

To evaluate the effectiveness of our cross feature generation methodology, we ran a set of experiments on the Criteo dataset. For this set of experiments, we randomly split 33% of data as a test set and make sure that both train and test sets include 50% accept and 50% reject records.

We used accuracy as an evaluation metric and evaluated 3 models which are *Deep*, *Wide & Deep* and *Wide & Deep model with automatically crafted features*. We also measured evidence and uncertainty values of predictions to obtain the threshold levels for the models in which models are confident.

To find a common parameter setting that gives the best performance on all models, we performed a careful parameter evaluation step. The resulting parameter settings are listed below:

1. Network Structure: 200, 100, 75, 50, 10
2. Learning rate: 0.01
3. Droppout Rate: 0.05
4. Activation function: relu
5. Optimizer: adam

For the Criteo dataset, the problem becomes more complex, and a decision tree needs to have at least 13 levels to achieve above 70% accuracy. Larger tree structures increase the number of categorical and cross features created. The hash bucket size values also grow larger than desired, since decision paths become too long.

For sparse datasets like Criteo, we made a minor change on the proposed method and applied a filter based on the *Gini importance* values of features while selecting features. For our sample Criteo dataset, we set this threshold as 0.01 and eliminate features, which has less importance in comparison to the threshold value. With this small modification, we can find cross features as good as previous dense datasets.

B.2 Results on Criteo Dataset

In this section, we trained and tested 3 network models which are *Deep*, *Wide & Deep* and proposed *Wide & Deep model with automatically crafted features* on Criteo dataset. With this experiment our goal is to see if the proposed cross feature generation methodology improves model accuracy on a different problem with different characteristics.

The accuracy values obtained with each model are listed in Table 12, which also includes mean evidence and uncertainty values for each network model.

Table 12: Accuracy and uncertainty results on Criteo dataset

	Deep	*Wide&Deep*	*DtreeBasedWide&Deep*
accuracy	0.753	0.748	0.798
fail evidence	42.32	36.69	41.59
evidence	60.20	55.93	69.84
success evidence	66.05	62.04	76.97
fail uncertainty	0.23	0.28	0.24
uncertainty	0.14	0.17	0.13
success uncertainty	0.12	0.13	0.10

The highest accuracy value observed in this experiment is around 79.5% and belongs to *Wide & Deep model with automatically crafted features* while the lowest accuracy observed is around 74.8% and belongs to *Wide & Deep* model. Deep model, on the other hand, performs slightly better than *Wide & Deep* model, which indicates that adding cross features randomly without any expert knowledge may even degrade

the performance.

Figure 59 shows the change of accuracy for all three models during 100 epochs. In Figure 59, it is seen that all three models have similar accuracy change trends. All models show a rapid increase in accuracy for the first 20 epochs and slightly improves afterward. Proposed decision tree based cross feature generation methodology seems to outperform others in terms of accuracy throughout the run.

Besides deep models, we also trained and tested logistic regression and decision tree models to see the performance of other models on the Criteo sample dataset. We trained decision tree models from 2 levels up to 30 levels. The highest accuracy obtained with decision tree models is around 72%. This accuracy is achieved by a tree model with 27 levels. The accuracy of the logistic regression model is around 67%.

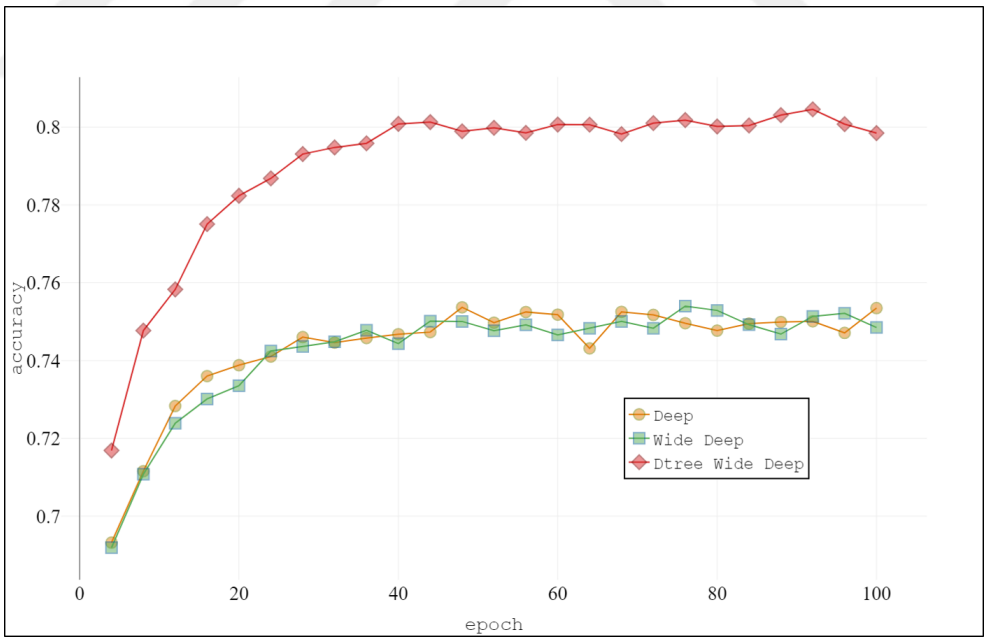

Figure 59: The change of accuracy per epoch for deep, Wide & Deep and Wide & Deep with automatically crafted features on Criteo sample dataset

Figure 60 and Figure 61 display evidence and uncertainty metrics measured during the execution of *Wide & Deep model with automatically crafted features*. Figure 60

plots change of evidence for correctly classified and incorrectly classified samples separately and shows that the neural network generates much more evidence for correctly classified samples and evidence tends to increase throughout the run while it generates much less evidence for incorrectly classified ones. Figure 61 shows change of uncertainty on correctly classified and incorrectly classified samples besides the change of accuracy on all test samples. Figure 61 shows that the model has a lower uncertainty (around 0.10) for correctly classified samples and uncertainty measured for incorrectly classified samples is twice as high as correct classifications (around 0.24).

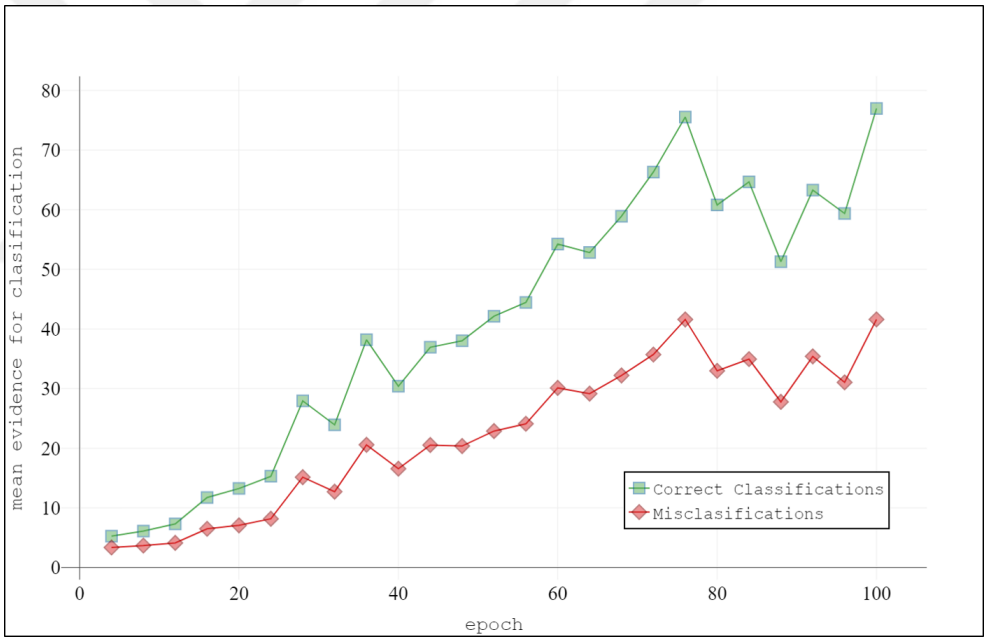

Figure 60: The change of evidence per epoch on Criteo sample dataset for Wide & Deep Model with automatically crafted features

Figure 62 plots how test accuracy changes if the model rejects making predictions above a varying threshold. According to Figure 62 model accuracy increases as uncertainty threshold decreases. For each threshold value, the neural network rejects to predict for a set of samples with the assumption that samples above a predefined uncertainty threshold will be classified incorrectly.

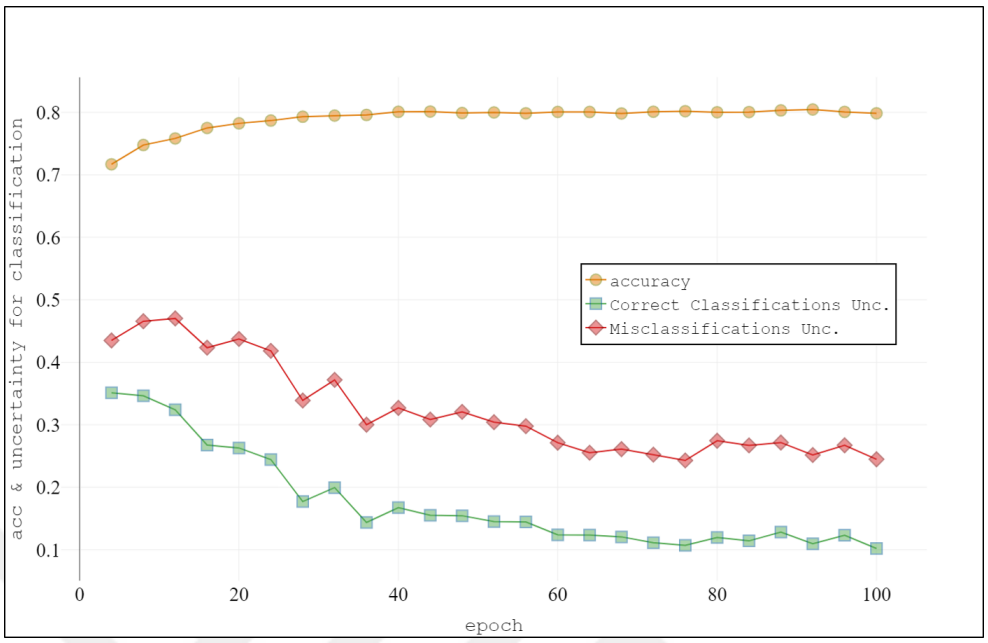

Figure 61: The change of accuracy and uncertainty on correctly classified samples and misclassifications on Criteo sample dataset for Wide & Deep Model with automatically crafted features

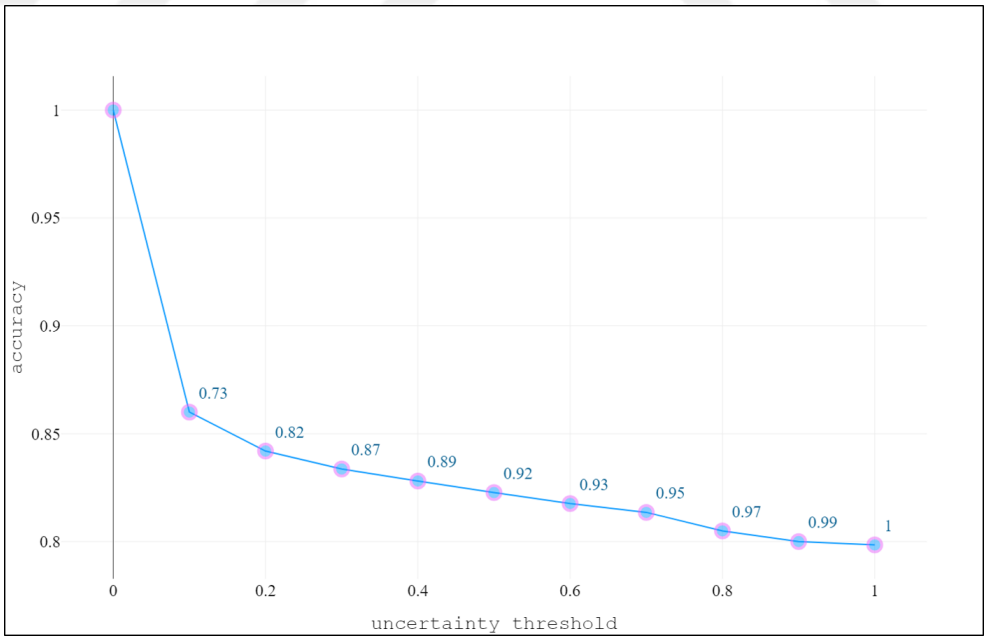

Figure 62: The change of accuracy with respect to uncertainty threshold on Criteo sample dataset for Wide & Deep model with automatically crafted features

Figure 63 and Figure 64 show change of accuracy with a varying uncertainty threshold for *Wide & Deep* and *Deep* models, respectively. The results indicates that it is not possible to abtain 99% accuracy on criteo dataset due to nature of this data but it is still possible to increase model accuracy with uncertainty-aware networks.

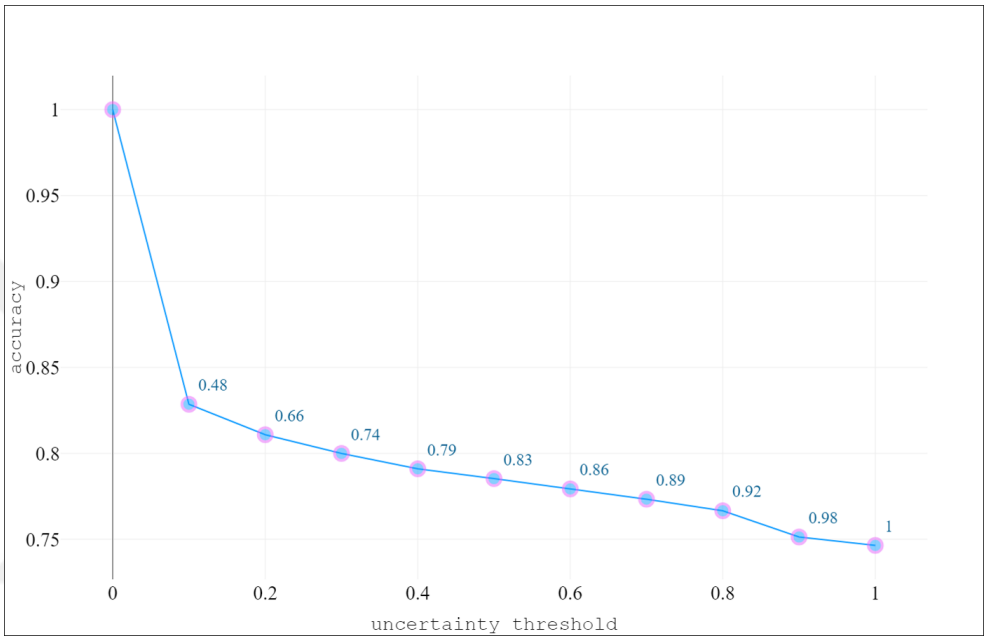

Figure 63: The change of accuracy with respect to uncertainty threshold on Criteo sample dataset for Wide & Deep model

B.3 Training and Evaluation Time Comparison

Figure 65 shows average execution time per sample measured during testing and training time for each model. The results indicate that all models are time efficient enough to be used in real time. *Deep* model has the least evaluation time and it is nearly 2 times faster than *Wide & Deep* models. The maximum measured average value belongs to *Wide & Deep model with automatically crafted features* and it is around 0.31 milliseconds. The overall picture shows that model training and evaluation time increases linearly as number of features used by the model increases.

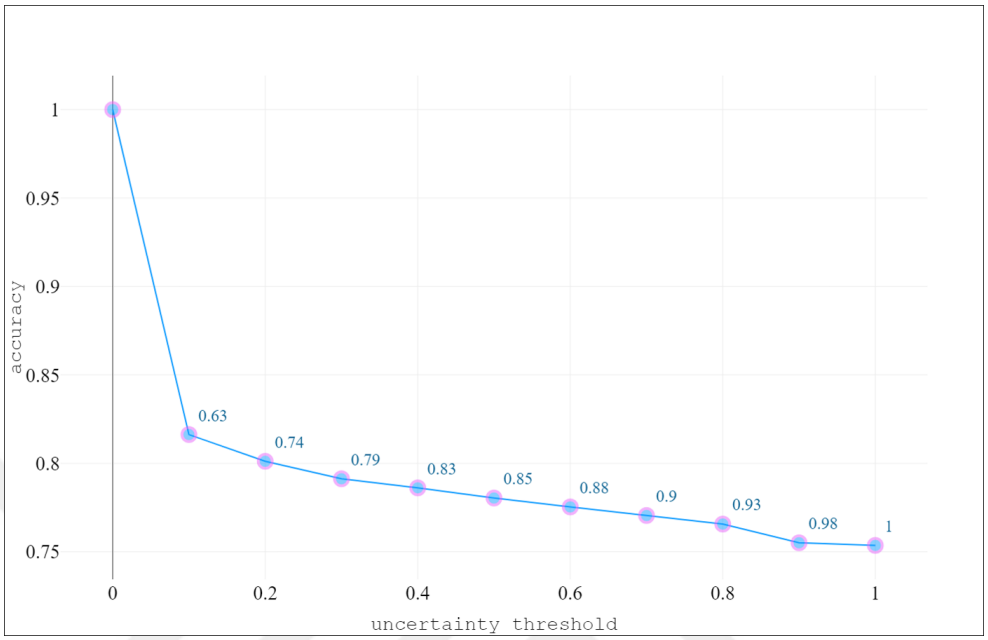

Figure 64: The change of accuracy with respect to uncertainty threshold on Criteo sample dataset for Deep model

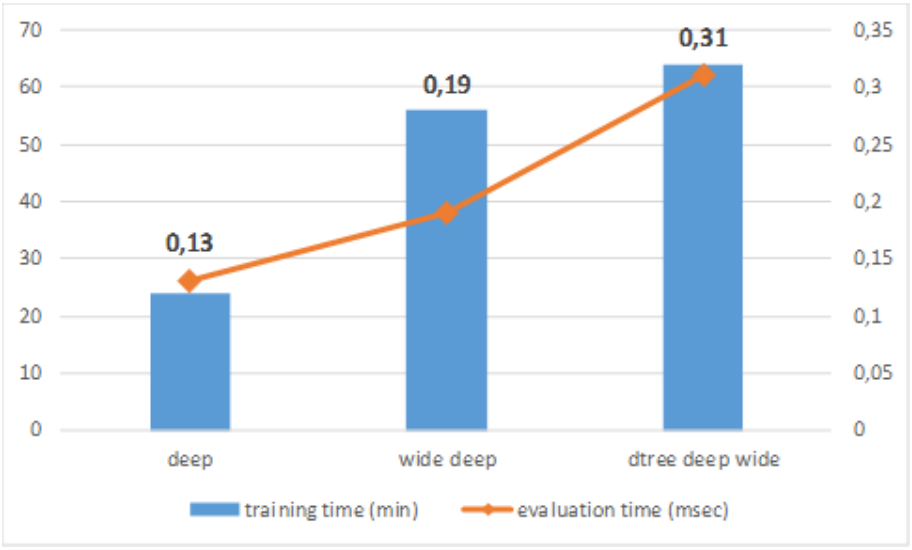

Figure 65: Evaluation time per item and total training time per model on Criteo sample dataset

Bibliography

[1] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, 2019.

[2] J. H. T. S. T. C. H. A. G. A. G. C. W. C. M. I. Heng-Tze Cheng, Levent Koc, "Wide deep learning for recommender systems," in *1st Workshop on Deep Learning for Recommender Systems*, pp. 7–10, ACM, 2016.

[3] TensorFlow, "Feature columns — tensorflow core — tensorflow," 2017. https://www.tensorflow.org/guide/feature_columns, (Last accessed 17 June 2019).

[4] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: A factorization-machine based neural network for ctr prediction," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 1725–1731, AAAI Press, 2017.

[5] Y. Shan, T. Ryan Hoens, J. Jiao, H. Wang, D. Yu, and J. Mao, "Deep crossing: Web-scale modeling without manually crafted combinatorial features," pp. 255–262, 08 2016.

[6] A. P V, "A survey of recommender system types and its classification," *International Journal of Advanced Research in Computer Science*, vol. 8, pp. 486–491, 09 2017.

[7] L. Fulop, G. Flp, R. Tth, J. Rcz, T. Pnczl, A. Gergely, and . Beszdes, "Survey on complex event processing and predictive analytics," 08 2010.

[8] S. Zhang, L. Yao, and A. Sun, "Deep learning based recommender system: A survey and new perspectives," *CoRR*, vol. abs/1707.07435, 2017.

[9] M. Sensoy, L. Kaplan, and M. Kandemir, "Evidential deep learning to quantify classification uncertainty," in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 3179–3189, Curran Associates, Inc., 2018.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[11] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, pp. 1798–1828, Aug. 2013.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, no. 2, pp. 119 – 130, 1988.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[15] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 2342–2350, JMLR.org, 2015.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[17] S. Rendle, "Factorization machines," in *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, (Washington, DC, USA), pp. 995–1000, IEEE Computer Society, 2010.

[18] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, pp. 81–106, Mar. 1986.

[19] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[20] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," vol. 29, no. 2, pp. 119–127, 1980.

[21] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[22] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2008.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[24] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in neural information processing systems*, pp. 2773–2781, 2015.

[25] A. P. Dempster, *A Generalization of Bayesian Inference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[26] A. Jøsang, *Subjective Logic: A Formalism for Reasoning Under Uncertainty*. Springer Publishing Company, Incorporated, 1st ed., 2016.

[27] J. Engel, M. Hoffman, and A. Roberts, "Latent constraints: Learning to generate conditionally from unconditional generative models," *CoRR*, vol. abs/1711.05772, 2017.

[28] K. Oflazer and H. C. Bozsahin, "Turkish natural language processing initiative: An overview," in *East Technical University*, 1994.

[29] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Trans. Manage. Inf. Syst.*, vol. 6, pp. 13:1–13:19, Dec. 2015.

[30] L. B. M. K. K. K. R. Collobert, J. Weston and P. Kuksa, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[31] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Trans. Manage. Inf. Syst.*, vol. 6, pp. 13:1–13:19, Dec. 2015.

[32] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 191–198, ACM, 2016.

[33] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath, "The youtube video recommendation system," in *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pp. 293–296, ACM, 2010.

[34] S. Okura, Y. Tagami, S. Ono, and A. Tajima, "Embedding-based news recommendation for millions of users," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1933–1942, 2017.

[35] D. Billsus, C. A. Brunk, C. Evans, B. Gladish, and M. Pazzani, "Adaptive interfaces for ubiquitous web access," *Commun. ACM*, vol. 45, pp. 34–38, May 2002.

[36] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, "Recommending and evaluating choices in a virtual community of use," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 194–201, ACM Press/Addison-Wesley Publishing Co., 1995.

[37] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, pp. 76–80, Jan. 2003.

[38] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "Movielens unplugged: Experiences with an occasionally connected recommender system," in *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pp. 263–266, ACM, 2003.

[39] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: An open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pp. 175–186, ACM, 1994.

[40] U. Shardanand and P. Maes, "Social information filtering: Algorithms for automating “word of mouth”," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 210–217, ACM Press/Addison-Wesley Publishing Co., 1995.

[41] D. H. Park, H. K. Kim, I. Y. Choi, and J. K. Kim, "A literature review and classification of recommender systems research," *Expert Systems with Applications*, vol. 39, no. 11, pp. 10059 – 10072, 2012.

[42] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, pp. 5–53, Jan. 2004.

[43] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, pp. 30–37, Aug. 2009.

[44] J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang, "Collaborative filtering and deep learning based recommendation system for cold start items," *Expert Systems with Applications*, vol. 69, pp. 29 – 39, 2017.

[45] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, pp. 1–127, Jan. 2009.

[46] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A survey on deep learning for big data," *Information Fusion*, vol. 42, pp. 146 – 157, 2018.

[47] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11 – 26, 2017.

[48] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surv.*, vol. 52, pp. 5:1–5:38, Feb. 2019.

VITA

Demet Ayvaz received her B.Sc. degree in Computer Engineering from Marmara University in 2004. Later in 2006, she received her M.Sc. degree in Computer Engineering from Boğaziçi University. She started her professional career in the area of Telecommunications in 2006 and joined Turkcell Technology in June 2011. Prior to her current position, she worked as Postpaid Solutions Specialist at Avea İletişim Hizmetleri A.Ş. Currently, she is working as a Senior Expert Data Analytics Developer at Artificial Intelligence & Analytic Solutions department of Turkcell Technology.