DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2018

An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing

CEDRIC SEGER

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing

CEDRIC SEGER

Bachelor of Science in Information and Communication Technology
Date: September 25, 2018
Supervisor: Johan Montelius
Examiner: Henrik Boström
Principal: Ather Gattami
Swedish title: En undersökning av kodningstekniker för diskreta variabler inom maskininlärning: binär mot one-hot och feature hashing
School of Electrical Engineering and Computer Science

Abstract

Machine learning methods can be used for solving important binary classification tasks in domains such as display advertising and recommender systems. In many of these domains categorical features are common and often of high cardinality. Using one-hot encoding in such circumstances lead to very high dimensional vector representations, causing memory and computability concerns for machine learning models. This thesis investigated the viability of a binary encoding scheme in which categorical values were mapped to integers that were then encoded in a binary format. This binary scheme allowed for representing categorical features using $log2(d)$-dimensional vectors, where d is the dimension associated with a one-hot encoding. To evaluate the performance of the binary encoding, it was compared against one-hot and feature hashed representations with the use of linear logistic regression and neural networks based models. These models were trained and evaluated using data from two publicly available datasets: Criteo and Census. The results showed that a one-hot encoding with a linear logistic regression model gave the best performance according to the PR-AUC metric. This was, however, at the expense of using 118 and 65,953 dimensional vector representations for Census and Criteo respectively. A binary encoding led to a lower performance but used only 35 and 316 dimensions respectively. For Criteo, binary encoding suffered significantly in performance and feature hashing was perceived as a more viable alternative. It was also found that employing a neural network helped mitigate any loss in performance associated with using binary and feature hashed representations.

Keywords: categorical features; feature hashing; binary encoding; classification

Sammanfattning

Maskininlärningsmetoder kan användas för att lösa viktiga binära klassificeringsuppgifter i domäner som displayannonsering och rekommendationssystem. I många av dessa domäner är kategoriska variabler vanliga och ofta av hög kardinalitet. Användning av one-hot-kodning under sådana omständigheter leder till väldigt högdimensionella vektorrepresentationer. Detta orsakar minnes- och beräkningsproblem för maskininlärningsmodeller. Denna uppsats undersökte användbarheten för ett binärt kodningsschema där kategoriska värden var avbildade på heltalvärden som sedan kodades i ett binärt format. Detta binära system tillät att representera kategoriska värden med hjälp av $log2(d)$ -dimensionella vektorer, där d är dimensionen förknippad med en one-hot kodning. För att utvärdera prestandan för den binära kodningen jämfördes den mot one-hot och en hashbaserad kodning. En linjär logistikregression och ett neuralt nätverk tränades med hjälp av data från två offentligt tillgängliga dataset: Criteo och Census, och den slutliga prestandan jämfördes. Resultaten visade att en one-hot kodning med en linjär logistisk regressionsmodell gav den bästa prestandan enligt PR-AUC måttet. Denna metod använde dock 118 och 65,953 dimensionella vektorrepresentationer för Census respektive Criteo. En binär kodning ledde till en lägre prestanda generellt, men använde endast 35 respektive 316 dimensioner. Den binära kodningen presterade väsentligt sämre specifikt för Criteo datan, istället var hashbaserade kodningen en mer attraktiv lösning. Försämringen i prestationen associerad med binär och hashbaserad kodning kunde mildras av att använda ett neuralt nätverk.

Nyckelord: kategoriska variabler; feature hashing; binär kodning; klassificering

Contents

Chapter 1

Introduction

Machine learning, or pattern recognition, has become an immensely popular approach for solving a wide variety of problems. Yahoo makes use of machine learning to classify email as spam [1] while Google uses machine learning for recommending apps in the Google Play store [2] and recommending videos on its YouTube platform [3]. In particular, deep learning approaches based on neural networks have gained a lot of attention ever since a convolutional neural network won the ImageNet Large Scale Visual Recognition Challenge in 2012 [4]. With the success of machine learning techniques, today many of the top technology companies have made machine learning an integral part of their business, including Facebook, Google, Apple, NVIDIA, Baidu, and Microsoft. Further, with academic and industry interest many tools and software frameworks are being worked on to enable better development and research: Tensorflow [5], Caffe [6], Pytorch [7] and ONNX [8] are some examples. All together this makes machine learning an interesting and worthwhile area of study.

1.1 Background

Many important problems that machine learning try to solve are of a binary nature. Recommendation systems in which the goal is to recommend a product can be phrased as a binary classification problem by predicting whether a person may like an item or not. The quality of such recommendation engines can have far reaching business and customer impact as ev-

ident by large scale systems such as YouTube's recommendations that impact more than a billion users [3]. Another important binary classification task is user response prediction in online display advertising. With digital advertising being a multi-billion dollar industry and click-prediction systems being a central part of online advertising systems [9], the quality of predictions are critical.

For both recommendation and ad-click prediction, generalized linear logistic regression models are widely used [2]. Recently there has also been a surge in applying deep learning techniques in an attempt to overcome cumbersome feature engineering [10]. Whether using a simple linear model or a deep neural network, one central problem, however, is how to represent discrete categorical features as input to the models. The standard technique applied is the use of one-hot encoded features. For linear models, non-linear cross-product transformations of the one-hot encoded features are also often included [2]. This approach, although promising, encounters problems with scale when dealing with very high dimensional feature spaces, common in recommendation and ad prediction tasks [11]. As a result methods such as feature hashing [12] or random projections [13] that try to compress feature representations have become relevant for large-scale linear models. Similarly, deep neural networks also experience computational problems when dealing with high dimensional one-hot representations. This has partly led to the use of embeddings in order to convert high dimensional, sparse input features into dense vectors better suited for computation in a neural network [2, 3, 14].

1.2 Problem

A one-hot representation, although commonly used, has several disadvantages. For example, one-hot requires storing a dictionary that maps categorical features to vector indices. When the cardinality of the categorical features are large these dictionaries can pose a significant strain on a computer's memory resources [1]. In addition, in sparse and high dimensional feature domains, storing the parameter vectors for one-hot encoded data becomes troublesome [15], even for simple models. Thus the problem that we seek to solve is to find different ways to represent categorical data

as to avoid the problems inherit in using a one-hot encoding.

Feature hashing has emerged as a popular approach for solving the scalability problems associated with using one-hot encoding. Feature hashing does so by removing the need for storing a dictionary and by allowing for dimensionality reduction. The approach has been succesfully applied to large-scale machine learning tasks [12, 1, 15]. The main problem with feature hashing is the potential occurence of hashing collisions: when two different values hash to the same index. Empirically it has been shown that the presence of hashing collisions still allows for good model performance, however. One reason for this is that feature hashing methods make use of 'sufficiently' high dimensional representations to mitigate hashing collisions: in several examples [15, 12] the resulting dimension of the new, feature hashed vector ranges on the scale of approximately 16,000 to 4-million dimensions.

As an alternative approach to solving the problems associated with one-hot encoding, we propose the use of a binary encoding scheme. That is, a feature with eight unique values will be represented as a vector with three dimensions ($log_2(8)$). This requires, as in one-hot, a mapping from categorical values to integers, but uses a binary representation of the integer. A categorical value mapped to an integer value of five will be represented in a three dimensional vector as $[1, 1, 0]$ (five in binary format). Using one-hot encoding one would have to use a five dimensional vector: $[0, 0, 0, 0, 1]$. This binary approach, to the best of our knowledge, has not extensively and formally been studied in literature (with the exception of a small-scale study comparing the performance of several encoding techniques [16]). Further, using binary encoding of categorical variables achieves a compressed representation without the explicit loss of information that is possible when using feature hashing. Together this suggests that there exists a scientific need and a practical interest in studying the use of binary encoding of categorical features.

1.3 Purpose

The purpose of this report is to investigate the relative performance differences that results from when using different encoding techniques of

categorical data to train a machine learning model. In particular, binary encoding is compared against one-hot and feature hashed representations for both a linear logistic regression- and neural networks based model.

1.4 Objectives

In order to answer the research question and achieve the aim of this study several goals need to be accomplished: data with a large number of categorical features needs to be collected. Using this data, binary classification models - using one-hot, binary and feature hashed representations of categorical input - need to be trained. Lastly, suitable performance measurements need to be defined and used to compare the trained models.

1.5 Methodology

To accomplish the goals, this study employs a quantitative, empirical research approach. For the data collection part, two publicly available datasets - Census income data [17] and Criteo ad-click prediction data [18] - have been chosen due to their different characteristics. This should help in providing a more general answer to the research question. Further, any continuous data features are either discarded or converted to categorical features. This allows the research to focus solely on the impact of encoding categorical features and limits influence of external factors.

In terms of models, a linear logistic regression model is used since this is a widely used model in practice [2]. However due to the popularity of neural networks, a neural network based logistic regression model is also trained. This follows our hypothesis that the type of feature representation and amount of compression will matter less for a neural network than for a simple linear model. For each model, dataset, and input encoding, a model is trained, resulting in a total of 12 trained models.

To evaluate the trained models - and thereby gauge the performance implications of the various input encoding strategies - precision, recall and area-under-curve for the precision-recall curve are used as performance metrics. These metrics are commonly employed in binary classification tasks [19].

1.6 Outline

In order to familiarize readers with the essential concepts discussed in this report a comprehensive background is given in chapter two. The method is described in chapter three and details the data, models, metrics and experiments. Chapter four describes the results, including a discussion and limitations section. Chapter five concludes and suggests potential future works.

Chapter 2

Background

This section aims to present a more comprehensive background for readers unfamiliar with the topics related to machine learning and feature representation.

2.1 Classification

Classification problems are concerned with classifying samples into distinct categories. In terms of binary classification, the two classes are often referred to as the positive class and the negative class and the goal is to determine whether a sample belongs to the positive or negative class.

2.1.1 Linear Logistic Regression

Logistic regression models the probability that a sample x belongs to a particular category or class. In terms of a binary classification problem, logistic regression can model the probability of a sample belonging to the positive class:

$$P(Y = 1|x).$$

In order to create a discrete output rather than a probability, one is free to choose a threshold. For example it is possible to choose a threshold of 0.5 and classify any sample for which $P(Y = 1|x) \geq 0.5$ as belonging to the positive class. It is equally possible to set a threshold of 0.8 if one wishes to

be more conservative or if the cost of wrongly labeling a sample as positive is high.

To model the relationship between the input x and the probability $P(Y = 1|x)$ a typical approach is to employ an affine transformation of the input data followed by the logistic, also known as the sigmoid, function:[20]

$$\begin{aligned} a &= w^T x \text{ (Affine transformation)} \\ P(Y = 1|x) &= \frac{e^a}{1 + e^a} \text{ (Logistic function)} \\ P(Y = 1|x) &= sigmoid(a) = \hat{y} \text{ (Short version)} \end{aligned} \tag{2.1}$$

where w represents a parameter vector and x the input to the model. While the relationship between input and ouput is modeled with the help of the model parameters, the sigmoid function in 2.1 restricts the output of the model to the range $(0, 1)$ and allows for the output to be interpreted as a probability.

2.1.2 Artificial Neural Networks

While a linear logistic regression model is appealing and often used in practice, an obvious limitation is that the functions it can express is limited to linear functions of the input x. For example a linear model cannot model the interaction between any two input variables [21]. In response to this limitation artificial neural networks and their derivatives have become a popular approach for automatically modeling linear and non-linear functions of the input x. Neural networks accomplishes this by specifying a fixed a number of basis functions (non-linear transformations) of the input but allow the transformations to be adaptive. This enables the network to learn appropriate feature transformations as part of the learning process [22]. Figure 2.1 shows a graphical representation of a feed-forward neural network.

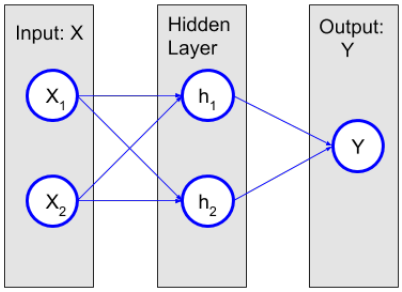

Figure 2.1: Graphical representation of simple neural network

The above network in figure 2.1 can be mathematically specified as:

$$f(X; W_1, W_2, b_1, b_2) = h_2(W_2^T h_1(W_1^T X + b_1) + b_2)$$

where $h_1()$ is a non-linear activation function such as a sigmoid function. The h_2 function computes the final output Y and is chosen according to the type of output required - for binary classification one may choose to use a sigmoid activation function.

In practice, neural networks have been successful at dealing with several complex tasks such as image classification [23], autonomous driving [24] and natural language processing [25].

2.1.3 Learning

In order for a model to be useful it is necessary to learn the parameters of the model. Logistic regression models, and many other machine learning models, makes use of the principle of maximum likelihood to fit the parameters of the model [21]. Particularly it is common to minimize the negative log-likelihood of the data rather than maximizing the likelihood. For logistic regression models that differentiate between only two classes, it is possible to write the negative log-likelihood as

$$l(w, x_i, y_i) = -y_i log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i) \tag{2.2}$$

where y_i is the label of sample i and $\hat{y}_i$ is the probability of the sample belonging to the positive class as defined in equation 2.1. The expression in equation 2.2 is also commonly referred to as a cost function in the machine learning literature.

With a cost function, whose value we seek to minimize, defined it is possible to update parameters of a logistic regression model in an incremental fashion using stochastic gradient descent. The update equations can be written as:

$$\begin{gathered} gradient = \frac{1}{m}\nabla_w \sum_{i=1}^{m} l(w, x_i, y_i) \\ w_{new} \leftarrow w_{old} - \epsilon(gradient) \end{gathered} \tag{2.3}$$

where m represents the batch size and ϵ is a constant called the learning rate. Learning is thus made possible by iterating through the training data and performing the updates as shown in equation 2.3. Note that neural network based logistic regression and linear logistic regression models can both be trained using the maximum likelihood approach and the concept of following gradient information.

2.2 Feature Representation

Models act on data - machine learning models such as the logistic regression model require instances of input, x, to produce an output. In this context, x is generally a set of attributes, also known as features, that describe a particular sample point instance. If one uses n features to describe a particular sample, x becomes a sample point in an n-dimensional data space: $x = [x_1, x_2, ..., x_n]$.

In the broadest sense one can distinguish between two types of features: numerical and categorical. Numerical features are usually represented by either floating-point or integer numbers and arise naturally in many fields. In predicting the salary of a person, the age of the person can be used as a numerical feature. The hope is that by knowing the age of a person the model can more accurately predict that person's salary. The representation of categorical variables is less obvious as there is no natural way to perform numerical computation on categories.

This section will discuss various approaches to feature representation with particular emphasis on categorical features.

2.2.1 One-hot

The most common approach to converting categorical features to a suitable format for use as input to a machine learning model is one-hot encoding. Continuing with the example of predicting a person's salary it is possible that the person's type of employment is an important factor to consider. For example, a lawyer tends to make more money than a student. Assuming that we wish to differentiate between four types of employment - student, teacher, doctor and banker - it is possible to represent this information using one-hot encoding as shown in figure 2.2.

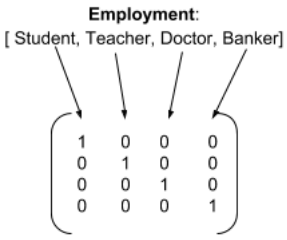

Figure 2.2: Graphical representation of one-hot encoding

Each category value in figure 2.2 is represented as a 4-dimensional, sparse vector with zero entries except for one of the dimensions for which the value is one. In general, for variables of cardinality d, the vectors would have d-dimensions. An interesting property of the one-hot encoding is that the categories are represented as independent concepts - one way to see this is to note that the inner product between any two vectors is zero, each vector is equally far from each other in euclidean space.

Since the data using one-hot encoding is of numerical nature, a machine learning model can easily incorporate such categorical feature information by learning a separate parameter, w, for each dimension. One of the problems with using one-hot encoding in practice, however, is that the cardinality of variables can be large. Recommendation systems such as click through prediction models can be forced to deal with million-, billion- or even trillion-dimensional feature spaces. In such settings, efficient processing and even storing of the data using one-hot encoding becomes a problem [11]. Also, the number of parameters to be learned

become very large. If one additionally considers cross-product transformations - common in logistic regression models [2] - the problem is further exacerbated. To try to alleviate the problems inherit in the standard one-hot representation in the case of high cardinality, data compression techniques become relevant and are discussed next.

2.2.2 Binary

Categorical data can be represented in a binary format by first assigning a numerical value to each category and then converting it to its binary representation. For a feature with d-unique values, this results in a $log2(d)$-number of on or off discrete values. The process is shown graphically in figure 2.3.

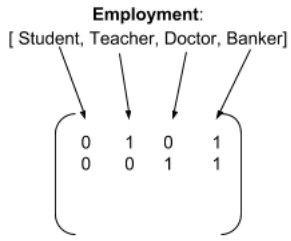

Figure 2.3: Graphical representation of binary encoding

To the best of our knowledge, few attempts [16] have been made to study binary encoding in a formal setting.

2.2.3 Feature hashing

The use of hash functions has been proposed as an alternative to the one-hot encoding of categorical features [26, 12]. The approach is particularly popular when dealing with large-scale datasets and has become part of many of the popular machine learning software and services [27, 15].

In order to understand feature hashing we first review the definition of a hash function. In general, hash functions are functions that map an input U (usually referred to as a key) to a number:

$$f : U \rightarrow \{0, 1, ..., m\}$$

where m is an integer. An example of a hashing function is:

$$f(x) = (3x + 5) \; mod \; 5.$$

If we assume x is an integer key, then the hash function maps any integer, x, to another integer in the set $\{0, 1, 2, 3, 4\}$. An important point is that it is possible to design hash functions for a variety of keys - a hash function that maps string keys to integers is an example. This property has classically allowed hash functions to be used for creating efficient data structures such as hash-tables.

The application of hashing in a machine learning context becomes clear by noting that raw categorical data is usually stored in string format. Thus it is possible to treat the raw string as a key for input to a hash function. For example when dealing with categorical features, in practice it is common to concatenate the name of the category and its actual value at a point [15, 1]. If the category is 'employment' and a particular sample has the value 'student', then the input to the hash function would be 'employment=student'. By design, the output of the hash function is an integer number that can be used to index into a feature vector, similar to how a hash-table look-up is performed. The process of converting categorical values to a suitable feature vector using hashing is illustrated in figure 2.4.

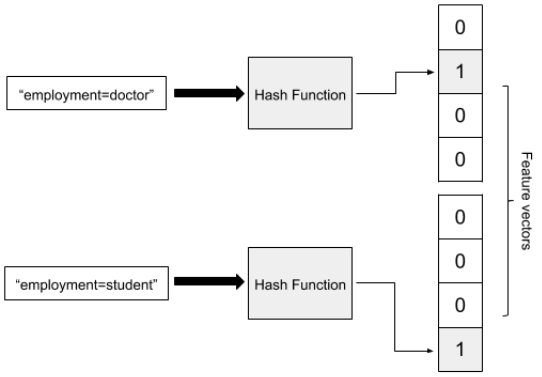

Figure 2.4: Graphical representation of feature hashing

The hash function used in figure 2.4 hashes keys to integers in the range $[0, 3]$ and hence results in 4-dimensional vectors. In practice we are free to

choose the range of the ouptut and thereby allow for dimensionality reduction. It is possible to choose to hash the values of the employment category to the set $\{0, 1\}$ and hence represent the employment category using a 2-dimensional vector. The cost of reducing the dimension of a vector, however, is the potential loss of information: two keys can hash to the same index. Hash collisions become more likely with a smaller number of dimensions. Empirically it seems that hash collisions does not significantly impact prediction performance of machine learning models [26, 12]. To reduce the impact of hashing collisions, using a second hashing function has been proposed [12].

Other than reducing the dimensionality, other interesting properties of feature hashing include the ability to use it in an online fashion and its ability to handle variable length vocabularies. For example feature hashing can be used as a fast method for text-feature vector extraction [28]. Also Weinberger et al. [12] argue that feature hashing preserves information as well as random projections and show that hashed feature vectors approximately preserve similarity measures such as inner products between sample data points.

Chapter 3

Method

This study aims to answer the research question through a quantitative, empirical study. It does so by investigating and comparing the performance of machine learning models trained to perform binary classification. The choice of binary classification tasks is not essential as there are many other valuable machine learning tasks that can be studied such as regression models or multi-class classification models. Nevertheless, binary classification tasks are important in machine learning as evident by large scale recommendation systems [3] and ad-click prediction systems that are part of multi-billion dollar industries [9].

In order to train and evaluate machine learning models it is required to choose appropriate datasets. The datasets to be considered need to conform to binary classification tasks since the goal is to study binary classification models. In order to emphasize reproducible results, two publicly available datasets are used: Census [17] and Criteo[18]. While other datasets are possible, both Census and Criteo have a large amount of categorical features. Further, Census and Criteo exhibit different characteristics: the Census data has lower cardinality features and is of smaller size while Criteo has features of much higher cardinality, has more features in total and is a dataset of larger scale (more samples). This is useful in order to investigate if binary encoding, feature hashing and one-hot perform differently under different circumstances. Hence it allows us to answer the research question more broadly.

Considering that one-hot encoding encounters memory problems even

for simple models [15], a linear logistic regression model is used for prediction. Also a neural networks based logistic regression model is tested. This follows from our hypothesis that the type of feature representation will matter less for a neural network than for a linear model. Generalized linear logistic regression models and neural networks based approaches are widely used in industry [29, 2, 10], making them interesting models to study. Other interesting, alternative models to study could be tree-based models but were chosen not to be included due to time limitations.

This chapter continues by describing these choices in greater detail, including the specific experimental setup.

3.1 Data

Two datasets - Census and Criteo - are used for conducting experiments. Both datasets contain a large number of categorical features, making them ideal for testing performance implications for categorical feature compression. The two datasets also have different characteristics, these characteristics are outlined in the next sections.

3.1.1 Census

The Census Income data[17] consists of 45,222 samples of income data for adults in the United States taken from the census bureau database. The goal is to predict whether a person makes more than 50,000 USD in salary. Each sample consists of 14 mixed continuous and categorical features. Some features contained little or highly sparse information and as such were discarded from the data. Any remaining continuous features were converted to categorical by discretizing into 10 equal-sized bins. The resulting categorical features and their cardinality is shown in table 3.1a. The class distributions for the complete data is shown in table 3.1b.

Table 3.1: Census dataset description

(a) Feature description

Feature	*Cardinality*
age	10
workclass	7
education	16
marital status	7
occupation	14
relationship	6
race	5
sex	2
hours per week	10
native country	41
TOTAL	118

(b) Class distributions

Class	*%*
1	24.4%
0	75.6%

3.1.2 Criteo

The Criteo dataset[18] is a real world dataset compromised of seven days of display ad logs from Criteo. Each ad is described by 13 integer and 26 categorical features and the goal is to predict click or no click for each ad. The original data contains some categorical features with very high cardinality and rare occurrences. In order to reduce the cardinality of such feature, all infrequently occurring values (feature values occuring less than 500 times) were mapped to a new, common category. Further, all continuous features were discretized by mapping them into bins derived from the relevant feature's 95th percentile. If the 95th percentile of a continuous feature turned out to be larger than 100, that feature was simply mapped into single-sized bins from zero to 150 (resulting in 150 bins of size one). The resulting features used for prediction had cardinalities in the range of three to 6,899 with a total sum of cardinalities of 65,953. The class distribution for the complete data can be seen in table 3.2.

Table 3.2: Criteo data - class distributions

Class	%
1	24.4%
0	75.6%

3.2 Models

To test our hypothesis we run experiments on the two datasets using both a linear and non-linear model. An overview of these models is illustrated in figure 3.1.

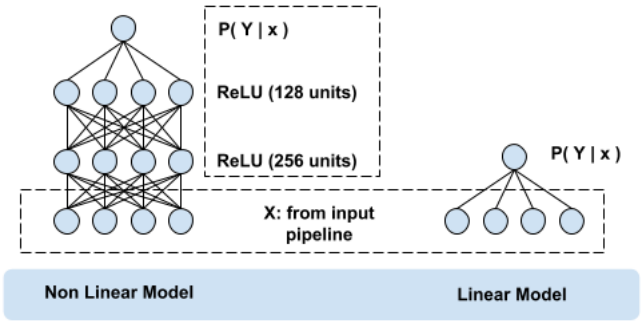

Figure 3.1: Illustration of the models used

The linear model is represented by an affine transformation in the form:

$$P(Y|x) = sigmoid(w^T x + b)$$

where $w = [w_1, w_2, ...w_d]$ are the model parameters to be learned, b is a bias term and $x = [x_1, x_2, ..x_d]$ are the transformed features received from the input pipeline.

The non-linear model can be seen on the left in figure 3.1 and is a two-layer feed-forward neural network with ReLU activations in the hidden layers. Additionally, batch normalization layers are included between each hidden layer since this is known to stabilize the training procedure of

neural networks [21]. The network consists of 256 and 128 units in the first and second layer respectively. Specifically, the neural network computes the following:

$$l_1 = h(w_1^T x + b_1)$$
$$l_2 = h(w_2^T l_1 + b_2)$$
$$P(Y|x) = sigmoid(w_3^T l_2 + b)$$

where l_1, l_2 represent the two layers, h is the ReLU activation function, w_1, w_2, w_3 the model parameters and b_1, b_2, b_3 are bias terms.

3.2.1 Learning

All the learning problems considered are binary classification tasks. Correspondingly the final output of the models is $P(Y|x) = sigmoid(...)$ and represents the probability of a sample, x, belonging to the positive class. Learning is done by minimizing the binary cross entropy between the true labels and predicted conditional labels. The cross entropy or negative log likelihood is widely used as it provides well behaved gradient updates [21] - required for learning to be efficient.

In addition to the log loss, an l2 regularization term is also included as part of the cost function for the neural network models. L2 regularization was not included for the linear model as we found this to worsen the performance. The gradient of the loss is propagated through the model to update the parameters using stochastic gradient descent. Learning is done on minibatches of size 32 with the Adam optimizer [30]. Due to the large size of the Criteo data, a larger batch size of 512 was used in order to speed up the training procedure. In general, a small mini-batch size is motivated by recent research by Masters and Luschi [31] that suggest that smaller batch-sizes improves stability and reliability of learning by providing more up-to-date gradient calculations. Further, some of the learning problems become very high dimensional when encoding input as one-hot vectors; a smaller batch-size therefore also helps to reduce the memory footprint.

In order to train and evaluate the models, the datasets were split into training and test sets. For the Census data, training and test sets were constructed by randomly partitioning the data into 80% training and 20%

for testing. The models were then trained for 40 epochs[1] on the training data and finally evaluated once on the test data. For neural networks, due to their non-convex optimization, this process was repeated ten times and the results averaged for the Census dataset.

For Criteo, due to its large size, training for 40 epochs is infeasible. Instead, the original data with 45,840,617 samples was split into train and test sets by taking the last 6,548,660 samples to form the test set. Similar procedures have been done by others [32] and the reason is that the Criteo data is chronologically ordered: the last 6,548,660 samples roughly correspond to the 7th day of the collected data. Training was carried out for a total of one epoch on the training set and the model was evaluated once on the test set.

3.3 Input Pipeline

The goal is to compare one-hot encoding, feature hashing and a new binary encoding scheme of the input. The input pipeline used to transform raw data values into suitable representations using one of these encoding schemes is illustrated in figure 3.2.

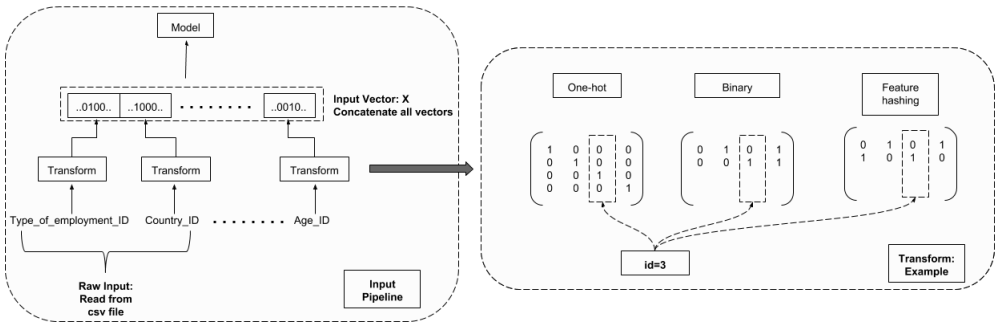

Figure 3.2: Illustration of input pipeline

The raw input is read-in from a csv file and each feature is separately transformed into the chosen representation such as one-hot or binary en-

[1]One epoch corresponds to iterating over the full training data once

coding. The transformed representations are then concatenated into a single vector, X, which is used as input to the models in figure 3.1.

Using a binary encoding scheme results in a $log2$ compression compared with the dimensionality of a one-hot encoded feature. Hence if the one-hot encoding results in 4-dimensional vectors, the corresponding binary encoding has only 2-dimensions. This is illustrated on the right in figure 3.2. While we are free to choose the dimensionality for feature hashing, we chose to hash the input to the same dimensionality as that of the binary encoding. The reason is that we want to compare a binary compressed representation with that of a feature hashed compressed representation. Note that it is possible to hash keys to the same value and is the reason why some feature values are represented as the same vector in figure 3.2.

3.4 Metrics

To compare the performance implications of the different encoding techniques precision, recall and area-under-curve for the precision-recall curve are used as evaluation metrics. The metrics are defined as follows:[19]

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$

where TP is true positives, FN is false negatives and FP stands for false positives. The recall metric measures the fraction of positive examples that are labeled correctly. Precision measures the fraction of times that the classifier is correct when predicting a positive class. For a logistic regression model that has a probabilistic output, the precision and recall values are associated with a chosen threshold. A precision-recall curve can be constructed by plotting points in a (recall, precision)-space by calculating precision and recall at various thresholds. The area under the precision-recall curve can be used a simple metric for comparing the capacity of the models.

These are common performance measures for binary classification tasks in other research [2] and are particularly well suited for when the data exhibits class imbalances [19]. Since both Criteo and Census are imbalanced

with respect to class distribution and have the goal of binary classification, these metrics are suitable to use. Specifically, the precision-recall curve is a better measure than accuracy for imbalanced datasets since it takes into account the trade-off in enhancing accuracy by biasing the classifier towards positive examples [33, 34].

To give a concrete example of why accuracy is not a sufficient metric for imbalanced data: in the case of a dataset exhibiting 99% positive samples, any naive classifier can achieve 99% accuracy by simply predicting positive classification for all samples.

Chapter 4

Results

This section describes the results of the experiments.

4.1 Results for Census Data

The area-under-curve metric for the precision-recall curve (PR-AUC) with respect to the different input representations is illustrated in table 4.1. The one-hot encoding technique resulted in a 118-dimensional vector representation for each sample point. The compressed representations, by using $log2$ number of bits to represent each feature, resulted in each sample point being encoded as 35-dimensional vectors. Compared with the one-hot representation, the compressed representations therefore achieved a compression rate of approximately three.

Using a simple affine transformation of the input, as represented by the linear model results in table 4.1a, the one-hot representation resulted in the model with the greatest capacity. Using a binary representation the model performed slightly worse while using feature hashing as an encoding technique resulted in the worst performance. Using a neural network model on top of the representations resulted in an increase in model performance as measured by PR-AUC for both the binary and feature hashing approaches - results are illustrated in table 4.1b. The difference in performance between the various encoding techniques is also less: the difference in performance between binary and one-hot is no longer significant using a margin of one standard deviation.

The precision and recall metrics for each model and input representation tell a similar story to that of PR-AUC. While precision is relatively similar for all input encoding techniques and models, recall shows a bigger difference. In particular, feature hashing gained a significant increase in recall when using a neural network model than when using a linear model.

Table 4.1: Performance on Census Data

(a) Linear Model Performance

Input Type	*PR-AUC*	*Precision*	*Recall*	*Dimension*
One hot	0.730	0.72	0.58	118
Binary	0.664	0.70	0.51	35
Feature Hashing	0.600	0.66	0.33	35

(b) Non-Linear Model Performance

Input Type	*PR-AUC*	*(+/-) Std*[a]	*Precision*	*Recall*
One hot	0.728	0.002	0.72	0.57
Binary	0.714	0.006	0.71	0.57
Feature Hashing	0.691	0.006	0.70	0.53

[a]The standard deviation is based on 10 re-runs of training and evaluating each model. The PR-AUC reported is the mean of these 10 runs.

4.2 Results for Criteo Data

The results from the experiments performed on the Criteo data are illustrated in table 4.2. Using a one-hot encoding resulted in a 65,953-dimensional vector representation for each sample point. The compressed representations (binary and feature hashing) made use of a 316-dimensional vector representation. Comparing the dimensionality, the compressed representations achieve a compression rate of approximately 209.

Model performance using a linear transformation of the encoded input is shown in table 4.2a. Using a linear model, the one-hot representation resulted in the best performing model. Feature hashing resulted in a worse performing model but a better model than when using a binary encoding.

Results from using a neural networks based approach are shown in table 4.2b. One-hot resulted in the best performing model for the non-linear approach. The performance is worse, however, when compared with a one-hot encoding using a linear model. The performance of both the binary and feature hashing encoding techniques increased when using a non-linear model than compared to a linear model.

The precision and recall metrics are also shown. The precision metrics are relatively more similar across model type and input representation than when comparing model performance using recall. Particularly, recall for one-hot encoding is significantly higher than when using other encoding techniques.

Table 4.2: Performance on Criteo Data

(a) Linear Model Performance

Input Type	*PR-AUC*	*Precision*	*Recall*	*Dimension*
One hot	0.573	0.64	0.36	65,953
Binary	0.466	0.59	0.17	316
Feature Hashing	0.471	0.60	0.18	316

(b) Non-Linear Model Performance

Input Type	*PR-AUC*	*Precision*	*Recall*
One hot	0.530	0.65	0.24
Binary	0.484	0.61	0.19
Feature Hashing	0.496	0.66	0.14

4.3 Discussion

All the models, with their respective encoding techniques, performed better than a random model would perform. A random model would be expected to achieve a PR-AUC of approximately 0.240[1], considerably worse than the worst performing models on Census and Criteo. This suggests that the models learned are somewhat useful for making predictions. Overall a linear model combined with a one-hot encoding of categorical variables consistently gave the best results. This can be expected since a one-hot encoding implies all categories to be independent and learns a separate parameter for each concept, there is no sharing of the parameters between categories. In contrast a binary encoding, by using fewer parameters to represent input, imposes a different assumption about the categories. For example using a binary coding for a category with four unique values, it is possible to use the following encoding:

category 1: $[0, 0]$
category 2: $[0, 1]$
category 3: $[1, 0]$
category 4: $[1, 1]$

which implies that category four is made up of category two and category three - category four shares the parameters of these other categories. Feature hashing achieves the same compression rate as binary and while feature hashing tries to preserve the structure of a one-hot encoded vector there inevitably occurs hashing collisions. This effect should be especially pronounced for the Census data as the compressed vector only has 35 dimensions. Feature hashing performs worse than binary on Census. For the Criteo data, using a larger compressed representation of 316 dimensions, however, feature hashing outperforms binary. This seems to indicate that the binary representation, by imposing explicit parameter sharing between categories, makes it more difficult for a model to perform well. It seems that an independent encoding achieves better performance in general.

[1]Calculation takes into account the class distributions of the data sets.

Applying a neural network on top of the one-hot encoded input did not seem to increase performance. That the linear model with one-hot encoding performed the best seems to be in line with other research [14, 2] that often makes use of generalized linear logistic models for similar prediction tasks. Thus it seems that using a neural network to extract useful feature interactions from a one-hot encoded input is a non-trivial task using the standard multi-layer perceptron model. In fact other research tries to find ways to structure neural networks to better learn such interactions [14, 10, 32].

For feature hashing and binary encoded representations applying a neural network did, however, increase the performance of the resulting model. If we regard the binary representation as a non-linear transformation of a one-hot encoded input, then a neural network could have the ability to disentangle some of the non-linearity inherit in the representation and thereby perform better. Similar reasoning can be made for the case of feature hashing.

Whether using a binary or feature hashed representation of the input is practical is also interesting to consider. For the Census data the dimensionality of the one-hot encoding is already relatively small - using only 118 dimensions - and so compressing the representation is not of great practical interest (modern computers easily handle 118 floating points for computation). More interesting is to consider the Criteo data in which the compression is more significant. It is clear that compressed representations comes with a drop in performance. For a similar level of precision, the compressed representations (binary and feature hashing) have a significantly worse recall of about 20% in the case of the linear model.[2]

4.3.1 Limitations

The results presented are limited in that only two datasets have been considered: Census and Criteo. In addition, the evaluation data was chosen in a simple way; for census a random 20% of the data was chosen for evaluation while for Criteo the last day of data was used. For more robust

[2]This indicates that it is harder to detect the positive samples using a compressed representation, but having labeled a sample as positive, the probability for the models to be correct is about the same.

results, techniques such as cross-validation could be used, although more time consuming.

Further, the learning algorithms used have a stochastic nature: for example certain initialization of model parameters may lead to sub-optimal convergence behaviour of the algorithms. This is particularly true for neural networks that suffer from a non-convex optimization problem. Several re-runs of each algorithm may enhance the reliability of the results.

The number of epochs for which the algorithms were trained can have an impact on the final results presented. For Census the number of epochs was chosen in an ad-hoc manner based on several trial runs. Learning on Criteo, however, was limited by the size of the dataset.

Lastly, the focus on feature hashing and one-hot encoding may falsely lead to the impression that binary encoding is the only other alternative for reducing the dimensionality of the data. In practice, singular value decomposition, principal component analysis, random projections and other methods can be used to achieve a similar goal.

Chapter 5

Conclusion

The aim of his report was to investigate how the performance of binary classification models are affected when using a binary encoding of categorical features rather than a one-hot encoding. Performance was also measured against a feature hashed representation of input. The input encoding schemes were tested on two binary classification tasks and experiments with both a linear and non-linear model were carried out.

The results provide no evidence in favor of a binary encoding with respect to predictive performance. For high cardinality data, in particular, encodings more similar to one-hot seem to be easier to optimize and yield better results. Considering that compression is of greatest practical interest when the one-hot dimensionality is large, feature hashing may provide a better alternative to a binary encoding. This is indicated by the results from the Criteo data.

Further, applying a neural network on top of the compressed representations led to better performance but at the cost of introducing more parameters and thereby more time consuming computations.

Due to the limitations in the number of experiments carried out, the results should not be interpreted as definitive. Instead the results show an indication of the potential performance aspects of the various models and encoding techniques. More testing and experimentation is encouraged and required, suggestions are given in the next section.

5.1 Further Research

This research did not investigate the speed of execution associated with the various encoding techniques. Binary or feature hashing may yield interesting speed improvements by using fewer parameters than a one-hot encoded input. Since machine learning models are popular for many online services, speed of operation is an interesting characteristic to investigate.

Another interesting area of further research is to extend the experiments presented in this research to new and different datasets. Together, such results could give a more reliable indication of performance implications of the various encoding techniques.

Lastly, the experiments conducted in this research assumed all input features were categorical and proceeded to encode all features using the desired encoding technique (one-hot, binary, feature hashed). It might be more relevant to only encode very high cardinality features using a compressed representation and to encode other features using a one-hot encoding - mixing different encoding techniques. This would be an interesting approach to consider further.

Bibliography

[1] J. Attenberg, K. Q. Weinberger, A. Dasgupta, A. Smola, and M. Zinkevich, "Collaborative email-spam filtering with the hashing-trick," in *Sixth Conference on Email and Anti-Spam*, 2009.

[2] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & deep learning for recommender systems," *CoRR*, vol. abs/1606.07792, 2016. [Online]. Available: http://arxiv.org/abs/1606.07792

[3] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B.

Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *CoRR*, vol. abs/1408.5093, 2014. [Online]. Available: http://arxiv.org/abs/1408.5093

[7] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS 2017 Autodiff Workshop*, 2017.

[8] J. Q. Candela. (2017) Facebook and microsoft introduce new open ecosystem for interchangeable ai frameworks. Accessed: 2018-05-16. [Online]. Available: https://research.fb.com/facebook-and-microsoft-introduce-new-open-ecosystem-for-interchangeable-ai-frameworks/

[9] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. n. Candela, "Practical lessons from predicting clicks on ads at facebook," in *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ser. ADKDD'14. New York, NY, USA: ACM, 2014, pp. 5:1–5:9. [Online]. Available: http://doi.acm.org/10.1145/2648584.2648589

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

[11] A. Shrivastava. (2017) 2017 rice machine learning workshop: Hashing algorithms for large-scale machine learning. [Online]. Available: https://www.youtube.com/watch?v=tQ0OJXowLJA

[12] K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. Smola, "Feature Hashing for Large Scale Multitask Learning," *ArXiv e-prints*, Feb. 2009.

[13] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: Applications to image and text data," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: ACM, 2001, pp. 245–250. [Online]. Available: http://doi.acm.org/10.1145/502512.502546

[14] W. Zhang, T. Du, and J. Wang, "Deep learning over multi-field categorical data: A case study on user response prediction," *CoRR*, vol. abs/1601.02376, 2016. [Online]. Available: http://arxiv.org/abs/1601.02376

[15] N. Pentreath. (2017) Feature hashing for scalable machine learning: Spark summit east talk by: Nick pentreath. [Online]. Available: https://www.youtube.com/watch?v=Uv9dY6Obv-s

[16] K. Potdar, T. S. Pardawala, and C. D. Pai, "A comparative study of categorical variable encoding techniques for neural network classifiers," *International Journal of Computer Applications*, vol. 175, no. 4, pp. 7–9, Oct 2017. [Online]. Available: http://www.ijcaonline.org/archives/volume175/number4/28474-2017915495

[17] Census income data set. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Census+Income

[18] Kaggle display advertising challenge dataset. [Online]. Available: http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/

[19] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 233–240. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143874

[20] T. Hastie, *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, 2nd ed., ser. Springer Series in Statistics. Springer, 2009.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[22] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. New York: Springer, 2006.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[24] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," *ArXiv e-prints*, Apr. 2016.

[25] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," *ArXiv e-prints*, Oct. 2013.

[26] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan, "Hash kernels for structured data," *J. Mach. Learn. Res.*, vol. 10, pp. 2615–2637, Dec. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755873

[27] Microsoft. (2018) Feature hashing. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/feature-hashing

[28] G. Forman and E. Kirshenbaum, "Extremely fast text feature extraction for classification and indexing," in *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, ser. CIKM '08. New York, NY, USA: ACM, 2008, pp. 1221–1230. [Online]. Available: http://doi.acm.org/10.1145/1458082.1458243

[29] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu,

M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica, "Ad click prediction: a view from the trenches," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[31] D. Masters and C. Luschi, "Revisiting Small Batch Training for Deep Neural Networks," *ArXiv e-prints*, Apr. 2018.

[32] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," *CoRR*, vol. abs/1708.05123, 2017. [Online]. Available: http://arxiv.org/abs/1708.05123

[33] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One-sided selection," in *In Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 179–186.

[34] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, Sept 2009.

TRITA EECS-EX-2018:596