2020

SCAM JOB ALERT

---a web application, to protect job seekers from being deceived by scam jobs.

PATTERN RECOGNITION SYSTEMS PROJECT

TAO XIYAN (A0215472J)
ZHANG ZEKUN (A0215485B)
SHI ZHAOHENG(A0215413U)

NUS National University of Singapore | ISS INSTITUTE OF SYSTEMS SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE | INSTITUTE OF SYSTEM SCIENCE

Contents

Executive Summary

Since the COVID-19 global pandemic hit in December 2019, its outbreak has dramatically impacted the world on every level. In Singapore, people have experienced changes in the job market, job postings fell by around 15 percent year-on-year since April 2020 and hiring growth rates have declined since February 2020. Unemployment is expected to spike to 5 percent or more.

Under this situation, more and more companies post their job positions online, remote-working also becomes mainstream. On the other hand, job scams are on the rise as well. According to an annual Flewjob survey[1], nearly 20% of job seekers have been the victim of a job scam (up from 13% in 2016).

In response to this new challenge in the COVID-19 pandemic, our team decided to build an intelligent pattern recognition system to help job seekers to avoid fraudulent job ads. This system can learn patterns from training data – Employment Scam Aegean Dataset from the University of the Aegean(http://emscad.samos.aegean.gr/), and be able to predict potential fraudulent job ads.

We began by dealing with the unbalanced dataset, and utilize ensemble learning to learn patterns from structured data as well as unstructured data (Text) simultaneously. After that, a web-based user interface was implemented then the system becomes useable.

We hope this system can help job seekers still be struggling to find jobs during a time of uncertainty.

Project overview

Problem Statement

To learn the pattern from the dataset and make accurate predictions, this project facing these major problems:

Unbalanced data: Over 95% of job ads in the training dataset are legitimate, fraudulent job ads only a small portion.

Structured vs Unstructured Data: A job ad is a combination of structured data (e.g. location, salary range…) and unstructured data (text like the description of a job). Our machine learning algorithm needs to learn these two kinds simultaneously to make unbiased predictions.

Project Objective

The object of our project is to build a web-based system to help job seekers to find potential fraudulent job ads. Users can enter job ads' information through a web page then the system will predict if this ad is fake or not and return the result. Our team is expected to

use and apply various techniques learned in the Graduate Cert in Pattern Recognition Systems and other modules in the MTech course.

Project Scope

The proposed system will include:

(1) A preprocess program that can augment unbalanced raw data. This program should not only be able to process the dataset we selected, but also need to handle subsequent new data because, in reality, fraudulent job ads will always be the minority. That means although we may update the training dataset, the requirement of unbalanced data augmentation will always exist.
(2) Training an ensemble learning classifier. Giving the information about certain job ads, it can predict whether it's fake or not.
(3) Deploy a website with the classifier running on the server and make some web pages as the user interface.

Business value

In recent years, the unemployment rate has increased, and a job is very important for people. Therefore, a lot of lawbreakers may issue some fraudulent job advertisements to defraud. If people are cheated, they will lose not only a job but also money, future, and important personal information. Therefore, we hope to develop a fraudulent job detection system to provide a reference for people who need a job.

Besides, For the operators of the job ads website, our system is also very useful, because They care very much about whether their information is true or not and whether anyone is cheated because of their information.

The Employment Scam Aegean Dataset

The Employment Scam Aegean Dataset (EMSCAD) is a publicly available dataset containing 17,880 real-life job ads that aim at providing a clear picture of the Employment Scam problem to the research community and can act as a valuable testbed for scientists working in the field. The first publication is available online by MDPI Future Internet Journal. EMSCAD records were manually annotated and classified into two categories. More specifically, the dataset contains 17,014 legitimate and 866 fraudulent job ads published between 2012 to 2014.

There are 3 main-type features of our dataset: phrases, boolean values, long text(sentence). this is the distribution of fraudulent feature, as Table 1 showing below.

String	
Name	Description
Title	The title of the job ad entry

Location	Geographical location of the job ad
Department	Corporate department (e.g. sales)
Salary range	Indicative salary range (e.g. $50,000-$60,000)
HTML fragment	
Company profile	A brief company description
Description	The details description of the job ad
Benefits	Enlisted offered benefits by the employer
Requirements	Enlisted requirements for the job opening
Binary	
Telecommuting	True for telecommuting positions
Company logo	True if company logo is present
Questions	True if screening questions are present
Fraudulent	Classification attribute
In balanced	Selected for the balanced dataset
Nominal	
Employment type	Full-type, Part-time, Contract, etc
Required experience	Executive, Entry level, Intern, etc
Required education	Doctorate, Master's Degree, Bachelor, etc
Function	Consulting, Engineering, Research, Sales etc
Industry	Automotive, IT, Health care, Real estate, etc

Table 1. Dataset description

The distribution of positive and negative samples of the dataset is very unbalanced. So the data augmentation for the minority class is necessary. We explain how we use the ESAD to train our system at the finding and discussions.

Tools/Techniques

Tools

Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

Flask offers suggestions but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that makes adding new functionality easy.

Algorithm

EDA

Introduction

We use EDA: easy data augmentation techniques, for boosting performance on our dataset. EDA consists of four simple but powerful operations: synonym replacement, random insertion, random swap, and random deletion.
Refer to the paper, *EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks* by Jason Wei and Kai Zou[1], on five text classification tasks, it shows that EDA improves performance for both convolutional and recurrent neural networks. EDA demonstrates particularly strong results for smaller datasets; on average, across five datasets, training with EDA while using only 50% of the available training set achieved the same accuracy as normal training with all available data. It also performed extensive ablation studies and suggest parameters for practical use.
Frustrated by the measly performance of text classifiers trained on small datasets, the paper tested a number of augmentation operations loosely inspired by those used in computer vision and found that they helped train more robust models. Here, it presents the full details of EDA. For a given sentence in the training set, it randomly chooses and performs one of the following operations[1]. The Table 2 shows the samples after used four operations:

1. **Synonym Replacement (SR):** Randomly choose n words from the sentence that are not stopwords. Replace each of these words with one of its synonyms chosen at random.
2. **Random Insertion (RI):** Find a random synonym of a random word in the sentence that is not a stopword. Insert that synonym into a random position in the sentence. Do this n times.
3. **Random Swap (RS):** Randomly choose two words in the sentence and swap their positions. Do this n times.
4. **Random Deletion (RD):** Randomly remove each word in the sentence with probability *p*.

Operation	Sentence
None	A sad, superior human comedy played out on the back roads of life.
SR	A **lamentable**, superior human comedy played out on the **backward** road of life.
RI	A sad, superior human comedy played out on **funniness** the back roads of life.
RS	A sad, superior human comedy played out on **roads** back **the** of life.
RD	A sad, superior human out on the roads of life.

Table 2. Sentences generated using EDA. SR: synonym replacement. RI: random insertion. RS: random swap. RD: random deletion.

Experimental training result

In the paper, it chooses five benchmark text classification tasks and two network architectures to evaluate EDA.

Overfitting tends to be more severe when training on smaller datasets. By conducting experiments using a restricted fraction of the available training data, it shows that EDA has more significant improvements for smaller training sets. it runs both normal training and EDA training for the following training set fractions (%): {1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}. Figure 1(a)-(e) shows performance with and without EDA for each dataset, and 1(f) shows the averaged performance across all datasets. The best average accuracy without augmentation, 88.3%, was achieved using 100% of the training data. Models trained using EDA surpassed this number by achieving an average accuracy of 88.6% while only using 50% of the available training data.[1]

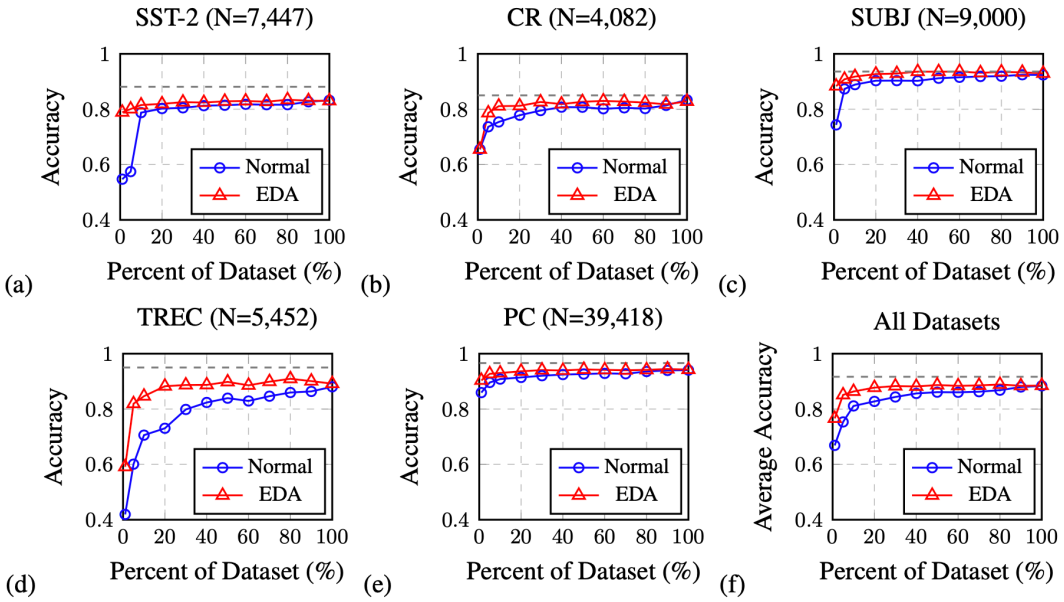

Figure 1.Performance on benchmark text classification tasks with and without EDA, for various dataset sizes used for training. For reference, the dotted grey line indicates best performances from Kim (2014) for SST-2, CR, SUBJ, and TREC, and Ganapathibhotla (2008) for PC.

The challenge of working with unbalanced datasets is that most machine learning techniques will not learn the knowledge of the minority class well, and turn have poor performance on the minority class. although typically the performance of the minority class

is most important. The distribution of positive and negative samples of the dataset is very unbalanced. So the data augmentation for the minority class is necessary.[1]

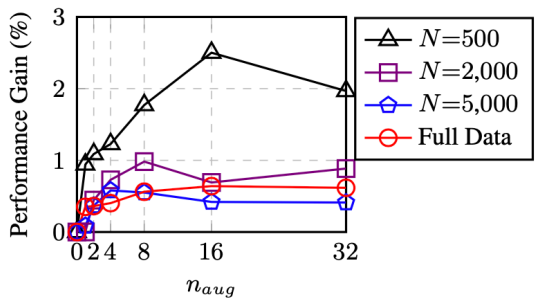

Figure 2. Average performance gain of EDA across five text classification tasks for various training set sizes. n_{aug} is the number of generated augmented sentences per original sentence.

Appling in EMSCAD

According to the paper, it depends on the size of the training data. When the training data is very small, the model is easier tooverfit , so it is recommended to generate more data-enhanced samples. When the training data is very large, it may not be helpful to add a large number of data enhancement samples, because the model itself may have been able to generalize. it turned out that it is not that the more sentences added, the better the performance is, it will reach a peak and then fall.

As it is shown in Figure 3, the original dataset is an unbalanced dataset. We apply EDA to increase the smaller part of the dataset. After increasing it by 10 times, it is shown in Figure 4.

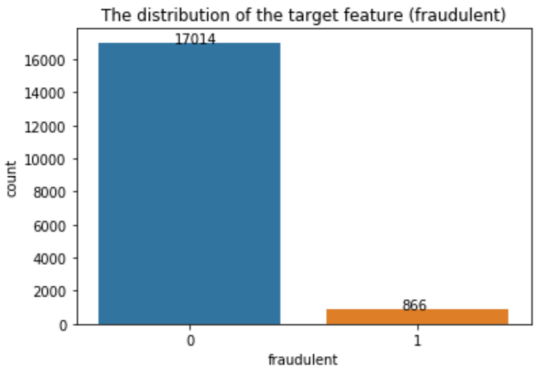

Figure 3. the distribution of the original EMSCAD (fraudulent)

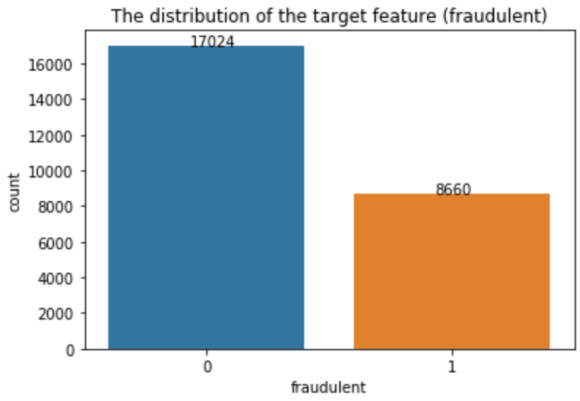

Figure 4. After 10-times EDA, the distribution of the original EMSCAD (fraudulent)

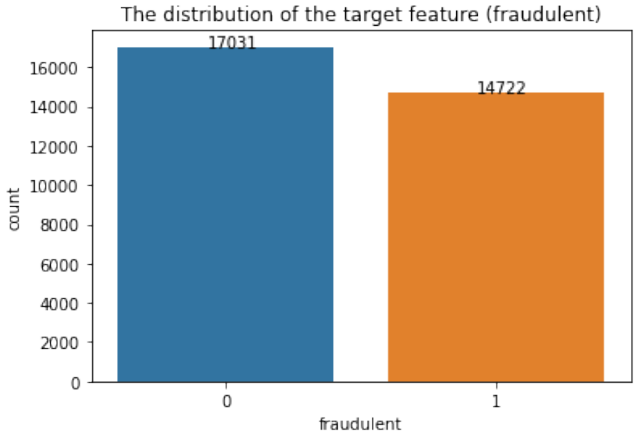

Figure 5. After 18-times EDA, the distribution of the original EMSCAD (fraudulent)

Finally, we chose the 18-times EDA Figure 5 as the final training dataset, as it performed better than others.

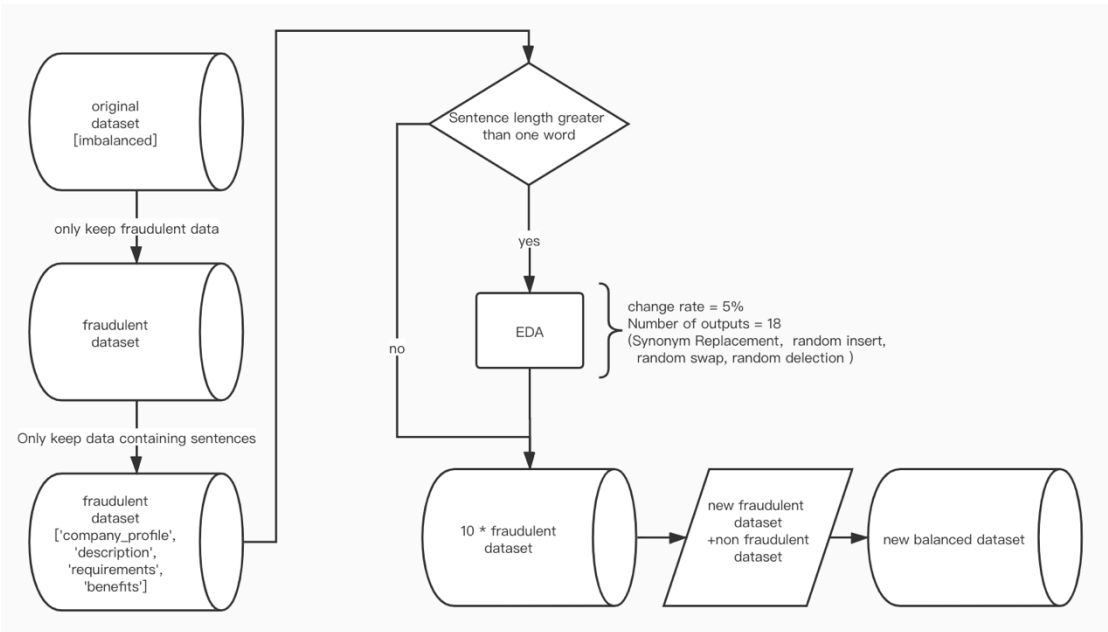

Figure 6.the workflow of the EDA

Global Vectors for Word Representation

Introduction

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Semantic vector space models of language represent each word with a real-valued vector. Most word vector methods rely on the distance or angle between pairs of word vectors as the primary method for evaluating the intrinsic quality of such a set of word representations. Recently, Mikolov et al. (2013c) introduced a new evaluation scheme based on word analogies that probe the finer structure of the word vector space by examining not the scalar distance between word vectors, but rather their various dimensions of difference. For example, the analogy “king is to queen as man is to woman” should be encoded in the vector space by the vector equation king − queen = man − woman. This evaluation scheme favors models that produce dimensions of meaning, thereby capturing the multi-clustering idea of distributed representations (Bengio, 2009). The two main model families for learning word vectors are: 1) global matrix factorization methods, such as latent semantic analysis (LSA) (Deerwester et al., 1990) and 2) local context window methods, such as the skip-gram model of Mikolov et al. (2013c). Currently, both families suffer significant drawbacks. While methods like LSA efficiently leverage statistical information, they do relatively poorly on the word analogy task, indicating a sub-optimal vector space structure. Methods like skip-gram may do better on the analogy task, but they poorly utilize the statistics of the corpus since they train on separate local context windows instead of on global co-occurrence counts.

The paper - *GloVe: Global Vectors for Word Representation*, wrote by Jeffrey Pennington, Richard Socher. it analyzes the model properties necessary to produce linear directions of meaning and argue that global log-bilinear regression models are appropriate for doing so. We propose a specific weighted least squares model that trains on global word-word co-occurrence counts and thus makes efficient use of statistics. The model produces a word vector space with meaningful substructure, as evidenced by its state-of-the-art performance of 75% accuracy on the word analogy dataset. it also demonstrates that the methods outperform other current methods on several word similarity tasks, and also on a common named entity recognition (NER) benchmark. [3]

Performance in EMSCAD

we download the pre-trained word vectors on github, choosing Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download) apply on the EMSCAD.

Logistic regression

Introduction

In statistics, the logistic model is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object that is detected in the image would be assigned a probability between 0 and 1, with a sum of one.

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling; the function that converts log-odds to probability is the logistic function, hence the name. The unit of measurement for the log-odds scale is called a logit, from the logistic unit, hence the alternative names. Analogous models with a different sigmoid function instead of the logistic function can also be used, such as the probit model; the defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given outcome at a constant rate, with each independent variable having its own parameter; for a binary dependent variable, this generalizes the odds ratio.

In a binary logistic regression model, the dependent variable has two levels (categorical). Outputs with more than two values are modeled by multinomial logistic regression and, if the multiple categories are ordered, by ordinal logistic regression (for example the proportional odds ordinal logistic model). The logistic regression model itself simply models probability of output in terms of input and does not perform statistical classification (it is not a classifier), though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other; this is a common way to make a binary classifier. The coefficients are generally not computed by a closed-form expression, unlike linear least squares. The logistic regression as a general statistical model was originally developed and popularized primarily by Joseph Berkson, beginning in Berkson (1944), where he coined "logit". [4]

Definition of the logistic function

An explanation of logistic regression can begin with an explanation of the standard logistic function. The logistic function is a sigmoid function, which takes any real input t, $(t \in \mathbb{R})$,

and outputs a value between zero and one; for the logit, this is interpreted as taking input log-odds and having output probability. The standard logistic function $\sigma: \mathbb{R} \to (0,1)$ is defined as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{e^t}{1 + e^{-t}}$$

A graph of the logistic function on the t-interval (−6,6) is shown in Figure 7.

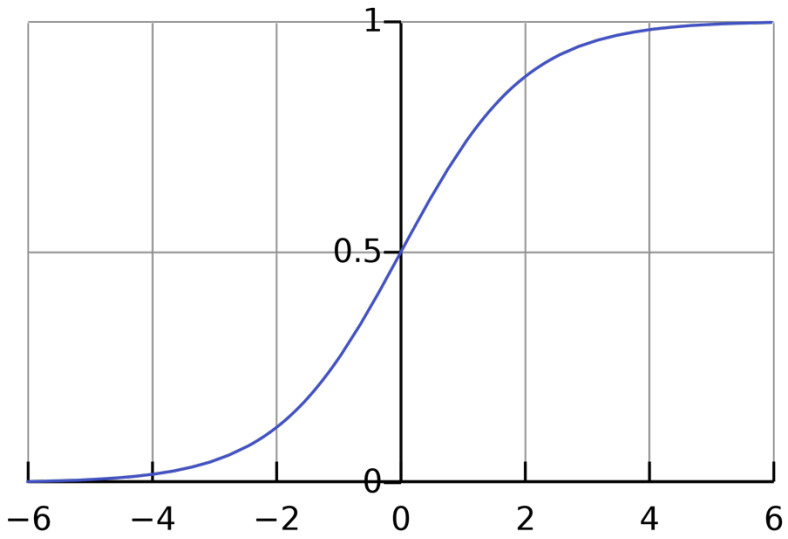

Figure 7. The standard logistic function $\sigma(t)$; note that $\sigma(t) \in (0,1)$ for all t.

Let us assume that t is a linear function of a single explanatory variable x (the case where t is a linear combination of multiple explanatory variables is treated similarly). We can then express t as follows:

$$t = \beta 0 + \beta 1 x$$

And the general logistic function $p: \mathbb{R} \to (0,1)$ can now be written as:

$$p(x) = \sigma(t) = \frac{1}{1 + e^{-(\beta 0 + \beta 1 x)}}$$

In the logistic model, $p(x)$ is interpreted as the probability of the dependent variable Y equaling a success/case rather than a failure/non-case. It's clear that the response variables Yi are not identically distributed: $P(Yi = 1|X)$ differs from one data point Xi to another, though they are independent given design matrix X and shared parameters β.

Interpretation of these terms:

In the above equations, the terms are as follows:

- $p(x)$ is the probability that the dependent variable equals a case, given some linear combination of the predictors. The formula for $p(x)$ illustrates that the probability of the dependent variable equaling a case is equal to the value of the logistic function of the linear regression expression. This is important in that it shows that the value of the linear regression expression can vary from negative to positive infinity and yet, after transformation, the resulting expression for the probability $p(x)$ ranges between 0 and 1.

- $\beta 0$ is the intercept from the linear regression equation (the value of the criterion when the predictor is equal to zero).
- $\beta 1x$ is the regression coefficient multiplied by some value of the predictor.
- Base e denotes the exponential function.

Examples

Let us try to understand logistic regression by considering a logistic model with given parameters, then seeing how the coefficients can be estimated from data. Consider a model with two predictors, $\mathrm{x1}$ and $\mathrm{x2}$, and one binary (Bernoulli) response variable Y, which we denote $p = P(Y = 1)$. We assume a linear relationship between the predictor variables and the log-odds of the event that $Y = 1$. This linear relationship can be written in the following mathematical form (where ℓ is the log-odds, b is the base of the logarithm, and βi are parameters of the model):

$$\ell = log_b \frac{p}{1-p} = \beta 0 + \beta 1x1 + \beta 2x2$$

We can recover the odds by exponentiating the log-odds:

$$\frac{p}{1-p} = b^{\beta 0+\beta 1x1+\beta 2x2}$$

By simple algebraic manipulation, the probability that $Y = 1$ is

$$p = \frac{b^{\beta 0+\beta 1x1+\beta 2x2}}{b^{\beta 0+\beta 1x1+\beta 2x2}+1} = \frac{1}{1+b^{-(\beta 0+\beta 1x1+\beta 2x2)}}$$

The above formula shows that once βi is fixed, we can easily compute either the log-odds that $Y = 1$ for a given observation, or the probability that $Y = 0$ for a given observation. The main use-case of a logistic model is to be given an observation$(x1, x2)$, and estimate the probability p that $Y = 1$. In most applications, the base b of the logarithm is usually taken to be e. However, in some cases, it can be easier to communicate results by working in base 2, or base 10.

We consider an example with$b = 10$, and coefficients $\beta 0 = -3$, $\beta 1 = 1$, and $\beta 2 = 2$. To be concrete, the model is

$$log_{10} \frac{p}{1-p} = \ell = -3 + x1 + 2x2$$

Where p is the probability of the event that $Y = 1$.

This can be interpreted as follows:

- $\beta 0 = -3$ is the y-intercept. It is the log-odds of the event that $Y = 1$, when the predictors $x1 = x2 = 0$. By exponentiating, we can see that when $x1 = x2 = 0$ the odds of the event that $Y = 1$ are 1-to-1000, or 10^{-3}. Similarly, the probability of the event that $Y = 1$ when $x1 = x2 = 0$ can be computed as $\frac{1}{1000+1} = 1/1001$.
- $\beta 1 = 1$ means that increasing $x1$ by 1 increases the log-odds by 1. So if $x1$ increases by 1, the odds that $Y = 1$ increase by a factor of 10^1. Note that the probability of $Y = 1$ has also increased, but it has not increased by as much as the odds have increased.

- $\beta 2 = 2$ means that increasing $x2$ by 1 increases the log-odds by 2. So if $x2$ increases by 1, the odds that $Y = 1$ increase by a factor of 10^2 Note how the effect of $x2$ on the log-odds is twice as great as the effect of $x1$, but the effect on the odds is 10 times greater. But the effect on the probability of $Y = 1$ is not as much as 10 times greater, it's only the effect on the odds that is 10 times greater.[4]

In order to estimate the parameters βi from data, one must do logistic regression.

Discussion

Logistic regression can be binomial, ordinal or multinomial. Binomial or binary logistic regression deals with situations in which the observed outcome for a dependent variable can have only two possible types, "0" and "1" (which may represent, for example, "dead" vs. "alive" or "win" vs. "loss"). Multinomial logistic regression deals with situations where the outcome can have three or more possible types (e.g., "disease A" vs. "disease B" vs. "disease C") that are not ordered. Ordinal logistic regression deals with dependent variables that are ordered.

In binary logistic regression, the outcome is usually coded as "0" or "1", as this leads to the most straightforward interpretation. If a particular observed outcome for the dependent variable is the noteworthy possible outcome (referred to as a "success" or an "instance" or a "case") it is usually coded as "1" and the contrary outcome (referred to as a "failure" or a "noninstance" or a "noncase") as "0". Binary logistic regression is used to predict the odds of being a case based on the values of the independent variables (predictors). The odds are defined as the probability that a particular outcome is a case divided by the probability that it is a noninstance.

Like other forms of regression analysis, logistic regression makes use of one or more predictor variables that may be either continuous or categorical. Unlike ordinary linear regression, however, logistic regression is used for predicting dependent variables that take membership in one of a limited number of categories (treating the dependent variable in the binomial case as the outcome of a Bernoulli trial) rather than a continuous outcome. Given this difference, the assumptions of linear regression are violated. In particular, the residuals cannot be normally distributed. In addition, linear regression may make nonsensical predictions for a binary dependent variable. What is needed is a way to convert a binary variable into a continuous one that can take on any real value (negative or positive). To do that, binomial logistic regression first calculates the odds of the event happening for different levels of each independent variable, and then takes its logarithm to create a continuous criterion as a transformed version of the dependent variable. [4]

Performance in EMSCAD

Using the EMSCAD to train our model by the logistics regression. The performance of the logistics regression using the EMSCAD is as Table 3 below:

Precision:	0.9651022864019254

Recall:	precision	recall	f1-score
F	0.97	0.96	0.96
R	0.96	0.97	0.97
accuracy			0.97
macro avg	0.97	0.97	0.97
weighted avg	0.97	0.97	0.97

Table 3. performance of the LR model in EMSCAD

Attention mechanism

Introduction

Attention is a way of obtaining a weighted sum of the vector representations of a layer in a neural network model. It is used in diverse tasks ranging from machine translation (Luong et al., 2015), language modeling to image captioning (Xu et al., 2015), and object recognition (Ba et al., 2014). Apart from substantial performance benefit (Vaswani et al., 2017), attention also provides interpretability to neural models (Wang et al., 2016; Lin et al., 2017; Ghaeini et al., 2018) which are usually criticized for being black-box function approximators.

There has been substantial work on understanding attention in neural network models. On the one hand, there is work on showing that attention weights are not interpretable and altering them does not significantly affect the prediction (Jain & Wallace, 2019; Serrano & Smith, 2019). While on the other hand, some studies have discovered how attention in neural models captures several linguistic notions of syntax and coreference (Vig & Belinkov, 2019; Clark et al., 2019; Tenney et al., 2019). Amid such contrasting views arises a need to understand the attention mechanism more systematically. In this paper, we attempt to fill this gap by giving a comprehensive explanation that justifies both kinds of observations.[5]

Self-attention-based models

There are three categories of tasks on self-attention-based models. For single and pair sequence tasks, we fine-tune pre-trained BERT model[7] on the downstream task. In pair sequence tasks, instead of independently encoding each text, we concatenate both separated by a delimiter and pass it to BERT model. Finally, the embedding corresponding to token is fed to a feed-forward network for prediction. For neural machine translation, we use Transformer model proposed by Vaswani et al. (2017)[6] with base configuration.

Transformer model

Introduction

Recurrent neural networks, long short-term memory and gated recurrent neural networks, in particular, have been firmly established as state-of-the-art approaches in sequence modeling and transduction problems such as language modeling and machine translation. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures.

Recurrent models typically factor computation along with the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previously hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks and conditional computation, while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences. In all but a few cases, however, such attention mechanisms are used in conjunction with a recurrent network.

In the paper, *Attention Is All You Need*, the authors propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output.

Model architecture

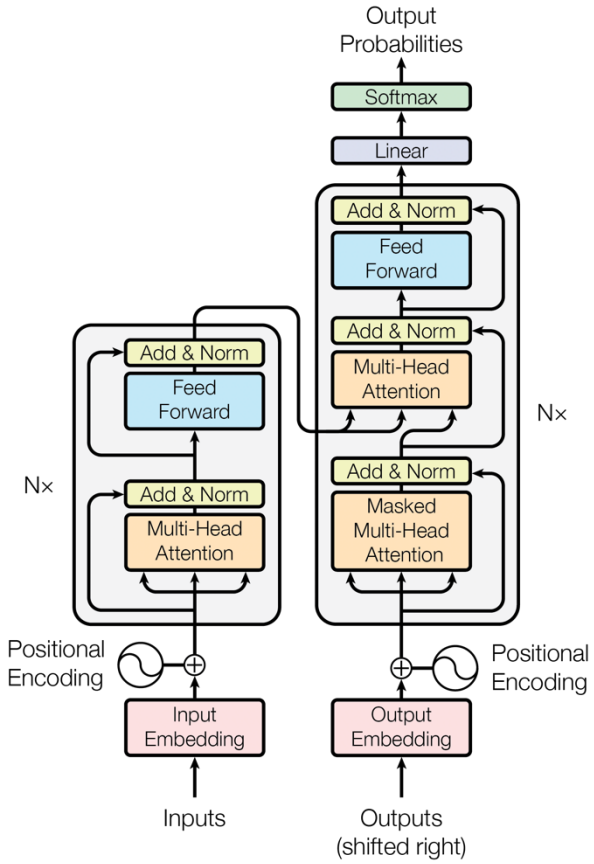

Figure 8. The Transformer - model architecture.

Most competitive neural sequence transduction models have an encoder-decoder structure. Here, the encoder maps an input sequence of symbol representations $(x1, \ldots, xn)$ to a sequence of continuous representations $z = (z1, \ldots, zn)$. Given z, the decoder then generates an output sequence $(y1, \ldots, ym)$ of symbols one element at a time. At each step the model is auto regressive, consuming the previously generated symbols as additional input when generating the next.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 8, respectively.

Encoder and Decoder Stacks

- **Encoder:** The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [10] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ is the function implemented by the sub-layer itself. To facilitate these

residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $dmodel = 512$.

- **Decoder:** The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Scaled Dot-Product Attention

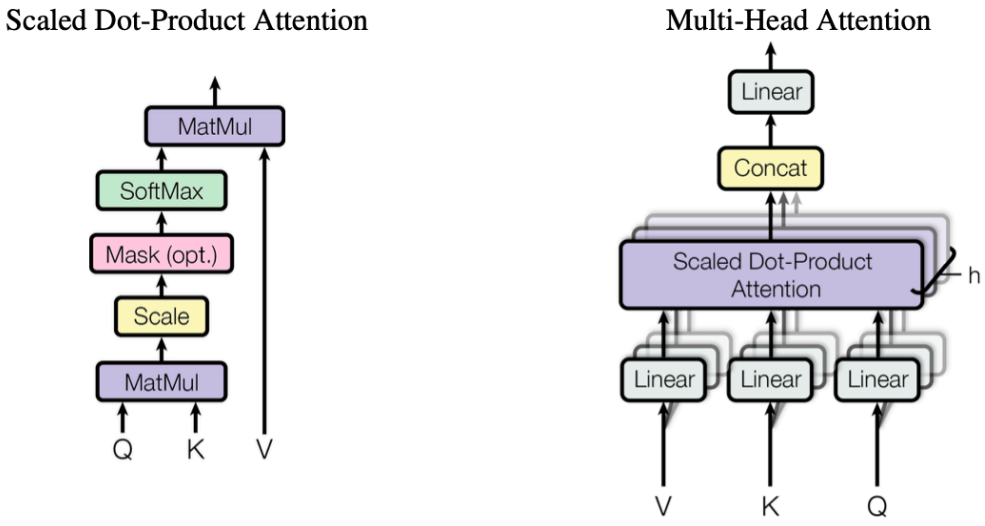

Figure 9.(left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

We call our particular attention "Scaled Dot-Product Attention" (Figure 9). The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V . We compute the matrix of outputs as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

The two most commonly used attention functions are additive attention, and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has 41extremely small gradients. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

Multi-Head Attention

Instead of performing a single attention function with $dmodel$-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 10.

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$MultiHead(Q, K, V) = Concat(head1, \ldots, headh)W^O$$

$$\text{where } headi = Attention(QWi^Q, KWi^K, VWi^V)$$

Where the projections are parameter matrices $Wi^Q \in \mathbb{R}^{d_{model} \times dk}$, $Wi^K \in \mathbb{R}^{d_{model} \times dk}$ and $W^O \in \mathbb{R}^{hdv \times d_{model}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $dk = dv = d_{model}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

BERT

Introduction

The full name of Bert is bidirectional encoder representation from Transformer model, that is, the encoder of the bidirectional transformer, because the decoder cannot obtain the information to be predicted. The main innovation of the model is pre-trained method, which uses masked LM and next sentence prediction to capture the representation of words and sentences respectively. That is, the Bert is a pre-trained model based on transformer.
We introduce BERT and its detailed implementation in this section. There are two steps in our framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For finetuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters. The question-answering example in Figure 1 will serve as a running example for this section.
A distinctive feature of BERT is its unified architecture across different tasks. There is minimal difference between the pre-trained architecture and the final downstream architecture.

Model Architecture

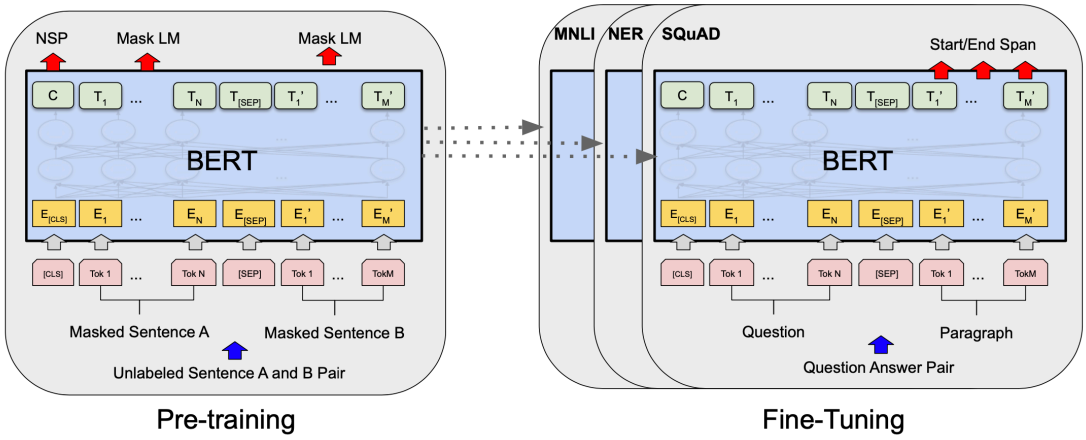

Figure 10. Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Vaswani et al. (2017). In this work, the authors denote the number of layers (i.e., Transformer blocks) as L, the hidden size as A.
BERT$_{BASE}$ was chosen to have the same model size as OpenAI GPT for comparison purposes. Critically, however, the BERT Transformer uses bidirectional self-attention, while the GPT Transformer uses constrained self-attention where every token can only attend to context to its left.[7]

Input/Output Representations

To make BERT handle a variety of down-stream tasks, our input representation is able to unambiguously represent both a single sentence and a pair of sentences (e.g., ⟨ Question, Answer ⟩) in one token sequence. Throughout this work, a "sentence" can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. A "sequence" refers to the input token sequence to BERT, which may be a single sentence, or two sentences packed together.
The authors use WordPiece embeddings (Wu et al., 2016) with a 30,000-token vocabulary. The first token of every sequence is always a special classification token ([CLS]). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks. Sentence pairs are packed together into a single sequence. The authors differentiate the sentences in two ways. First, the authors separate them with a special token ([SEP]). Second, the authors add a learned embedding to every token indicating whether it belongs to sentence A or sentence B. As shown in Figure 1, we denote input embedding as E, the final hidden vector of the special [CLS] token as $C \in \mathbb{R}^H$, and the final hidden vector for the i^{th} input token as $T_i \in \mathbb{R}^H$.
For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings. A visualization of this construction can be seen in Figure 2.[7]

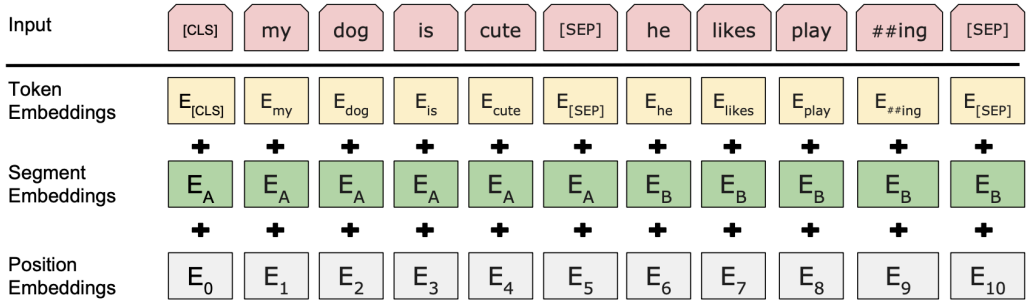

Figure 11. BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Performance in EMSCAD

We use the simpleclassification python library. This library is based on the Transformers library by HuggingFace. Simple Transformers lets us quickly train and evaluate

Transformer models. Only 3 lines of code are needed to initialize a model, train the model, and evaluate a model, support Binary Classification.
Train and Evaluation data needs to be in a Pandas Dataframe of two columns. The first column is the text with type str, and the second column is the label with type int. it performs predictions on a list of text.
Using the EMSCAD to train our model by the logistics regression, we can see the performance of the BERT using the EMSCAD is as the table below:

Accuracy is :	0.9849465705683419		
F1 Score is :	0.9892016618728028		
	precision	recall	f1-score
0	0.99	0.99	0.99
1	0.99	0.97	0.98
Accuracy			0.98
macro avg	0.96	0.97	0.99
weighted avg	0.99	0.99	0.99

Table 4. performance of the BERT model in EMSCAD

How BERT leverage attention mechanism and transformer to learn word contextual relations

GA ensemble

Introduction

Many simple and complex methods have been developed to solve the classification problem. Boosting is one of the best-known techniques for improving the accuracy of classifiers. However, boosting is prone to overfitting with noisy data and the final model is difficult to interpret. Some boosting methods, including AdaBoost, are also very sensitive to outliers.
In the article, *GA-Ensemble: a genetic algorithm for robust ensembles*, the authors propose a new method, GA-Ensemble, which directly solves for the set of weak classifiers and their associated weights using a genetic algorithm.[8]

Performance in EMSCAD

Following is the training and testing flow of GA-Ensemble system.

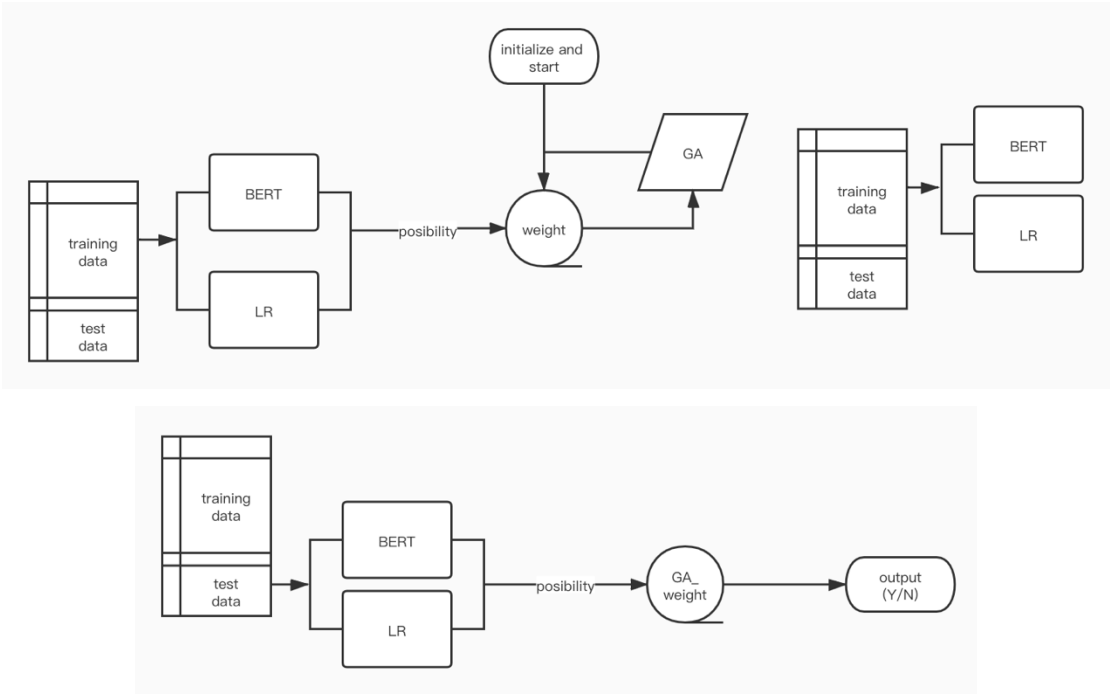

Figure 12. the training and testing flow of GA-Ensemble system

in the training, we set $num\_generations = 1000$, then we got the $Best\ solution$: [[[1.17620871 1.18895564 0.62855357 1.2274774]]], when $fitness = [0, 0, 0, 0, 0, 0, 4, 0]$.

Web design

Frontend

We decide to build our system as a website that has 5 pages including start page, form page, loading page, result page, and error report page.

The start page is just a guide page, which tell users the name of the system. When user clicks the button below, it will enter to next page---form page.

The form page ask user to fill in the job information that the model need to make decision. It is unnecessary for users to complete all the questions of the form and users can just write down answers you know from job information. Of course, the more information the system gets, the more accurate judgement it will make, so we recommend users to complete as many questions as they can. If you want to test the system first, there are three example cases for users and users just click one of three buttons below the form title, the form will be completed automatically.

As user finishes the form and submit the job information, it will be sent to the backend and be processed by ensemble model, which will make user wait for the result in the loading page. After a few minutes of waiting, the result that system gives will present in the result page. If the result shows that the job is a real job, the page will show the result and confidence of this result. Confidence means how confident the system is in the result, which is a percentage value. If the result shows that the job is fake, except those two things that

are mentioned, system will not only show some link for teaching users how to classify whether a job is a real job or not, but also has a button for user to go to the error report page.
In the error report page, it will highlight some suspicious sentences in red color to tell users those sentences may be the key sentences that make the system consider this job as a fake job.
When users finish reading error report, they can back to the start page to do another job fraudulence classification.

Backend

We choose flask as our web framework to build the backend system. Flask provides us with tools, libraries and technologies that allow us to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.
There are two significant functions of the backend:

- Get the job information from form page and build information into a special data structure for model to deal with, finally display the result.
- Highlight the suspicious sentences of the fake job in order to let users know why system thinks it is a fake job.

The backend will get the job information from form page. Firstly, system put all text values together as a parameter for bert model, a pretrained model that has ability to understand the context of text. Then, this information will be also converted to DataFrame construction for logistic regression model. The predicted result of two models will be put in the genetic algorithm to get the predicted result and confidence which will display on the result page. Also, the system will get suspicious words list from the algorithm.
If model predicts that it is a fake job, the backend will highlight the suspicious sentence. As we have mentioned above, system gets a list of suspicious words, so the system will match every word in every sentence with those words in the list. Finally, we can know how many suspicious words in every sentence and if the number of those words in a sentence is larger than a threshold, this sentence will be considered as a suspicious sentence. After finishing this work, the processed text will be sent to the error report page.

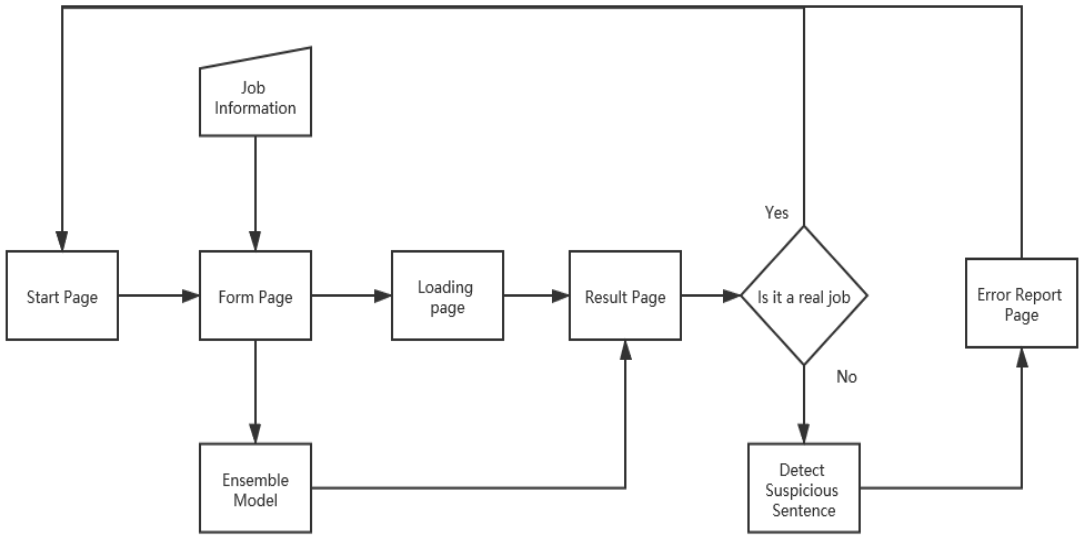

Figure 13. System design

System performance

Show the performance of the logistic regression and BERT Table 3 algorithms in performance of the LR model in EMSCAD and Table 4 performance of the BERT model in EMSCAD. Then, we got the GA weight $Best\ solution$: $[1.17620871, 1.18895564, 0.62855357, 1.2274774]$.

Findings and discussions

To solve the unbalanced dataset

What will the unbalanced cause?
Unbalanced datasets are prevalent in a multitude of fields and sectors, and of course, this includes financial services. From fraud to non-performing loans, data scientists come across them in many contexts. The challenge appears when machine learning algorithms try to identify these rare cases in rather big datasets. Due to the disparity of classes in the variables, the algorithm tends to categorize into the class with more instances, the majority class, while at the same time giving the false sense of a highly accurate model. Both the inability to predict rare events, the minority class, and the misleading accuracy detracts from the predictive models we build.
The challenge of working with unbalanced datasets is that most machine learning techniques will not learn the knowledge of the minority class well and turn have poor performance in the minority class. although typically the performance of the minority class is most important.

The class imbalance problem between the majority and minority is frustrating, but not unexpected. We will now discuss the main techniques and methods available when dealing with this type of data.

1. Resample the dataset:

- **Under-sampling:** The idea is to reduce the ratio of instances in the majority and minority levels. We can randomly select observations in the desired ratio — 50/50, 60/40 in a binary case, or 40/30/30 if we have three classes. In this case, taking a random sample without replacement would be enough. We can carry out an informed under-sampling by looking at the distribution of the data and selecting the observations to discard. In this last case, we can first try using a clustering technique or k-NN (k-nearest neighbors algorithm) to obtain a down-sampled dataset. This dataset includes observations of every natural group of data inside the majority class. This way, we retain all the underlying information in the sample. Otherwise, random down-sampling may select all of one type of observation, and we will lose valuable information from the sample. During the resampling, we can try different ratios as each class does not have to contain the same number of observations. We can see it clearly in Figure 14.

Figure 14. Resample the dataset

- **Overs-sampling:** We focus on the minority class based on the available data. Some algorithms that allow for this include Variational Autoencoders (VAE), SMOTE (Synthetic Minority Over-sampling Technique) or MSMOTE (Modified Synthetic Minority Over-Sampling Technique) or EDA (Easy Data Augmentation Techniques).

  For SMOTE: we select some observations (20, 30 or 50, the number is changeable) and use a distance measure to synthetically generate a new instance with the same "properties" for the available features. Analyzing one feature at a time, SMOTE takes the difference between an observation and its nearest neighbor. It multiplies the difference with a random number between zero and one. Then, it identifies a new point by adding the random number to the feature. This way, SMOTE does not copy observations and instead creates a new, synthetic one.

2. Collect more data from the minority class

This option appears trivial, but it solves the problem when it is applicable.

3. Use the "adequate" correct algorithm

Some algorithms are more robust than others. A mastery of the theory behind each algorithm will help us understand their strengths and weaknesses in various situations. Remember, machine learning algorithms are chosen based on the input data and learning task at hand.

4. Change our approach

Instead of building a classifier, sometimes it is beneficial to change our approach and the scope; one option would be to analyze our data from the 'anomaly detection' point of view. Then, we can apply from 'One Class SVM' to 'Local Outlier Factor (LOF)' algorithms.

5. Use penalized models

Many algorithms have their own penalized version. Usually, algorithms treat all misclassifications the same, so the idea is to penalize misclassifications from the minority class more than the majority. Mistakes made during training carry an additional cost (that is why they are called cost-sensitive classifiers), but in theory, these penalties help the model improve the attention given to the minority class. Sometimes, the penalties are called weights. Achieving the correct matrix of penalties can be difficult and sometimes does not do much to improve results, so try several schemas until we find the one that works best for our situation.

In this project we use a way of oversampling, called EDA, to make our dataset balanced.

Fine-tune different models

In this project, we try four algorithms on the dataset, EMSCAD. Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.

The four models contain four algorithms random forest, naïve bayes, logistic regression and BERT. We divided the same range of training and test dataset for those four models. And, after finetuning them, the performance of these four models are different from each other. As we can see the results shown on the tools/techniques and appendix of the project report. Even though, the goal of any machine learning problem is to find a single model that will best predict our wanted outcome. Rather than making one model and hoping this model is the best/most accurate predictor we can make, ensemble methods take a myriad of models into account, and average those models to produce one final model.

To make the model visualized

How BERT leverage attention mechanism and transformer to learn word contextual relations? Underlying BERT is the Transformer model, a type of neural network that accepts a sequence as input (e.g. a sequence of words) and produces some output (e.g. sentiment prediction). Unlike traditional recurrent networks such as LSTMs, which process each sequence element in turn, the Transformer processes all elements simultaneously by forming direct connections between individual elements through a process known as attention. Not only does this enable greater parallelization, but it also results in higher accuracy across a range of tasks.

The above visualization shows one attention mechanism within the model. BERT actually learns multiple attention mechanisms, called heads, which operate in parallel to one another. As we'll see shortly, multi-head attention enables the model to capture a broader range of relationships between words than would be possible with a single attention mechanism.[13]
BERT also stacks multiple layers of attention, each of which operates on the output of the layer that came before. Through this repeated composition of word embeddings, BERT is able to form very rich representations as it gets to the deepest layers of the model.
We tried to extract the self-attention scores according to [CLS] token in the last hidden layer of the BERT model. After added and modified some code on the original simpleclassification library and transformer python documents. Then, we can make the model return the wordlist refers to the attention score. those words highlighted is the suspicious part of scam job ads or a real job. and the wordlist is not the best way for our human-being to understand the meaning of the sentences. So, we choose to highlight the whole sentence which contains the word in the wordlist.
According to the knowledge above, now we can highlight the scam part of a job advertisement.

Appendix

Tentative algorithm

We tried those flowing techniques in the EMSCAD, the average performance of these two algorithms was about 82%. It is quite low, so we finally give them up.

TF-IDF

Introduction

the TF-IDF is a kind of preprocessing method we chose for the naïve bayes algorithm.
Typically, the TF-IDF weight is composed of two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

- **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

$$TF(t) = (Number\ of\ times\ term\ t\ appears\ in\ a\ document) / (Total\ number\ of\ terms\ in\ the\ document).$$

- **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scaling up the rare ones, by computing the following:

$$IDF(t) = log\_e(Total\ number\ of\ documents / Number\ of\ documents\ with\ term\ t\ in\ it).$$

Appling in EMSCAD

Figure 15. Word-cloud in fraudulent job

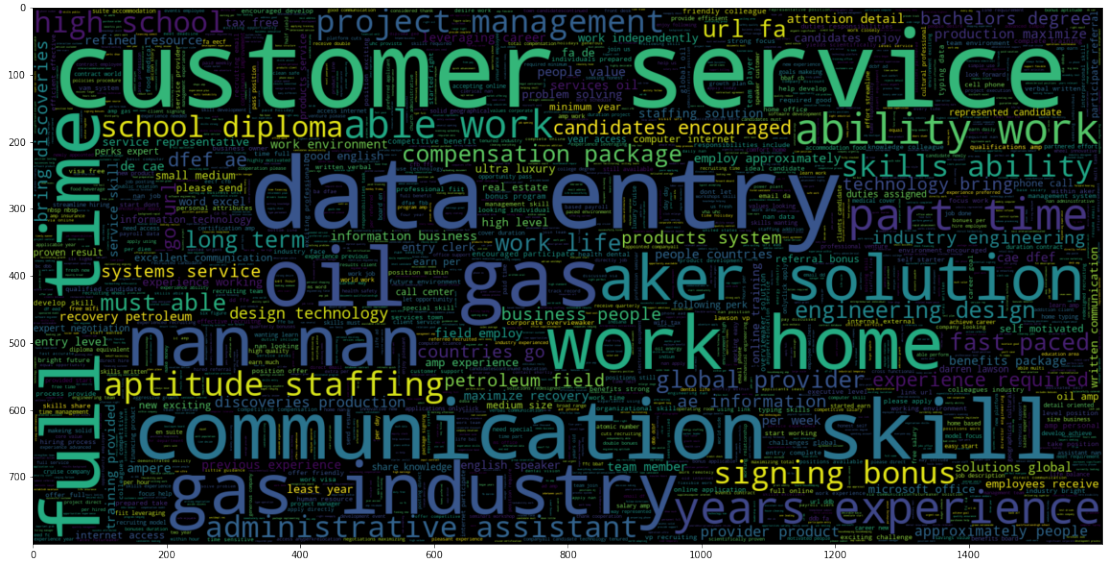

Figure 16. Word-cloud in real job

Random forest

Introduction

Recently there has been a lot of interest in “ensemble learning” — methods that generate many classifiers and aggregate their results. Breiman (2001) proposed random forests, which add an additional layer of randomness to bagging. In addition to constructing each tree using a different bootstrap sample of the data, random forests change how the classification or regression trees are constructed. In standard trees, each node is split using the best split among all variables. In a random forest, each node is split using the best among a subset of predictors randomly chosen at that node. This somewhat counterintuitive strategy turns out to per-form very well compared to many other classifiers, including discriminant analysis, support vector machines and neural networks, and is robust against overfitting (Breiman, 2001). In addition, it is very user-friendly in the sense that it has only two parameters (the number of variables in the random subset at each node and the number of trees in the forest) and is usually not very sensitive to their values.

The algorithm

The random forests algorithm (for both classification and regression) is as follows:

1. Draw n_{tree} bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an unpruned classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample m_{try} of the predictors and choose the best split from among those variables. (Bagging can be thought of as the special case of random forests obtained when $m_{try} = p$, the number of predictors.)
3. Predict new data by aggregating the predictions of the n_{tree} trees (i.e., majority votes for classification, the average for regression).

An estimate of the error rate can be obtained, based on the training data, by the following:

1. At each bootstrap iteration, predict the data not in the bootstrap sample (what Breiman calls “out-of-bag”, or OOB, data) using the tree grown with the bootstrap sample.
2. Aggregate the OOB predictions. (On average, each data point would be out-of-bag around 36% of the times, so aggregate these predictions.) Calculate the error rate, and call it the OOB estimate of error rate.

Performance in EMSCAD

Using the EMSCAD to train our model by the random forests, we show how it work here:

```
## Lets build our hyperparameter search grid
## since we have a large space of hyperparamter we will use randomisedSearchCV
# Number of trees in random forest
>       n_estimators = [int(x) for x in np.linspace(start = 10, stop = 800, num = 5)]
```

```
# Maximum number of levels in tree
>       max_depth = [int(x) for x in np.linspace(10, 50, num = 5)]
>       max_depth.append(None)
# Minimum number of samples required to split a node
>       min_samples_split = [2, 5]
# Minimum number of samples required at each leaf node
>       min_samples_leaf = [1, 2]
# Create the random grid
>       random_grid = {'n_estimators': n_estimators,
                       'max_depth': max_depth,
                       'min_samples_split': min_samples_split,
                       'min_samples_leaf': min_samples_leaf}
>       rf = RandomForestClassifier()
>       rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 5, cv = 4,scoring = 'f1')
>       rf_random.fit(X, Y)
```

Average of the best f1-score in various folds during cross validation = 0.8329387433584526. The best parameters found during k-fold cross validation is = {'n_estimators': 405, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 20}

Average of the best f1-score：	0.8329387433584526
n_estimators	405
min_samples_split	2
min_samples_leaf	1
max_depth	20

Table 5. Using random forest on EMSCAD

Naïve bayes

Introduction

The naive Bayes classifier greatly simplifies learning by assuming that features are independent given class. Although independence is generally a poor assumption, in practice naive Bayes often competes well with more sophisticated classifiers.

The algorithm

Performance in EMSCAD

Using the EMSCAD to train our model by the Naïve bayes, we show how it work here:

```
>       from sklearn.metrics import roc_auc_score
>       from sklearn.naive_bayes import MultinomialNB
```

```
>    alpha_set=[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
>    Train_AUC_BOW = []
>    CrossVal_AUC_BOW = []
>    for i in alpha_set:
    naive_b=MultinomialNB(alpha=i)
    naive_b.fit(tv_train_reviews, y_train)
    Train_y_pred =    naive_b.predict(tv_train_reviews)
    Train_AUC_BOW.append(roc_auc_score(y_train,Train_y_pred))
    CrossVal_y_pred =    naive_b.predict(tv_cross_reviews)
     CrossVal_AUC_BOW.append(roc_auc_score(y_cross,CrossVal_y_pred))
>    from numpy import math
>    Alpha_set=[]
>    for i in range(len(alpha_set)):
    Alpha_set.append(math.log(alpha_set[i]))
>    optimal_alpha=alpha_set[CrossVal_AUC_BOW.index(max(CrossVal_AUC_BOW))]
>    Classifier1=MultinomialNB(alpha=optimal_alpha)
>    Classifier1.fit(tv_train_reviews, y_train)
>    auc_train_bow = roc_auc_score(y_train,Classifier1.predict(tv_train_reviews))
>    print ("AUC for Train set", auc_train_bow)
>    auc_test_bow = roc_auc_score(Y_test,Classifier1.predict(tv_test_reviews))
>    print ("AUC for Test set",auc_test_bow)
>    from sklearn.metrics import accuracy_score, log_loss, f1_score
>    preds = Classifier1.predict(tv_test_reviews)
>    acc = accuracy_score(Y_test, preds)
>    f1 = f1_score(Y_test, preds, average='macro')
```

The performance of the Naïve bayes is as the table below:

Accuracy is :	0.8023465703971119		
F1 Score is :	0.7932016618728028		
Confusion Matrix of Test Data:		[[1683 9] [648 984]]	
	precision	recall	f1-score
0	0.72	0.99	0.84
1	0.99	0.60	0.75
Accuracy			0.80
macro avg	0.86	0.80	0.79
weighted avg	0.85	0.80	0.79

Table 6. performance of the NB model in EMSCAD

Reference

[1] How to Avoid Work-from-Home Job Scams: 6 Tips. By Rachel Jay, Content Specialist. https://www.flexjobs.com/blog/post/how-to-find-a-real-online-job-and-avoid-the-scams-v2/

[2] EDA- Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. Jason Wei, Kai Zou. Protago Labs Research, Tysons Corner, Virginia, USA 2Department of Computer Science, Dartmouth College 3Department of Mathematics and Statistics, Georgetown University. 901.11196v2 [cs.CL] 25 Aug 2019

[3] GloVe: Global Vectors for Word Representation, Jeffrey Pennington, Richard Socher, Christopher D. Manning, Computer Science Department, Stanford University, Stanford, CA 94305

[4] Logistic regression and artificial neural network classification models. Stephan Dreiseitla, and Lucila Ohno-Machadob. Journal of Biomedical Informatics 35 (2002) 352–359.

[5] ATTENTION INTERPRETABILITY ACROSS NLP TASKS. Shikhar Vashishth, Shyam Upadhyay, Gaurav Singh Tomar, Manaal Faruqui. 1909.11218v1 [cs.CL] 24 Sep 2019.

[6] Attention is all you need. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. In *Proc. of NIPS*, 2017.

[7] Bert: Pre-training of deep bidirectional transformers for language understanding. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. In *Proc. of NAACL*, 2019.

[8] GA-Ensemble- a genetic algorithm for robust ensembles. Dong-Yop Oh & J. Brian Gray. Received: 9 June 2011 / Accepted: 2 February 2013 / Published online: 1 March 2013 © Springer-Verlag Berlin Heidelberg 2013

[9] Evolutionary Ensembles Combining Learning Agents using Genetic Algorithms. Jared Sylvester and Nitesh V. Chawla. Department of Computer Science and Engineering University of Notre Dame.

[10] An empirical study of the naive Bayes classifier. I. Rish. T.J. Watson Research Center.

[11] Interpretable Machine Learning. A Guide for Making Black Box Models Explainable. Christoph Molnar. 2020-10-19.

[12] Random forest classifier for remote sensing classification. M. Pal (2005) International Journal of Remote Sensing, 26:1, 217-222, DOI: 10.1080/01431160412331269698

[13] What Does BERT Look at An Analysis of BERT's Attention. Kevin Clark, Urvashi Khandelwal, Omer Levy, Christopher D. Manning. Association for Computational Linguistics, 276–286.