# Can you predict which company will hire you?

## I Introduction

In this essay, I attempt to make use of techniques from Supervised Machine Learning Literature to provide a general framework for building a 'personalized job classifier'. The purpose of this classifier will be to take in as its input, a company's feature vector and output a label (e.g. 0/1) that would inform its 'user' whether or not he gets a job in that company.[1, 2] 'Personalized' means that the classifier is specifically trained and tuned to a specific 'user' and is rendered unusable or at least unfit for anyone else to use. I aim to layout a general framework for building such a classifier.

I will start by discussing and solving a similar classification problem which once solved will provide all the tools required to tackle the original problem.[3] Then, I will layout the original problem and work towards constructing an algorithm independent general framework. This framework will abstract out all the algorithmic idiosyncrasies and focus on explaining and solving the core of the problem, which once understood can be implemented using a classification algorithm of one's choice. Finally, I will conclude by discussing how to scale up such a model to more than one user.

## II A Similar Problem

A similar problem focuses on building a classifier for a specific company '$C_1$'. This classifier would take in as its input an 'applicant feature vector' and output whether or not the applicant gets a job in company $C_1$. [4,5]

This problem is relatively easy to tackle than the original problem. Here, the focus is not on the applicant but on the company. Later I will show how by repeating this process for companies $C_2$ to $C_n$ we get very close to solving the original problem.

To build a classifier for $C_1$ that outputs a label (0/1) for an applicant requires us to create a dataset from the company's job applicants' history.

[1] 'feature vector' for any entity 'E' refers to a list of attributes that characterize 'E'. For e.g. if the entity is a 'House' then its feature vector includes, its price, location, number of bedrooms, size etc.
[2] 'user' refers to the person for whom the classifier is trained and tuned. I use 'applicant' and 'user' interchangeably.
[3] Original problem is: Can you predict which company will hire you?(focus is on the 'user'/applicant).
[4] Similar problem is : Can you predict which applicant will get the job in your company?(focus is on the company).
[5] Note that the original problem focuses on building a classifier for a specific 'user'.

Figure 1 below displays the structure of such a dataset.[6]

| Applicant Name | $\text{Feature}_1$ | $\text{Feature}_2$ | ………. | $\text{Feature}_n$ | Label(0/1) |
|---|---|---|---|---|---|

Figure 1: Applicant Dataset Structure

It is crucial that this Applicant Dataset contains both positive (label = 1) and negative (label = 0) training examples. This is an important concern as we want the algorithm to learn and identify the combination of features that count towards getting a job (label = 1) and also the combination that count towards not getting a job (label = 0).[7] In other words, the Applicant Dataset should contain both types of applicants, the ones that got the job and also the ones that didn't.

A single row in the dataset corresponds to a single applicant, for whom we have available, a ground truth label (0/1). Next, we have to construct a feature vector for the applicant, which together with the applicant's name and ground truth label will give us a single row in our dataset or in other words a single training example. By repeating this process for all applicants we will have our training set which can then be fed to the learning algorithm.

For constructing the feature vector I assume that the training set is rectangular.[8] A rectangular training set implies that the size of the feature vector constructed for each applicant would be same. For doing that, a predefined set of categories has to be selected, on which to collect data for each applicant. Some examples of such categories are as follows; years of education, years of experience, number of connections etc. Next step requires collecting data on those pre defined categories for each applicant. This can be done in part via a business and employment oriented social networking service like LinkedIn or the like. Within a LinkedIn profile of an applicant we can leverage languages like Python which has support for web scraping packages like 'Beautiful Soup', which creates a parse tree for parsed pages that can be used to extract data from HTML and XML documents. Now we can scrape off all the data for the pre defined set of categories for each applicant one by one from their profiles.

One important thing to realize is that, it is relatively easy to construct training examples that have a ground truth label of '1'. In other words, from a data source like LinkedIn, we can only find people who 'got the job' (label = 1) in a particular company ($C_1$). No one specifies on his/her LinkedIn profiles, the companies by which they were 'turned down' (label = 0). But, as mentioned earlier, it is crucial that our Applicant dataset for $C_1$ contains both positive (label = 1) and negative (label = 0) training examples.

[6] Labels: 0 = 'does not get the job', 1 = 'gets the job'. These labels are also referred to as ground truth labels.
[7] One 'training example' corresponds to one applicant or in other words a single row in Figure 1: Applicant Dataset Structure. Together all such training examples form a 'training set'.
[8] Rectangular data: collections of values that are each associated with a variable and an observation.

Unlike data captured from LinkedIn profiles, where people willingly make their private information available to a wider audience. Getting data on those job applicants of company $C_1$ that were turned down (label = 0) is less easy. People do not usually want to share such information and the companies having such information (e.g. $C_1$'s past job applicant history) have legal constraints on sharing such information. But, there are workarounds which allows us to remain within the legal boundaries and still get access to the required information.

One workaround is to buy from company $C_1$, the required dataset (e.g. records/results of past job applications), but with personally identifiable information (information that makes it possible for us to identify specific individuals) removed. For our application to work well, we need to know the combination of features that count towards getting a job and the combination of features that count towards not getting a job. It is certainly not necessary for us to know the names, home addresses, etc of the applicants (even though that information might in and itself count as interesting features, we may have to let it go). Other workarounds include repeating a similar process as above but instead of buying data from company $C_1$, we could buy data from more 'general purpose' data brokers. These brokers might be able to offer a relatively richer (in terms of features) dataset, still remaining within the legal boundaries. Once we have the required datasets, we can then use them to construct training examples with label = 0.[9] Next, I will focus on the process of 'feature extraction'.

Please note that I assumed that the Applicant dataset we constructed is rectangular. This assumption is made to simplify the setting in which we construct our model. But one should know that, if we want to leverage information from datasets that are unstructured (non-rectangular), there are techniques that allows us to do that. Similarly, if we want to leverage information from different 'types' of data (e.g. images, video files, etc.), we can. But, throughout this essay, I will continue following the assumption that our datasets are rectangular so as to keep our primary focus on the question at hand (Can you predict which company will hire you?) and not get lost in the implementational details. But, even constructing a rectangular dataset in and itself requires a fair bit of work. Let's go on to discuss the process of feature extraction.

The categorical data that we have collected for each applicant is not directly usable in its current state.[10] It has to be first pre-processed and then normalized to be declared 'fit' for training. This part of the process is called 'feature extraction'.[11] 'Extraction' means that we do not directly transition from the data collection phase to the model training phase. The feature extraction phase is not entirely synonymous with the cleaning of data, as the data we collect/acquire might be clean but in a 'state' that makes it use limited.

[9] It is of 'extreme' importance to make sure that one has the appropriate legal clearance in trying to get access to/ creating datasets that might contain 'Personally identifiable information'. Legal clearance might be needed from IRB's (Institutional Review Boards) and/or equivalent institutions for applications that either directly or indirectly involves human subjects and/or makes use of information that is potentially 'personally identifiable'.

[10] At least the non-numeric categories are not directly usable.

[11] Normalization of data is done to bring different features into similar ranges. This improves learning and makes algorithms like gradient descent converge faster. This is usually done by subtracting off the mean of the feature from itself and then dividing the result by the feature's standard deviation.

Constructing some features require minimal work as they are already in the state we want them to be. For e.g. the ‘number of connections’ block in a LinkedIn profile page is directly usable as it’s data type is an integer, other such examples include, ‘yrs of experience’, ‘yrs of education’ etc. Such features require minimal work like normalization and then they are good to go.

Other features might not be directly usable. For e.g., We might think that an applicant’s Alma mater is a feature that is an important determinant in predicting whether or not he gets the job. But, we usually do not directly feed in the html block containing the Alma mater information into the dataset. We can do that and the machine learning algorithms will make sense of it (given enough labeled training examples). But in practice (or at least for the purposes of this essay) we establish some form of structure (e.g. a hash table, associative array, etc.) that makes learning faster, intuitive and more efficient.[12]

So, we could map each college name (value) to a number (key). This way we could construct a mapping that would help us efficiently incorporate one’s Alma mater as a feature in our dataset. In some cases such mapping might not be very useful due to the hash table being very unevenly distributed (when there are either too many or too less keys).

If hash tables are not useful, a different way to capture the ‘Alma mater’ feature would be to use yes/no (0/1) questions. Examples of such questions are as follows, ‘Did you attend Harvard as your Alma mater?’, ‘Did you attend MIT as your Alma mater?’, ‘Do you have more than a year’s experience working with the R programming language?’, etc. This is a very useful and fast way to construct new features that are binary (0/1) in nature. These types of features captures interesting information, take up significantly less memory and are relatively easy to construct such that a lot of these can be constructed relatively fast. Moreover, there is no need to clean or normalize them due to their binary nature. But, these features require hand designing on our part. This hand designing is based on what we think counts as interesting features (so there is an added risk of human added biases). We are forgoing benefits of techniques such as ‘automatic feature selection’ using train/development/test sets that the field of Machine Learning has to offer. These techniques aim at choosing the subset of features that minimize their corresponding cost function on the cross-validation set.[13]

Usually the process of feature extraction involves both hand designing features and leveraging techniques from the field of Machine Learning to automate feature selection process.[14]

Figure 2 summarizes the typical workflow for constructing a single training example/single applicant.[15]

[12] Hash table data structures are daily used in applications like spam classification to convert an email into an array of numbers, which then with a label (0/1), serves as a single training example in the training set for our spam classification dataset.

[13] Intuitions of what one thinks are ‘interesting’ features is a very subjective matter. Using handpicked features (as opposed to using an algorithm to min a cost function) might result in us unknowingly adding a small bias term to our model.

[14] Sometimes, carefully hand designed features saves a lot of computing time that machine learning algorithms will otherwise take to learn what subset of features are interesting. It is like leveraging the best computing machine we currently have (our brain) for a little time (maybe a few minutes) to save probably hours of computing needed to find the best subset of features.

[15] Figure 2 is just meant for illustrative purposes, i.e. we do not go through the entire workflow one training example at a time. Rather, we first collect data on all applicants from the data source, then we perform feature extraction for all applicants and then we get our entire training set, all at once.

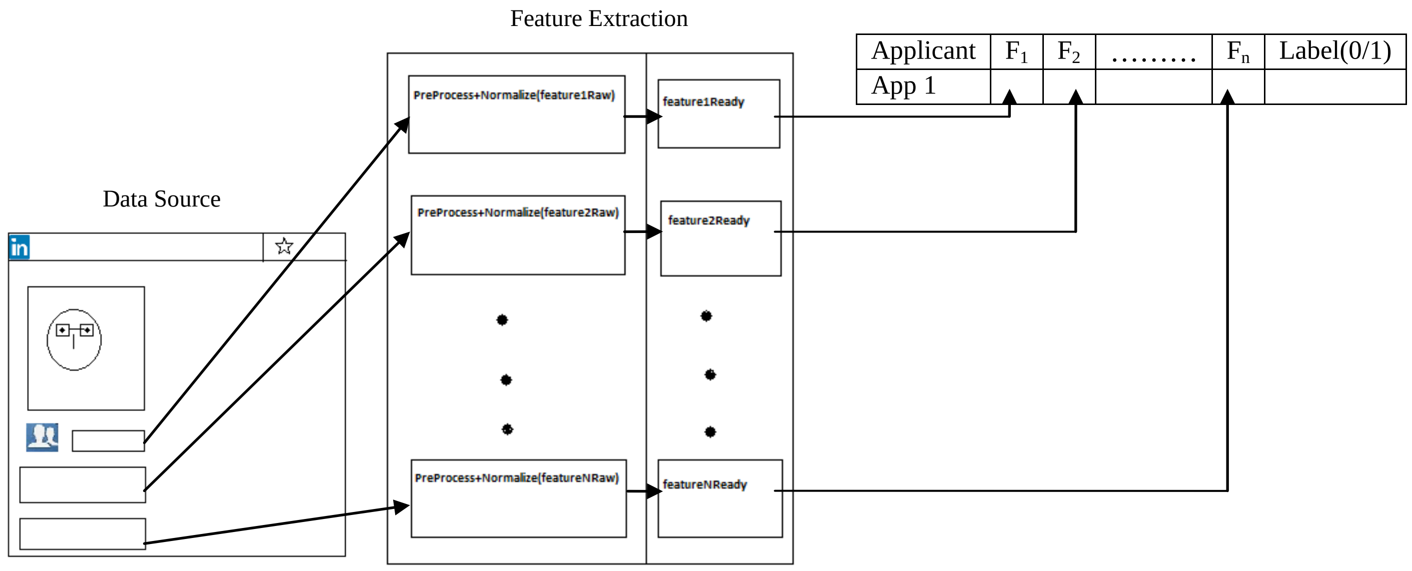

Figure 2: Typical Workflow to construct a single training example

We repeat the 'typical workflow' (shown in Figure 2 above) 'm' times (where, m is the number of training examples that we want in our Applicant dataset) to get the entire training set. We then feed that training set to an algorithm e.g. logistic regression. This algorithm will further use some optimizer like gradient descent to minimize a cost function (e.g. cross-entropy cost function) and hand out the learned parameters which can then be used to make a prediction on new unseen examples.[16]

This initial implementation of the model is dirty. Our objective is to be able to perform well on previously unseen examples and it is unlikely that our first implementation would be the one that serves our purpose. It is very important for us to not fall for pre mature optimization while building such models. It is recommended that we build a model, test it, rebuild it, rather than trying to optimize before building. A lot of tools are available for tuning the model once its initial implementation is ready. Some examples are, dividing the entire dataset into training, development/CrossValidation and test subsets and then using that to systematically select hyper parameters like the 'learning rate', or the 'regularization parameter' Λ (lambda).[17] Another example involves using techniques like regularization to address over fitting.[18]

[16] Note that logistic regression for an applicant outputs Probability(label = 1 | training set), so in the above example, if suppose we input a new applicant's feature vector we get as the output of the hypothesis function, 0.8, this means that there is an 80% chance that the applicant will get the job. Now, to output a label for the applicant we have to set a threshold e.g. 0.5, above which we predict 1 else 0.

[17] We train different models only on a small part of the 'entire dataset', and call that part the 'training/train' set. Then we use the cross-validation set to choose the best performing (in terms of minimizing the cost function) model from these different models. Further, we use the chosen model and evaluate its performance by testing it on the 'test' set, which gives us a reasonable estimate of the model's generalizability.

[18] Regularization (in a few words) makes it costly for our model to have 'big' weights/parameters. Adding the regularization term to the cost function and then minimizing it, adds an additional incentive to push 'weights' towards 'zero'. The size by which the weights are pushed towards 'zero' depends on the size of the regularization parameter Λ (lambda).

Given the availability of all these techniques, premature optimization is not a good idea. So, rather than being fixated on to decide what the learning rate or the number of iterations of gradient descent should be, we should use these machine learning techniques of automatic model selection to serve that purpose.

Now, let's assume that we have trained and tuned our model such that it is giving us a high predictive accuracy and also good performance on the test set.[19] We can now use it to classify whether or not a new applicant will get a job in company $C_1$. Now, having solved this problem we have already come very close to solving the original problem (see pg. 1, footnote no. 3). Now, let's tackle the original problem.

## III Original Problem

In section II, I provided a framework to build a model that addresses the question 'Can you predict which applicant will get the job in your company?'. There, the input to the algorithm was the applicant feature vector and the output was a label (0/1) which informed the applicant whether or not he got the job in that company. Note that this model was built for a specific company $C_1$, so if an applicant wants to know whether or not he gets a job in company $C_2$, he cannot use this model. To solve this problem, we have to build a general model. This model will take in as its input, not the applicant feature vector, but rather the 'company feature vector' and inform the applicant/user of his job prospects (0/1) for any company he is interested in. The original problem presented ahead aims at building such a model.

The original problem states, 'Can you predict which company will hire you?'. It aims at building a model for a particular user/applicant (see pg.1, footnote no. 2), the focus is on the user/applicant. All that the user has to do is to enter the company feature vector for the company he is interested in and have as the output of the model a label that will inform him (with certain probability; see pg. 4, footnote no. 13) whether or not he will get a job in that company. So, in this model the focus will be on a specific 'user'. Also, the model that is built will be trained and tuned to the specific user and will be rendered unusable or at least unfit for anyone else to use. I assume from now on that the model is built for a user named User1. Next, I will layout the entire process of building this model in a few lines and then expand on it.

In section II, we built the model for a specific company $C_1$ (let's call that model $M_1$). Now, we have to repeat the entire process used to construct $M_1$, for companies $C_2$ to $C_n$ , which gives us models $M_2$ to $M_n$ (corresponding to companies $C_2$ to $C_n$). We have 'n' models each corresponding to a specific company. Now, User1 can in turn use $M_1$ to $M_n$ to get 'n' labels $L_1$ to $L_n$, each of which will inform him of his job prospects (0/1) in companies $C_1$ to $C_n$. In section II, we built an applicant feature vector, one for each applicant, in a similar way now, we have to build a company feature vector, one each for companies $C_1$ to $C_n$. Once we do that we will have 'n' company feature vectors ($V_1$ to $V_n$) each of which is 'n' dimensional. Having done that, for User1, we have labels $L_1$ to $L_n$ (each of which tells him whether or not

[19] Predictive accuracy of the model is equal to the proportion of the training examples it classifies correctly.

he got a job in companies $C_1$ to $C_n$). Also, we have 'n' feature vectors $V_1$ to $V_n$, corresponding to companies $C_1$ to $C_n$.

Now, we have all we need to create our new 'Company Dataset'. In this dataset, each training example contains information for a single company in the form of 'n' features and a label. So, $row_1$ in the training set corresponds to $C_1$ which possesses a 'n' dimensional feature vector $V_1$ and a label $L_1$. Similarly $row_2$ in the training set corresponds to $C_2$ which possesses a 'n' dimensional feature vector $V_2$ and a label $L_2$. This continues in a similar fashion till $row_n$ in the training set which corresponds to $C_n$ which possesses a 'n' dimensional feature vector $V_n$ and a label $L_n$.

Figure 3 below represents the structure of the 'Company dataset'.

| Company Name | $Feature_1$ | $Feature_2$ | ……..… | $Feature_n$ | Label(0/1) |
|---|---|---|---|---|---|

Figure 3: Company Dataset Structure

Note that we get labels $L_1$ to $L_n$ by running a 'specific user's', applicant feature vector 'in turn' through models $M_1$ to $M_n$. This means that the Company Dataset represented above is 'personalized' to a specific user and is not suitable for anyone else to use. This idea is better represented by an example.

First, let's see what $row_1$ in the above Company Dataset represents. First comes the Company Name ($C_1$), that is followed by a 'n' dimensional company feature vector ($V_1$), which defines $C_1$ in terms of n features (some examples of features could be "Is the company a fortune 500", 'stock price', 'Number of employees', etc.), after that comes a label ($L_1$), this label $L_1$ comes from running a 'specific user's' (let's assume that user is named User1) applicant feature vector through the model $M_1$ (remember, $M_1$ took as its input an applicant feature vector and outputted whether or not that applicant got the job in company $C_1$), which outputs a label (let's assume that label was 1, i.e. User1 got the job). So, an informal way to read $row_1$ of the table is,

*"As per $M_1$, User1 got the job in company $C_1$ which has features $V_1$ (number of employees, stock price, etc.) "*

If we follow a similar procedure for $row_2$ to $row_n$, while keeping in mind that, just like $L_1$, each label $L_2$ to $L_n$ comes from running User1's applicant feature vector in turn through models $M_2$ to $M_n$. (this is what makes the model 'personalized' to User1, as all labels $L_1$ to $L_n$ in our entire Company Dataset correspond to User1). So, we can now informally read all other rows just like we read $row_1$ above as follows. (For illustration purposes, I assume that $row_2$ has label = 1, $row_3$ has label = 0, $row_n$ has label = 1).

*"As per $M_2$, User1 got the job in company $C_2$ which has features $V_2$ (number of employees, stock price, etc.) "*

*"As per $M_3$, User1 didn't get the job in company $C_3$ which has features $V_3$ (number of employees, stock price, etc.) "*

*"As per $M_n$, User1 got the job in company $C_n$ which has features $V_n$ (number of employees, stock price, etc.) "*

There are few important things to notice in the above statements. First, across all statements the 'user' remains the same (User1), this is what I mean by a model 'personalized' to User1. Second, each statement corresponds to a specific row in the Company Dataset. Third, each row corresponds to a company, a model, a feature vector and a label, e.g. $row_1$ corresponds to company $C_1$, model $M_1$, feature vector $V_1$, and label $L_1$. Finally, the 'Company Dataset' is also a rectangular dataset (see pg. 2, footnote no. 8).

Once we feed the entire Company Dataset to our learning algorithm like logistic regression or the like, it will go on to learn (begin training) what combination of company features count towards User1 getting a job and also what combinations of company features count towards User1 not getting a job. Once trained and tuned, the model serves as a 'personalized job classifier' for User1, which means that the model can now take any new company that it has not seen before, distill that company's characteristics into a company feature vector, take that vector as its input and output a label (0/1) which will inform User1 (with certain probability, see pg. 4, footnote no. 13), whether or not he gets a job in that new company. Also, User1 can do this for any number of new companies, all he has to do is enter as an input to the model the company feature vector of the company he is interested and get a label as an output that will inform him of his job prospects in that company.

With that, the model is complete. I answered how someone can build a model that lets him predict whether or not a particular company will hire him. Next, I will expand on a few important details of the model and then I will summarize the entire process of building the whole model (personalized job classifier) for a specific user (User1) in a diagram

First, I will discuss, How to construct a company feature vector?. In section II (Figure 2), I showed how we can construct an applicant feature vector using the LinkedIn profile of an applicant. Like an applicant, a company may not have a LinkedIn profile. But, every company has some form of online presence, e.g. a website, blog, etc. So, the ideas discussed, to construct an application feature vector in section II, can in similar ways be leveraged to construct a company feature vector. To highlight some of the important things, first we have to decide the categories on which to collect data (from a data source, like a company website) for each company. After that we have to perform feature extraction, keeping in mind that our dataset is rectangular so that the features we decide to extract from the categorical data we collected for '$C_1$' (e.g. stock price, number of employees) are same for all other companies $C_2$ to $C_n$. In other words, if we have two features (stock price, number of employees) for $C_1$, we will have those same features for $C_2$ (with values corresponding to $C_2$ filled in), and the same for $C_3$ to $C_n$ (with their corresponding values filled in). Next, we may want to leverage data structures like hash tables (as explained in section II) to construct various features. Also, we discussed ways of creating interesting binary features, in the same way, we can create binary features that capture important information about the companies, some examples are, 'Is this company a fortune 500', 'Is this company listed in a stock exchange' etc.

One advice for anyone building such a model would be to focus only on the set of companies relevant to him. For e.g. If an IT professional wants to build such a model for himself, then maybe he should streamline his training set to only include companies from the IT sector (and/or, some other sectors where he thinks there are hints of job prospects). Including companies from sectors where his job prospects are 'nil' might not hurt, but unnecessarily crowding the training set with the companies for which he is sure of not getting a job (as those are the companies that don't match his skill set and are already classified with label = 0.) might not be a good idea.[20]

Next concern is regarding the quality and quantity of features and the size of the training set. By quality features, I mean those features that strongly count towards predicting the variable we are interested in. In other words, features that explain most of the variance in the underlying variable we want to predict. For e.g. if I want to predict housing prices, some quality features might include 'size of the house', 'location of the house', etc. Other not so good features that might somehow contribute towards deciding the price of a might include 'distance to the nearest train station'. One of the ways to choose quality features is to handpick the ones we 'think' are good. Other ways (that are usually complementary) include using train/CV/test sets to systematically choose the set of features that correspond to the model which minimizes the cost function on the cross validation set.

Finally, let's discuss one more concern, the quantity of features and the size of the training set. The total number of features we choose to have, has a bearing on the total number of training examples. To illustrate, suppose we decided to collect five hundred features (i.e. five hundred columns) for each row in our training set. So, if we had a thousand training examples (rows), the size of our training set would be (1000, 500). But, now suppose we only have five training examples such that the size of the training set is (5, 500). The difference between a training set size of (1000, 500) and (5, 500) is that in the latter case, our learning algorithm has only five instances of data to decipher patterns from 500 (and much more in terms of total permutations /combinations) features , which is not nearly enough to learn the underlying general statistical patterns. Given that, it is important that our training set size (total number of instances

[20] Adding companies from sectors where job prospects are 'nil' can ideally be done, if someone wants to build a very 'general purpose' personalized job classifier. If someone considers building such a model than an important thing to remember is that, adding a few additional training examples of companies from these 'other sectors' might not be enough, for our algorithm to decipher meaningful patterns from it. So, we may have to provide a lot of new training examples of the 'other sector' type.

of data) is big enough (relative to the total number of features) so that our learning algorithm could learn a reasonable representation (that will perform good on the test set) of the underlying statistical patterns in the data. In other words, if we have a very large training set, then we can accommodate a lot of features into our model. Exactly how large the training set size should be relative to the number of features is debatable. But, the training set size should, at the least be equal to the total number of features (for e.g. if we have two features, than we at the least need two training examples, simply because in order to solve for two variables, at the least we need two equations.)

So, if our training set is big enough, such that we can accommodate a lot of features, than we should, as that increases the granularity of the model. For example, in our original model, if we assume that our training set size is significantly larger than the total number of features (e.g. 20x larger), than we can accommodate a lot of features so that our model is able to more finely recognize the 'company features' that count towards User1 getting a job and hence is able to pick very small differences between different company feature vectors. This might not be possible with a small set of features (There is an additional concern of 'over fitting' in models with high dimensional feature spaces, but with machine learning techniques like 'regularization' in our toolbox, that can be handled quite reasonably.)

Below, I summarize the entire process of building a 'personalized job classifier' for 'User1' in a diagram,

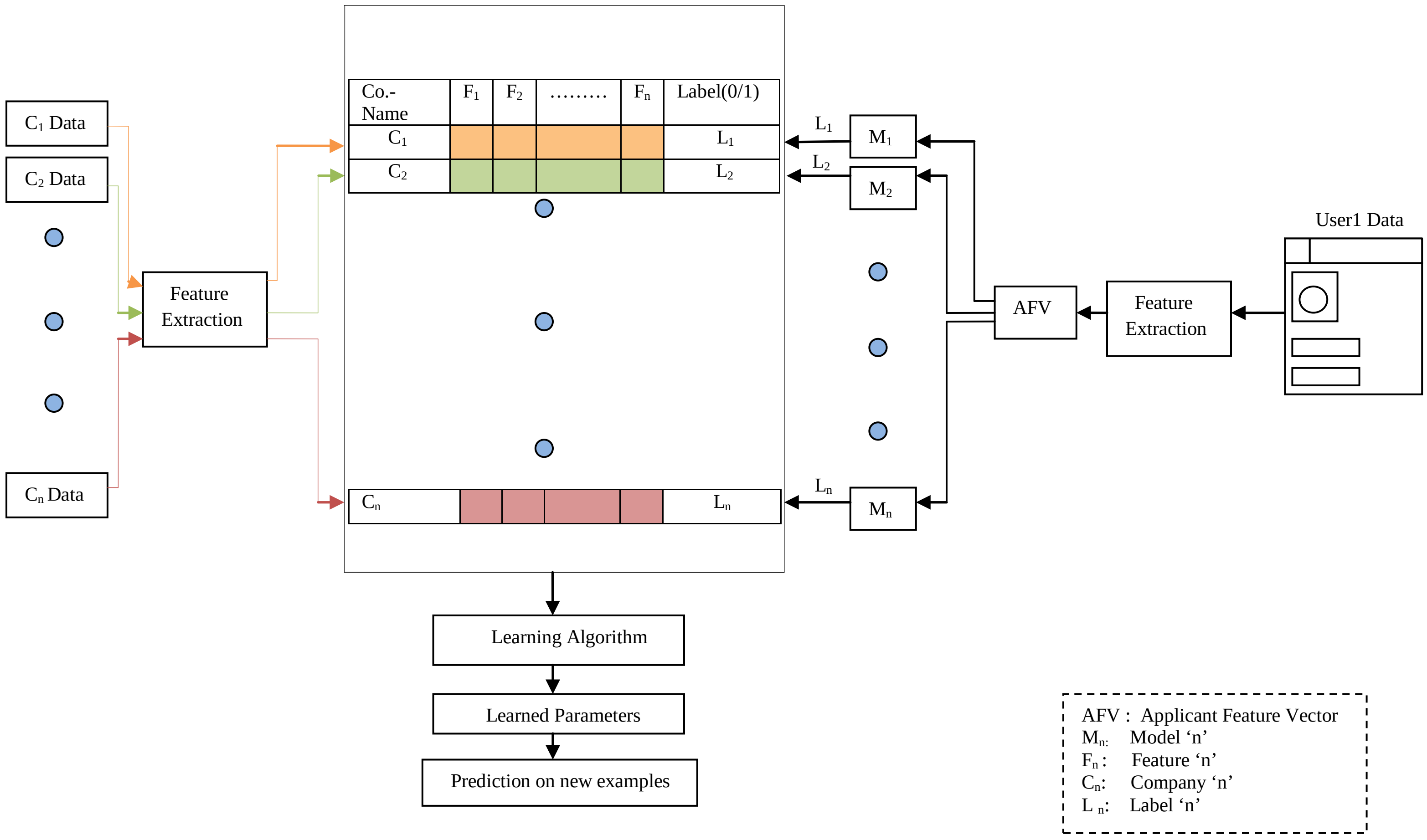

Figure 4: Process of building a personalized job classifier for User1

## IV Conclusion

Figure-4 summarizes the entire process for building a personalized job classifier for a specific user (User1). Throughout in the essay, I mentioned that the model is built for a specific user (User1). Now, in these last few lines I will discuss, how we can very easily scale up this model to any number of users (assuming that these other users belong to the same profession as User1 does, and/or these other users belong to the same group of people as User1 does, e.g. the group of Mathematics Graduates).

Let's assume that User1 is a Mathematics Graduate and wants to apply for a job. Now, User1 builds for himself a personalized job classifier. To build a model for User2 (who is also a Mathematics graduate) takes very less time as we already have the infrastructure for User1's model, the only things that change while building the model for User2 is the applicant feature vector and the labels ($L_1$ to $L_n$). In other words, as User2 belongs to the same group of people as User1 (Mathematics Graduates), it is reasonable to assume that while applying for a job, User2 also aims at applying at the companies that User1 aims at, so that there is no need to reconstruct the company feature vectors. Given that, the only thing that User2 need to do is to construct his own 'personalized labels', which he will do by constructing his applicant feature vector and passing it through models $M_1$ to $M_n$ (which are the same models that User1 used and hence are reutilized by User2) to get his own personalized labels ($L_1$ to $L_n$). These labels to together with the company feature vectors can then be passed to the learning algorithm, which will output the learned parameters that will then be used to make prediction on new unseen examples for User2. Similarly, we can scale up this process to any number of users (given they belong to the same group/category of people).

But, what if User1 and User2 belonged to different professions?. In that case, if we assume that the companies relevant to User1 are not relevant to User2 (because they don't match User2's skill set). Then, at best, the only things that User2 can leverage from User1's already built infrastructure are its company feature vectors. For User2 that would serve as data with label = 0 (as User2 is sure of not getting a job in companies relevant to User1 such that he can safely classify them with label = 0, and add them to his training set). Note that this borrowed infrastructure might not serve as quality data for User2 as these training examples are from a sector where the job prospects of User 2 are 'nil' (see pg. 9, footnote no. 20). (Besides that, remember that User2 also has to construct his personalized labels for which he has to construct models $M_1$ to $M_n$ and his applicant feature vector).

I believe that the general framework provided in my model above will help serve as a guide to anyone wanting to build a personalized job classifier.

(Thank you for your kind attention, I eagerly look forward to the opportunity.)