Interview Questions

Answer 1: A/B Testing

Q1: We A/B tested two styles for a sign-up button on our company's product page. 100 visitors viewed page A, out of which 20 clicked on the button; whereas, 70 visitors viewed page B, and only 15 of them clicked on the button. Can you confidently say that page A is a better choice, or page B? Why?

A/B Testing is a way of comparing two or more test groups by randomly assigning each group a specific single variable treatment. In this example: showing page A and page B to two groups of users respectively.

	Views (V)	**Clicks (C)**	**Conversion Rate (C/V)**
Page A (control)	100	20	20.0%
Page B (treatment)	70	15	21.4%

Usually, page with higher conversion rate is better. We may be tempted to declare Page B good, but how do we know the variation is not due to random chance? What if instead of 70 visitors we only had 5 visitors? Would we still be so confident? Hypothesis testing is all about quantifying our confidence. Let us aim for 95% confidence interval.

Our null hypothesis will be - The conversion rate of Page A (control treatment) is no less than the conversion rate of Page B (experiment treatment).

$$H_0: R_b - R_a <= 0$$

Where, R_a is the conversion rate of Page A and R_b is the conversion rate of Page B

The alternative hypothesis is therefore - Page B has a higher conversion rate. This is what we want to see and quantify.

$$H_1: R_b - R_a > 0$$

We know the probability distribution of Page A conversion rates (R_a) is some kind of binomial distribution. Similarly, the probability distribution of Page B conversion rates (R_b) is another binomial distribution. We know for BN distribution the standard deviation is given by this formula:

$$\boldsymbol{\sigma}_a = \text{sqrt}(R_a * (1 - R_a) / N_a),\ \text{cos variance fo BN distribution is } r(1-r)$$

Likewise we can calculate $\boldsymbol{\sigma}_b$. These binomial distributions can be approximated to normal distributions. We are interested in the difference between the two conversion rates. If the difference is large enough we conclude that the treatment really did alter user behavior. The probability distribution of the “difference in conversion rates” is another normal distribution. Therefore, z-score of the “difference in conversion rates” distribution can be calculated with this formula:

$$z = (P_b - P_a) / \text{sqrt} (((P_a * (1 - P_a)) / N_a) + ((P_b * (1 - P_b)) / N_b))$$

Where N is the respective sample size of each experiment.

Since our alternative hypothesis is of form X > 0, we are interested in only right tailed test. At 95% CI, the cutoff z-score is 1.65. In other words, we can reject the null hypothesis with 95% confidence if the z-score is higher than 1.65. So let us now calculate z score for Page B, using above formula:

	Views (V)	**Clicks (C)**	**Conversion Rate (C/V)**	**Z-Score**
Page A (control)	100	20	20.0%	N/A
Page B (treatment)	70	15	21.4%	0.2257

0.2257 is below our required cutoff z-score of 1.65. Hence we failed to reject null hypothesis. In other words, Page A and Page are not significantly different. Neither is better than the other. That was a trick question, huh? **:)**

Answer 2: Group Twitter Users

Q2: Can you devise a scheme to group Twitter users by looking only at their tweets? No demographic, geographic or other identifying information is available to you, just the messages they've posted, in plain text, and a timestamp for each message. Assuming you have a stream of these tweets coming in, describe the process of collecting and analyzing them, what transformations/algorithms you would apply, how you would train and test your model, and present the results.

Right away this sounds like a unsupervised learning problem, specifically clustering problem. For such problems my goto algorithm is Gaussian Mixture Model clustering (GMM), rather than K-Means clustering, because GMM clustering is a soft clustering method. In KMeans, a point belongs to one and only one cluster, whereas in GMM a point belongs to each cluster to a different degree. Twitter Users can belong to multiple clusters, based on the overlapping features (such as, age, gender, ethnicity, location, profession, political affiliation, sports following etc) that we will be able to extract from their tweets. Hence GMM is better suited.

Since our data (tweets) is streaming, we will use a streaming framework such as, Apache Spark cluster to gather, preprocess, transform and finally analyze data.

Our data-processing pipeline will look something like this:

- Collect data: As data arrives we can store it in hdfs. We can use timestamp of tweet to partition data. User Id will be the key of the records.
- Pre-process data: Periodically run spark job to pre-process data:
    - Remove stop words (like the, i, you, me etc) from tweets using some NLP framework (eg: NLTK).
    - Stem the words in each tweets to get the root of the words. (eg: change words like responsiveness, responsible, responsibility, nonresponse to ‘respon’).
    - TFIDR: May be we can use tfidr too.
- Feature extraction: We will need a corpus of words that will help us extract features (such as, age, gender, ethnicity, location, profession, political affiliation, sports following etc) from each tweet. We will give higher weight to the hashtag words in corpus matching.
    - Age: If user uses words like ‘dude’, ‘wassup’ - probably teenager
    - Ethnicity: If user uses words like ‘Bonjour’ - probably french
    - Profession: If user uses words like Java, C++ - probably a programmer
- Feature selection/reduction: Since we will end with lot of features per user, we need to reduce it. We can take top 10 percentile of features first. Sklearn automatically does this. Then we can perhaps apply PCA, to preferably reduce features to 3 principle components, so that clustering algorithm can run fast.
- Analyze data: Once we have data transformed by feature extraction and reduction, we can run python’s sklearn GMM algorithm to generate clusters.

- Visualization: We can use some WebGl based JS plotting library to render our clusters in a browser. We will color code each cluster and plot the cluster points in 3D space. Each cluster will appear in some kind of spherical-ish blob in 3D space. Since we had reduced our features to 3 principle components, it is coming handy in visualization as well. Visualization job will run periodically and update the UI input data structure.

Since the number of clusters is not known beforehand, there is no guarantee that computed number of clusters above best segments the data. However, we can quantify the "goodness" of a clustering by calculating each data point's silhouette coefficient. The silhouette coefficient for a data point measures how similar it is to its assigned cluster from -1 (dissimilar) to 1 (similar). Calculating the mean silhouette coefficient provides for a simple scoring method of a given cluster.

Space required to store the tweets data is not a concern. Disk space on HDFS cluster is cheap. Trained GMM model itself has a small memory footprint and stays in RAM. Because of the amount of tweets data is so large, training and updating the model is bit time consuming. We can reduce this impact by parallelizing the computing by distributing the workload over many machines, that Apache Spark cluster will automatically do for us. Model will update, as frequently as we run the update jobs. So it is one of the tunable parameters that we have to play with to get the best results.

Answer 3: Prevent Overfitting in Classification Setting

Q3: In a classification setting, given a dataset of labeled examples and a machine learning model you're trying to fit, describe a strategy to detect and prevent overfitting.

Training and testing a model on the same data is wrong. Such model would just repeat the labels of the samples that it has just seen and would have a perfect score but would fail to predict anything useful on unseen data. This situation is called overfitting. To avoid it, it is a common practice in supervised machine learning to split the data between training and testing subsets. Often training set is further divided into training and cross validation sets. Training set is to build the model. Cross validation is to fine tune the model parameters on the unknown instances. To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds. Finally, test set is to check for accuracy for the model.

To detect Overfitting measure error on a training and test set. If we see low error on training set and high error on test and validation set then we have overfitted the model. On the other hand, if we see high error on training set then we have high bias model.

To prevent overfitting:

- K-Fold cross validation: In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. The advantage of this method for Grid Search over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once.

Without cross-validated set, if we keep training the model several times over different hyper parameter combinations, using only a single validation set to check the model's performance, the model will slowly become biased towards the validation set being used. It will later fail to generalize to new unseen data. Such models are called over fitted models. Therefore it is highly recommended to use k-fold cross-validation with GridSearch.

- Regularization: Regularization penalizes certain sources of overfitting. A commonly used regularization technique is Lasso Regularization, where it penalizes model if the sum of squared errors is too low, thus smoothing high variance case.

- Simple Model: Occam's razor favors simpler models over more complicated models. Generally, the fewer parameters that we have to tune the better. If you can remove one of our parameters without significantly increasing the error, that is better.

Answer 4: Smart Context Menu

Q4: Your team is designing the next generation user experience for your flagship 3D modeling tool. Specifically, you have been tasked with implementing a smart context menu that learns from a modeler's usage of menu options and shows the ones that would be most beneficial. E.g. I often use Edit > Surface > Smooth Surface, and wish I could just right click and there would be a Smooth Surface option just like Cut, Copy and Paste. Note that not all commands make sense in all contexts, for instance I need to have a surface selected to smooth it. How would you go about designing a learning system/agent to enable this behavior?

Right away this sounds like a "learning" system, rather than "learned" system. This also looks very similar to our smartcab project, where a virtual car learnt do's and dont's of US traffic rules. Smartcab was able to learn by making use of Reinforcement Learning techniques, getting +ve rewards for doing right things and -ve rewards for doing wrong things.

We can deploy similar Reinforcement technique, Q-Learning, to design our Smart Context Menu.

- State Management: First thing we need is a way to capture system's State. State captures values of all the attributes in the system that are necessary to make a decision. For example: In our case, State will include objects that are currently in the user's workspace (such as text in wordApp, circle-square-line in drawApp, aeroplane parts in planeAssemblyApp). Capturing this State is app specific. For instance in drawingApp it means capturing the coordinates of shapes.
- Actions Management: Next we need a way to capture valid Actions from each state. System can be preconfigured with some "initial" set of valid actions from each State. For example: In wordApp, when some text is highlighted, valid actions can be Copy, Delete, Bold, Align etc.

Out of these valid actions set, our AI algorithm will slowly learn the “prefered” action by the user in each state. By pre configuring the system with valid actions in each state we have already specified the “known constraints” (or invalid actions) for respective states. For example: “Smooth Surface” action is not available if State does not include a selected surface.

- Rewards Management: As user interacts with the application, a background thread will take note of what action was taken by the user when the workspace was in a certain State. It will increase the q value for that particular state-action pair. Next time workspace is back in the same State, our system will prompt the highest q value action to the user in a dialog box. If the user accepts the recommendation, system is positively rewarded, and corresponding state-action q value is further increased. If the user rejects the dialog box (suggested action), then a -ve reward is received and system accordingly decreases the q value of the corresponding state-action pair. As this interaction cycle continues, system slowly “learns” the user’s preferred actions in each state. System will prompt for user feedback on actions, by balancing exploration and exploitation by exploration factor, epsilon. In each State when right click context menu is shown, we query the q-table and show actions that have top 5 q values.

Thus overtime, our context menu will start showing action that are most relevant in the given state and prefered by the user.

- Space requirement: Q-learning at its simplest uses tables to store data. This very quickly loses viability with increasing sizes of state/action space of the system it is monitoring/controlling. Space requirement is of concern here. We must chose our States very wisely.

  One answer to this problem is to use an (adapted) artificial neural network as a function approximator. More generally, Q-learning can be combined with function approximation. This makes it possible to apply the algorithm to larger problems, even when the state space is continuous, and therefore infinitely large. Additionally, it may speed up learning in finite problems, due to the fact that the algorithm can generalize earlier experiences to previously unseen states.

- Time requirement: Although it will take system infinitely long time to learn the optimal policy, meaning figure out best actions in each State, we don’t need optimal policy. Our system can already declare success, as soon as it can show some preferred actions in context menu for the most prominent States, which will happen within first few interactions. Hence model train and update will happen within 5-6 episodes. Learning time is not of a concern here.

Answer 5: Regularization

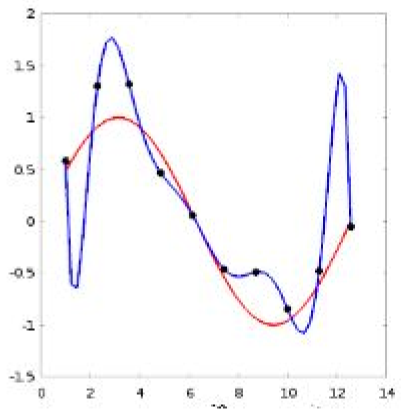
Q5: Give an example of a situation where regularization is necessary for learning a good model. How about one where regularization doesn't make sense?

Regularization refers to the method of preventing overfitting, by explicitly controlling the model complexity. It leads to smoothing of the regression line and thus prevents overfitting. It does so by penalizing the bent of the regression line that tries to closely match the noisy data points. Lasso Regularization is a popular technique to achieve this. Let us try to understand Regularization with an example:

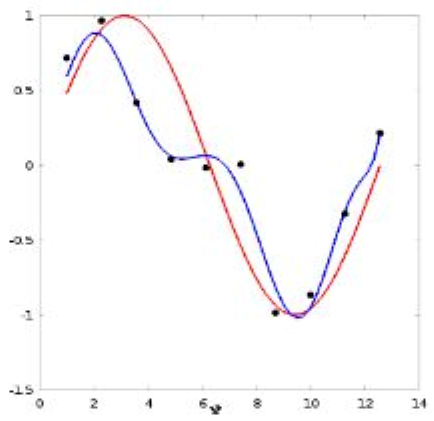
Let us say we are trying to predict the wage of someone based on his age. We will first try a linear regression model with age as an independent variable and wage as a dependent one. This model will mostly fail, since it is too simple. Then, we might think: well, we also have the age, the sex and the education of each individual in our data set. We could add these as explaining variables. Our model becomes more interesting and more complex. We measure its accuracy regarding a loss metric L(X,Y) where X is our design matrix and Y is the observations (or labels) vector (here wages). We find out that our result are quite good but not as perfect as we wish. So we add more variables: location, profession of parents, social background, number of children, weight and so on and so forth. Now our model will do good but it is probably overfitting, i.e. it will probably have poor prediction and generalization: it sticks too much to the data and the model has probably learned the background noise while being fit. This isn't of course acceptable. It is here where the regularization technique comes in handy. We penalize our loss function by adding a regularization parameter (λ) to our weights vector w. In intuitive terms, we can think of regularization as a penalty against the complexity. Increasing the regularization strength penalizes "large" weight coefficients. Therefore, our goal is to prevent our model from picking up "peculiarities" or "noise," and to generalize well to new, unseen data.

$$L(X,Y) + \lambda N(w)$$

This will help us avoid overfitting and will perform features selection also. To tune the regularization term λ, we divide our training data. Then train our model for a fixed value of λ and test it on the remaining subsets and repeat this procedure while varying λ. Then we select the best λ that minimizes our loss function. The plot below shows over-fitting, where the derived function (the blue line) fits well with the training data but does not resemble the original function (red line).



After using regularization, the derived function looks much closer to the original function, as shown below:



Regularization does not makes sense where models are simple, such as linear model. For example: at constant speed, the distance traveled with time is linear. The more time we drive the more distance is traveled. If we are modeling this scenario to predict how far we will travel in given time t then our model is simply: s = wt, where w is some weight which will come out to be the speed of travel, t = time taken, s = distance traveled. In this simple case, there is no need to regularize the model.

Other situation that I can think of is where there is absolutely no noise in the examples collected. The computed model by linear regression is pretty accurate in representing the underlying data. If we know this is the case, then we should not regularize the model. However, on the cautious side, we should still investigate whether our model has overfitted by calculating training and testing errors and making sure testing error is not high. In other words, model has generalized well to the unseen data.

Answer 6: Targeted Coupons

Q6: Your neighborhood grocery store would like to give targeted coupons to its customers, ones that are likely to be useful to them. Given that you can access the purchase history of each customer and catalog of store items, how would you design a system that suggests which coupons they should be given? Can you measure how well the system is performing?

I think Collaborative Filtering with k-means is most appropriate for this usecase.

First, we need a database to store item catalog and user purchase history. Catalog table will contain Item number, description, price and product category. User table will contain User information such as User Id, name, age, gender, address etc. Purchase history table will have user Id, product id and qty purchased. Every time a user makes a purchase new records are added to this table. Since user's buying patterns change as family and other circumstances change (for example baby born), we just need to keep 2 year worth of buying history for a user in this table. Finally, we need a Collaborative Filter table, which is a giant matrix of each user as rows and every product item in the store as columns. This table is a scratch pad in that it is populated whenever machine learning algorithm is run and erased clean after targeted coupons are generated.

We are planning to generate targeted coupons once a month as an incentive based on the user's entire month's buying pattern.

Coupon generation will be done in two steps.

- k-means - By applying K-means unsupervised learning, we can form small clusters of our user base. This will group all the users who have similar tastes, or similar purchasing power, or something else common among them based on their user profile.
- Collaborative Filtering - Then in each cluster, we can generate coupons for each user by using Collaborative Filtering. For example: Let us say user A usually buys lots of ice creams. We can look at other users in the same cluster and see what they are buying. Find users who also buy ice creams. Then find other products that matching users are purchasing that our user A has not been buying (for example cake). Like this we can collect all the recommendable products. Then we can calculate probabilities of each recommendable product by using Naive Bayes rule. Then we will generate coupons for user A for the products that have the 3 highest probabilities.

Next time the user comes to the store and is making a purchase, checkout counter system will print coupons and hand over to him.

Whenever the user uses the coupon in the store, our system will make note of the purchase and exclude this product from the recommendable products list for that user. This coupon has done its job of introducing a new product the user. Its job ends there. We don't want to regenerate coupon for this item again.

The metrics such as precision and recall is useful to assess the quality of a recommendation method. Informally, the performance of the recommendation system can also be measured by counting the number of coupons that were actually used by the user.

Answer 7: At Dream Company

Q7: Pick a company of your choice and briefly describe a hypothetical Machine Learning Engineer role at that company you would like to apply for. Now, if you were hired for that position starting today, how do you see your role evolving over the next year? What are your long-term career goals, and how does this position help you achieve them?

I have always been inclined towards academia. Learning and Teaching is something that I enjoy. My colleagues and friends have always flocked to me for solutions. I also like to see the practical application of what I learn/research make real difference in the real world. Ubercity is one such company, which I think offers a right balance of both the worlds.

In a short period, Ubercity has become a global company. Its users span entire global. With such a global footprint, the amount of data its systems produce is enormous. There is so much that can be learnt from its data and improve Ubercity's offerings. The ultimate goal is to better serve its students. For example: The important insight that we can learn from the data as Machine Learning engineer at Ubercity is areas where some students are struggling to learn. What steps can be taken to ease the learning curve? We can text analyze course forums and identify topics that are most confusing for students and then take remedial actions. Another area of improvement is grading the projects/papers. This can be automated somewhat as well with appropriate text analysis. At the minimum we can detect plagiarism, if we want to. Not necessarily as a tool to punish someone, but as additional datapoint for our internal analysis. Other obvious Machine Learning tasks are monitoring the server logs, detecting early on any degradation in quality of service etc. We can generate reports to find out what is working well and things that need improvment.

The number of courses offered will only skyrocket from here. I can totally see that we will offer new courses related to Healthcare and advanced Genetics shortly. IoT and Home Automation related courses are no brainer. I also envision that nature of courses will morph quite a bit. From Audio Video training + Capstone model, I think Ubercity will organically transform into more hands on type of model. True Learning happens when students actually do things. May be small chemistry/dna kits (health), or small programmable robots (autonomous), or raspberry pi kits (IoT) will be integral part of our future courses. I can see myself designing such courses and keeping tab of ever increasing user base needs with appropriate Machine Learning techniques. Without appropriate automations to detect and fill gaps in student learning, Ubercity's rapid expansion can be painful. I see lot of work to be done in coming years. Let us get Started!

I'm super excited about all the recent changes that are happening in the areas of AI, ML, RL, CNN, CV, NLP, and Robotics. I've decided to make a career change and be part of all of it. I'm on my way to enter this new field with full energy. I want to make a big difference in a niche. I see myself owning an entire problem end to end and solving it with a small fun team (engineers, designers, teachers). Together we will change a small part of the world in a big way. It will be our own startup within an org. Everything else in life will automatically follow, be it remuneration, position or career.

Hope to see you soon! Toodle-oo!