Machine Learning Nanodegree

ML Interview Practice Project

Question 1

Taking a quick look at these numbers, I can't say with much confidence that either A or B is more effective in causing users to want to sign-up for the product/service. For one, the sample sizes for both group A and group B are quite small, so its unwise to draw strong conclusions from these results to begin with. A hypothesis test for differences between population proportions would support my case. We can consider this a two-tailed test, since we aren't trying to say that style B is more effective than style A or vice-versa, but just that one of them is a *better* choice.

For our null hypothesis, we are going to assume that in fact the true proportion of users that sign-up for the product is the same in both group A and B:

$H_o : p_a = p_b$

We are looking for a result that will allow us to reject that hypothesis so we can determine which style attracts more sign-ups. The alternative hypothesis that would permit that is:

$H_a : p_a \neq p_b$

To test the null hypothesis, let's *assume* the true population proportion of users that sign-up for the product is just the weighted mean between p_a and p_b:

$$\hat{p} = \frac{Y_a + Y_b}{n_a + n_b} = \frac{20 + 15}{100 + 70} \approx 0.206$$

Now let's use this weighted proportion along with the known data to generate a Z-statistic from which we can derive a p-value:

$$Z = \frac{(\hat{p_a} - \hat{p_b})}{\sqrt{\hat{p}(1-\hat{p})(\frac{1}{n_a} + \frac{1}{n_b})}} = \frac{0.2 - 0.214}{\sqrt{0.206(1-0.206)(\frac{1}{100} + \frac{1}{70})}} \approx -0.227$$

On a normal distribution table, $P(Z \leq -0.227) \approx 0.41$. We need to double this since we need to consider $P(Z \geq 0.227)$ as well. With that said, we are looking at a p-value of 0.82. In other words, there is an 82% chance that that one could observe differences at least as extreme as those shown in the sample data even if there were no true difference in the proportions at the population level. P-values have been criticized by some in the scientific and mathematics community for being carelessly applied, however in this case the p-value *far* exceeds any reasonable significance level (0.01, 0.05, 0.1), so we cannot confidently say that either style A or B is better with the data given here.

Question 2

A useful way to group tweets would be to cluster them by topic, so we can see which tweets are similar to each other by their content. A straightforward way to parse incoming tweets would be label them by hashtag, however this approach assumes that each tweet has one or more hashtags, which is not necessarily the case. What could be done instead for tweets that don't have hashtags would be to apply part-of-speech (POS) tagging, and then retrieve the nouns and verbs - the content words - and discard the rest. After transforming the tweets in this way, each one would have a user_id, a time-stamp, and a label. As we went about trying to cluster the tweets by label, we would quickly discover that there are thousands upon thousands of different labels, so meaningful clusters would never materialize. In order to create topics, what we need is a way to group labels that have similar meaning and/or functions into discrete clusters.

To accomplish this, I would download a pre-trained English word embedding scheme that a major company such as Google or Facebook has made available on Github (https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md)*. I'd then replace each text label with a 300-dimension word vector directly drawn from the scheme. Once I had a matrix of all the word-embeddings in the labels data, I'd apply t-distributed stochastic neighbor embedding (t-SNE) to reduce the feature space into two dimensions. I'd then use K-Means clustering to group words by similarity in the space. I'd optimize the number of clusters by experimenting with different values and checking the silhouette scores of the clusters that were produced. After settling on clustering scheme, I'd create a visualization of the clusters. If the process was successful, then the clusters would have some correspondence to the real world, for example representing topics such as "food", "sports", "music", "politics", etc. Once I could explain the data both quantitatively and qualitatively, I'd present it to our marketing department so that they could use it to better understand the kind of broad topics that users in the dataset like to create discussions around. If these users were our customers or potential customers, we could use insights derived alongside additional data that we gather to build a system that recommends them content, services, and products.

* *In a real interview, I wouldn't know the source off the top of my head, but at least that they are readily available if needed.*

Question 3

Since the question specifically states that I should devise a strategy to prevent overfitting *given a machine learning model*, then I interpret that as meaning that I couldn't have free choice over the model to use in the first place. I need to apply general techniques that could work well to reduce overfitting with *any model*.

Overfitting is prone to occur when either the model is too complex or the dataset is too small. Larger datasets allow for more complex models, but before I even consider the model, I'd need to ascertain whether not we have enough to sample data to create a model that generalizes well to the population from which the data was derived. If the dataset was too small, I'd consider going about acquiring more data. If that was unfeasible, then I would know going forward that overfitting is a high risk, and be able to take appropriate measures to reduce it. I'd want to reduce the dataset's feature space to only a few major features that have major explanatory ability. To accomplish this, I could use decision trees to perform feature selection, or dimensionality reduction techniques such as PCA to create a new and more compact feature space out of the original one.

Next, I'd transform the data so that the scales of input features wouldn't determine their relative impacts on the output predictions. I'd then split the dataset into 80% training and 20% test. This exact split isn't a hard rule and might change according to the size of the dataset, but it's a good place to start. I'd then perform K-Fold cross-validation with the training set, splitting the data into 10 folds, training on 9 of them, and validating on the leftover one for 10 consecutive runs. At this stage, I'd run the model with default settings and wouldn't focus much on accuracy. I'd instead compare the training performance to the validation performance by plotting both of them on a learning curve. If I found that training accuracy improved with the number of training samples, but the validation accuracy "peaked out" after a certain point, then I'd know that overfitting is occurring, and need to make appropriate adjustments, such as reducing the dataset's feature space further. Once I was sure that I couldn't do more to prevent overfitting to on the data side, I'd turn to the model. Most machine learning models have one or more hyper-parameters that can be tuned to adjust complexity. For decision trees, it's the maximum depth of the tree. For K-Nearest-Neighbors, it's the number of neighbors. For logistic regression and support-vector machines, it's their regularization parameter. Given a model, I'd create a list of 10 or 20 values that the hyper-parameter could take, then create a validation curve, plotting training and validation performance at each value. On the plot, I'd be looking for the "sweet spot" - the *maximum point* for which the training and validation have *minimal divergence*. Here, I could be confident that accuracy was maximized while overfitting was limited. The ultimate degree of overfitting could be detected in performance on the test set, but by that point it would be too late to prevent it.

Question 4

The fact that this problem has a state-action procedure points me more towards some kind of reinforcement learning approach, however I think that would be a bit overkill for this problem. In fact, we wouldn't need to devise a learning system at all for this task - a simple weighting and ranking system would serve it much better. Considering that a smart-context menu needs constant refreshing

each time a new action is used, one of the biggest priorities is designing a system that doesn't not add much computational overhead to the software. To this end, applying machine learning be counter-productive and excessive. We need something light-weight that smoothly runs in the background and does not interfere with the user experience first and foremost.

To implement this, each user's context-menu would start with the same default actions that we decide on in advance. When they first right-click, only essential options such as cut, copy, and paste would be available. The software would also have a hidden log that maintained the history of all advanced actions, such as Edit-Surface-Smooth Surface, that the user has taken since activating the software. Because one's usage behavior can frequently change over time, more recently used actions would have more weight than past ones, and in fact weight would degrade exponentially of the form $\{weight = 2^{-t}$; where $t = 0$, $1, \ldots\}$. A particular action's score for any given day t in the log would be $\{score_t = n_t * weight_t$, where n is the number of times the action was used on that day$\}$ For example if I used Edit-Surface-Smooth Surface eight times today $(t = 0)$, then its current score in the log would be eight $(8 * 2^{-0} = 8)$, however the day after tomorrow, it would be only two $(8 * 2^{-2} = 2)$. We would increment t only for days that the software was actually used, so that a user's action scores didn't diminish into nothingness while they were on vacation. Finally, each action would be given a "grand score" which is the sum of its action scores over all days m: $\{grand = \sum_{t=0}^{m} score_t\}$. In all likelihood, we would truncate the log at $m = 30$ so that space requirements don't ever become an issue.

Using *grand*, we could easily rank all the advanced actions to decide which ones to insert into the context-menu for any given context. Each time the user entered a new context, an agent would go into the log and "highlight" all the actions that are possible to use. Lists of possible actions for each context would have already been assembled by previous developers. Since each action's *grand* would automatically be calculated and updated each time that action was used, the agent could simply choose from the highlighted actions that have highest *grand* score to send to the menu. A fixed amount of advanced options would be included in each context-menu, say three or five, so that it didn't get too cluttered. During testing, if we found that the constant updating was causing performance issues, then we could instead have the log only perform score updates at startup instead of each time an action was selected, although this would reduce the agent's responsiveness to changes in user behavior.

Question 5

Regularization makes sense in situations where a dataset has a large number of features, say 10,000, and we'd like to prevent overfitting by having "distributed" contribution towards the model's predictions. This is especially important if the dataset isn't particularly large or completely representative of the target population to begin with. It's possible that as our model started optimizing its

weights, a few weights could "explode" and start having outsize contribution on predictions. However, we didn't put together training data with 10,000 features just to have predictions more or less determined by 10 or even 100 of them. Instead we'd like to train the model in such a way that it doesn't heavily overweight certain features to the expense of others. Regularization is necessary in an image classification task where the size of the feature space is equal to the number of pixels in an image. It's easy for a complex model to overfit to the training images by "memorizing" an object's most significant features, while ignoring the rest of it.

For example, I could be training a model to distinguish between cats and dogs, where all or most of the dog images feature dogs with long, hanging ears. Without regularization, the model might start to make its decision between cats and dogs solely on the ears by placing massive coefficients on the set of pixels that represents them. It's simple thinking process would be that long, hanging ears equals dog and small, pointing ears equals cat. As one can imagine, this model would fail to generalize well to all dogs. All you would have to do is give it a picture of a chihuahua and watch it predict "cat". Using regularization, the classifier could gain a more complete picture of what a truly constitutes a dog beyond its ears, which would prevent overfitting and lead to better out-of-sample accuracy.

On the other hand, regularization would be counter-productive in a situation where you are confident that your sample dataset is a perfect or near-perfect representation of the population that it comes from and explaining how the model works isn't particularly important. You just want good results. Perhaps you are fortunate enough to be working with a very large and balanced dataset and you want to model the relationship between input and output by any means. In that case, you could be sure that out-of-sample data nearly mirrors in-sample data so wouldn't need to be concerned so much with overfitting. You could instead strive for maximum accuracy, even if it meant creating a somewhat overly complex and elaborate model configuration.

Question 6

Since this is a small business with data only coming from one source - the customers' purchase history, we can go ahead and eliminate building a recommendation system that uses techniques such as collaborative filtering, because such a system is more suitable in situations involving big data coming in from many different sources on a very large number of customers. To use a system effectively, we would need to invest in physical or cloud infrastructure, which is likely beyond our budget to begin with.

A simple approach to implementing a targeted coupon program for customers would be to give each one random coupons based on their purchase history. If John bought cereal two months ago, then perhaps offering him a coupon for

cereal would encourage him to buy it again. This program wouldn't fulfill a business goal such as increasing revenue though, because the customers would in effect be spending less on what they would probably buy at full price in the first place. What we need is a way to drive customers to *new* purchases that they wouldn't make in the absence of coupons, and for them to keep making those purchases after the coupons were withdrawn. In order to do that, we need to take advantage of the relationships between different customers' purchase histories. If the customer data was in a log format, then would first need transform into it a vectorized format in order to perform analysis. We'd make a matrix where the columns are all items in the item catalog and the rows are the customers. For each customer, we'd insert 1s where they bought items, and 0s where they didn't. Then a clustering scheme could be applied, using an algorithm such as K-Means, that tries to segment the customers into groups that share similar purchase histories. Once we had a rough idea of the clusters that existed in the data, we would come up with a list of coupons that would appropriate for each cluster. It's okay if these clusters overlap somewhat, as long as it isn't too severe.

Let's start by making two sets for each cluster. Set A will include items that are common to all members of the cluster, and Set B will include any and all items that were purchased by any and all members of the cluster *minus* all of the items in Set A. Since Set A represents the "core" purchases of a given cluster, we would be be wise not to write coupons on these items, since its likely that the cluster's members would purchase the items without coupons. We can use Set B instead to target coupons to the customers on an individual basis. When deciding what coupons to offer an individual, we could look at their purchase history and compare it to Set B of the cluster that they reside in. We could then pick out several items from Set B that don't exist in the individual's purchase history and offer those to them. Basically, we are trying to offer them coupons on items that that they didn't buy but that other members of the cluster bought. A ideal result would be that the customer goes on to use the coupons to make purchases on their next visit. If a customer didn't use a particular coupon after say three visits, we could be reasonably sure that they don't benefit from the coupon, and therefore would replace it next time around with a new one drawn from Set B.

Since we probably wouldn't be making profit on the coupon purchases, it would not make sense to keep offering the same coupons that the customer used in the past. In order to truly reap the benefits of such a program, a customer needs to make a purchase on an item at full-price that they previously purchased with a coupon. This indicates that the coupon succeeded in driving the customer to a new full-price purchase that they wouldn't have otherwise made without it, so measuring this would allow us to quantify the success of the program. In order to so, purchases made with coupons would not be logged in future data. Future purchase data would only include items made without coupons. We could run an A/B pilot version of the program for a given amount of time, say three to six months, to see if there is a significant difference between Group

A that received targeted coupons and Group B that received random coupons. An A/B test would be useful because there are a variety of exogenous factors that could produce changes in purchase data over time, so by conducting the test in this way, we could better isolate the effect of the targeted coupons on customer purchases. For each customer in Group A, we would compare their previous purchase history with their purchase history after the program and calculate the difference in their total purchases. By averaging these differences for every customer in the group, we could get a sense of the average effect of the targeted coupon scheme on a randomly selected customer. We'd then do the same for Group B and compare the effect to Group A. If the effect in Group A was stronger, then we'd know that we were on the right track to driving more purchases and ultimately more revenue for the business.

Question 7

Going forward as Data Scientist at Kabbage, my primary goal would be to apply my knowledge and skills to improve the company's existing data products and services. Each day, I'd be asking myself, how can we use our data to not only better understand our clients so that we can make appropriate lending decisions, but also how can we help them deploy capital in a way that best suits their specific needs? On one hand, I'd be always on the lookout for new sources of data that could improve our system's modeling capabilities. On the other, I'd seek to always stay on top of the latest developments in the field so that we'd have access to state-of-art technologies and methods. Data science is a rapidly evolving area, so having this kind of nimbleness and flexibility is invaluable. As I settled into the role, I'd more draw on my experience in coordinating research projects to shine light on new aspects of the business that data science could add value to, whether on the sales side, the operations side, or elsewhere. For each new area that I could demonstrate was worth exploring, I'd see if we had data on relevant metrics, and if we didn't I'd devise a way to collect that data. After testing and shipping appropriate predictive models, I'd go about working with the Data Platform Engineers to build pipeline solutions that integrate them into the goals of the overall business.

Over the medium-term I'd like to take more responsibility on the team, especially on the production-side. I envision not only building upon existing projects, but also taking leadership on new initiatives as well. My passion is creating impact for small businesses in the form of products that utilize machine learning, so in that regard I would describe myself as more of an entrepreneur than an employee. A tremendous amount of small and medium-sized enterprises (SMEs) today could greatly benefit having data science impact their businesses, but more often than not they don't have the infrastructure or the know-how to do so. To address that opportunity, I'd like to find a way to take advantage of Kabbage's vast database of SMEs to help position the company as a a supplier of business intelligence services. Beyond lending our clients capital, we'd be able to provide

them knowledge in the form of a powerful analytics platform that they could depend on for their data needs. To this end, my long-term career goal is to be a project manager that oversees development of these kind of end-to-end projects that utilize data science and machine learning. I believe that this Data Scientist position would allow me to obtain valuable experience in working in fast-paced teams, working with massive real-world datasets, and contributing to enterprise-scale systems that impact the lives of real people. I can't imagine a more perfect fit to get me started on that journey.