Changelog:

- Clarified reasoning behind impact that experimental model will make on the evaluation metric
- Specified which hyperparameters of interest would be tuned, how they would be tuned, and included the settings currently being used in our experiments
    - Including: variance for the baseline model, threshold for the experimental classification model
- Ran both baseline and experimental models (homoscedastic vs heteroscedastic) and included plots and metrics comparing the two on the toy dataset
- Included citations from further research done

Motivation:

Churn rate, or the annual percentage rate at which customers stop subscribing to a service, is an important piece of information for companies to gather year after year as a measure of company success, and as a predictor of future performance. With the ability to understand which customers are likely to cancel a subscription, companies can leverage targeted incentives to convince these customers to stay. With churn analysis, understanding the likelihood that a prediction will be correct is critical: labeling a customer as likely to churn when they actually are not will result in wasted resources on incentives, while overlooking a customer who is likely to churn will result in lost business.

For evaluation of our models' performances, we will focus mainly on maximizing the F_β-score, which is a weighted average of the recall and the precision. The recall represents the proportion of customers who churn that our model is able to correctly identify. Maximizing recall minimizes the number of customers lost; customers that leave a service usually leave for good. One customer lost costs more than one distribution of an incentive to a customer who is not at risk of leaving. However, because recall can be maximized by predicting that every customer will churn, we need to balance recall with precision. So, β will be set to be greater than 1 (weighting recall more heavily than precision); the specific value of β is yet to be set, but it will be determined after doing further research into the corresponding value of a lost customer versus the cost of providing incentives for customers believed to be leaving a service in the context of telecom companies. We will show that any significant differences in the F_β-score between our baseline model and our experimental model are due to the modeling of heteroscedasticity by tuning the hyperparameters of each model to find the combination that results in the best performance, before comparing the baseline to the experimental, averaging the results across 10 training sessions each.

Our project's main objective will be to predict as a classification task if a current customer is at significant risk of changing providers or not. Towards this goal, we will train a Bayesian neural network classifier. Specifically, we will explore the effect on performance of modeling the aleatoric uncertainty across our dataset. We will compare the resulting predictions and confidence levels against models that assume a homoscedastic variance across the input

feature space. Specifically, we expect to observe more accurate probabilities associated with our predictions when our model incorporates heteroscedastic uncertainty. We will measure this with calibration plots. We expect our model which incorporates heteroscedastic uncertainty to yield better F_β scores due to its ability to more fully represent the relationship in our data.

Aleatoric uncertainty is uncertainty whose source cannot be explained by the data; we will model it by predicting the likelihood in addition to the classification of customers as “at risk for churning”/“not at risk” as a function of the input features. The motivation behind exploring this heteroscedastic variance in the input data’s feature space is the nature of real-world scenarios: the variance across any classification problem’s input feature space should not remain constant.

Because of the uneven noise we expect across the input features in the dataset, we predict that if we include the modeling of heteroscedastic variance as a function of the input, the model will have a higher F_β-score than a Bayesian neural net that assumes constant variance. We believe that it will also provide a more nuanced confidence of predictions that the model makes, which will be evaluated by using calibration plots. If our model accurately models change in variance across the input feature space, the hope is that the model would be able to ask for more information, or perhaps make more educated predictions about what the level of risk associated with allocating resources towards retaining each customer is and what should be allocated given that risk.

Methods:

We build a BNN which can model heteroscedastic variance in the dataset using variational inference to learn the probability distribution over our neural network's weights. The BNN will accomplish this by having two outputs for every input: the predicted mean $\hat{f}$(X, W, b), and the predicted variance $\hat{\sigma}$(X, W, b) (Kendall, 2017). This predicted mean and predicted variance will be used in our model's likelihood function. We estimate the likelihood of a particular class given feature values (X), and the weights (W) and biases (b) of our neural network as follows. First we get a real valued *s*:

$$s \sim N(\hat{f}(X, W, b), \hat{\sigma}(X, W, b))$$

Then we get a $\hat{p}$ where $\hat{p}$ is a probability between 0 and 1, by:

$$\hat{p} = \text{Sigmoid}(s)$$

$\hat{p}$ represents the probability, based on one sample from the above distribution, that the input features, X, belong to a customer who churned. To get an accurate estimate, we will use Monte Carlo sampling, with T samples. Thus, we will estimate our likelihood as:

$$p(y|X, W, b) = \begin{cases} \frac{1}{T}\Sigma_{l=0}^{T}\hat{p}(s_l) & y = 1 \\ 1 - \frac{1}{T}\Sigma_{l=0}^{T}\hat{p}(s_l) & y = 0 \end{cases}$$

We will pick a threshold, *t**, such that we predict the following where 1 corresponds to churn and 0 corresponds to no churn:

$$\hat{y} = \begin{cases} 1 & \hat{p} > t^* \\ 0 & \hat{p} \leq t^* \end{cases}$$

We will choose *t** by examining precision and recall at different *t** values; similarly to how we will be comparing results from and selecting other hyperparameter values, we will choose the value of t* that maximizes the F_β-score.

We define the prior distribution over the weights and biases of our neural network as follows:

$$p(W, b) \sim N(0, 1)$$

We estimate our posterior distribution with variational inference. We define our variational distribution *q ~ N(m, s)* where *m* is a vector the same size as the number of trainable parameters of our network representing the means of the trainable parameters, and *s* diagonal matrix, whose diagonal is the same size and represents standard deviations of our trainable parameters. We will use as our loss function the negative of the ELBO function, which is an upper bound on the KL divergence between our variational distribution, *q*, and the actual posterior distribution of our model. We will use gradient descent on our loss function, defined below, to find the *m*, s** which minimize loss. We use the reparameterization trick in our

estimation of the ELBO function to give a low variance estimate of the gradient of the ELBO (Kingma & Welling, 2014).

$$\mathcal{L}(m,s) \quad = \quad -\mathbb{E}_{q(w,b|m,s)}[\log p(y|x,w,b) \; + \; \log p(w,b) \; - \; \log q(w,b|m,s)]$$

By using a likelihood which incorporates heteroscedastic variance into our ELBO function, we hypothesize the BNN will better capture aleatoric uncertainty in the data than if our likelihood assumed constant variance; thus, we believe this model will lead to improvements in accuracy. In order to test this hypothesis, we will compare our results against a baseline BNN with identical architecture, but which assumes a constant variance in its likelihood.

For our baseline, our neural network will only predict one value, the predicted mean, $\hat{f}$(X, W, b). All calculations will be the same, except, our likelihood will assume a constant variance, so we calculate *s* as follows:

$$s \sim N(\hat{f}(X,W,b),\sigma)$$

Where σ is a constant. We will try three different values for σ : 0.5, 1, and 1.5.

Experimental Setup:

The dataset we are using was originally made public for a data science competition. As part of the competition, the performance of multiple baseline models on the dataset, including the company's existing solution, was made public. Both BNN models will also be compared to these models in order to validate variational inference against other machine learning techniques.

We will use the Orange Telecom dataset from the 2009 Knowledge Discovery in Data Competition. For this competition, a French telecom company, Orange, provided customer data with 230 features and a label for whether or not each customer will churn. The dataset has 50,000 entries with imbalanced classes. 7.5% of the instances belong to the churn class, while 92.5% will not churn. The company inflated the proportion of people who churn in the sample, so it is actually higher than under natural circumstances. In order to protect the privacy of its customers and its proprietary information, the company removed all variable names, encoded all categorical features, and multiplied all continuous features by a scalar value. Although we do not know what any of the features in our dataset represent, this dataset is ideal to explore the effect of modeling aleatoric uncertainty, as it is large enough that issues arising from epistemic uncertainty should be minimal.

For many of our models' hyperparameters, we will use the same tuning procedures across models. In both our focus and baseline models, we will gridsearch our learning rates optimizing for F_β, and, for simplicity, we will fix the mean and variance of our prior at mean=0, variance=1 for the weights and biases of our model. For both models, we will test pytorch optimizers including SGD and Adam. We will leave the number of Monte Carlo samples fixed for both models for simplicity. We will test two architectures for our focus model. Both will have 5 hidden layers, but one will share the final hidden layer between both heads of the network, while the other will have 4 shared hidden layers, with the final hidden layer independent for each head of the network. For simplicity, we will not vary hidden size and all hidden layers will have hidden size of 25. Our baseline model will have 5 hidden layers with hidden size of 25 to ensure the two networks are of comparable complexity.

We will stratify our data when we split into training and test sets, and because our dataset is large, we will use a 90 : 10 train test split. We will run our variational inference until the difference in loss between epochs falls below 0.05%.

The baseline models mentioned in the report on the Knowledge Discovery in Data Competition are evaluated using AUC. We will use AUC as a secondary metric so that we can compare our results with results previously published on this dataset. We will create precision/recall (PRC) plots to show the precision and recall of our models because these curves are shown to be most effective on imbalanced datasets (Saito, 2015). As described in the motivation, so we will seek to maximize the F_β-score, with a β value that balances the cost of a

churn with the cost of providing an unnecessary incentive to continue services. This will allow us to compare the performance of our models, and understand whether accounting or heteroscedastic variance in our dataset results in more desirable predictions.

In order to get an approximate evaluation for how accurate the class probabilities generated from our model are, we will use calibration plots. To create these plots, we will group our predictions by ranges of probabilities assigned by our model, and then count the number of instances from each group for which our model made the correct classification.

We will be using pytorch to implement our program. The variational inference will be written by ourselves.

Experiments

To validate our approach. We tested a simplified version of our model on a toy dataset. This dataset was comprised of X values between -10 and 10 with Y = X - 5 + noise. The noise was equal to samples from a normal distribution ~N(0, x + 10.0000001). The hyperparameters used to train the model are given below.

Hyperparameters	
Batchsize	100
Number of Epochs	100
Learning Rate	1e-3
NN Hidden Layers	0

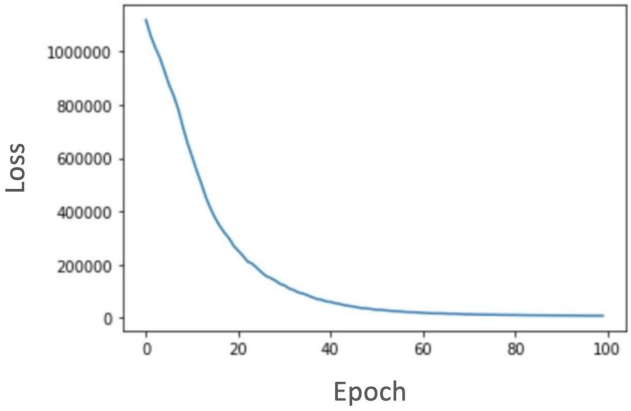

Figure 1: Loss for a linear model on our toy dataset. Hyperparameters used are batchsize of 100, 100 epochs, and learning rate of 1e-3

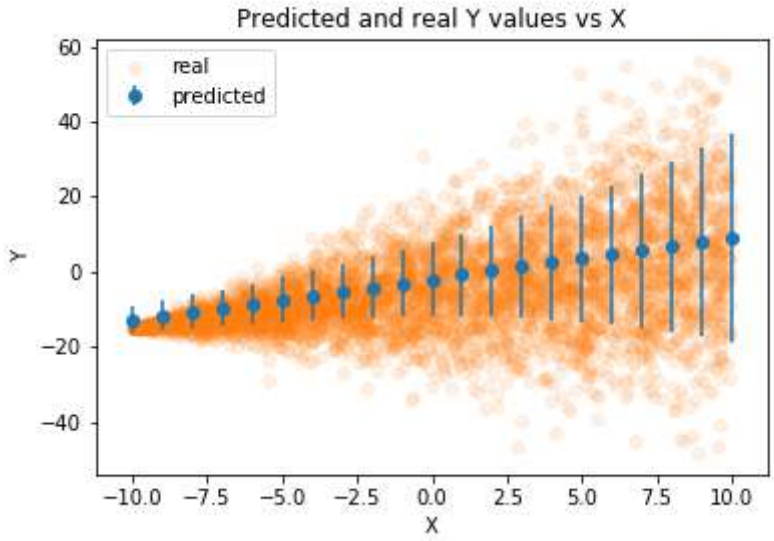

Figure 2: Predicted means and standard deviations plotted against the real Y values in a toy dataset.

After training the model, we obtained predicted means and standard deviations for various X values. In the above figure we can observe these predicted means and standard deviations against the actual Y values. As X increases in value, the standard deviation increases.

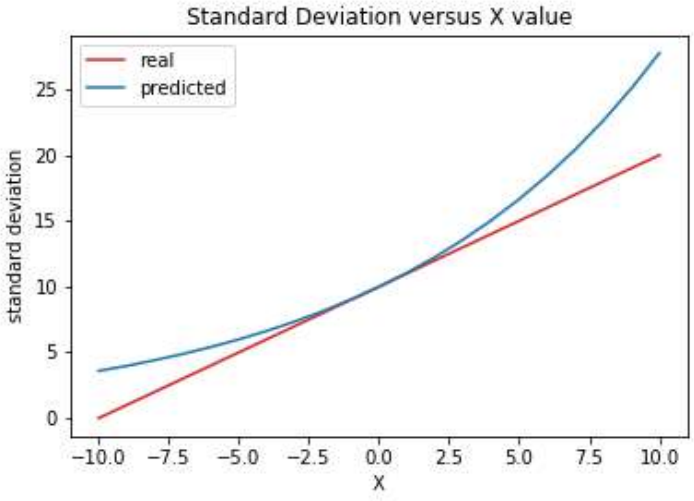

Figure 3: Predicted standard deviations over X values against the standard deviation used to generate the dataset

In figure 3 we explicitly graph the standard deviation predicted from the network at various X values and compare it against the real standard deviation used to generate the data the model trained against. In this graph we can see our model outputs an exponential standard deviation. We believe this occurs because the simplified model is linear and outputs the log standard deviation. Therefore, the predicted log standard deviation is linear and the predicted standard deviation is exponential. After incorporating more layers into our network the output standard deviation will not be linear.

On the same toy dataset parameters, we trained a baseline model that does not model heteroscedasticity. The hyperparameters were kept the same as the set used for the experimental model. The graphed results are as follows:

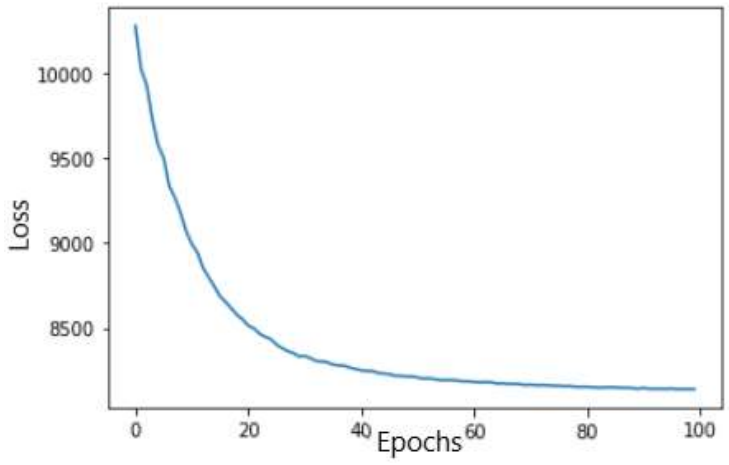

Figure 4: Loss for a homoscedastic linear model on our toy dataset. Hyperparameters used are batchsize of 100, 100 epochs, and learning rate of 1e-3

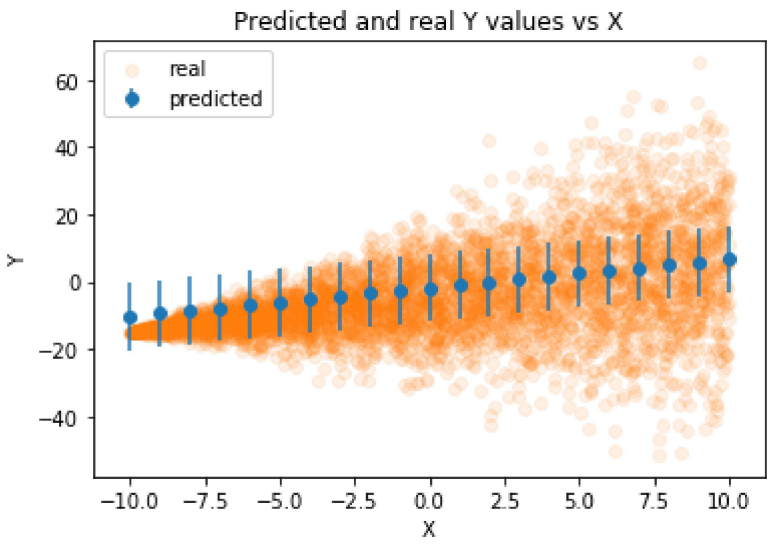

Figure 5: Predicted means and constant standard deviations plotted against the real Y values in a toy dataset.

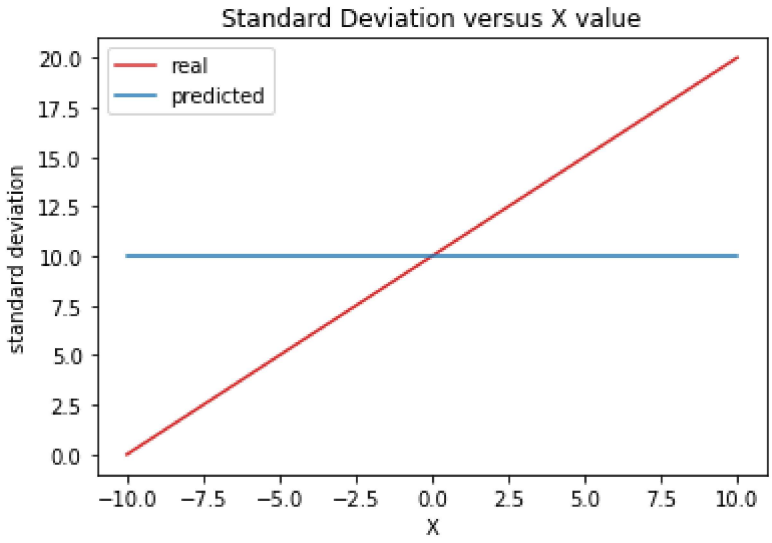

Figure 6: Constant standard deviations against the standard deviation used to generate the dataset

As can be seen by the graphs, the baseline model and the experimental model predicted relatively similar mean values; however, it's clear that the experimental model's learned variance values describe the data more completely. In terms of basic metrics, the baseline model's final batch resulted in a loss value of 8139.12 and a MAE value of 8.81, compared to the experimental

model's slightly higher loss value of 8230.07 and a slightly lower MAE value of 8.60. We have yet to confirm what the best value of β to use for the F_β, but once that is confirmed we will be using that primarily as the metric for evaluating and comparing the two models.

Timeline:

~~10/21~~ ~~Additional literature review, confirm dataset~~

~~10/28~~ ~~Noise modeling research~~

~~11/04~~ ~~Noise modeling implementation w/ simple model~~

~~11/18~~ ~~DNN prototyping~~

~~11/25~~ ~~DNN prototyping p2~~

12/02 Final Presentations

12/18 Paper

Looking Ahead / Potential Roadblocks:

We had a lot of trouble getting a working linear regression with heteroscedastic variance. Thankfully we've completed that task in this checkpoint. The remaining steps involve converting this to logistic regression, increasing the complexity of the model, and applying the complex model to the actual dataset. We have already written the code for the logistic regression. With that in mind, of the three tasks, we suspect increasing the complexity of the model will be the most precarious part of our remaining work. We've spent more time reading additional journal papers related to churn prediction, but many of these have had different enough datasets that it's not possible for us to copy their network architecture and retrofit variational inference to it. Because of that, our current plan is to simply increase the number of nodes and layers arbitrarily until our model fails to improve. Once this happens, we will start tuning hyperparameters in earnest to find the truly optimal set of hyperparameters for both the baseline and experimental model to compare the two for evidence of significant results.

Kendall, A., & Gal, Y. (2017). What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? *Advances in Neural Information Processing Systems 30* (pp. 5574–5584).

Kingma, Diederik P, and Max Welling. (2014) “Proceedings of the 2nd International Conference on Learning Representations.” ArXiv, *Auto-Encoding Variational Bayes*

Saito, T., & Rehmsmeier, M. (2015). The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE*, *10*(3), e0118432. https://doi.org/10.1371/journal.pone.0118432

Vafeiadis, T., Diamantaras, K. I., Sarigiannidis, G., & Chatzisavvas, K. Ch. (2015). A comparison of machine learning techniques for customer churn prediction. *Simulation Modelling Practice and Theory*, *55*, 1–9. https://doi.org/10.1016/j.simpat.2015.03.003

Zhang, R., Li, W., Tan, W., & Mo, T. (2017) "Deep and Shallow Model for Insurance Churn Prediction Service," *2017 IEEE International Conference on Services Computing (SCC)*, Honolulu, HI, pp. 346-353. https://doi.org/10.1109/SCC.2017.51