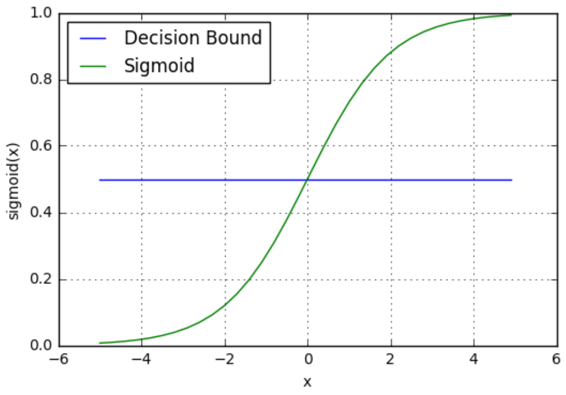
Data Cleaning and Preprocessing

This was a fun and interesting dataset to work on, with multiple areas that needed tweaking and pitfalls during the data preparation and cleaning process. This data set was cleaned and models were produced as a fun project. This data is completely made up/fabricated (not by me) so it is hard to draw real world predictions out of it. The fact that this data is random also plays into feature selection and the cleaning of the data. Since the data is seemingly made up, especially the categorical variables, I did not do any feature engineering. Instead I just cleaned up the data and replaced any missing values with the most common categorical variable, or for integer data types, replaced it with the mean. All of the categorical variables were then feature engineered by One-Hot Encoding.

I tried to do dimensionality reduction using principal component analysis (PCA). However, the principal components which describe 90 percent of the data only end up reducing the dimensionality by around 25%, which in this case is not appreciable. This means that the variance within each feature of the dataset is quite low, with no singular dimension being able to describe a majority of the variance. Furthermore, employing the following algorithms on the reduced dataset did not appreciably improve the AUC scores, which is how this dataset is going to be evaluated. Also, other standard classification algorithms were used including ensemble methods (Random Forest), boosting methods (Gradient Boosting and Adaboost), and standard probabilistic classifiers (Gaussian Naïve Bayes and Maximum Likelihood), but none of them worked as well as Logistic Regression and a Support Vector Machine Classifier (SVM). Clustering algorithms were also thought of, but due to the high dimensionality of the data it is most likely going to be prone to high False Positive and False Negative rates—which should be avoided given the scoring system.

Machine Learning Algorithm 1 – Logistic Regression

Logistic regression is a commonly used linear classification technique to draw a decision boundary. Due to the sigmoidal nature of the function, it works remarkably well when dealing with binary classification. Furthermore, the decision boundary can be adjusted as to whether the algorithm classifies an instance as a 0 or 1, which can be useful in many classification situations. It is defined as the following:



$$Eq.1 \quad g(x) = P(y = 1|x;\theta) = \frac{1}{1+e^{-x}}$$

$$Eq.2 \quad h_\theta(x) = g(\theta + \theta x_1 + \theta x_2 \dots \theta x_n)$$

Logistic regression will output the probability of a prediction being associated with a positive outcome (Eq.1). For instance, if the algorithm predicts the output of the sigmoid function to be 0.35, and the decision boundary, as seen in the blue is 0.5, this would then be classified as having failed and then set to

0. This decision boundary is drawn by our hypothesis function, Eq. 2, where θ represents weights (parameters) that can be learned.

These parameters, θ, are learned by utilizing a cost function and gradient descent. When doing linear regression, one usually minimizes the cost function by using mean squared error. However, due to the non-linearity of the logistic regression function this would cause local minima to arise during gradient descent. To avoid this problem, logistic regression utilizes a different cost-function:

$$Cost(h_\theta(x), y) = -ylog\left(h_\theta(x)\right) - (1-y)\log(1-h_\theta(x))$$

Not only does this deal with the nonlinearity problem, but this cost-function also has the property of penalizing wrong predictions. For example, if y = 1, but our hypothesis is predicting zero, the cost function would equal infinity. The inverse is also true, in that the closer our prediction is to the actual, it minimizes the cost function. If y = 1 and our hypothesis is predicting 1, then the cost function would equal 0. It does the same thing when y = 0, penalizing wrong predictions while rewarding the others. We would then need to calculate this cost function for all the parameters θ and then update them for gradient descent to occur. While I will not go into gradient descent here, or it's other incarnations, it is the fundamental way in which machine learning algorithms optimize parameters to fit a local minima in its cost function. This can be described for the case of Logistic Regression as:

$$Calculate\ Cost\ Function\ of\ Parameters$$

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^i \log\left(h_\theta(x)\right) + (1-y)\log\left(1-h_\theta(x)\right)\right]$$

$$Update\ Parameters\ with\ Gradient\ Descent$$

$$\theta_j := \theta_j - \alpha\frac{\delta}{\delta\theta_j} = \theta_j - \alpha\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^i\right)x_j^{(i)}$$

Pros of Logistic Regression

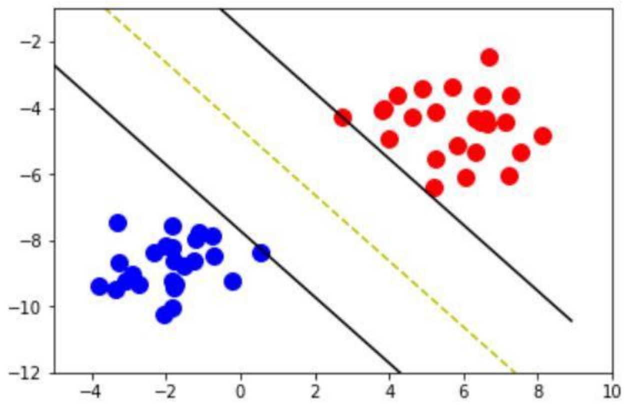
- Logistic regression is computationally relatively inexpensive when only a few binary classifications are being performed, with many vectorized implementations already available in many popular libraries and languages.
- Due to the maximum-likelihood implementation of the cost-function and the output of the sigmoid function, logistic regression can output probabilities of a class. This is particularly useful when showing how sure one is of a prediction and would be a great tool especially in the insurance industry when analyzing risk and can also be used for ranking.
- You can easily introduce both L1 (Lasso) and L2 (Ridge) regularization in logistic regression to combat over-fitting of high dimensionality datasets.

- One can change the cutoff threshold, which can influence precision and recall of your model.

Cons of Logistic Regression

- Logistic regression is very sensitive to outliers. It may change your model significantly due to the robust cost-function and sigmoidal nature of the activation function.
- At the heart, logistic regression is a linear model. Complex non-linear decision boundaries cannot be implemented.
- It becomes very computationally expensive when trying to classify more than two or three binary classifiers and simply does not work well. If there are 5 binary classifiers, then there would be 2^5 or 32 different states to predict and other methods may work faster and better. However, for simple binary classification it is very good.

Machine Learning Algorithm 2 –Support Vector Machine



Support Vector Machines, are known as large margin classifiers that try to draw a decision boundary such that it maximizes the space between the two classes. Using kernels, or a similarity function used to project your features into higher dimensional space to calculate the decision boundary, which can then be mapped back onto lower dimensional features, this allows SVMs to be a powerful non-linear classifier. In practice SVMs utilize a very similar cost function to logistic regression. Keeping in mind the cost function for logistic regression, as seen above, the cost function for SVMs is:

$$min_\theta\ C\sum_{i=1}^{m}\left[y^{(i)}cost_1\left(\theta^T x^i\right)+\left(1-y^{(i)}\right)cost_0\left(\theta^T x^i\right)\right]+\frac{1}{2}\sum_{i=1}^{m}\theta_j^2$$

Where: $cost_1\left(\theta^T x^i\right)=\max(1,-x)\ when\ y=1\ and\ cost_0\left(\theta^T x^i\right)=\max(-1,x)\ when\ y=0$

These are ReLU functions which have become popular activation functions in the field of Deep Learning. However, one can notice the similarity in the cost function with the classification variables of y being set similar to logistic regression. The intuition behind this, for example, is that when y = 1, the cost will be 0 if $\theta^T x^i(prediction) \geq 1$. If it is less than 1, the cost function increases linearly. Like logistic regression, the cost function in SVM penalizes incorrect predictions, while rewarding correct ones. One parameter that is unique to SVMs is the C parameter. As C gets bigger, the space that is in the margin of your decision boundary gets smaller. This could be a good thing is some cases, as it will let one fit more complicated decision boundaries, however it could also be a case for overfitting of the data. So, care needs to be taken that the correct value of C is chosen.

An interesting intuition into how SVMs choose their decision boundaries, lies within their vector optimization of the above cost functions. If one projects the feature space (x) as individual vectors onto the vectorized parameter θ, it turns out that the minimization of the cost function above will correspond to the choosing of a correct decision boundary. This changes the $\theta^T x$ term above, to a vector feature projection of the training example onto the parameter θ, $p^{(i)} * \|\theta\|$. When this is minimized, it corresponds to the largest overall distance between the decision boundary and the various classification points.

Pros of SVMs

- Have been largely optimized to run quickly and efficiently and is available in a multitude of libraries.
- Can fit complicated non-linear boundaries and simple linear boundaries.
- Can easily handle multiclass classification problems
- Parameter tuning is quite simple, with the main parameter to tune being C. Depending on kernel used other parameters will need to be tuned as well.
- It is robust to overfitting, as it is in the nature of the algorithm to maximize the margin between the decision boundary and your classes.
- Can be used to tackle several different problems, and even has implementations for regression.

Cons of SVMs

- Since it is built very similarly to logistic regression, outliers can have a large effect on your data and should be considered carefully.
- Need to be careful when approaching a problem to know the correct kernel to use.
- Does not natively output probabilities (but sklearn has a built-in function to carry this out, but it is slow).

Steps to Improve the Models (or make a new one)

Utilizing a randomized cross validation approach in parameter tuning, allowed me to rather quickly build robust models. The SVM seems to be exceptionally strong when compared to my development data set and calculating the AUC of the ROC curve (0.96). To me, this would indicate that there is non-linearity within the data, as the logistic regression algorithm did not fare as well (0.79). Utilizing the time and the tools that I have available, I would choose SVM over the logistic regression algorithm. It is interesting that other ensemble methods like Random Forest were not able to work well in this dataset. It would be quite interesting to examine this problem further.

To try to mitigate the difference between the much higher precision and lower recall that is observed in both of the models, I propose that I could change the ratio of the data between the training set and the development set. This may even out the precision and recall for both proposed models. However, since the SVM is already so robust, and the use case is not known in this example (how important are false-negatives?), this may not be needed. Furthermore, something else that could be done with the data is examine it closely for outliers. Both of the algorithms are sensitive to outliers, so removing some may especially help the logistic regression.

Looking to the future, I believe that this dataset would be a great candidate to build a neural network that would classify the data. However, due to my personal technology constraints, a graduate student's budget is unforgiving when it comes to having state of the art laptops, and in the time allotted an NN would not have been feasible. Furthermore, when the SVM is already doing so well at the prediction, I did not see a reason to try and implement it. While it would be an interesting intellectual endeavor, and one that I will surely build, for the sake of this work it seemed unnecessary.

References:

1) Andrew Ng's Machine Learning course and the chapters on Logistic Regression and SVM. (equations and most of the material)
2) Logistic Regression Picture. https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
3) SVM Picture. https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be