2017

# Ensemble Classifiers – Stacking Ensembles

CSC529 – CASE STUDY 1

Gunjan Pravinchandra Pandya

ADVANCED DATA MINING | Jonathan Gemmell

# 1. Introduction

As a part of CSC529 – Case Study 1, I am going to work on the Titanic: Machine Learning from Disaster Dataset from Kaggle. In this dataset, the objective is to create a machine learning model to predict the survival of passengers of the RMS Titanic, whose sinking is one of the most infamous event in the history. While there are number of machine learning branches or topics, the one I will work on is “Supervised Learning” where we assign each of the records to a pre-determined set of categories. So, the goal of creating a machine learning model here is to successfully classify which of the passengers will survive and which will not. The dependent variable or the variable of interest if “Survived” whose possible values are 0 and 1, where 0 is a negative class and indicates passenger didn’t survive and 1 is a positive class which indicates passenger did survive the Titanic disaster.

Based on the given data, my goal is creating a machine learning model using the Stacking technique of ensembles or popularly called Stacking Ensemble, to create an ensemble classifier and analyze how it performs compared to the individual classification algorithms. In stacking ensemble technique, we use the output from various models as features and create a model using those outputs/features to predict the target class. In this case study, I am going to use the class probabilities from each of the individual classifier and create a model using logistics regression on those probabilities, to see how ensemble classifier performs compared to individual classifiers. The class probabilities will come from individual classifiers/algorithms like Two-class Bayes Point Machine, Two-class Boosted Decision Tree, Two-class Average Perceptron and Two-class Decision Forest. I have selected Microsoft Azure ML Studio as a platform to do this case study.

I am going to preprocess and clean data using the inbuilt capabilities of Microsoft Azure ML studio and Python. I will do data analysis using Python as Azure ML studio has limitations on visualization, so I will use an iPython notebook to show the visualizations. I am going to use cross validation over a range of parameters for tuning the parameters of the algorithms, as well as will perform cross validation for reporting accuracies and their confidence intervals. I will use precision and recall along with accuracy to compare various models. This case study will help me to learn about ensemble classifier, the implementation of stacking ensemble and see how it compares with individual classifiers.

A comparison table of Bagging, Boosting and Stacking Ensemble:

| | **Bagging** | **Boosting** | **Stacking** |
|---|---|---|---|
| **Partitioning of data** | Bootstrapping – Sampling with replacements | Misclassified samples are given higher weights/preference | Various |
| **Goal to achieve** | Minimize variance | Increase predictive power | Both |
| **Methods where this is used** | Random subspace | Gradient descent | Blending |
| **Function to combine single models** | Weighted average/majority voting | Weighted average/majority voting | Logistics regression |

*Table 1: Comparison of Bagging, Boosting and Stacking*

## 2. Data Description

Dataset: Titanic: Machine Learning from Disaster Dataset
Format: Comma Separated Values (csv)
Source: Kaggle
Link to dataset: https://www.kaggle.com/c/titanic/data

Description:

- PassengerId: ID of the embarked passenger on RMS Titanic
- Pclass: Passenger Class
    - Values: 1 = 1st, 2 = 2nd, 3 = 3rd
- Name: Name of passenger
- Sex: Sex of passenger
    - Values: Male, Female
- Age: Age of passenger
- Sipsp: Number of Siblings/Spouses Aboard
- Parch: Number of Parents/Children Aboard
- Ticket: Ticket number
- Fare: Passenger Fare
- Cabin: Cabin number
- Embarked: Port of Embarkation
    - Values: C = Cherbourg, Q = Queenstown, S = Southampton
- Survived: Passenger survived or not?
    - Values: 0 = No, 1 = Yes

Special notes on dataset as is from Kaggle:

```
Pclass is a proxy for socio-economic status (SES)
 1st ~ Upper; 2nd ~ Middle; 3rd ~ Lower

Age is in Years; Fractional if Age less than One (1)
 If the Age is Estimated, it is in the form xx.5

With respect to the family relation variables (i.e. sibsp and parch)
some relations were ignored.  The following are the definitions used
for sibsp and parch.

Sibling:  Brother, Sister, Stepbrother, or Stepsister of Passenger Aboard Titanic
Spouse:   Husband or Wife of Passenger Aboard Titanic (Mistresses and Fiances Ignored)
Parent:   Mother or Father of Passenger Aboard Titanic
Child:    Son, Daughter, Stepson, or Stepdaughter of Passenger Aboard Titanic

Other family relatives excluded from this study include cousins,
nephews/nieces, aunts/uncles, and in-laws.  Some children travelled
only with a nanny, therefore parch=0 for them.  As well, some
travelled with very close friends or neighbors in a village, however,
the definitions do not support such relations.
```

Size: Training: 891 rows and 12 columns, Test: 418 rows and 11 columns

- ➔ As the test data set has no column of the target class as it is supposed to be for Kaggle competitions, I am using that data during preprocessing and stuff but not for model building as it has no class labels or as well as not for checking accuracy.

- ➔ There is no source mentioned by Kaggle from where data was collected and there are a lot of missing values and preprocessing possibilities in this one.

- ➔ Before starting the data cleaning and feature engineering I have combined the training and test datasets, using a Python script in Microsoft Azure ML studio experiment and then separated them as test data cannot be used for model building or validation due to absence of labels.

## 3. Data Analysis & Visualizations

➔ To analyze the dataset and see distributions and interactions of variables, I have used iPython notebook as the visualization capabilities of Microsoft Azure ML studio are limited. The analysis and graphs/plots from iPython notebook are shown below. Visualization will also help us decide on some of the cleaning, preprocessing steps which I have performed in the next section based on the insights we obtain here.

Distributions of Categorical variables:

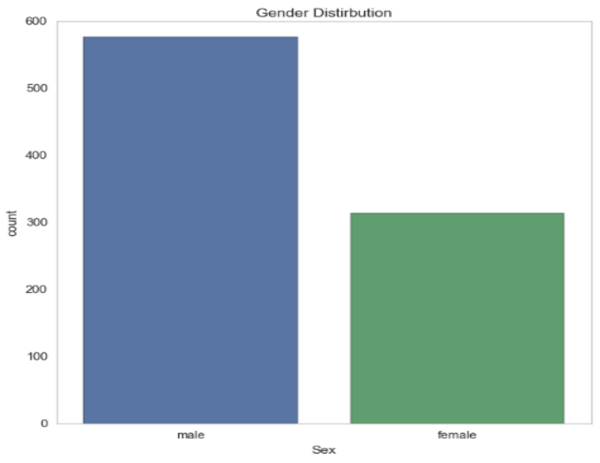
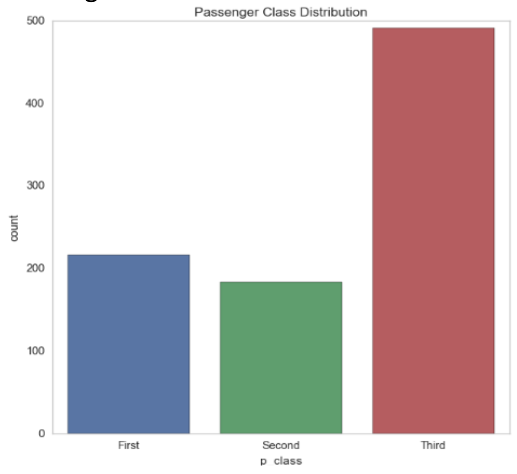
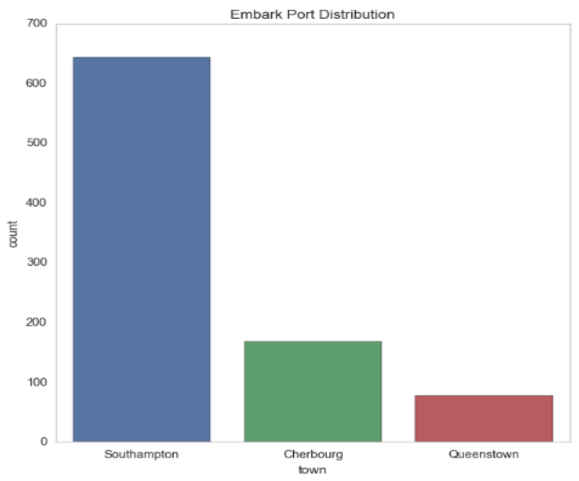
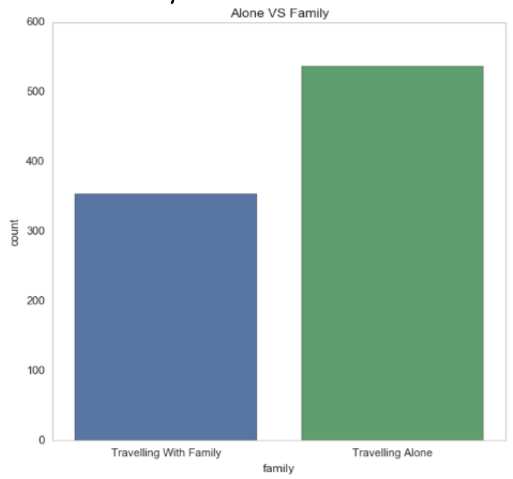
| | |
|---|---|
| Gender Distribution:  | Passenger Class Distribution:  |
| Embark Port Distribution:  | Alone vs Family:  |

*Figure 1: Categorical Variables Distribution and One New Variable*

➔ We can see that distributions of "Gender", "Passenger Class" and "Embarked" is comparatively biased towards the Male, Third and Southampton values respectively. What we can infer is that most of the travelers on Titanic were Male, belonged to Third Class and travelled from Southampton. Also, looking at the special notes we can say that most of the people travelling in Titanic had lower socio-economic status. I have also created a new field, which indicates whether they were travelling with family or along based on the fields “SibSp” and “Parch” and we can see around 350 passengers were travelling with family while approximately 550 were travelling alone.

Survived by variables:

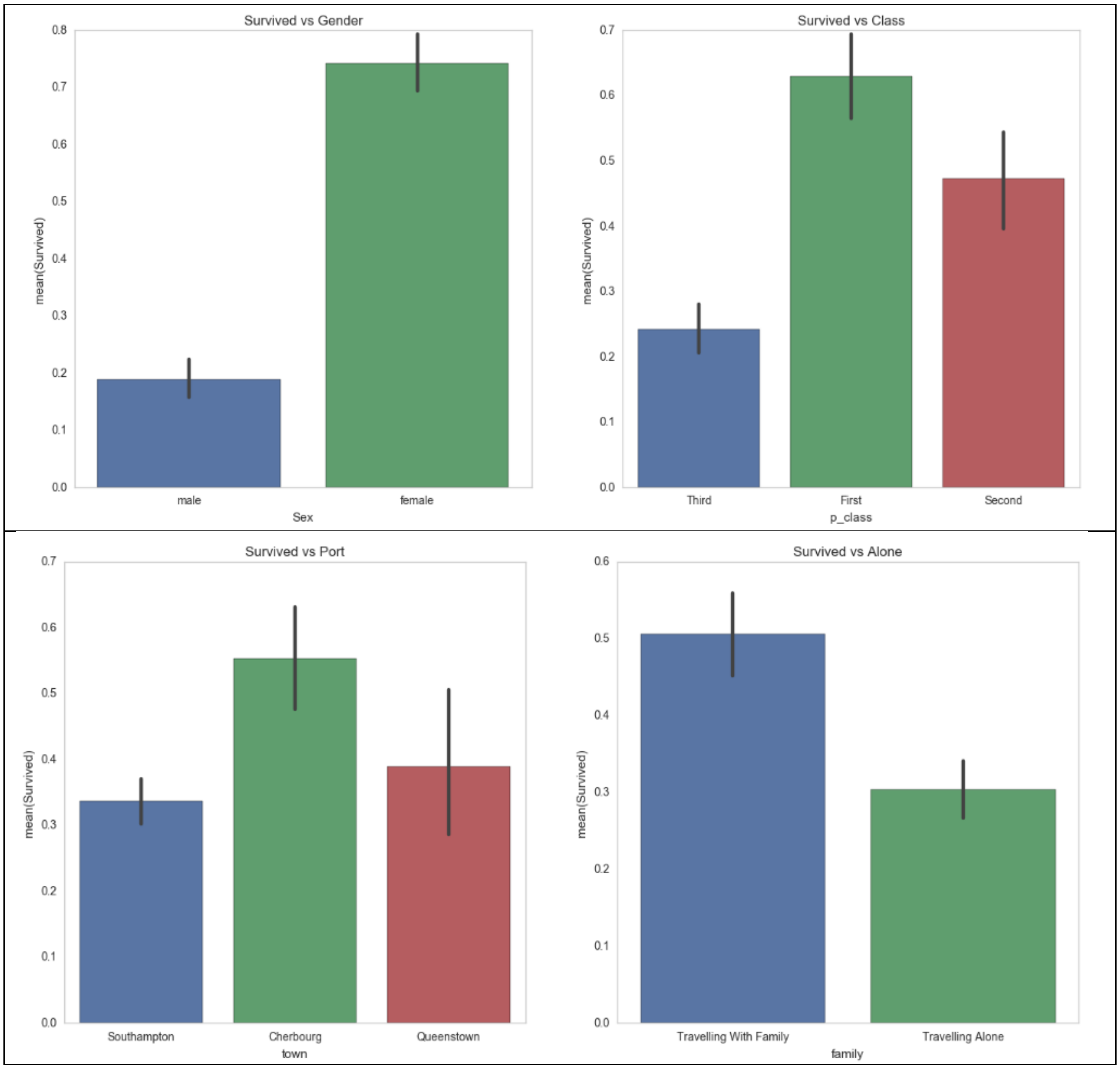


*Figure 2: Survival rate by variables*

➔ We can see from the graphs that the survival rate of Women is higher than Male, passengers travelling in First Class were most amongst those who survived, passengers embarking from Cherbourg were the most in those who survived, also people travelling with family are more likely to survive than those travelling alone.

How do variables interact to affect the survival rate?

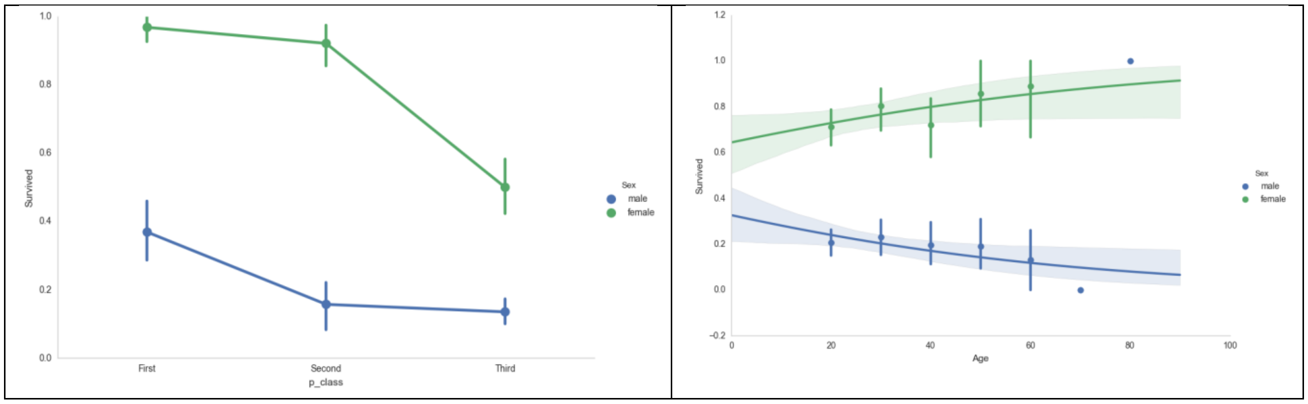


*Figure 3: How Gender and Age/Class interacts to affect Survival rates*

➔ From the plots above we can see that, females were given priority in boarding the life boats irrespective of class and age so their survival rates were higher, so the variable "Gender" will contribute a lot when creating a model.

Looking at the "Survived" Distribution:

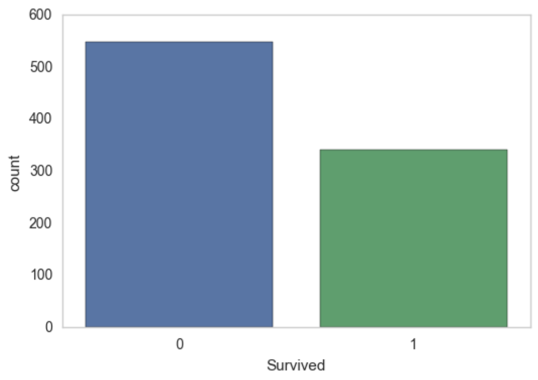


*Figure 4: Distribution of target variable*

➔ We can say that there are more instances of the class 0 which is the negative class and less instances of 1 which is the positive class but the distribution is not as biased and it is balanced.

➔ So, looking at the visualizations we can say that a new variable which shows family size would help predict the survival rates and the class distribution is not unbalance so stratified sampling will not be required and two-class classifiers will work to solve the task of classification.

## 4. Data Cleaning, Preprocessing & Feature Engineering

➔ Dataset has missing values and ambiguous values as well, so there is need of data cleaning and I am creating some new features from the existing features or modifying existing features to enhance the information data facilitates, making it less ambiguous so that I can increase the performance of the classifiers.

➔ For cleaning the dataset, I started with checking the missing values and moving forward from there. Table below shows the number of number missing value counts for each of the column/variables in the dataset.

| Column Name/Variable | Missing Value Counts |
|---|---|
| Age | 263 |
| Cabin | 1014 |
| Embarked | 2 |
| Fare | 1 |
| Name | 0 |
| Parch | 0 |
| PassengerId | 0 |
| PClass | 0 |
| Sex | 0 |
| SibSp | 0 |
| Survived | 0 |
| Ticket | 0 |

*Table 2: Missing value counts*

➔ Data cleaning, preprocessing and feature engineering steps:

1) Fill the missing values in "Embarked" column with mode value "S" = Southampton
2) Fill the missing values in "Fare" column with median
3) "Cabin" column has almost 1014 missing values, so I am recoding that column with just two values based on whether there is value in that column or not, so new values are 0: Missing and 1: Present
4) If we look at the names, we see a lot of titles, so I have processed the name field using a regular expression and have saved the title in a separate new field "Title" and have recoded it in the following way:

| Old Titles | New Title |
|---|---|
| 'Don.', 'Major', 'Capt.', 'Jonkheer.', 'Rev.', 'Col.', 'Sir.' | 'Mr.' |
| 'Countess.', 'Mme.', 'Lady.' | 'Mrs.' |
| 'Mlle.', 'Ms.', 'Dona.' | 'Miss.' |
| 'Dr.' & Gender = Male | 'Mr.' |
| 'Dr.' & Gender = Female | 'Mrs.' |

*Table 3: Recoding for new "Title" field*

After recoding the values in "Title" are: 'Mr.', 'Mrs.', 'Miss.' and 'Master.'

5) Fill in missing values in "Age" column based on "Sex", "PClass" and "Title". I am grouping by the data based on "Sex", "PClass" and "Title" and using the median value of each group to fill in the missing

values of “Age” variable based on values of “Sex”, “PClass” and “Title” for that record. The reason behind doing the group by and then taking the median is that the “Sex”, “PClass” and “Title” together can give a better estimate rather of age than simply taking the median or doing group on one of the fields and then taking median.

6) New variable “Family_Size” which is sum of values in field “SibSp” and “Parch” for each record
7) One-hot dummy encoding for the variables “Embarked”, “Title”, “PClass”, “Family_Size” and “Sex” which are categorical in nature.
8) Min-max normalization for numeric/continuous variables “Age” & “Fare”
9) Removing “PassengerId” and “Name”, as it provides no information for the classification task

➔ After doing data preprocessing and feature engineering, we are separating out the training samples we have as our test data does not have labels, so we cannot use that anymore in the data modelling or for checking the accuracies of the model.

➔ Moving forward we will use 80% of our data for training and 20% of our data for testing. Out of the 80% training data we will use 80% data for tuning the model hyperparameters using cross validation and 20% as a validation set for tuning the model hyperparameters, then we will test the accuracy of model over whole training dataset (80%) from first split and accuracy of model over testing dataset (20%) from first split.

## 5. Experimental Results

➔ The machine learning algorithms I used for this case study are:
1) Two-class boosted decision tree
2) Two-class logistics regression
3) Two-class Bayes point machine
4) Two-class averaged perceptron

➔ In the stacking ensemble technique, I will use "Scored Probabilities" from each of the model and will create a logistics regression model over the "Scored Probabilities" from four models and will try to predict the class label and see the accuracy. I will also tune the hyperparameters of the ensemble model.

The details of the algorithms, range of parameters used, tuned hyperparameters and experimental results for each of the algorithms are:

### 1) Two-class Boosted Decision Tree

Two-class boosted decision tree is an ensemble of decision tress, in which using the boosting concepts, second tree corrects the error of first tree, the third tree corrects the error of the second tree and predictions are based upon the ensemble of trees produced in this fashion until one of the criterion is reached. The two-class boosted decision tree implementation in Microsoft Azure ML Studio is a memory-intensive implementation and the hold tree is hold in memory so it would create memory bottlenecks if it was not being executed on Azure cloud platforms and so it might not be able to handle very large datasets which other linear algorithms can. Also, visualizations are a problem for Two-class boosted decision trees in Azure ML studio as sometimes if trees created are large or more in numbers. In our case, I was not able to visualize the best selected model and its parameters so I am not reporting it.

Range of parameters used:

| **Parameters** | **Range/Values** |
|---|---|
| Maximum number of leaves per tree | 2, 8, 32, 128 |
| Minimum number of samples per leaf node | 1, 10, 50 |
| Learning rate | 0.025, 0.05, 0.1, 0.2, 0.4 |
| Number of trees constructed | 20, 100, 500 |

*Table 4: Range of parameters for Two-Class boosted decision tree*

Relevant performance metrics:

| **Dataset** | **Accuracy (%)** | **95% CI (%)** | **Precision (%)** | **Recall (%)** |
|---|---|---|---|---|
| Training | 83.17 | 83.17 - 83.18 | 0.8113 | 0.7543 |
| Testing | 77.48 | 76.06 – 78.9 | 0.7314 | 0.6047 |

*Table 5: Performance metrics for Two-class boosted decision tree*

Below is ROC Curve for Two-class boosted decision tree, where blue line indicates performance over training data and red line indicates performance over testing data.

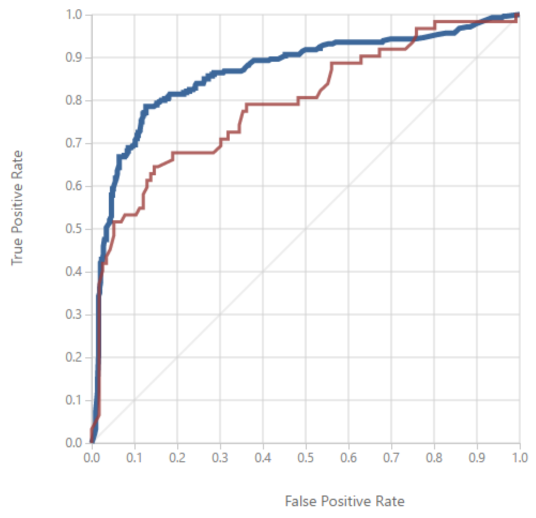


*Figure 5: ROC Curve for Two-class boosted decision tree*

## 2) Two-class Logistics Regression

It creates a logistics regression model predicts the probability of occurrence of an event by using a logistics function. Details of the implementation can be found here: https://msdn.microsoft.com/library/azure/b0fd7660-eeed-43c5-9487-20d9cc79ed5d#bkmk_Notes.

Range of parameters used:

| Parameters | Range/Values |
|---|---|
| Optimization tolerance | 0.0001, 0.0000001 |
| L1 regularization weight | 0.0, 0.01, 0.1, 1.0 |
| L2 regularization weight | 0.01, 0.1, 1.0 |
| Memory size for L-BFGS | 5, 20, 50 |

*Table 6: Range of parameters for Logistics Regression*

Tuned parameters for the best model:

| Parameters | Range/Values |
|---|---|
| Optimization tolerance | 0.0000001 |
| L1 regularization weight | 0.01 |
| L2 regularization weight | 1.0 |
| Memory size for L-BFGS | 20 |

*Table 7: Tuned parameters for Logistics Regression*

Relevant performance metrics:

| Dataset | Accuracy (%) | 95% CI (%) | Precision (%) | Recall (%) |
|---|---|---|---|---|
| Training | 83.74 | 83.37 – 84.10 | 0.8004 | 0.7870 |
| Testing | 76.93 | 75.73 – 78.12 | 0.7358 | 0.5633 |

*Table 8: Performance metrics for Two-class Logistics Regression*

Below is ROC Curve for Two-class logistics regression, where blue line indicates performance over training data and red line indicates performance over testing data.

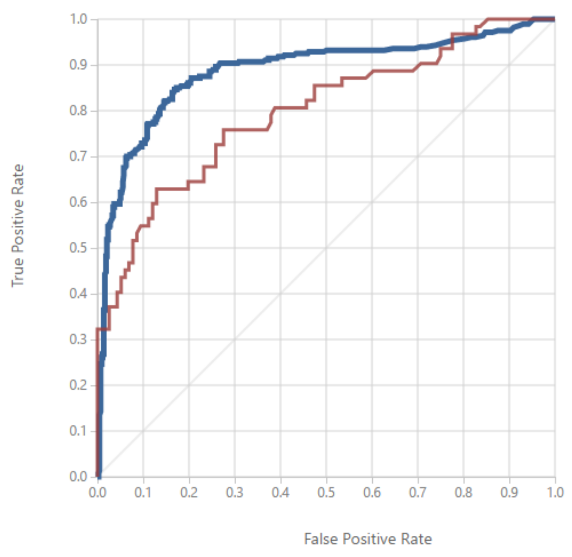


*Figure 6: ROC Curve for Two-class Logistics Regression*

### 3) Two-class Bayes Point Machine

The Bayes Point Machine algorithms in Microsoft Azure ML studio is a Bayesian approach to classification. It approximates the theoretically optimal Bayesian average of linear classifiers (in terms of generalization performance) by choosing one "average" classifier, the Bayes Point. Because the Bayes Point Machine is a Bayesian classification model, it is not prone to overfitting to the training data. It doesn't require parameter tuning and doesn't need data to be normalized.

Number of training iterations: 30

Relevant performance metrics:

| Dataset | Accuracy (%) | 95% CI (%) | Precision (%) | Recall (%) |
|---|---|---|---|---|
| Training | 81.47 | 81.05 – 81.90 | 0.8065 | 0.7092 |
| Testing | 74.74 | 73.05 – 76.41 | 0.7280 | 0.4489 |

*Table 9: Performance metrics for Two-class Bayes Point Machine*

Below is ROC Curve for Two-class logistics regression, where blue line indicates performance over training data and red line indicates performance over testing data.

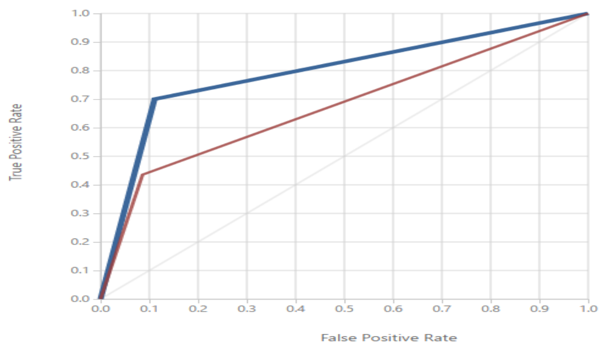


*Figure 7: ROC Curve for Two-class Bayes Point Machine*

## 4) Two-class Averaged Perceptron

Two-class averaged perceptron in Microsoft Azure ML studio is a simple version of neural network. In this method, inputs to the algorithm are classified into various possible outputs based on a linear function, and then they are combined with a set of weights that are derived from the feature vector—hence the name. Perceptrons are faster and can learn linearly separable patterns.

Range of parameters used:

| **Parameters** | **Range/Values** |
|---|---|
| Learning rate | 1.00e-4 - 1.00 |
| Maximum number of iterations | 1 - 50 |

*Table 10: Range of parameters for Two-class Averaged Perceptron*

Tuned parameters for the best model:

| **Parameters** | **Range/Values** |
|---|---|
| Learning rate | 1 |
| Maximum number of iterations | 17 |

*Table 11: Tuned parameters for Two-class Averaged Perceptron*

Relevant performance metrics:

| **Dataset** | **Accuracy (%)** | **95% CI (%)** | **Precision (%)** | **Recall (%)** |
|---|---|---|---|---|
| Training | 84.03 | 83.72 – 84.33 | 0.8029 | 0.7935 |
| Testing | 78.04 | 76.77 – 79.40 | 0.7472 | 0.5744 |

*Table 12: Performance metrics for Two-class Averaged Perceptron*

Below is ROC Curve for Two-class Averaged Perceptron, where blue line indicates performance over training data and red line indicates performance over testing data.

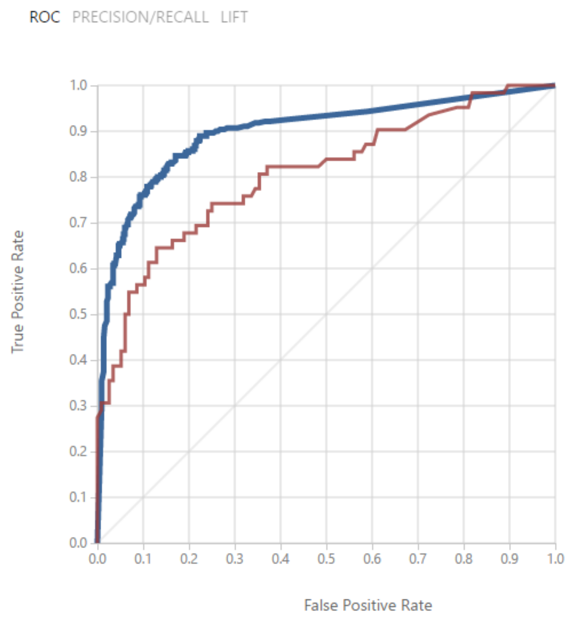
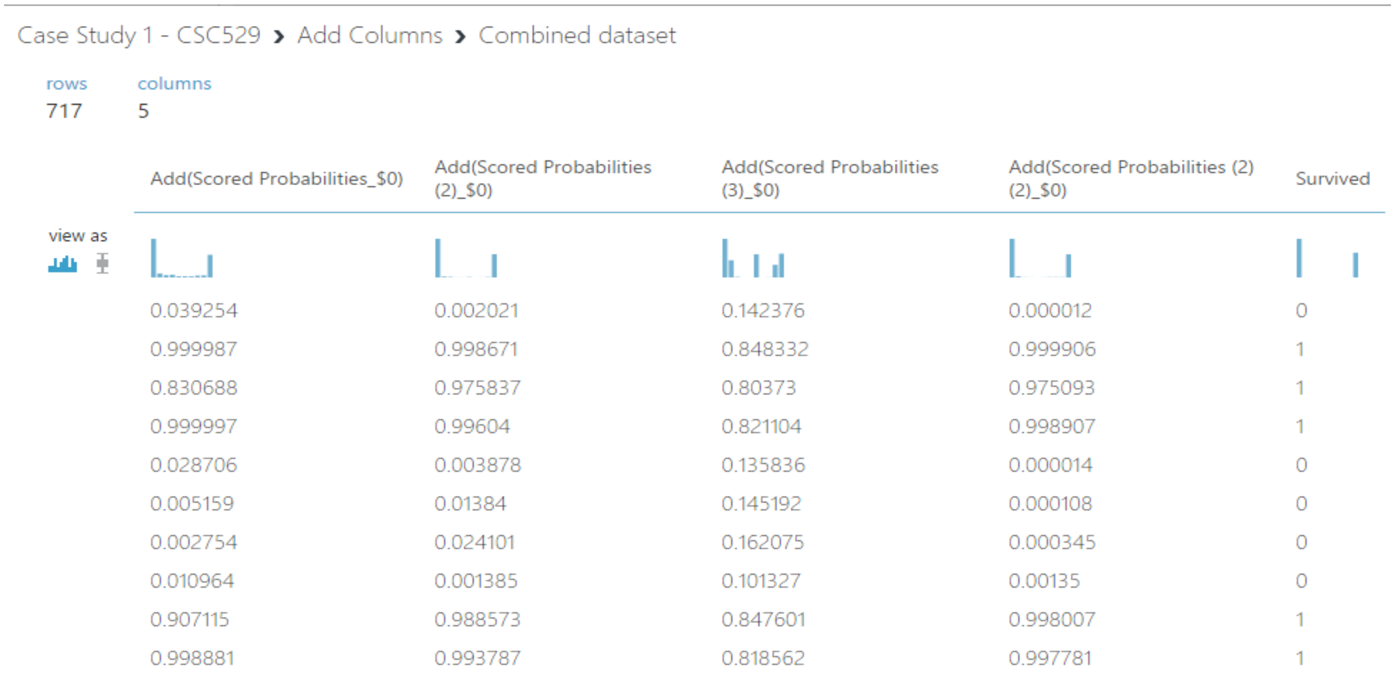


*Figure 8: ROC Curve for Two-class Averaged Perceptron*

## 5) Stacking Ensembles

Here I am going to implement the stacking ensemble or stacking technique for building ensemble of classifiers. I am taking the "Scored Probabilities" of training and testing dataset from each of the scored models and then I will model those using a Two-class Logistics Regression by tuning its hyperparameters using the same method I used for individual classifiers. So, Logistics Regression will take as input four features which will look as follows, where Add(Scored Probabilities_$0) comes from Two-class Boosted Decision Tree, Add(Scored Probabilities (2)_$0) comes from Two-class Logistics Regression, Add(Scored Probabilities (3)_$0) comes from Two-class Bayes Point Machine and Add(Scored Probabilities (2) (2)_$0) comes from Two-class Averaged Perceptron:


Case Study 1 - CSC529 › Add Columns › Combined dataset

rows 717 columns 5

| Add(Scored Probabilities_$0) | Add(Scored Probabilities (2)_$0) | Add(Scored Probabilities (3)_$0) | Add(Scored Probabilities (2) (2)_$0) | Survived |
|---|---|---|---|---|
| 0.039254 | 0.002021 | 0.142376 | 0.000012 | 0 |
| 0.999987 | 0.998671 | 0.848332 | 0.999906 | 1 |
| 0.830688 | 0.975837 | 0.80373 | 0.975093 | 1 |
| 0.999997 | 0.99604 | 0.821104 | 0.998907 | 1 |
| 0.028706 | 0.003878 | 0.135836 | 0.000014 | 0 |
| 0.005159 | 0.01384 | 0.145192 | 0.000108 | 0 |
| 0.002754 | 0.024101 | 0.162075 | 0.000345 | 0 |
| 0.010964 | 0.001385 | 0.101327 | 0.00135 | 0 |
| 0.907115 | 0.988573 | 0.847601 | 0.998007 | 1 |
| 0.998881 | 0.993787 | 0.818562 | 0.997781 | 1 |

*Figure 9: Features for Stacking Ensemble*

Range of parameters:

| Parameters | Range/Values |
|---|---|
| Optimization tolerance | 0.0001, 0.0000001 |
| L1 regularization weight | 0.0, 0.01, 0.1, 1.0 |
| L2 regularization weight | 0.01, 0.1, 1.0 |
| Memory size for L-BFGS | 5, 20, 50 |

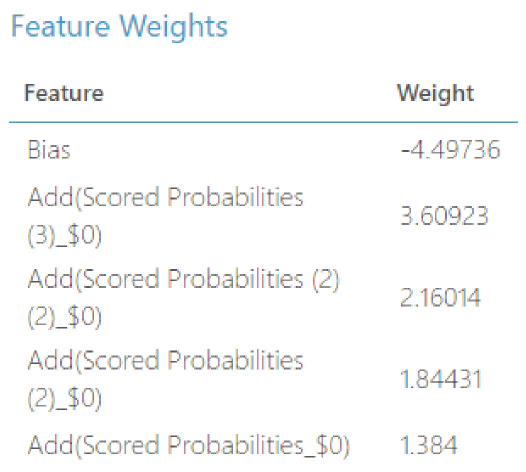
*Table 13: Range of parameters for Logistics Regression – Ensemble Classifier*

Tuned parameters for the best model:

| Parameters | Range/Values |
|---|---|
| Optimization tolerance | 0.0000001 |
| L1 regularization weight | 0.01 |
| L2 regularization weight | 1.0 |
| Memory size for L-BFGS | 20 |

*Table 14: Tuned parameters for Logistics Regression – Ensemble Classifier*

The feature weights produced by the Logistics Regression model is:



Feature Weights

| Feature | Weight |
|---|---|
| Bias | -4.49736 |
| Add(Scored Probabilities (3)_$0) | 3.60923 |
| Add(Scored Probabilities (2) (2)_$0) | 2.16014 |
| Add(Scored Probabilities (2)_$0) | 1.84431 |
| Add(Scored Probabilities_$0) | 1.384 |

*Figure 10: Weights after modeling features of Stacking Ensemble*

We can see that the model gives highest weight to the scored probabilities from Two-class Bayes Point Machine and weight to Two-class Averaged Perceptron, Two-class Logistics Regression and Two-class Boosted Decision Tree in decreasing order.

Relevant performance metrics:

| Dataset | Accuracy (%) | 95% CI (%) | Precision | Recall |
|---|---|---|---|---|
| Training | 96.51 | 96.27 – 96.75 | 0.9679 | 0.9410 |
| Testing | 89.07 | 88.23 – 89.91 | 0.8546 | 0.7747 |

*Table 15: Performance metrics for Two-class Logistics Regression - Ensemble Classifier*

Below is ROC Curve for Two-class Averaged Perceptron, where blue line indicates performance over training data and red line indicates performance over testing data.

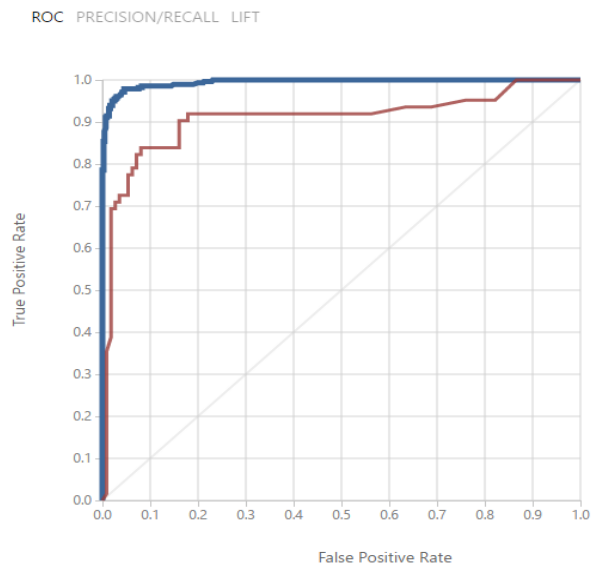


*Figure 11: ROC Curve for Two-class Logistics Regression*

## 6. Experimental Analysis

Consolidating the results of all the models what we get is:

| **Training** | **Accuracy** | **95% CI Accuracy (%)** | **Precision** | **Recall** |
|---|---|---|---|---|
| Two-class boosted decision tree | 83.17 | 83.17 - 83.18 | 0.8113 | 0.7543 |
| Two-class logistics regression | 83.74 | 83.37 - 84.10 | 0.8004 | 0.7870 |
| Two-class Bayes point machine | 81.47 | 81.05 - 81.90 | 0.8065 | 0.7092 |
| Two-class averaged perceptron | 84.03 | 83.72 – 84.33 | 0.8029 | 0.7935 |
| Stacking Ensemble | 96.51 | 96.27 – 96.75 | 0.9679 | 0.9410 |
| **Testing** | | | | |
| Two-class boosted decision tree | 77.48 | 76.06 – 78.9 | 0.7314 | 0.6047 |
| Two-class logistics regression | 76.93 | 75.73 – 78.12 | 0.7358 | 0.5633 |
| Two-class Bayes point machine | 74.74 | 73.05 – 76.41 | 0.7280 | 0.4489 |
| Two-class averaged perceptron | 78.04 | 76.77 – 79.40 | 0.7472 | 0.5744 |
| Stacking Ensemble | 89.07 | 88.23 – 89.91 | 0.8546 | 0.7747 |

*Table 16: Consolidates results of performance metrics*

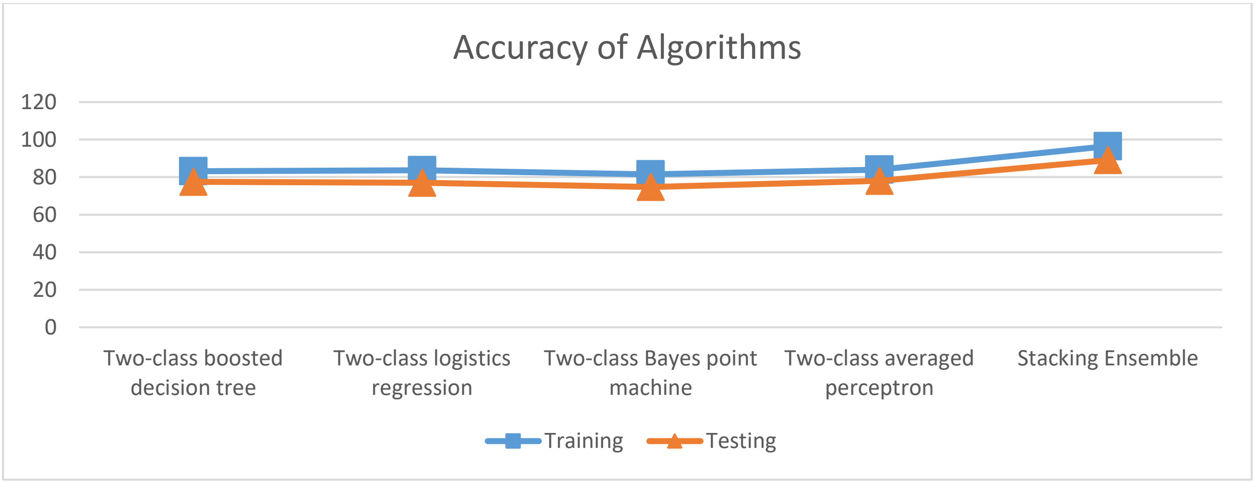


*Figure 12: Accuracy of Algorithms*

➔ So here we can see that the accuracies range from 81.47% to 84.03% for training dataset and 74.74% to 78.04% for testing dataset. Two-class averaged perceptron gives best results on both training and testing. Also, it is important to consider the 95% Confidence Intervals for accuracies when we talk about the best or worst performing algorithm on this dataset. We can see that the 95% CI is the smallest for Two-class boosted decision tree and so it shows that the cross-validation accuracies had lowest standard deviation for that algorithm. The highest 95% CI for accuracy is 83.72 – 84.33 for Two-class averaged perceptron and so we can say with 95% confidence that if we repeatedly apply this algorithm to the dataset the mean of the accuracies we get will be between 83.72 – 84.33 which is still the highest CI we get. The lowest CI we get is for Two-class Bayes point machine. Similar are the results for testing data as well. If we talk about the accuracy of the Ensemble on training and testing it comes out to be 96.51% and 89.07% and their 95% CI are 96.27 - 96.75 and 88.23 - 89.91 and it clearly outperforms the individual classifiers in terms of accuracy and 95% CI.

➔ If we look at the training and testing accuracies and their 95% CI's we can say that the algorithms including the ensemble were able to generalize well on testing data as well and we are not overfitting the model in any of the case, one way to look at this is that the training and testing accuracies are within 10% difference.

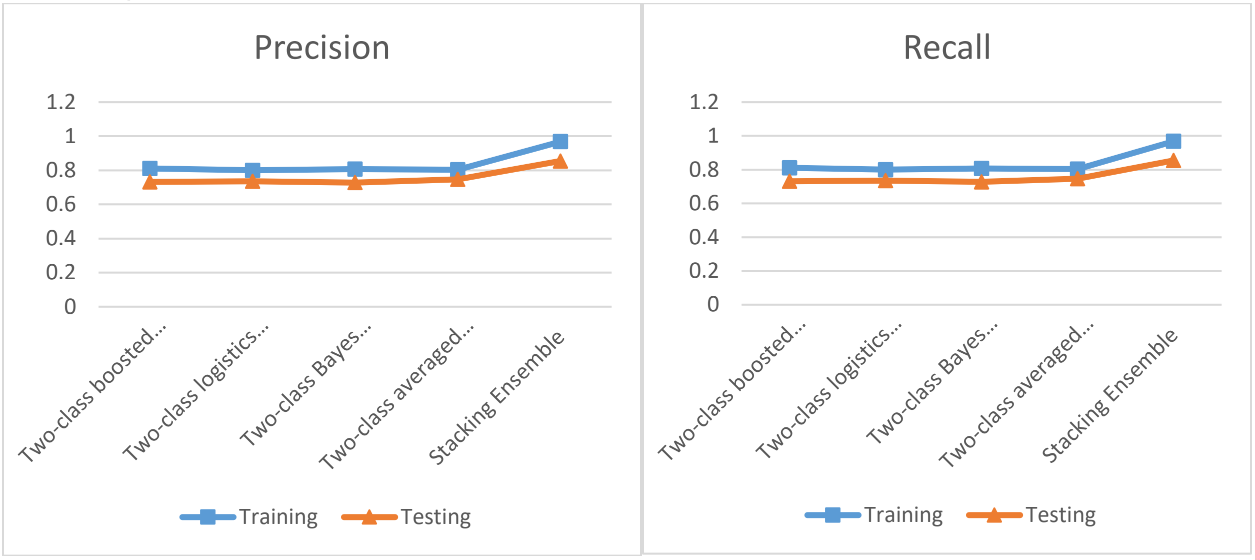


*Figure 13: Precision & Recall*

➔ If we talk about the precision, which is the fraction of the events where we correctly classified a record into one class out of all the classified tweets into that class, reported for all the algorithms we can see that the precision is same for all algorithms which is around 0.80-0.81 on training dataset. For testing, the precision we get is around 0.72-0.74. If we compare Recall values, which is fraction of the events where we correctly classified records into a class out of all the records in that class, it is 0.70 for Two-class Bayes Point machine which is lowest while highest recall we get for training is 0.79 using Two-class averaged perceptron algorithm. So the precision value here will specify how right we are predicting at the overall level and is the measure of exactness, so precision of 0.8113 for Two-class boosted decision tree for training shows that for example if we predict that 100 records are classified as "Survived: Yes (1)", algorithm was right around 81 of the times and 19 times that passenger didn't actually survived and recall shows how well we are doing in a particular class and is the measure of completeness, so recall of 0.7472 shows that for example out of 100 records in positive class we were able to correctly classify 74 records. Precision and recall rates are low on testing data as compared to training data and is the lowest for Two-class Bayes point machine. If we look at Precision and Recall for Ensemble classifier, they are the highest for ensemble at value 0.9675 and 0.9410 for training and 0.8546 and 0.7747 for testing, showing that stacking does work for this dataset and we were successfully able to combine individual learners and create a powerful ensemble algorithm. We can see that there is significant improvement in the Recall which shows that we are doing good in terms of performing on a class, where earlier it was 0.79 highest for two-class averaged perceptron for training and jumped to 0.94 which is a significant improvement of around 19% due to stacking, also the Recall improves for the testing dataset and increases from highest of 0.60 to 0.77.

- ➔ If we look at the ROC Curve, which compares two operating characteristics (True Positive Rate and False Negative Rate) for each of the algorithms, we see that the area under the curve is highest for Stacking Ensemble. The performance of the Two-class Bayes point machine was the worst amongst all four individual classifiers, the probabilities from that were given the highest weight in the model created by logistics regression and the weight given to Two-class averaged perceptron was second highest which was the best performing algorithm among the individual classifiers. So, the logistics function or logistics regression equation to estimate the probability of predicting a positive class can be given as:

**Probability of surviving = 1/ (1+exp (-(-4.49736 + 3.60923*Probabilities from Two-class Bayes Point Machine + 2.16014*Probabilities from Two-class Averaged Perceptron + 1.84431*Probabilities from Two-class Logistics Regression + 1.384*Probabilities from Two-class Boosted Decision Tree)))**

- ➔ So we can say due to algorithms being in the denominator, higher the weight lower the contribution of that algorithm in predicting the probability of surviving, so we can say that the algorithms with highest weight is the lowest contributor in deciding the outcome and vice-versa, so we can say Two-class boosted decision tree is contributing the highest and Two-class Bayes Point Machine is contributing the lowest, which is explainable looking at the fact that the accuracy of Two-class Bayes Point Machine was lowest and Two-class Boosted Decision Tree didn’t had the highest accuracy but its 95% CI was the least wide for training and so it was more precise in the predictions and which is evident from the fact that it precision was highest as well so that contributing the highest to the final stacking ensemble is not a surprise, so that’s what the power of ensembles is that it combines the best features or learning abilities of each of the algorithms and produces a strong learner which happened in our case study as well.

## 7. Conclusion

- ➔ The aim of the case study was to learn the implementation of one of the ensemble technique and the one I showcased here was Stacking technique in which we use the scored probabilities from each of the individual classifier and use them as features to model them using Logistics Regression and we see increase performance of the ensemble classifier.

- ➔ So, looking at the performance metrics for each of the algorithms and ensemble and experimental analysis we can say that the **stacking ensemble technique does improve the classification accuracy as well as precision and recall and could generalize well on the testing dataset** and can be used to work on bigger and complicated datasets as well.

- ➔ One of the limitation I faced due to Microsoft Azure ML studio was I was required to do visualizations separately in iPython notebook. Also, the dataset is not that large and it would be interesting to use the same stacking ensemble concept on bigger dataset with more data and variables. Visualization of Two-class Boosted Decision Tree was not possible and I was not able to report the parameters due to error raised by Microsoft Azure ML studio.

References:

1) Kaggle: https://www.kaggle.com/
2) Azure Machine Learning Documentation: https://docs.microsoft.com/en-us/azure/machine-learning/