EE660- Machine Learning Project Report

Learning from data to:
1) Predict bike-sharing users
2) Predict loan-defaults

Nadir Nibras

Project 1) Predict bike-sharing users from data

Abstract

The data used here is on Bike sharing systems, a new generation of traditional bike rentals where whole process from membership, rental and return has become automatic. Some of the features included in this dataset are season, month (1 to 12), hour (0 to 23), holiday, weekday: day of the week, working-day, weather-situation, temperature, humidity, windspeed and the count of total rental bikes used each day (including both casual and registered users). There are 731 data points. The dataset was used to predict the number (count) of ride-sharing bikes that will be used in any day given other features with the help of a regression algorithm.

Problem Statement and Goals

The goal, is to use machine learning algorithms we learned in class to predict the number of ride-sharing bikes that will be used in any day given other features with the help of a regression algorithm. The bike-sharing dataset had plenty of categorical data (ex: weather-situation for bike-sharing dataset) I had to be split up into multiple features so I could run regression algorithms on them. Some of the features had to selected and removed based on my understanding of the data and what I thought would not be helpful. Although there were no missing feature values, the pre-processing took a fair bit of time for me. The training samples were also not too extensive (731), so I had to rearrange the training set samples when doing multiple runs to get the average value of the weights using each combination of parameters for the algorithm. I experimented with ML algorithms covered in class: MLE, MAP and Random Forest and figured out the ideal parameters (weights and regularizer values) for the algorithm that provided the best result.

Prior and related work- NONE

Project Formulation and Setup

After trying out a several algorithms, I ended up going with the Maximum A Posteriori (MAP) method as it gave me the least error out of the three and it wasn't nearly as computationally expensive as Random Forest which took more than hour to run each trial with the 19 features I used.

The basis of MAP is we calculate the parameters (weight assigned to different features in this case) as a mode of the posterior distribution of a random variable.

$$\hat{\theta}_{\mathrm{MAP}}(x) = \arg\max_{\theta} f(\theta \mid x) = \arg\max_{\theta} \frac{f(x \mid \theta)\, g(\theta)}{\int_{\vartheta} f(x \mid \vartheta)\, g(\vartheta)\, d\vartheta} = \arg\max_{\theta} f(x \mid \theta)\, g(\theta).$$

For this prediction problem, I started with my homework solutions from our class, and adjusted it extensively to make it work for the dataset that I have.
I assumed that we know the distribution of **W** (weight vector) as a prior:

$$\underline{w} \sim N\left(\underline{m}_w, \tau^2 I\right)$$

Taking the derivative (gradient) of the negative log posterior function with respect to **W** and carrying out the minimization algebraically, and then by setting the expression equal to the zero vector, we get the value of the **W** to be found from the following expression:

$$\hat{w}_{MAP} = (X^T X + \lambda I)^{-1}(X^T y + \lambda m_w)$$

I found the optimal values of m_w and Tau^2 (and therefore **W**) by trying out many different values as I explain in the next section.

Methodology and Implementation

Feature Space and pre-processing

I started with a dataset that had the following features. All continuous features such as temperature or windspeed were already normalized to values between 0 and 1.

- instant: record index
- dteday : date
- season : season (1:springer, 2:summer, 3:fall, 4:winter)
- yr : year (0: 2011, 1:2012)
- mnth : month (1 to 12)
- holiday : weather day is holiday or not (extracted from [Web Link])
- weekday : day of the week
- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.

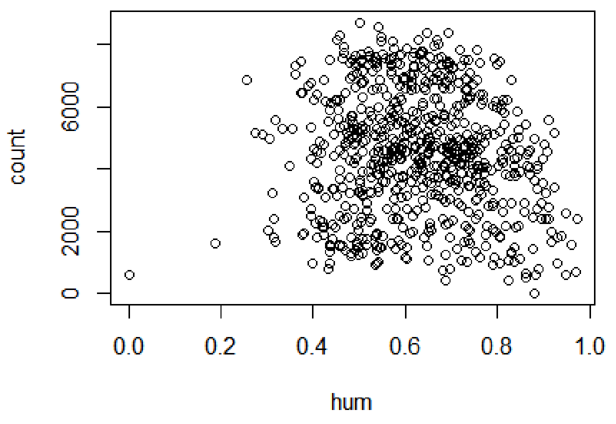
+ weathersit :

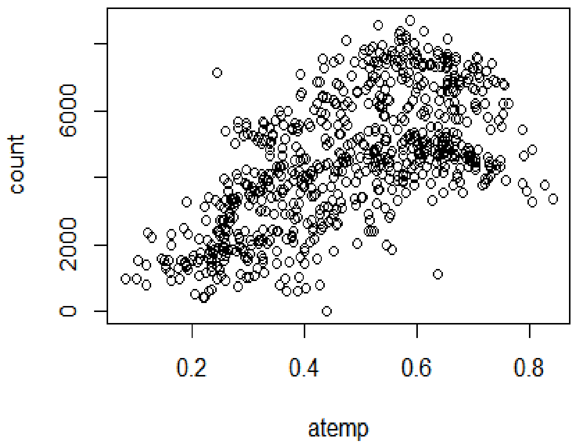
- 1: Clear, Few clouds, Partly cloudy, Partly cloudy
- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
- 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp : Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)
- atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-16, t_max=+50 (only in hourly scale)
- hum: Normalized humidity. The values are divided to 100 (max)
- windspeed: Normalized wind speed. The values are divided to 67 (max)
- casual: count of casual users
- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered

I used Matlab for feature pre-processing. After reading in the data, I initially got rid of features that I thought were unimportant from prior knowledge. I removed the index feature as it added no value. I removed the date as well, as I believed the other features would compensate for the information we miss out from the date. I removed the year as we only had 2 years available and I expected to find no meaningful information from this number. I also removed the “casual” and “registered” count values as I aimed to predict the total “count” only. Binary features needed no altering as it was already showing either 0 or 1 values. Categorical features (Month, Weather-sit, Season, Weekday) were split up into multiple features where each feature indicated whether a certain category for the previous feature was true or not (1 or 0). I removed the features “Weather Situation 4” and “Weekday 7” as all instances resulted in a value of 0 for these features (they never occurred). It might mean that the specific weather situation (4) is very rare and that bikes are not rented out on Sundays. I ended up with 19 features for the input, a 1D output and 731 data points.

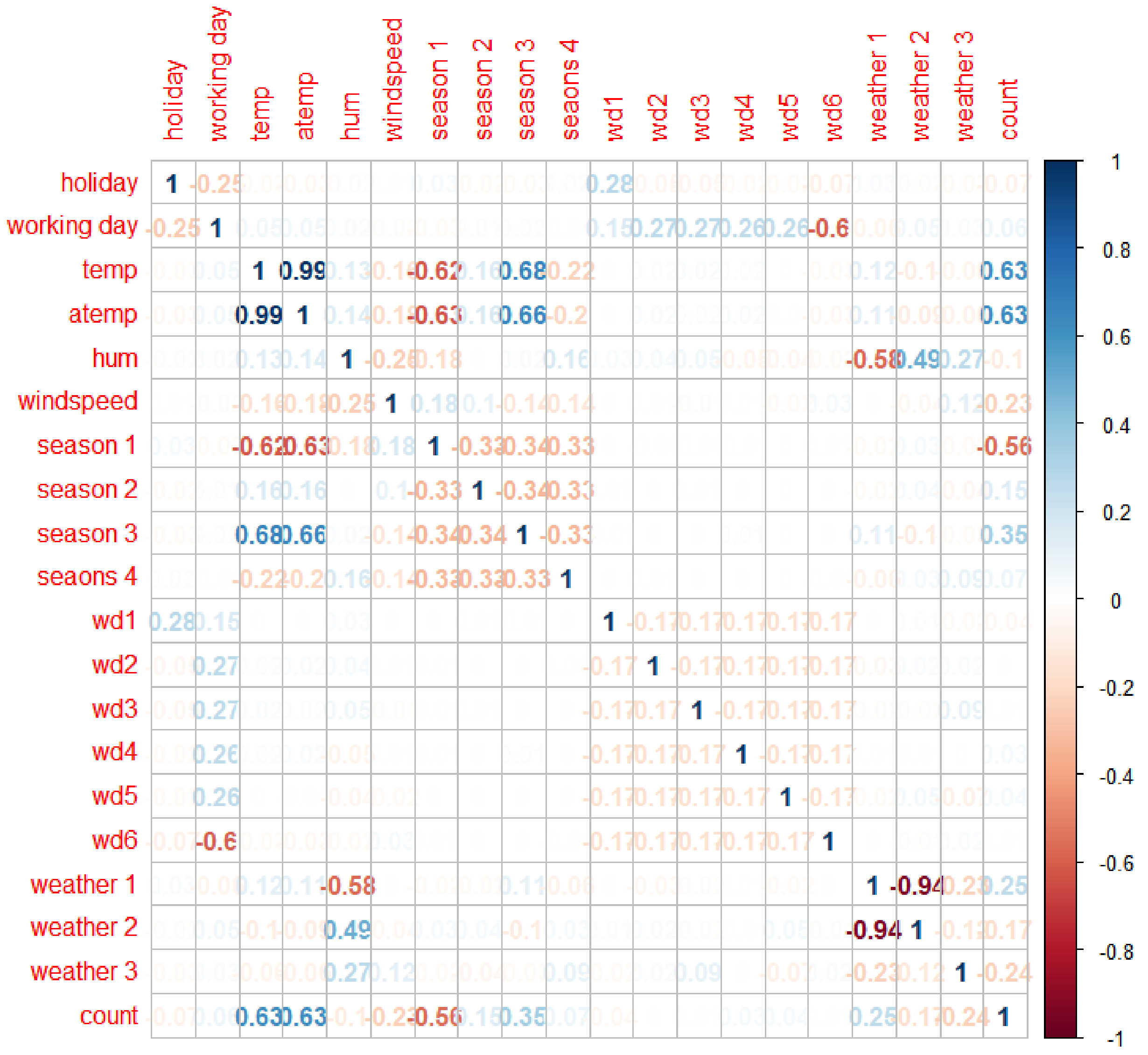
Exploratory data analysis

Before I started running any algorithm, I explored the data further to get a better understanding of what I was working with. I plotted scatter plots between the bike count and the continuous variables: adjust temp, windspeed, humidity. The plot for the adjusted temp looked most promising for regression purposes.





Next, I did a correlation plot between all features to get an overall idea of which features may be the most useful. Although none of the features were very strongly correlated to “count”,all the features working together could however serve as a good predictor for outcome.



Next, I checked and made sure none of the outcome values were outliers. There weren’t any but it was worth noting that the data was quite broadly spread out with a mean of 4504.349 and a standard deviation of 1937.211.

Training, Validation and Testing

I used Maximum Likelihood Estimate (MLE), MAP and Random Forest (RF) algorithms to test which ones perform the best. I also tried non-linear regression to check if it helped reduce the error. I wrote the code for all three algorithms on Matlab. I wrote the first two on the same script and the RF code on a different script.

I used the randperm function to scramble the data points every time I run my script at the start of the script. Then I separated 61 data points as a **test set** that I used later to test both the MLE and MAP.

MLE

I ran MLE over the training set 100 times as my data samples were very limited over here. Each validation set (selected randomly at each permutation from the samples left after deducting the test set) had 60 samples and the remaining 600 sets were used as training.

I used the following formula on Matlab (see below) 100 times while scrambling the dataset and found the average parameters to find the value of **W**.

$$\hat{w}_{MLE} = (X^T X)^{-1} X^T y$$

This formula was derived from finding the minimum of the negative log likelihood function similar to how I did it for MAP.

$$\{\hat{\theta}_{\text{mle}}\} \subseteq \{\underset{\theta \in \Theta}{\arg\max}\ \hat{\ell}(\theta\,;\, x_1, \ldots, x_n)\},$$

After finding **W** from the train set, I used it on the Validation Set features to predict the output of the validation set and then compared the MSE between the actual validation set output and my predictions in the 100 iterations from cross validation. I picked the **W** that produced the lowest MSE and used it to find the MSE on the test set for my final reported MSE.

MAP

I used the following function as mentioned above to derive the weights from MAP

$$\hat{w}_{MAP} = (X^T X + \lambda I)^{-1}(X^T y + \lambda m_w)$$

I used a validation set for MAP as I was testing 100,000 combinations for the unknown MAP parameters, m_w and Tau^2. I used 600 data points for the Train Set and 61 data points for the Validation Set and these were randomly selected each time.

I used for-loops to try different values of m_w and Tau^2 and to see which one gave the best results. I tried 1000 values of m_w ranging from 0.001-1 and 100 values of Tau^2 ranging from 0.1-10. Note that lambda = of m_w/Tau^2. For each value of m_w and Tau^2, I calculated **W** on a randomly generated train set of 600 samples and then, after using the **W** to calculate a prediction for outcomes for the 61-point validation set, I checked the MSE between the predictions and the actual validation test outputs. After going through 10,000 iterations of values for m_w and Tau^2 I picked the values that gave me the smallest MSE on the validation set. I saved this **W** as **W_ideal** and used this to find a prediction of the outputs on the test set we separated earlier. Once again, I found the MSE between the actual test outputs and the estimation of outputs from our best **W_ideal** to test the accuracy of our algorithm.

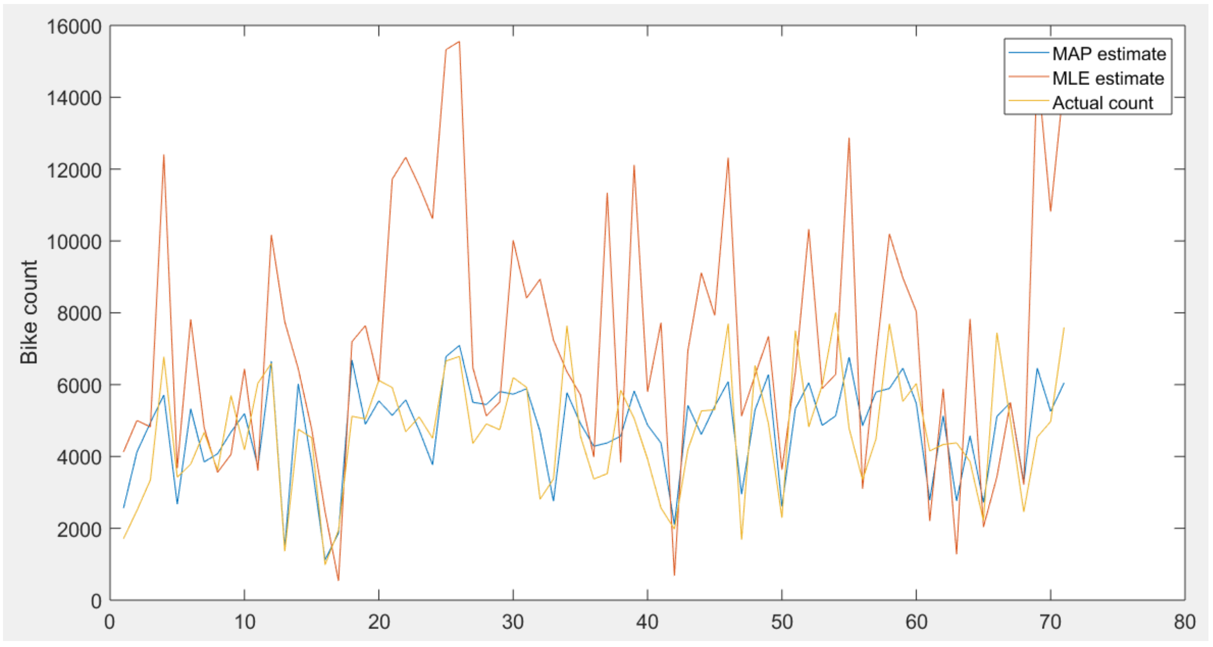
Both the MLE and MAP algorithms used linear regression, so my hypothesis set includes and infinite number of Hypotheses as **W** is a 19-element vector where each element could be any real number. I tried using non-linear regression and added one line of code to do polynomial regression (included in code) but it didn't give me any better results so I stuck to linear regression. This makes sense as most of my features are binary features with two possible outcomes where polynomial fits make no difference.

Random Forest

I used a separate Matlab script for trying out the RF algorithm as this was a lot more computationally expensive than the previous 2 algorithms, I used the "Pmtk3" package that we used in our assignments. At the beginning, I scrambled the datapoints and separated out 131 data points as our test set for the RF algorithm. I performed the experiments over a range for the number of trees ranging from 1 to 30.
For each number of trees (ntree), I repeated 10 iterations where I picked a new train set by picking 200 points with replacement from the old train set of 600 points. I found the "forest" variable for fitting the tree using "fitforest" function, 4 "random features" and a bag-size of 1/3. Then I found the MSE of this tree constructed for the train set and the test set. Finally for each value of ntree, I calculated the average MSE (for train set and for the test set) over the 10 iterations and recorded that value.

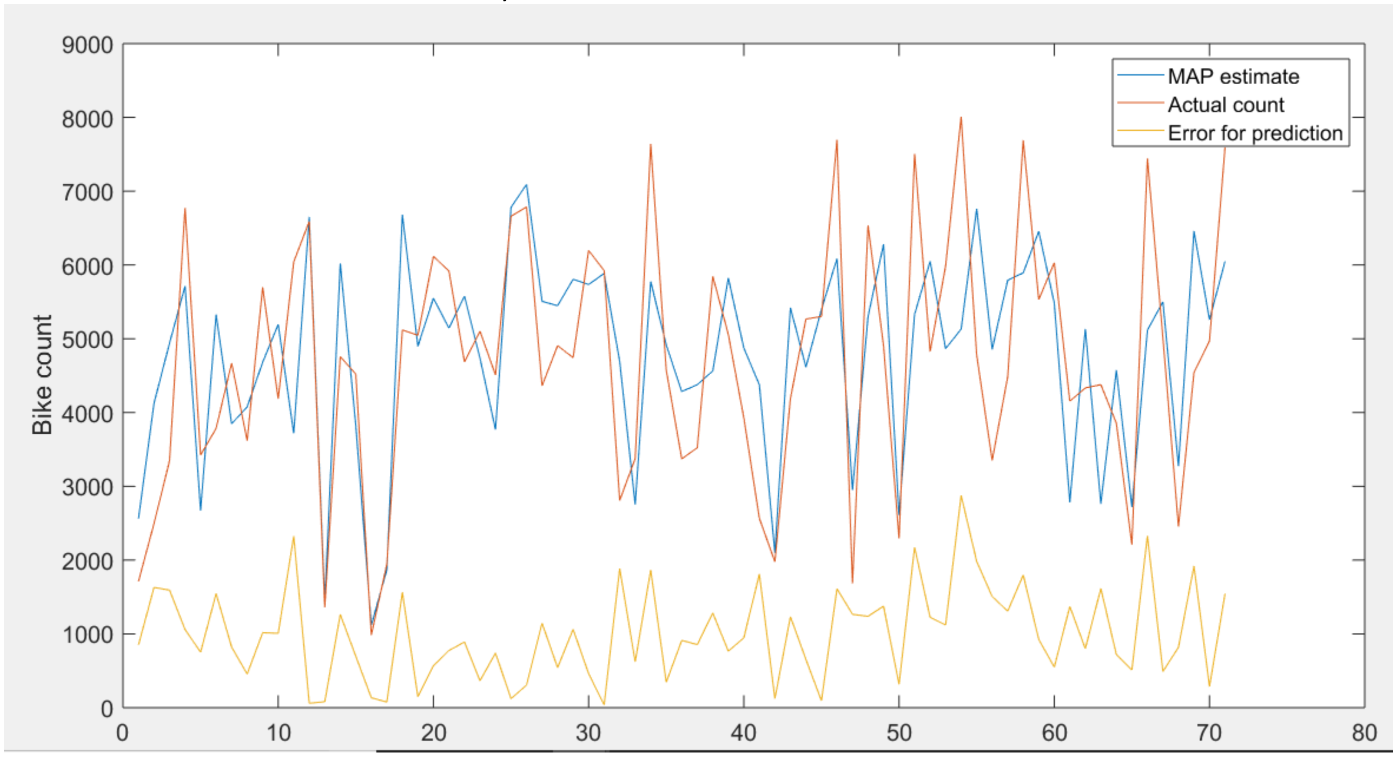
Final Results and interpretations

MLE and MAP
From MLE I found the MSE to be always greater than 10,000,000.
For MAP I could always get it in under 10,000,000, and sometimes less than 1,000,000. From the plot below it seems obvious that MAP is a lot better at predicting the actual "Count" than MLE, and that MLE fluctuates a lot more.



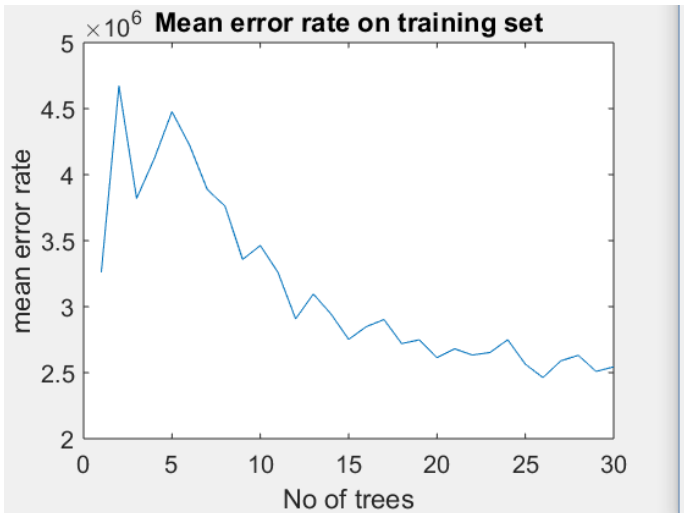
Looking further into how good the MAP estimate was, I found the **average difference between the MAP prediction and the actual test data** to be 995.9270 for one instance and almost always around 1000. Looking at the difference in terms of a percentage of the actual test output value, I found that on average the MAP prediction is 26% away from the Test output.

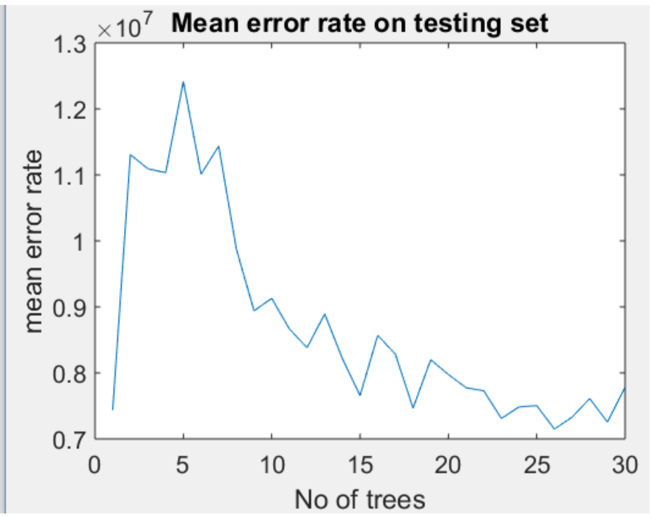
I have plotted the error value for each instance in the plot below. It is also interesting to note that the MAP estimate for the actual count seems to be almost always closer to the mean count and the MAP estimate doesn't fluctuate as wildly as the real data.

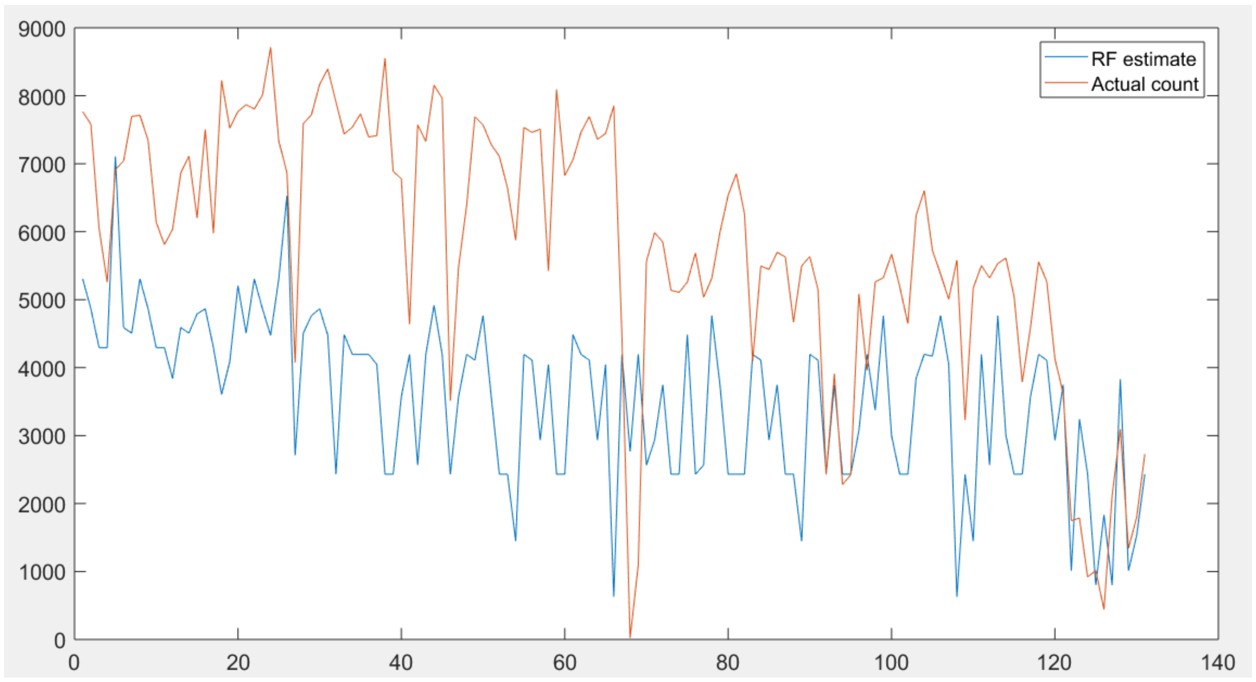


RF
For RF, as we increased the number of trees, the MSE started going down as expected for the training set, going down to 2,500,000 for ntree= 30. For the test set, the MSE also dropped sharply and then started rising again after crossing ntree= 26, providing a best MSE value of around 7,200,000, which is better than MLE but not as good as our MSE from MAP.

Looking further into how the RF estimates vary from the actual count, we see that just like MAP, the predictions are more likely to be highly inaccurate when the actual count deviates from the mean by a large value.







An important point to note is the computational time for each algorithm. For MLE the time taken was 0.22 seconds and for MAP using 100,000 iterations (100 times as many iterations than MLE) the time taken was around 19.5 seconds. The computational time for RF with up to 30 ntrees was over 1 hour and 30 minutes

Conclusion

It seems that MAP is the best algorithm for this project. It is possible that RF may produce slightly better results with further tweaking of the parameters but it is too computationally expensive in my opinion. The minimum MSE I found is quite large (close to 1,000,000) but that is largely because the output variable here is a large number and squaring the error gives an even bigger number. While I don't think this is the best algorithm out there, I think being able to predict the number of bikeshare counts per day with an average error of 26% could be useful for a company that needs these numbers. It might be worth looking into other algorithms however.

Contributions of each team member- 1 member

Project 2) Predict loan-defaults from customer data

Abstract

The dataset looks at the case of customers' default payments in Taiwan. The data features include the Amount of the given credit (NT dollar), Gender, Education, Marital status, Age and financial information related to credit card payments. There is information on 30,000 users. The dataset was used to predict the whether a customer will default on his/her loan based on other information regarding relating to his finances and what demographic group he belongs. My goal was to find a classification algorithm that predicts the outcome of a loan.

Problem Statement and Goals

The goal, is to use machine learning algorithms we learned in class to predict whether a customer will default on their loan given other features with the help of a classification algorithm. The dataset had categorical data (ex: educational level, marital status) I had to be split up into multiple features so I could run classification algorithms on them. Although there were no missing feature values, the pre-processing took a fair bit of time for me. None of the features were normalized so I had to normalize monetary values such as balances and payment histories by finding the range of the minimum-maximum for each feature and seeing where that value stands in relation to that. I experimented with ML algorithms covered in class (K-Nearest Neighbors and Random Forest) and also tried some algorithms that we haven't tried in class (Support Vector Machines and Linear Discriminant analysis) to figure out which algorithm is the best one to use.

Prior and related work- NONE

Project Formulation and Setup

After trying out a several algorithms, I ended up going with the Linear Discriminant Analysis (LDA) method as it gave me the least error out of the 4 and it also was not computationally expensive (only took a few seconds to calculate) as other ones like Random Forest and SVM which took more than an hour each to run with the 28 features I used and the 24,000 training data points I used. There are no parameters that I had to fine tune for LDA as I trained the algorithm with the Caret package in RStudio. I'll discuss the gist of **how LDA works** below:

LDA makes some simplifying assumptions about the data:
a) that the data is Gaussian, that each variable is shaped like a bell curve when plotted.
b) that each attribute has the same variance, that values of each variable vary around the mean by the same amount on average.

After making these assumptions, the LDA model estimates the mean and variance from the data for each class (default and non-default). The mean (mu) value of each input (x) for each class (k) can be estimated in the normal way by dividing the sum of values by the total number of values.

muk = 1/nk * sum(x)

Where muk is the mean value of x for the class k, nk is the number of instances with class k. The variance is calculated across all classes as the average squared difference of each value from the mean.

sigma^2 = 1 / (n-K) * sum((x – mu)^2)

Where sigma^2 is the variance across all inputs (x), n is the number of instances, K is the number of classes and mu is the mean for input x.

LDA makes predictions by estimating the probability that a new set of inputs belongs to each class. The class that gets the highest probability is the output class and a prediction is made. The model uses Bayes Theorem to estimate the probabilities.:

P(Y=x|X=x) = (PIk * fk(x)) / sum(PIl * f(x))

Where PIk refers to the base probability of each class (k) observed in the training data. In Bayes' Theorem this is called the **prior** probability.

PIk = nk/n

The f(x) above is the estimated probability of x belonging to the class. A Gaussian distribution function is used for f(x). Plugging the Gaussian into the above equation and simplifying leads to the equation below. This is called a discriminate function and the **class is calculated as having the largest value will be the output classification** (y):

Dk(x) = x * (muk/sigma^2) – (muk^2/(2*sigma^2)) + ln(PIk)

Dk(x) is the discriminate function for class k given input x, the muk, sigma^2 and PIk are all estimated from the data.

Methodology and Implementation

Feature Space and pre-processing

I started with a dataset that had the following features. The features were not normalized

X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
X2: Gender (1 = male; 2 = female).
X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
X4: Marital status (1 = married; 2 = single; 3 = others).
X5: Age (year).
X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

I used Matlab for feature pre-processing. Binary features needed no altering as it was already showing either 0 or 1 values. Categorical features (Education, Marital status) were split up into multiple features where each feature indicated whether a certain category for the previous feature was true or not (1 or 0). I had to normalize most of the features as they were large values such as the age or the value of customer balances or payments (which could be very large), to ensure that all features are treated equally in my algorithm.

Exploratory data analysis

Before I started running any algorithm, I explored the data further in Rstudio to get a better understanding of what I was working with. I looked at how many data points were actually default and found that defaults occurred 22.11% of the time in my dataset. Next, I summarized the dataset to see if there is any interesting information about the features that I cannot catch from a quick glance at the dataset. Here is what I got:

```
LIMIT_BAL           SEX                 AGE              PAY_0                PAY_2
 Min.   :0.0100   Min.   :-1.0000   Min.   :0.0000   Min.   :-0.222220   Min.   :-0.22222
 1st Qu.:0.0641   1st Qu.:-1.0000   1st Qu.:0.1207   1st Qu.:-0.111110   1st Qu.:-0.11111
 Median :0.1750   Median : 1.0000   Median :0.2241   Median : 0.000000   Median : 0.00000
 Mean   :0.2111   Mean   : 0.2109   Mean   :0.2497   Mean   :-0.001926   Mean   :-0.01459
 3rd Qu.:0.3000   3rd Qu.: 1.0000   3rd Qu.:0.3448   3rd Qu.: 0.000000   3rd Qu.: 0.00000
 Max.   :1.0000   Max.   : 1.0000   Max.   :0.9310   Max.   : 0.888890   Max.   : 0.88889
     PAY_3              PAY_4              PAY_5              PAY_6            BILL_AMT1
 Min.   :-0.22222   Min.   :-0.22222   Min.   :-0.22222   Min.   :-0.22222   Min.   :-0.099502
 1st Qu.:-0.11111   1st Qu.:-0.11111   1st Qu.:-0.11111   1st Qu.:-0.11111   1st Qu.: 0.002165
 Median : 0.00000   Median : 0.00000   Median : 0.00000   Median : 0.00000   Median : 0.013479
 Mean   :-0.01815   Mean   :-0.02424   Mean   :-0.02928   Mean   :-0.03234   Mean   : 0.030772
 3rd Qu.: 0.00000   3rd Qu.: 0.00000   3rd Qu.: 0.00000   3rd Qu.: 0.00000   3rd Qu.: 0.040044
 Max.   : 0.88889   Max.   : 0.88889   Max.   : 0.77778   Max.   : 0.88889   Max.   : 0.579600
   BILL_AMT2           BILL_AMT3           BILL_AMT4           BILL_AMT5           BILL_AMT6
 Min.   :-0.041931   Min.   :-0.094505   Min.   :-0.102160   Min.   :-0.036880   Min.   :-0.1256200
 1st Qu.: 0.001779   1st Qu.: 0.001623   1st Qu.: 0.001425   1st Qu.: 0.001071   1st Qu.: 0.0007626
 Median : 0.012758   Median : 0.012069   Median : 0.011418   Median : 0.010880   Median : 0.0102410
 Mean   : 0.029537   Mean   : 0.028155   Mean   : 0.025908   Mean   : 0.024184   Mean   : 0.0233087
 3rd Qu.: 0.038356   3rd Qu.: 0.035934   3rd Qu.: 0.032554   3rd Qu.: 0.030149   3rd Qu.: 0.0294830
 Max.   : 0.591270   Max.   : 1.000000   Max.   : 0.535780   Max.   : 0.557160   Max.   : 0.5778900
    PAY_AMT1            PAY_AMT2            PAY_AMT3            PAY_AMT4            PAY_AMT5
 Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000
 1st Qu.:0.0005884   1st Qu.:0.0004981   1st Qu.:0.0002316   1st Qu.:0.0001698   1st Qu.:0.0001526
 Median :0.0012468   Median :0.0011904   Median :0.0010687   Median :0.0008906   Median :0.0008906
 Mean   :0.0033455   Mean   :0.0034729   Mean   :0.0031329   Mean   :0.0029086   Mean   :0.0028623
 3rd Qu.:0.0029699   3rd Qu.:0.0029687   3rd Qu.:0.0026718   3rd Qu.:0.0023821   3rd Qu.:0.0024106
 Max.   :0.5186600   Max.   :1.0000000   Max.   :0.5320100   Max.   :0.3687100   Max.   :0.2532400
    PAY_AMT6            GRADSKL          UNIVERSITY         HIGHSKL          EDU_OTHER
 Min.   :0.0000000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
 1st Qu.:0.0000718   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000
 Median :0.0008906   Median :0.0000   Median :0.0000   Median :0.0000   Median :0.00000
 Mean   :0.0030762   Mean   :0.3531   Mean   :0.4684   Mean   :0.1627   Mean   :0.00425
 3rd Qu.:0.0023767   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.00000
 Max.   :0.3138900   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
    married            single       marital-others      default
 Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Length:24001
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   Class :character
 Median :0.0000   Median :0.0000   Median :0.00000   Mode  :character
 Mean   :0.4554   Mean   :0.2394   Mean   :0.06837
 3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.00000
 Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
```

It didn't show something too important but it was interesting to note that most of the customers had attended university or graduate school, and the most common marital status for a customer was "married".

Training, Validation and Testing

Disclaimer: One important thing to note here is that in this project, I have used the validation set as the one that predicts the final accuracy of a model. In our class, EE 660, we always did this with a test set, but the roles of validation and test set have been flipped here.

I used LDA, Support Vector Machines(SVM), K-Nearest Neighbors (KNN) and Random Forest (RF) algorithms to test which ones perform the best. I relied heavily on the "caret" package on R to train my algorithms so I didn't have to do much of the coding as was the case for the regression project.

To find out whether my models are any good. I split the loaded dataset into two, 80% (24,000) of it for training our models and 20% (6,000 samples) for using later as a validation dataset. I replaced the dataset variable with the 80% sample of the dataset to keep the code simple.

For evaluating our algorithm, I set up the test harness to use 10-fold cross validation to estimate accuracy. This split the dataset into 10 parts, train in 9 and test on 1 and then release for all combinations of train-test splits. The process was repeated 3 times for each algorithm with different splits of the data into 10 groups. This was done so I could get a more accurate estimate and was all done using "traincontrol" function, made available through the "caret" package.

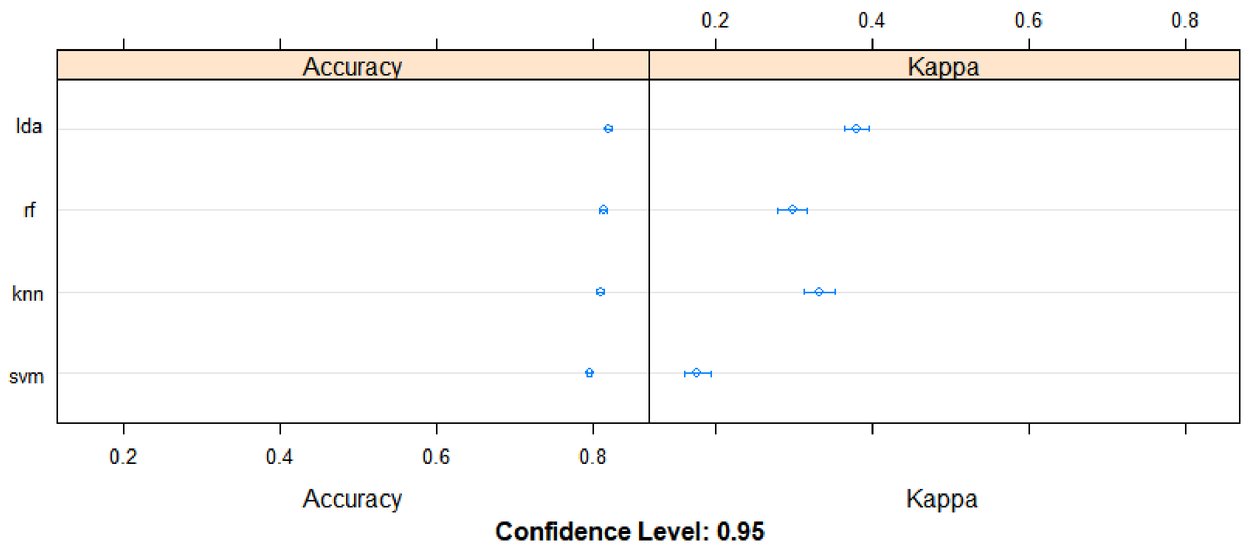
I used the metric of "Accuracy" to evaluate models (as can be seen in the R code). To elaborate further, this is a ratio of the number of correctly predicted instances (defaults/non-defaults) divided by the total number of instances in the dataset multiplied by 100 to give a percentage.

For the algorithms I tested, I used a mixture of different algorithm types. They were simple linear (LDA), nonlinear (kNN) and complex nonlinear methods (SVM, RF). I reset the random number seed before each run (using the "set.seed" command in R) to ensure each algorithm is evaluated using exactly the same data splits and that the results are directly comparable. Training the models is done simply using the "train" function (which is available from the "caret" package) and then specifying the dataset, the algorithm used, my metric (accuracy) and specifying how I train my control.

Next, I summarized the results from the cross-validation for each method and got the following summary:

```
Models: lda, knn, svm, rf
Number of resamples: 10

Accuracy
         Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
lda 0.8104956 0.8142518 0.8189583 0.8187998 0.8220833 0.8295833    0
knn 0.7938359 0.8040825 0.8093352 0.8083420 0.8137500 0.8158333    0
svm 0.7855060 0.7925000 0.7931681 0.7943005 0.7969792 0.8025000    0
rf  0.8034152 0.8067277 0.8145448 0.8126753 0.8162500 0.8233333    0
```



To better visualize the results (see above), I next used the dotplot function where we can see that the accuracy of all four models are very similar with LDA having a slight lead over the others. Comparing that with the fact that LDA takes only a few seconds while the others take hours, it is an easy decision to pick LDA as our classification algorithm

For the final step, after picking the LDA model from training I tested it on the validation set of 6000 samples that I had separated at the start of my script using the "predict" function

Final Results and interpretations

After fitting the LDA model on the validation set, I summarized the results using a confusion matrix in R. This is what I got.

```
Confusion Matrix and Statistics

             Reference
Prediction    Default Not default
  Default         318         154
  Not default    1009        4518

               Accuracy : 0.8061
                 95% CI : (0.7959, 0.8161)
    No Information Rate : 0.7788
    P-Value [Acc > NIR] : 1.218e-07

                  Kappa : 0.2686
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.23964
            Specificity : 0.96704
         Pos Pred Value : 0.67373
         Neg Pred Value : 0.81744
             Prevalence : 0.22120
         Detection Rate : 0.05301
   Detection Prevalence : 0.07868
      Balanced Accuracy : 0.60334
```

```
       'Positive' Class : Default
```

Conclusion

It seems that LDA works well for this set as it accurately predicts the outcome more than 80% of the time with a very tight 95% Confidence Interval and a p-value lower than 1.2*10^-7. It is important to note that most of errors are false negatives (default when predicted not-default) as opposed to false positives (not-default when predicted default), and credit lending companies may not like that. It is entirely reasonable for them to want to settle for a lower overall accuracy as long as the percentage of false negatives are reduced and they don’t lose money on defaulted loans. Without that prior information, however, LDA is still the best algorithm for this dataset from the ones I tried out, and it is also a relatively computationally efficient one.

Contributions of each team member- 1 member