**EC 48W Final Project Report**

**Group 9: Loan Default Prediction**

**Denizhan Özyurt**

**Ezgi Berksoy**

**Ozan Yagcilar**

**Ömer Bahar**

**Zeki Berkay Saygin**

**Abstract**

This is a final report for our project for EC48W. We tried to develop a Loan Default Prediction Model by using the dataset provided for a competition in Kaggle. We prioritized 166 features among 800 from an anonymous feature set for 200,000 observations and developed a Random Forest model using Python. Although our model performed well with cross validation score of 90.66%, it has a large headroom for improvement with optimizations when compared with winner in Kaggle competition. This report summarizes our efforts to apply a machine learning model for the first time.

**Introduction**

We picked the loan default prediction as a topic since it is such an knowledge-abundant and challenging topic as there are a great number of people working on the topic. A loan default prediction is essentially a model which calculates the probability of default for a company or an individual. With the recent regulations of Basel Committee, all commercial and public banks are expected to develop their own probability of default model and calculate the probability of default for every loan they issue. Their models also calculate the other features like Loss Given Default and Exposure at Default. Our kaggle challenge and the model we built only focuses on developing a model which calculates the Probability of Default. Aim of this competition is to maximize the cross validation score, which is calculated by the mean absolute of the model. Our data had 800 features for each of 200,000 observations and an output indicating weather the customer defaulted or not. After exploring and cleaning the data, we developed a Random Forest Model, which takes features that were prioritized by our model as input and gives a binary output whether the customer will go default or not. Our model had 0.9066 cross validation score. This is a very hot topic where banks worldwide are trying to develop better models to increase their profits. So our model performed sub-par compared to models that were posted in Kaggle competition. However, we developed ourselves with the skills to build a simple and sound Random Forest Model and test it.

**Methods Review and Background**

This competition case was posted on Kaggle.com by Imperial College London 5 years ago with the name of Loan Default Prediction. Loan Default Prediction requires a model that either comes up with a probability distribution model (e.g. logistic regression) or a model which can produce binary outputs. We discussed over a few models including: Logistic regression, k-nearest neighbour classification, support vector machine and random forest classification. Logistic model has been used over many years in the industry. So we wanted to apply a different model, which we can also apply in our future work with different datasets. As it was also suggested by our professor, we picked the Random Forest Classification method. We applied scikit-learn libraries on Python after covering the documentations about the Random Forest Model on scikit-learn. We used pandas, numpy, scikit-learn, matplotlib and seaborn libraries.

**Statistical Analysis, Data Exploration and Cleaning**

We had a massive data that we had to understand and clean before applying the Random Forest Classification Model. Our dataset (kaggle\_loan\_data.csv) consists of 105,471 observations with 778 variables and an output variable of loss. 653 of these variables were in float type, 99 was in integer type and 19 was in object type.

We looked at the main statistics of the data by using the . Describe function and checked the following statistics for each feature: count, mean, standard deviation, min, 25%, 50%, 75% and max. Since our data was anonymized, we couldn’t interpret much by just looking at these descriptive statistics but we realized that statistics were varying highly for each feature, so we decided to standardize our features before applying machine learning tools. We used a function that transforms a vector into standard scalar form.

Another important point we realized was the quality of the data. Our train data had a lot of missing values. 525 of our variables had missing values in at least one of their observations. For some features e.g. f662 17.9% of all observations were missing. So we filled these missing values with the mean of the observations for that variable. After that, we still had a few variables that included NAs. So we dropped down these variables. Our final data that had no NAs had 771 variables.

The third and the most important pillar of our data cleaning was to remove the collinear variables. Since our feature set was very large as 771, we thought that many of those features might actually be collinear. The more the number of features is, there is a better chance for features to be affected by similar causes and thus be collinear. So we created a correlation matrix and checked the correlations of each two features. Later we used a function that removes collinear features in a dataset if their correlation is above a threshold. We defined the threshold as 0.6, since the smaller the data the better it is for the Jupyter notebooks we worked on. So the number of features we have dropped down to 156. Later we randomly splitted our Train data into two data sets: 80% training and 20% test.

**Methods and Tools of Use**

We built a Random Forest model to predict the loan defaults by using the functions in the scikit-library. Random Forest is essentially a model built on a decision trees. Decision trees are supervised learning algorithms used for classification and regression tasks. Our data needs a classification on whether the customer will default or not. Decision trees are assigned to the information based learning algorithms which use different measures of information gain for learning. The main idea of decision trees is to find those descriptive features which contain the most "information" regarding the target, loss in our case. This process of finding the most helpful feature is done until it arrives to a stopping criteria where it ends up in leaf nodes.

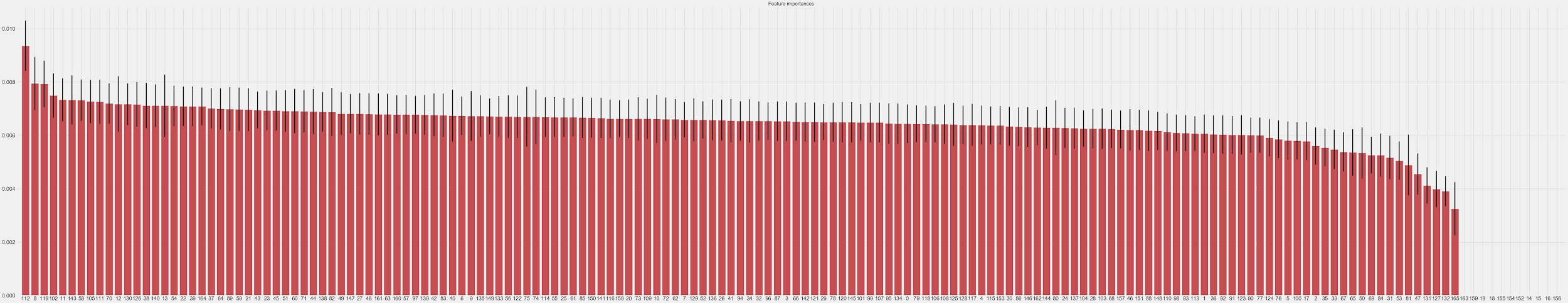
Random Forest builds an ensemble of decision trees and it combines different learning methods to increase overall result. That is the reason why this model can predict a more accurate and stable predictions. Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. As random forests select a subset of features in each of its decision trees thereby reducing the bias (because of high importance of single feature) of the model. The final output will be the mode of the outputs of all its decision trees which has better results than decision trees (which can possibly overfit).

Since our data is completely anonymous, we can’t prefer one feature over another discreetly. That’s why the way random forest treats each features equally helps us a great deal. So we used the random forest model for classification.

**Random Forest Model**

We separated the data we cleaned into two: Train and Test. We randomly splitted the data into half as the following: 80% Train, 20% Test.

Later we trained the model with the train data (X\_train, y\_train) by using “ExtraTreesClassifier” function from scikit-learn library with n\_estimators = 250) We fitted the model and calculated the importance score of each feature. Later we listed the features according to their importance scores. (Check the appendix for the importance-sorted list of features)



Using an extra tree classifier we analysed the importance of each feature and realized that some features have no impact on the classification, except one feature, mainly FX, rest of the features has pretty similar impact and the overall impact for each feature is low, meaning the combination of features is more important than a couple of main features. Check the appendix for the code we used for fitting the model.

**Results**

After fitting the train data (X\_train, y\_train) with a random forest model, we tested the success of our model by looking at Mean Absolute Errors with a cross validation score on the test data we previously separated randomly. (X\_test, y\_test) Check the appendix for the code we used for testing the model.

Our cross validation score was 0.9066, which was higher than some solution attempts posted on Kaggle.

As our decision tree had 166 features, it was not intuitive to plot the graph for this.

**Resources and References**

**Kaggle Competition**

<https://www.kaggle.com/c/loan-default-prediction/overview>

**Dropbox Link for Our Data** (kaggle\_loan\_data.csv)

<https://www.dropbox.com/s/t67zsa44iebunwa/kaggle_loan_data.csv?dl=0>

**Scikit Learn Random Forest Classifier Documentations**

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

<https://scikit-learn.org/stable/modules/cross_validation.html>

**Random Forest Blogs on Medium.com**

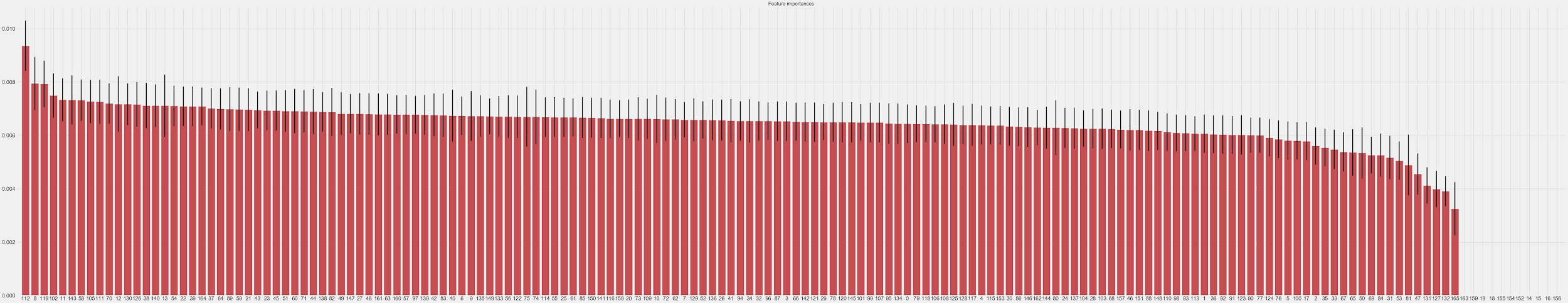
<https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d>

<https://medium.com/datadriveninvestor/k-fold-cross-validation-6b8518070833>

<https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd>

**Appendix**

**Feature List Importances Graph**



**Features - Sorted According to Their Importances**

1. feature 112 (0.009354)  
2. feature 8 (0.007943)  
3. feature 119 (0.007926)  
4. feature 102 (0.007490)  
5. feature 11 (0.007334)  
6. feature 143 (0.007327)  
7. feature 58 (0.007319)  
8. feature 105 (0.007268)  
9. feature 111 (0.007261)  
10. feature 70 (0.007196)  
11. feature 12 (0.007172)  
12. feature 130 (0.007171)  
13. feature 126 (0.007161)  
14. feature 38 (0.007115)  
15. feature 140 (0.007111)  
16. feature 13 (0.007109)  
17. feature 54 (0.007106)  
18. feature 22 (0.007087)  
19. feature 39 (0.007086)  
20. feature 164 (0.007085)  
21. feature 37 (0.007012)  
22. feature 64 (0.006993)  
23. feature 89 (0.006980)  
24. feature 59 (0.006978)  
25. feature 21 (0.006961)  
26. feature 43 (0.006950)  
27. feature 23 (0.006930)  
28. feature 45 (0.006927)  
29. feature 51 (0.006909)  
30. feature 60 (0.006906)  
31. feature 71 (0.006903)  
32. feature 44 (0.006892)  
33. feature 138 (0.006880)  
34. feature 82 (0.006875)  
35. feature 49 (0.006811)  
36. feature 147 (0.006811)  
37. feature 27 (0.006808)  
38. feature 48 (0.006802)  
39. feature 161 (0.006789)  
40. feature 63 (0.006788)  
41. feature 160 (0.006783)  
42. feature 57 (0.006781)  
43. feature 97 (0.006777)  
44. feature 139 (0.006767)  
45. feature 42 (0.006765)  
46. feature 83 (0.006750)  
47. feature 40 (0.006737)  
48. feature 6 (0.006732)  
49. feature 9 (0.006724)  
50. feature 135 (0.006722)  
51. feature 149 (0.006712)  
52. feature 133 (0.006705)  
53. feature 56 (0.006701)  
54. feature 122 (0.006698)  
55. feature 75 (0.006697)  
56. feature 74 (0.006694)  
57. feature 114 (0.006684)  
58. feature 55 (0.006680)  
59. feature 25 (0.006680)  
60. feature 61 (0.006674)  
61. feature 85 (0.006666)  
62. feature 150 (0.006660)  
63. feature 141 (0.006654)  
64. feature 116 (0.006625)  
65. feature 158 (0.006625)  
66. feature 20 (0.006623)  
67. feature 73 (0.006621)  
68. feature 109 (0.006618)  
69. feature 10 (0.006617)  
70. feature 72 (0.006603)  
71. feature 62 (0.006603)  
72. feature 7 (0.006589)  
73. feature 129 (0.006582)  
74. feature 52 (0.006581)  
75. feature 136 (0.006579)  
76. feature 26 (0.006570)  
77. feature 41 (0.006549)  
78. feature 94 (0.006542)  
79. feature 34 (0.006539)  
80. feature 32 (0.006538)  
81. feature 96 (0.006535)  
82. feature 87 (0.006528)  
83. feature 3 (0.006526)  
84. feature 66 (0.006514)  
85. feature 142 (0.006503)  
86. feature 121 (0.006498)  
87. feature 29 (0.006496)  
88. feature 78 (0.006491)  
89. feature 120 (0.006490)  
90. feature 145 (0.006489)  
91. feature 101 (0.006483)  
92. feature 99 (0.006482)  
93. feature 107 (0.006481)  
94. feature 95 (0.006442)  
95. feature 134 (0.006438)  
96. feature 0 (0.006438)  
97. feature 79 (0.006432)  
98. feature 118 (0.006430)  
99. feature 106 (0.006421)  
100. feature 108 (0.006417)  
101. feature 125 (0.006408)  
102. feature 128 (0.006392)  
103. feature 117 (0.006391)  
104. feature 4 (0.006384)  
105. feature 115 (0.006374)  
106. feature 153 (0.006373)  
107. feature 30 (0.006333)  
108. feature 86 (0.006322)  
109. feature 146 (0.006311)  
110. feature 162 (0.006296)  
111. feature 144 (0.006289)  
112. feature 80 (0.006288)  
113. feature 24 (0.006283)  
114. feature 137 (0.006269)  
115. feature 104 (0.006256)  
116. feature 28 (0.006255)  
117. feature 103 (0.006251)  
118. feature 68 (0.006240)  
119. feature 157 (0.006217)  
120. feature 46 (0.006206)  
121. feature 151 (0.006205)  
122. feature 88 (0.006174)  
123. feature 148 (0.006167)  
124. feature 110 (0.006120)  
125. feature 98 (0.006090)  
126. feature 93 (0.006085)  
127. feature 113 (0.006067)  
128. feature 1 (0.006065)  
129. feature 36 (0.006039)  
130. feature 92 (0.006032)  
131. feature 91 (0.006019)  
132. feature 123 (0.006019)  
133. feature 90 (0.006007)  
134. feature 77 (0.006005)  
135. feature 124 (0.005911)  
136. feature 76 (0.005853)  
137. feature 5 (0.005804)  
138. feature 100 (0.005794)  
139. feature 17 (0.005782)  
140. feature 2 (0.005599)  
141. feature 35 (0.005541)  
142. feature 33 (0.005473)  
143. feature 67 (0.005377)  
144. feature 65 (0.005360)  
145. feature 50 (0.005339)  
146. feature 69 (0.005257)  
147. feature 84 (0.005257)  
148. feature 31 (0.005170)  
149. feature 53 (0.005049)  
150. feature 81 (0.004891)  
151. feature 47 (0.004546)  
152. feature 131 (0.004123)  
153. feature 127 (0.003984)  
154. feature 132 (0.003908)  
155. feature 165 (0.003250)  
156. feature 163 (0.000000)  
157. feature 159 (0.000000)  
158. feature 19 (0.000000)  
159. feature 18 (0.000000)  
160. feature 155 (0.000000)  
161. feature 154 (0.000000)  
162. feature 152 (0.000000)  
163. feature 14 (0.000000)  
164. feature 15 (0.000000)  
165. feature 16 (0.000000)  
166. feature 156 (0.000000)

**Code for Fitting the Model**

# Build a classification task using 3 informative features

X, y = X\_train, y\_train

# Build a forest and compute the feature importances

forest = ExtraTreesClassifier(n\_estimators=250,

random\_state=0)

forest.fit(X, y)

importances = forest.feature\_importances\_

std = np.std([tree.feature\_importances\_ for tree in forest.estimators\_],

axis=0)

indices = np.argsort(importances)[::-1]

**Code for Testing the Model**

# Cross Validation Calculation with Mean Absolute Error

# Function to calculate mean absolute error

def cross\_val(X\_train, y\_train, model):

# Applying k-Fold Cross Validation

from sklearn.model\_selection import cross\_val\_score

accuracies = cross\_val\_score(estimator = model, X = X\_train, y = y\_train, cv = 5)

return accuracies.mean()

# Takes in a model, trains the model, and evaluates the model on the test set

def fit\_and\_evaluate(model):

# Train the model

model.fit(X\_train, y\_train)

# Make predictions and evalute

model\_pred = model.predict(X\_test)

model\_cross = cross\_val(X\_train, y\_train, model)

# Return the performance metric

return model\_cross

# # Random Forest Classification

from sklearn.ensemble import RandomForestClassifier

random = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy')

random\_cross = fit\_and\_evaluate(random)

print('Random Forest Performance on the test set: Cross Validation Score = %0.4f' % random\_cross)