Comparison of Statistical Learning Methods in Credit Risk Prediction

Chandler Chang and Ryan Hayden
Final Project for STP 598

1. Background

1.1. Brief Overview of Project Goals

In this project, we will investigate a variety of modern statistical learning methods based upon regression models, decision trees, and neural networks. In particular, we will focus on the use of methods that employ strategies like regularization, bootstrapping/bagging, and gradient boosting, as well as explain the statistical justification for such strategies. In order to illustrate the effectiveness of the chosen methods, as well as analyze their weaknesses, we will analyze a popular dataset known as the German Credit Dataset, wherein the goal is to classify people applying for loans as either credit worthy or credit unworthy, based on several features like age, employment status, and bank account balance. In addition to comparing and contrasting the methods, we will also provide the relevant portions of our code so that the results are reproducible.

We also note that this is a particularly interesting application of statistical learning, as the problem essentially boils down to one of risk prediction: given an individuals financial or personal attributes, can we assess whether lending them money is risky or not? This is a problem that is at the core of several modern industries, for example the fintech industry is full of billion dollar companies like SoFi who in large part aim to find ”under-banked” individuals who are supposedly worthy of loans/credit but previously would have been denied/overlooked. As such, improving credit risk models is quite literally a million dollar problem. With that being said, we hope that our investigation provides insight into some of the statistical methodology behind risk assessment and credit decisioning.

Fig. 1: Our goal is to classify credit applicants as either good credit or bad credit applicants based on a variety of traits like age, bank account status, and loan amount

1.2. Classification as a Statistical Learning Problem

We are interested in a subclass of statistical learning problems known as statistical classification. Given a set of predictors X, we want to predict an outcome Y_i, where the set $\{Y_i : i \in S \subseteq \mathbb{N}\}$ is the set of possible classes. Specifically, our credit risk prediction problem involves predicting 2 outcomes, credit worthy or credit unworthy, which is known as a binary classification problem. The prediction of discrete outcomes is what makes classification problems different from regression problems in a statistical learning context. Hence, while the statistical learning methods involved in both paradigms are deeply related, there is also a set of specialized techniques for tackling classification problems. In this project we will investigate these specialized methods including models like logistic regression and evaluation metrics like receiver operating characteristic (ROC) curves.

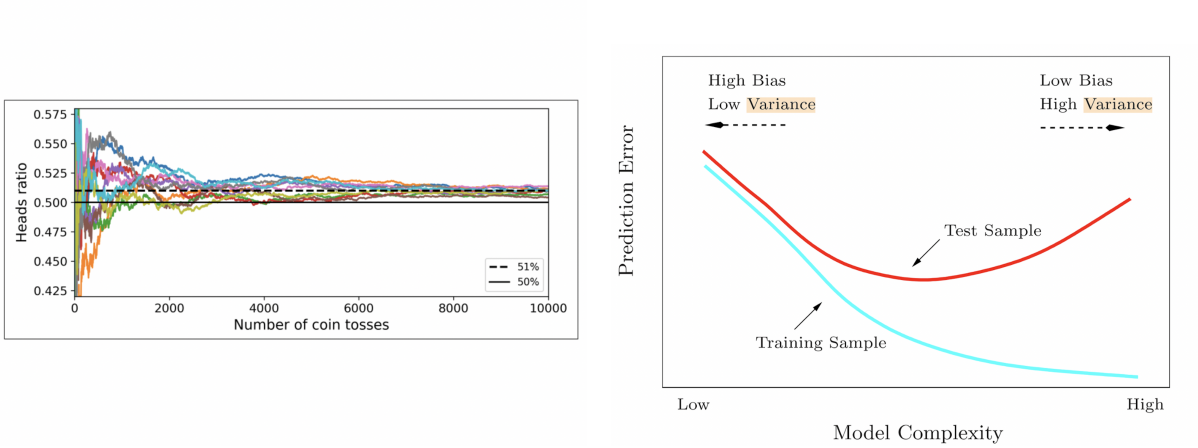


Fig. 2: The law of large numbers and the bias-variance trade-off are key concepts exploited in a lot of statistical learning methods, particularly in the context of flexibility and overfitting. Such methods include the use of regularization, ensemble learning, boosting, bagging, and dropout, all of which we will explore in our investigation

1.3. Dataset

The dataset was obtained from UC Irvine's Machine Learning Repository and classifies individuals described by a set of attributes as either good or bad credit risks. It contains 1000 rows of 20 features (both numeric and categorical) and a binary label for good or bad credit risk. The complete list of features can be found using the provided link, the most important ones will be articulated later.

Raw data link: https://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	...	A12	A13	A14	A15	A16	A17	A18	A19	A20	Label
0	A11	6	A34	A43	1169	A65	A75	4	A93	A101	...	A121	67	A143	A152	2	A173	1	A192	A201	1
1	A12	48	A32	A43	5951	A61	A73	2	A92	A101	...	A121	22	A143	A152	1	A173	1	A191	A201	0
2	A14	12	A34	A46	2096	A61	A74	2	A93	A101	...	A121	49	A143	A152	1	A172	2	A191	A201	1
3	A11	42	A32	A42	7882	A61	A74	2	A93	A103	...	A122	45	A143	A153	1	A173	2	A191	A201	1
4	A11	24	A33	A40	4870	A61	A73	3	A93	A101	...	A124	53	A143	A153	2	A173	2	A191	A201	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	A14	12	A32	A42	1736	A61	A74	3	A92	A101	...	A121	31	A143	A152	1	A172	1	A191	A201	1
996	A11	30	A32	A41	3857	A61	A73	4	A91	A101	...	A122	40	A143	A152	1	A174	1	A192	A201	1
997	A14	12	A32	A43	804	A61	A75	4	A93	A101	...	A123	38	A143	A152	1	A173	1	A191	A201	1
998	A11	45	A32	A43	1845	A61	A73	4	A93	A101	...	A124	23	A143	A153	1	A173	1	A192	A201	0
999	A12	45	A34	A41	4576	A62	A71	3	A93	A101	...	A123	27	A143	A152	1	A173	1	A191	A201	1

1000 rows × 21 columns

Fig. 3: A portion of the raw German credit data, containing 1000 samples and 20 features. Before our investigation begins we must one-hot encode the categorical variables and normalize the data to make it better suited for our classification methods

2. Methods and Investigation

2.1. Data Preparation

2.1.1. One Hot Encoding

The data set contains both categorical and numerical features. In order to perform analysis, we need to be able turn the categorical features into numerical ones do to numerical comparisons and computations. This is done using One Hot Encoding. For a categorical feature, each of the possible categories is made into a separate column and assigned a binary value of 0 or 1. In our example, the A1 feature is split into 4 columns: A1_A11, A1_A12, A1_A13, A1_A14, where a 1 in a given column (A1_A11) corresponds to the assigned category (A11). This turns the 20 features from the default data set into a total of 61 features.

2.1.2. Normalization

All of the features are now numerical, but have wide-ranging values among features. For example, the age feature can range from 18-99 but any categorical feature can only be 0 or 1. This is not ideal for methods such as regression-based methods and neural nets, as the difference in values will cause errors in how the models weight each feature. Since we have one hot encoded the categorical variables to 0s and 1s, we scaled all other numerical features from 0 to 1 to match. Thus, numerically, the appropriate algorithms will value each feature equally.

2.2. Evaluation Metrics

When evaluating the performance of classification models, we have to employ particular metrics, as standard metrics like RMSE which are used in regression problems. In particular, we will use what is known as a Receiver Operating Characteristic (ROC) Curve and the associated Area Under ROC Curve (AUC). These methods are preferred over methods like RMSE because they focus on the ability of the model to correctly distinguish different classes, while RMSE measures the distance between predicted result and actual result. In fact, RMSE's focus on squared distances can lead to flawed results and it has been shown that in the context of a binary classification problem, RMSE is not guaranteed to minimize the cost function.

As such, in this project we will evaluate our methods with ROC/AUC. As we learned in lecture, the ROC curve plots the false positive rate against the true positive rate at different classification thresholds. Moreover, the AUC is simply the 0-to-1 valued area under the ROC curve, and our goal is to get the AUC score as close to 1 as possible, as this indicates a model which correctly classifies 100% of the data. It should also be noted that AUC is scale-invariant, which is particularly relevant in a lot of statistical learning problems. Now that we have explained the data and evaluation methods, we will explain the models we will use in our investigation.

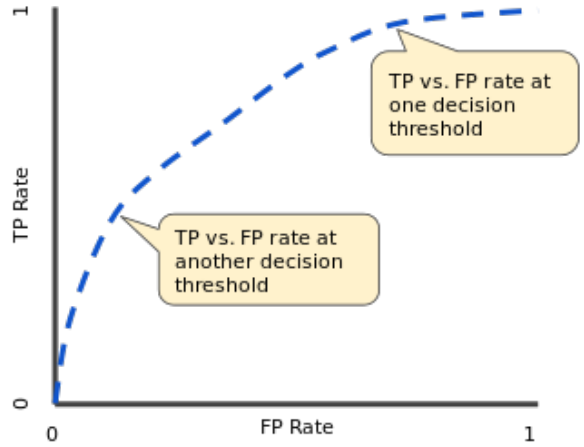


Fig. 4: Classification problems require the use of a specific evaluation metric known as an ROC curve, from which we can calculate the AUC score

2.3. Regression Models

2.3.1. Lasso Regression

LASSO, or Least Absolute Shrinkage and Selection Operator, is a type of linear regression that uses shrinkage, where data values are shrunk towards a central point like the mean. LASSO performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of the coefficients. This type of regularization encourages more sparse models with fewer features used, as this can set some of the coefficients to zero and eliminate the corresponding features from the model. As a result, the models from LASSO are often more interpretable than Ridge.

This type of model typically gives rational predictions instead of binary ones. We want binary predictions, so we will test different cutoffs for where to round predictions to 1 and accept the best cutoff in terms of error.

Minimization problem:

$$\sum_{i=1}^{n}\left(y_i - \sum_j x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Where λ is the shrinkage parameter. Note that if $\lambda = 0$, then we have least squares regression.

The figure below shows how our LASSO model selected coefficients for different values of λ for this data set. The x-axis is inverted logistically, so λ decreases as you go right on the x-axis. Note that as λ increases, the coefficients shrink towards zero. The dashed black line shows the cross-validated value for λ that our model chose to use.

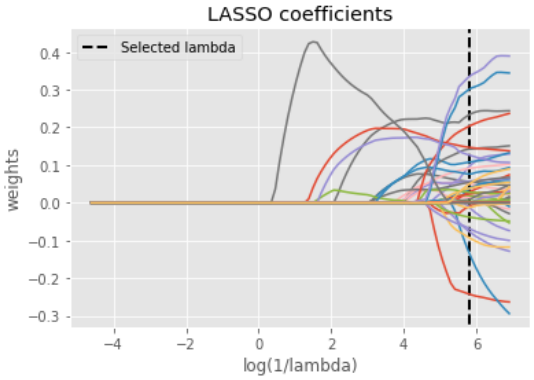


Fig. 5: LASSO coefficients as a function of lambda

2.3.2. Ridge Regression

Ridge Regression works very similar to LASSO, except that a penalty is added equal to the square of each coefficient (L2 regularization). Unlike LASSO, this does not result in a more sparse model, as each coefficient is never set to zero (although can be very close). Scikit has a method for Ridge Binary Classification, so there is no need to test different cutoffs in this method like in LASSO.

Minimization problem:

$$\sum_{i=1}^{n}\left(y_i - \sum_j x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

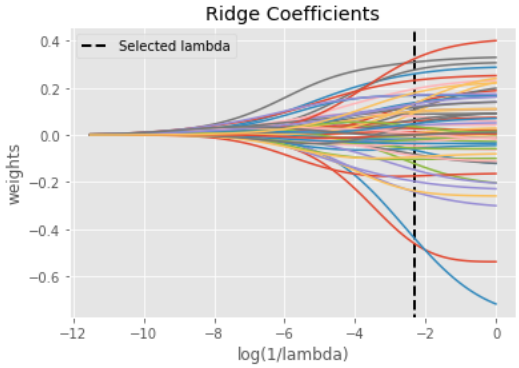


Fig. 6: Ridge coefficients as a function of lambda

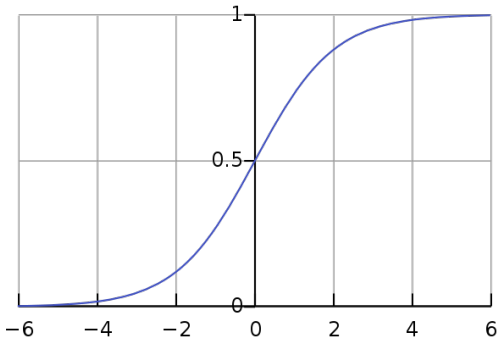


Fig. 7: The logistic function is used to convert log-odds to probabilities in logistic regression

2.3.3. Logistic Regression

As an alternative to the LASSO and RIDGE regression frameworks, we also investigate the Logistic regression model. In contrast with the linear models of LASSO and RIDGE, logistic regression makes use of a sigmoid function, most commonly the namesake logistic function, pictured in Figure 7. Logistic regression is often preferred over linear methods as it outputs a probability value which corresponds to the probability of belonging to a certain class (ie: credit worthy or credit unworthy). This is done by converting the log-odds via the logistic function, which outputs continuous values from 0 to 1. To be specific, the logistic regression model is not a classifier per se, but its results can easily be fed into a classification model. The logistic model is relatively simple in the sense that it maintains flexibility by maximizing entropy and therefor making minimal assumptions regarding the data.

In particular, the logistic model can be described as follows, where σ is a scale parameter and μ is a location parameter, and the output is a log-odds transformed into a class probability:

$$probability(x) = \frac{1}{1 + e^{-(\mu+\sigma x)}}$$

Our goal is to estimate the parameters σ and μ which yield the best "fit" as determined by our choice of performance metric. Maximum likelihood is often the method used to estimate these parameters. The results of the logistic model trained and tested on the German credit data set can be seen in Figure 8. We found that the logistic model out-performs the Ridge regression model and under-performs the LASSO classifier when evaluated using ROC/AUC. We will further analyze the reasons for these contrasting results in our analysis section.

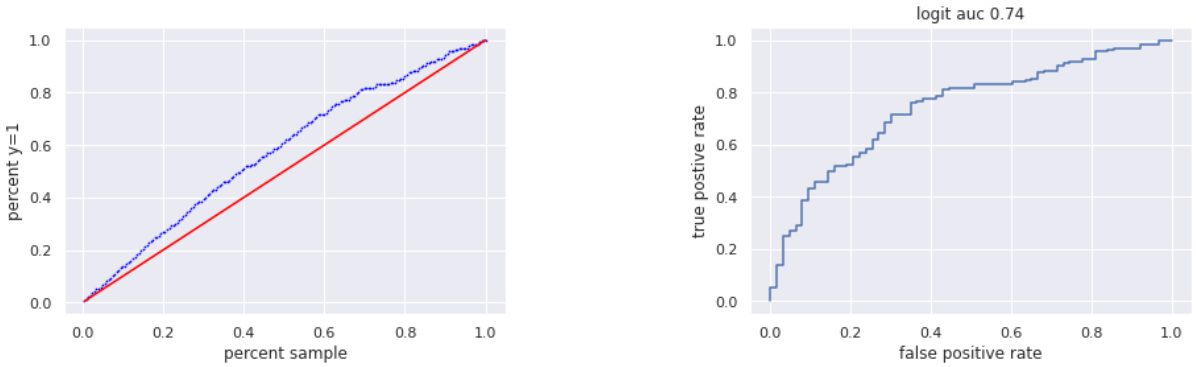


Fig. 8: Evaluation results from our implementation of logistic regression

2.4. *Tree Based Methods*

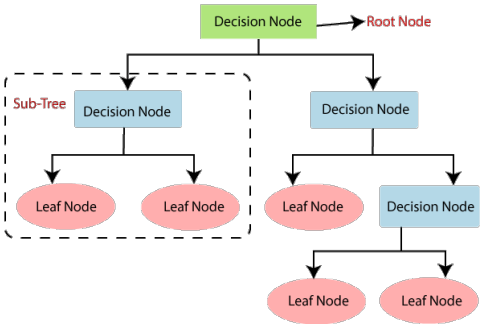


Fig. 9: The simple decision tree forms the basis of several popular methods in statistical learning, most notably random forests and gradient boosted trees

2.4.1. Simple Decision Tree

A completely different paradigm to the regression frameworks we have discussed is that of the decision tree. Decision trees are popular methods for a variety of reasons, particularily their model versatility, immunity to data scale/magnitude, and ability to accurately fit complicated datasets. While decision trees can be used in both the classification and regression settings, we will focus on classification trees and their application to the German credit data set. Essentially, a decision tree is exactly what it sounds like: a tree-like chart, where nodes represent decision rules and the leaf nodes represent final decisions. An example of a simple decision tree is pictured in Figure 9, where the simple flow-chart like nature of the tree can be seen.

In the context of statistical learning, we want to estimate the best decision tree for the given data, which means we want to choose which decision nodes to include in our tree. For example, in regards to the German credit data set, a possible decision node is one that asks whether the age of the credit applicant is above or below some threshold. This decision node then is able to partition the data into two branches, one which is below the age threshold and one which is above. Of course, in practice we will have much more than just one decision node though. It should also be noted that the primary metric we are concerned with in modeling a decision tree is the Gini impurity, $G_i = 1 - \sum_{k=1}^{n} p_{1,k}^2$, where $p_{i,k}$ is the ratio in node i of class k instances among all the training instances. The Gini impurity essentially is a measure of how well a certain decision node "splits" or "partitions" the data.

The results of our application of a simple decision tree on the German credit data can be seen in Figure 10. Although its performance is not as good as the more complicated methods like random forests, it performed surprisingly well for such a simple and easy to set-up method.

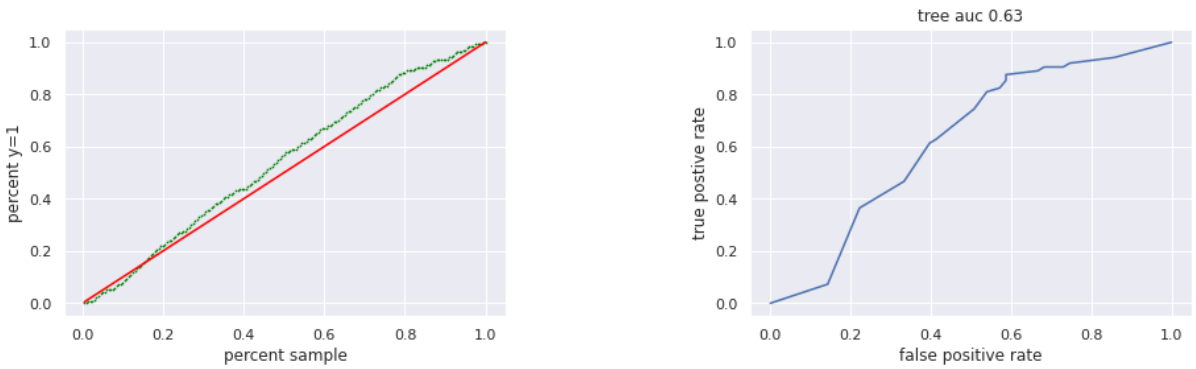


Fig. 10: Evaluation results from our implementation of a simple decision tree classifier

2.4.2. Random Forests

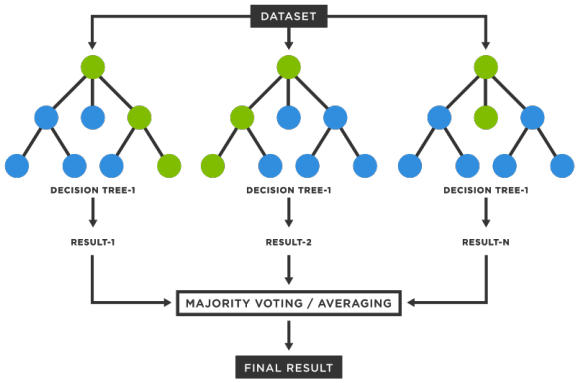


Fig. 11: random forests

Random forests are one of the most popular and effective methods in machine learning. It is a prime example of what is known as an ensemble learning method, which utilize simple methods like a decision tree as component classifiers which are trained in tandem and then a final classification output is decided by majority vote or other comparable methods (seen in Figure 11). This is a powerful technique which takes advantage of several statistical strategies such as bootstrap aggregation (bagging) and randomness, all of which allow the control of variance. As such, random forests are a great example of fundamental concepts in statistical learning such as the bias-variance trade off.

Random forests are named as such because they consist of a ”forest” of simple decision trees, each of which is trained on a random subset of the training data. As mentioned previously, the classification predictions of each of the trees is then averaged to make a final prediction. This process is known as aggregation and due to the law of large numbers, it is the principal source of a random forests effectiveness. Regarding the model details, each tree is trained as described in the above section on simple decision trees, moreover when we train each tree on a random subset of the data, this is done with sampling with replacement, which is a method known as bagging. In addition, random forests are popular for a variety of reasons beyond their high effectiveness, particularly their ability to yield feature importance insights, which we will elaborate on in the analysis section at the end.

The results of our application of a random forest classifier on the German credit data can be seen in Figure 12. The results are impressive and was only outperformed by the LASSO classifier, which helps justify the widespread use of random forests as an easy to use of-the-shelf classifier.

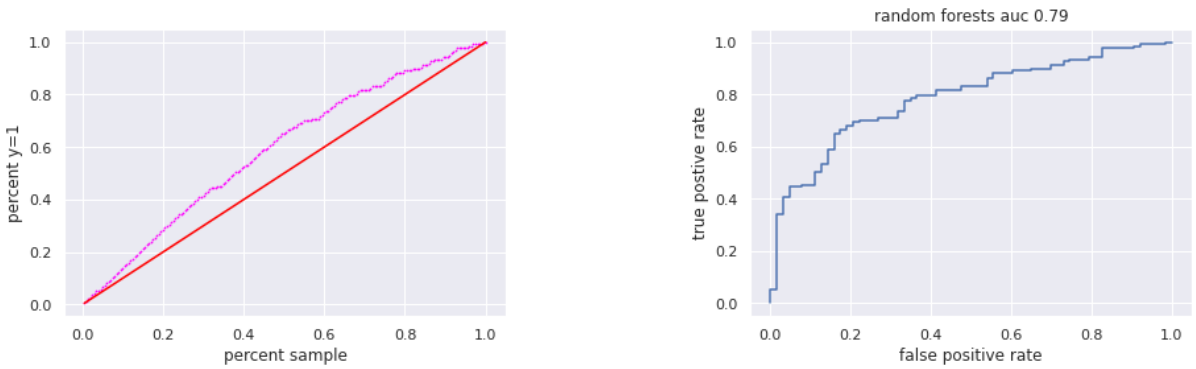


Fig. 12: Evaluation results from our implementation of a random forest classifier

2.4.3. Gradient Boosted Trees

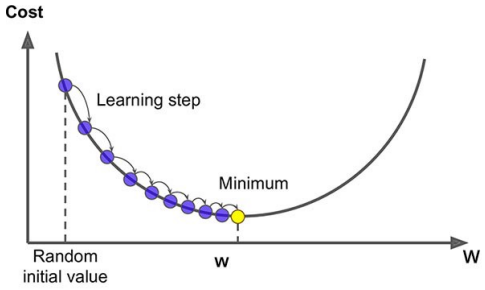


Fig. 13: The method of gradient descent is used to iteratively produce better fits to the data

An alternative to random forests which is just as popular is the method known as gradient boosted trees. This is another example of a ensemble learning method, except now we train the component trees sequentially in order to iteratively correct the previous trees errors in classification. This is where the idea of gradient boosting comes into play as we fit the next tree in the sequence to the residual errors from the previous trees using the methods of gradient descent. One of the most common machine learning libraries is XGBoost (extreme gradient boosting) which makes use of these same methods.

The results of our application of gradient boosted trees to the German credit data set can be seen in Figure 14, and a comparison of all 3 tree based methods can be seen in Figure 15 below.

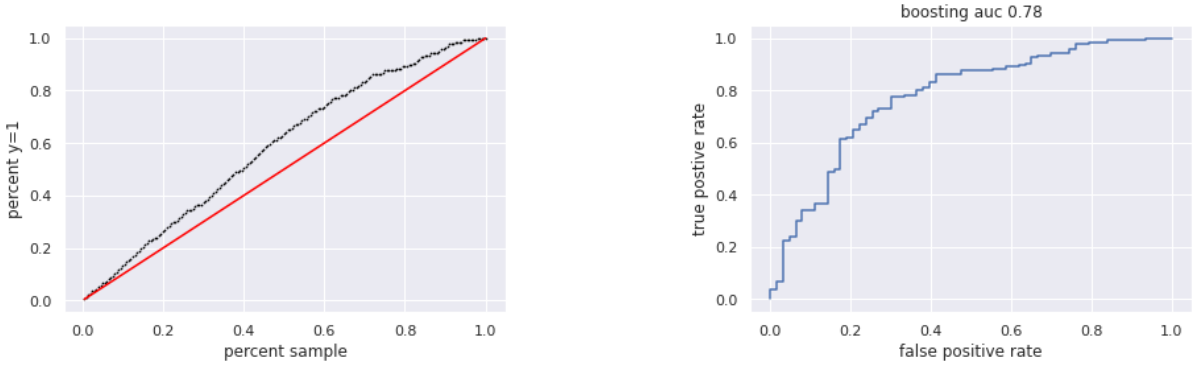


Fig. 14: Evaluation results from our implementation of gradient boosted decision trees

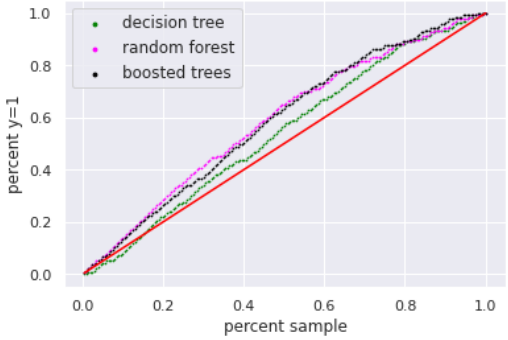


Fig. 15: Comparison of accuracy across the three different tree based methods

2.5. *Neural Networks*

The final statistical learning paradigm we will investigate is that of neural networks, which are highly popular and often impressively effective. A neural network is composed of node layers: an input layer for the data, hidden layers for optimization, and an output layer for the prediction. A node is simply where a computation occurs. Each node combines its input data with a set of coefficients, or weights, that either dampen of amplify that input. The selection of weights allows each node to choose how significant its input is with regard to what it is trying to model. These input-weight products are summed and then the sum is passed through a node's so-called activation function, to determine whether and to what extent that signal should progress further through the network to affect the ultimate outcome, in this case, a classification.

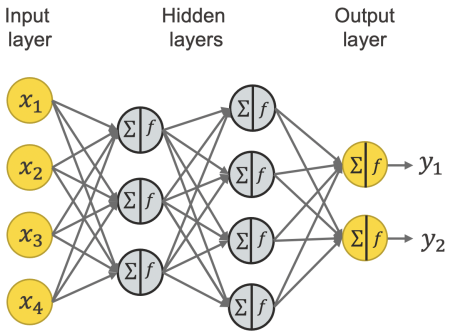
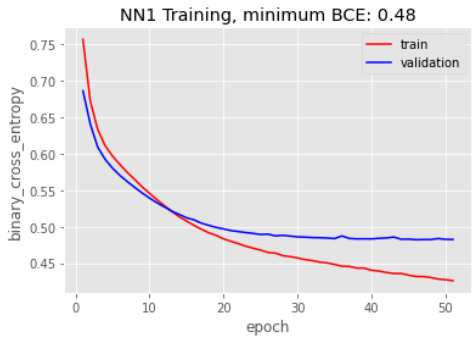


Fig. 16: Example of Neural Network

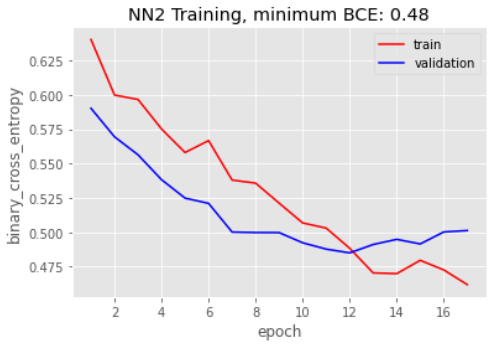
2.5.1. Activation Function, Early Stopping, and Dropout

There are many different variables to set when creating a neural network. We used RELU, or rectified linear unit, as the activation function in all nodes.

Early stopping prevents a neural network from overfitting by stopping its optimization when its out of sample error begins to climb. Our early stopping method stops the model after five epochs without reaching a new minimum error. Dropout is another way to regularize a neural net fit. It randomly picks some of the connections between nodes to eliminate.



(a) Training/Validation for Neural Net 1

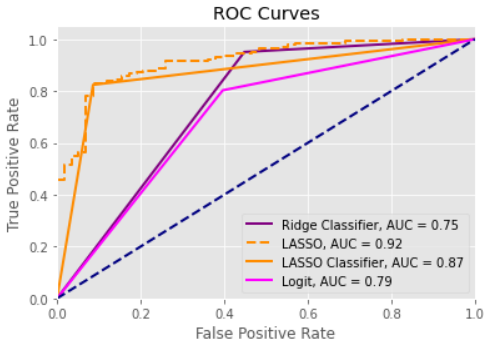


(b) Training/Validation for Neural Net 2

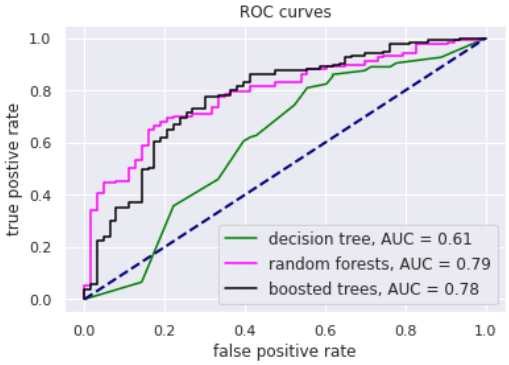
Fig. 17: Neural Net Optimization

For our analysis, we used two different neural network models. Neural Net 1 (shallow) uses a single hidden layer of 20 nodes and early stopping. Neural Net 2 (deep) uses five hidden layers of 50 nodes per layer, a dropout rate of 0.4, and early stopping.

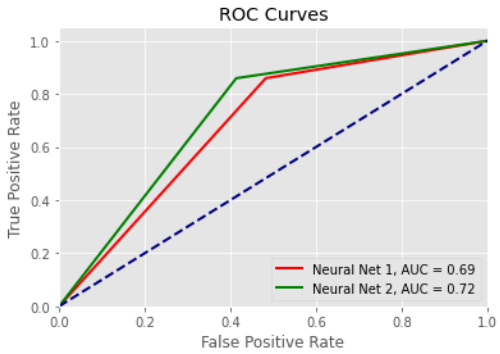
3. Analysis & Conclusion



(a) Regression-based methods



(b) Tree-based methods



(c) Neural Nets

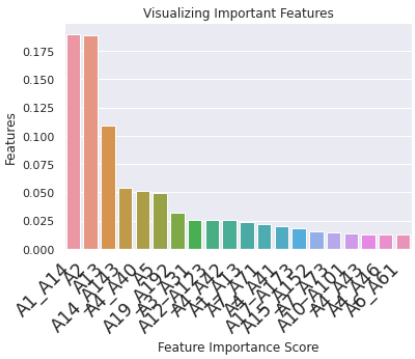
Fig. 18: Comparison of ROC curves and AUC scores across the 9 different classification methods

All of the investigated methods performed relatively well, especially considering the small amount of data each model was trained on (only 800 data points). However, the LASSO regression, logistic regression, random forest, and boosted trees classifiers performed exceptionally well (.79-.87 AUC scores). Unsurprisingly the simple decision tree and shallow neural net performed the least effectively. It should be noted that the fact the LASSO model performed the best could indicate that a simple model is inherently better suited for this data set. This is also evidenced by the fact that the deep neural net only slightly outperformed the shallow neural net. Overall the fact that we were able to consistently get results better than .75 AUC indicates that the 61 features we predicted on correlate pretty strongly with the risk evaluation outcomes. Figure 18 above indicates the comparative performance of the 9 different classification methods that we tested.

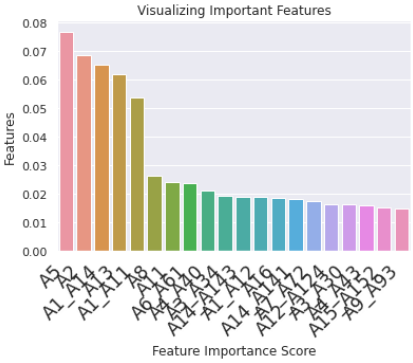
Moreover, a table summarizing these results is presented below:

Model	AUC	#1 Feature	#2	#3	#4	#5
LASSO Classifier (L1)	**0.87**	Loan duration	Good credit history	Negative balance	Doesn't pay housing	Foreign worker
Random Forest	0.79	Credit amount	Loan duration	No bank acct.	Age	Negative balance
Logistic	0.79	N/A	N/A	N/A	N/A	N/A
Boosted Trees	0.78	Credit amount	Loan duration	No bank acct.	Age	Negative balance
Ridge (L2)	0.75	Good credit history	Negative balance	No bank acct.	Loan duration	Bad credit history
Neural Net 1 (Deep)	0.72	N/A	N/A	N/A	N/A	N/A
Neural Net 2 (Shallow)	0.69	N/A	N/A	N/A	N/A	N/A
Simple Decision Tree	0.61	No bank acct.	Loan duration	Age	No other installments	For a new car

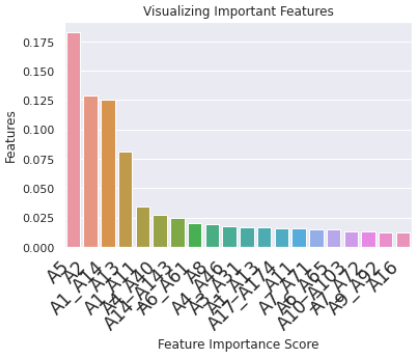
Besides just analyzing the performance metrics of the methods we investigated, we also looked into how each model determined important features in the data. Some features appeared multiple times in our analysis: Age, Loan duration, Negative balance, No bank acct. These all are pretty obvious proxy's for financial health and ability to pay back a loan, so it makes sense that a lot of the models would pick up on that fact. In particular, the simple decision tree valued No bank acct. the most, as did LASSO if its lambda parameter was set higher, so it can be speculated that the possession of a bank account is the single most relevant predictor. The full results of our feature importance analysis can be seen in Figure 19.



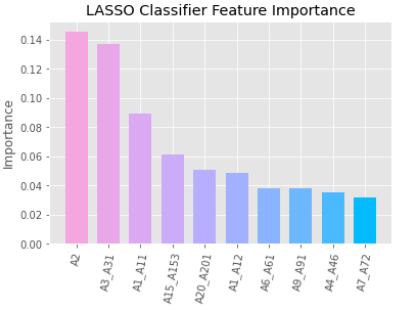
(a) Decision Tree



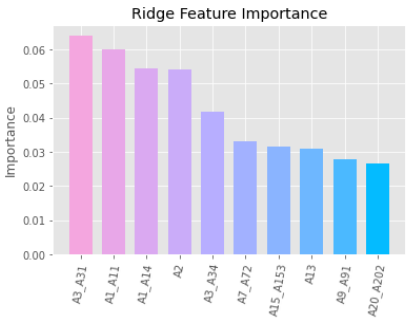
(b) Random Forest



(c) Boosted Tree



(d) LASSO



(e) Ridge

Fig. 19: Comparison of Feature Importance across the different classification methods

In conclusion, we explained the methodology and statistical context behind several of the most popular classification methods and we then investigated the performance of these methods on the German credit data set. This investigation revealed the relative strengths and weaknesses of each method and helped provide support for the popularity of methods like LASSO and Logistic regression as well as ensemble methods involving decision trees. In a future project we could investigate further optimization and fine-tuning of these methods, as well as their application to a larger dataset. Overall, our investigation can serve as a good starting point for someone interested in the methods used in credit risk decisioning as well as binary classification problems in general.

4. Code

We performed our investigation on Google Collab using Python. The associated code is included at the end of this paper.

References

1. T. Hastie, R. Tibshirani, and J. Friedman, “The Elements of Statistical Learning” , (Springer, 2017).
2. A. Geron, ”Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, (O’Reilly, 2019).

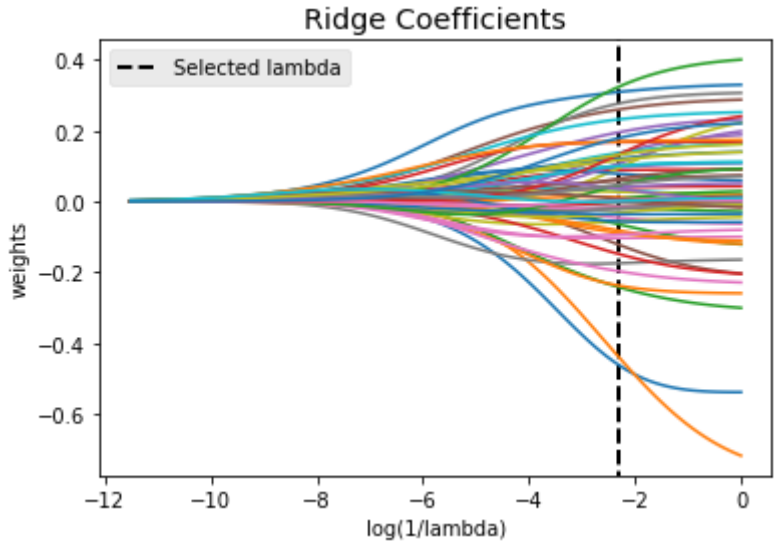
```
import numpy as np
import pandas as pd
from google.colab.data_table import DataTable
from sklearn.preprocessing import normalize, StandardScaler, MinMaxScaler
from sklearn.linear_model import RidgeClassifier, RidgeClassifierCV, Lasso, LassoCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve, auc
import matplotlib.pyplot as plt
import matplotlib.colors
import tensorflow as tf
import statsmodels.api as sm

DataTable.max_columns = 100

#import data, remove unnecessary 'index' column
df = pd.read_csv("credit_data.txt")
df = df.drop('index', 1)
num_columns = ['A2','A5', 'A8', 'A11', 'A13', 'A16', 'A18']; #numerical features
cat_columns = ['A1', 'A3', 'A4', 'A6', 'A7', 'A9', 'A10', 'A12', 'A14', 'A15', 'A17',

#find the frequency of each option for each categorical feature
df_cat = df.apply(lambda x: x.value_counts() if x.name in cat_columns else x).T.stack(
df_cat[cat_columns]

#find the distribution of each numerical feature
df[num_columns].describe()

#Change label to (1=good, 0=bad)
df['Label'] = -1*(df['Label']-2)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureW
  import pandas.util.testing as tm
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: FutureWarning:
```

```
#One-Hot Encoding for categorical variables
data = pd.get_dummies(df, columns = cat_columns)
data.insert(len(data.columns)-1, 'Label', data.pop('Label')) #move Label to the end

#normalized data frame
data_n = data.copy()
data_n[num_columns] = MinMaxScaler().fit_transform(data_n[num_columns])

col_names = np.array(data_n.drop(labels='Label', axis=1).columns)
n_classes = data_n.shape[1]
```

```
### Ridge Regression
## Visual of Coefficients
#train/test split
X = data_n.drop(['Label'], axis=1)
```

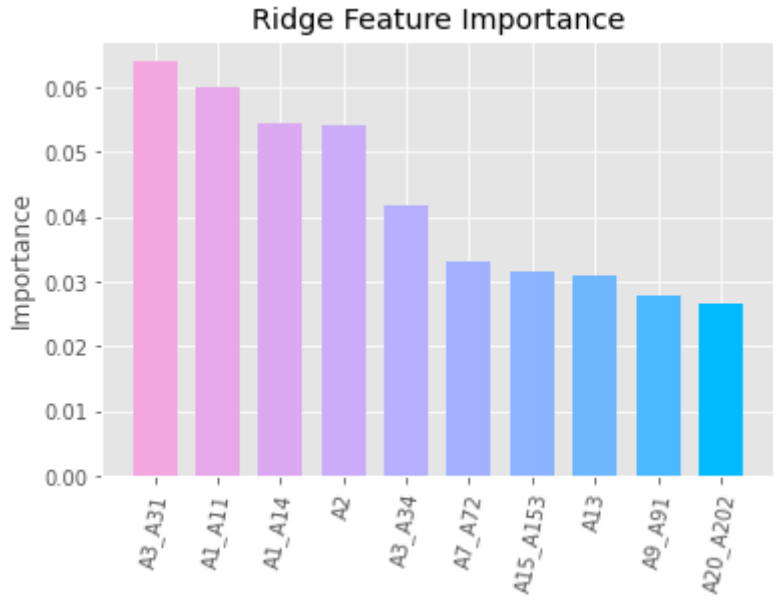
```
y = data_n['Label']
myseed = 314
Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,random_state=myseed, test_size=.2)

alphas = np.logspace(start=0,stop=5,num=101)
coefs = []

for a in alphas:
    rcv = RidgeClassifierCV(alphas=[a], cv=10, fit_intercept=False)
    rcv.fit(Xtrain, ytrain)
    coefs.append(np.ravel(rcv.coef_))

ax = plt.gca()
ax.plot(np.log(1/alphas), coefs)
plt.style.use("ggplot")
plt.xlabel("log(1/lambda)")
plt.ylabel("weights")
plt.title("Ridge Coefficients")
plt.axis("tight")
ax.vlines(x=[np.log(1/10)], ymin = -0.8, ymax = 0.5, colors='black', ls='--', lw=2, la
legend = plt.legend(loc="upper left")
plt.setp(legend.get_texts(), color='k')
plt.show()
```




```
## CV for actual model
rcv = RidgeClassifierCV(cv=10)
rcv.fit(Xtrain,ytrain)
alpha_rcv = rcv.alpha_
print("Ridge Cross-Validated alpha value:", alpha_rcv)

#use the best hyperparameter on the test set
rc = RidgeClassifier(alpha=[alpha_rcv])
rc.fit(Xtest,ytest)
yhatr_ridge = rc.predict(Xtest)
```

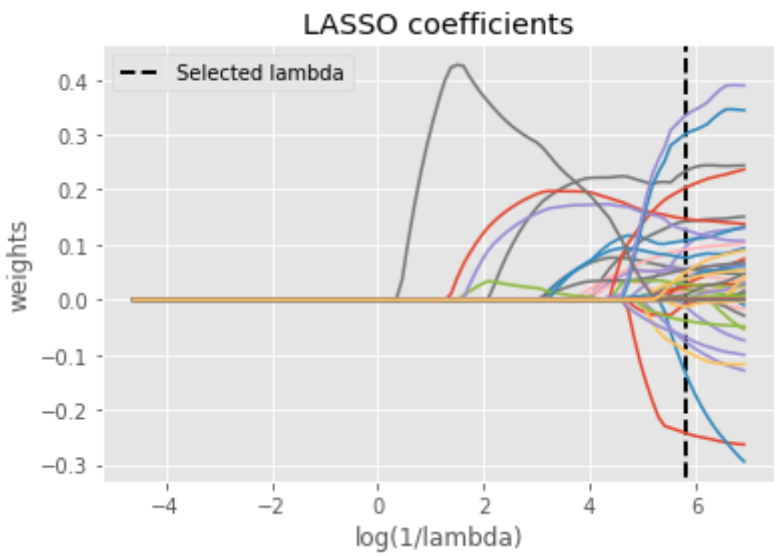
```
print("ROC: ", roc_auc_score(ytest,yhatr_ridge))

#obtain significance coefficients
rcoef_n = sum(abs(rc.coef_)) #turn to 1D array
rcoef_n = abs(rcoef_n)/sum(abs(rcoef_n)) #scaled as a percentage
rcoef_n = np.vstack((rcoef_n, col_names)) #combined with corresponding feature
rcoef_n = rcoef_n[:, rcoef_n[0, :].argsort()[::-1]] #sorted by importance

colors = ['#f4a6df', '#e8a7e8', '#daa9f1', '#caacf9', '#b7aeff', '#a2b1ff', '#8ab4ff',
          '#6eb7ff', '#4bb9ff', '#04bbff']

plt.figure()
plt.bar(rcoef_n[1, 0:10], rcoef_n[0, 0:10], color = colors, width = 0.7)
plt.xticks(rotation=80)
plt.ylabel('Importance');
plt.title('Ridge Feature Importance');
```

```
Ridge Cross-Validated alpha value: 10.0
ROC:  0.7512141816415735
```




```
### LASSO
## Visual of Coefficients
#train/test split
alphas = np.logspace(start=-3,stop=2,num=101)
coefs = []

for a in alphas:
    lcv = LassoCV(alphas=[a], cv=10, fit_intercept=False)
    lcv.fit(Xtrain, ytrain)
    coefs.append(np.ravel(lcv.coef_))

ax = plt.gca()
ax.plot(np.log(1/alphas), coefs)
plt.xlabel("log(1/lambda)")
plt.ylabel("weights")
plt.title("LASSO coefficients")
```

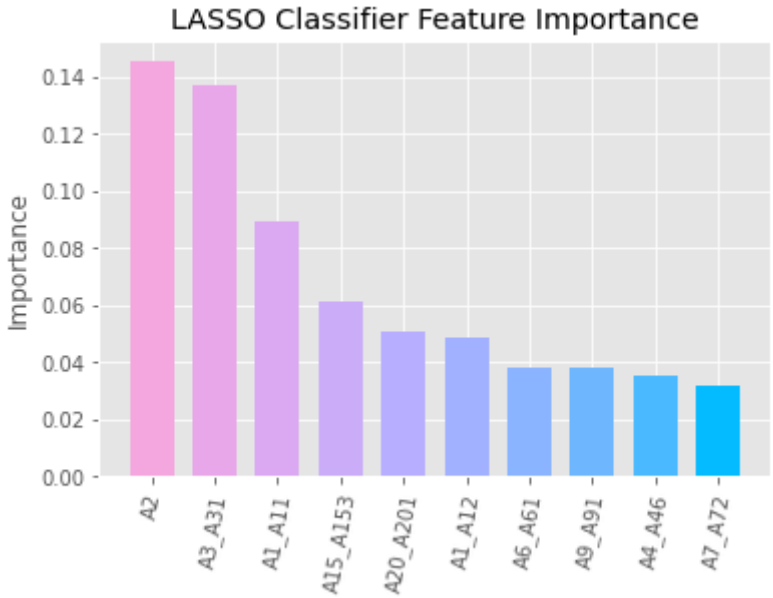
```
plt.title("LASSO coefficients")
plt.axis("tight")
ax.vlines(x=[np.log(1/0.0029488333489478087)], ymin = -0.4, ymax = 0.5, colors='black'
legend = plt.legend(loc="upper left")
plt.setp(legend.get_texts(), color='k')
plt.show()
```




```
## CV for actual LASSO model
lcv = LassoCV(cv=10)
lcv.fit(Xtrain,ytrain)
alpha_lcv = lcv.alpha_
mse_lcv = lcv.mse_path_
print("LASSO Cross-Validated alpha value: ", alpha_lcv)

#use the best hyperparameter on the test set
lc = Lasso(alpha=[alpha_lcv])
lc.fit(Xtest,ytest)
yhatr_lasso = lc.predict(Xtest)

#this gives us rational predictions,
#we need to find the best cutoff for predicting 0 or 1
yhatr_1 = [0] * len(yhatr_lasso)
yhatr_1_best = [0] * len(yhatr_lasso)
best_roc = 0
best_cutoff = 0

for a in np.arange(0.2, 1.0, 0.01):
  for i in range(len(yhatr_lasso)):
    if (yhatr_lasso[i] > a):
      yhatr_1[i] = 1
    else:
      yhatr_1[i] = 0
  current_roc = roc_auc_score(ytest,yhatr_1)
  if current_roc > best_roc:
    best_roc = current_roc
```

```
    best_cutoff = a
    yhatr_1_best = yhatr_1.copy()

print("Best cutoff for prediction: ", best_cutoff)
print("ROC: ", best_roc)

#obtain significance coefficients
lcoef_n = abs(lc.coef_)/sum(abs(lc.coef_)) #scaled as a percentage
lcoef_n = np.vstack((lcoef_n, col_names)) #combined with corresponding feature
lcoef_n = lcoef_n[:, lcoef_n[0, :].argsort()[::-1]] #sorted by importance

colors = ['#f4a6df', '#e8a7e8', '#daa9f1', '#caacf9', '#b7aeff', '#a2b1ff', '#8ab4ff',
          '#6eb7ff', '#4bb9ff', '#04bbff']

plt.bar(lcoef_n[1, 0:10], lcoef_n[0, 0:10], color = colors, width = 0.7)
plt.xticks(rotation=80)
plt.ylabel('Importance');
plt.title('LASSO Classifier Feature Importance');
```

```
LASSO Cross-Validated alpha value:  0.0029488333489478087
Best cutoff for prediction:  0.6500000000000004
ROC:  0.8688683827100534
```

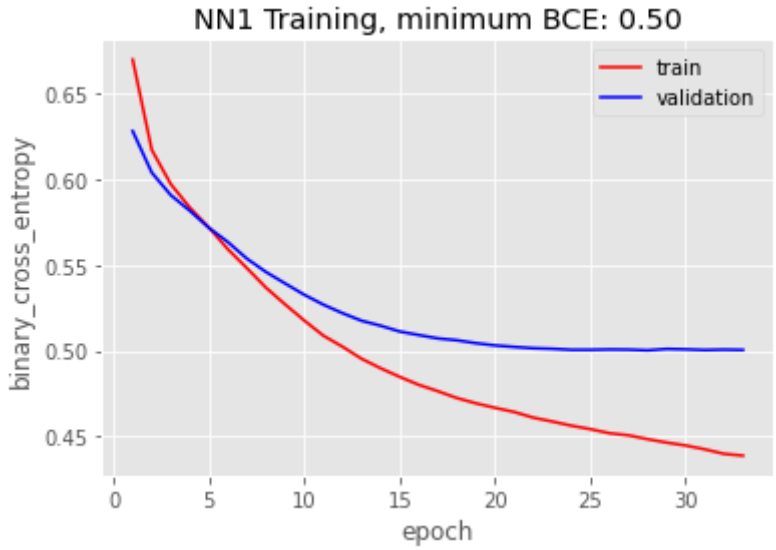


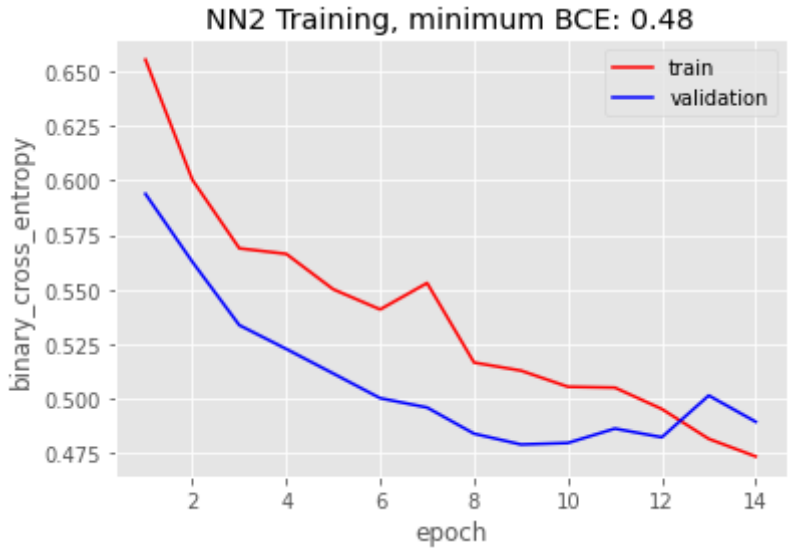

```
### Neural Nets with Keras
## Neural Net 1: Single Layer, AdaptSive Early Stopping
nunits = 20
ytrain_OHE = pd.get_dummies(ytrain)
ytest_OHE = pd.get_dummies(ytest)
nx = Xtrain.shape[1]
nn1 = tf.keras.models.Sequential()
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

# add one hidden layer, one binary probability output
nn1.add(tf.keras.layers.Dense(units=nunits,activation='relu',input_shape=(nx,)))
nn1.add(tf.keras.layers.Dense(units=2,activation='softmax'))
```

```
#compile model
nn1.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])

nepoch = 100
nhist = nn1.fit(Xtrain,ytrain_OHE,verbose=0,epochs=nepoch,batch_size=50,validation_dat

# plot training by epoch
yhatnn = nn1.predict(Xtrain)
yprednn = nn1.predict(Xtest)

trL = np.array((nhist.history['loss']))
teL = np.array((nhist.history['val_loss']))
epind = range(1,len(trL)+1)

plt.figure()
plt.plot(epind,trL,c='red',label='train')
plt.plot(epind,teL,c='blue',label='validation')
plt.xlabel('epoch'); plt.ylabel('binary_cross_entropy')
plt.legend()
minbce = teL.min()
plt.title(f'NN1 Training, minimum BCE: {minbce:0.2f}')

nn1_pred = nn1.predict(Xtest)
yhatr_nn1 = np.argmax(nn1_pred,axis=1)
print("ROC for NN1: ", roc_auc_score(ytest,yhatr_nn1))

## Neural Net 2: Two-Layers, Dropout = 0.4, Adaptive Early Stopping
nunits = 50
nn2 = tf.keras.models.Sequential()

# add two hidden layers with dropout, one binary probability output
nn2.add(tf.keras.layers.Dense(units=nunits,activation='relu',input_shape=(nx,)))
nn2.add(tf.keras.layers.Dropout(.2))
nn2.add(tf.keras.layers.Dense(units=nunits,activation='relu',input_shape=(nx,)))
nn2.add(tf.keras.layers.Dropout(.2))
nn2.add(tf.keras.layers.Dense(units=nunits,activation='relu',input_shape=(nx,)))
nn2.add(tf.keras.layers.Dropout(.2))
nn2.add(tf.keras.layers.Dense(units=nunits,activation='relu',input_shape=(nx,)))
nn2.add(tf.keras.layers.Dropout(.2))
nn2.add(tf.keras.layers.Dense(units=nunits,activation='relu',input_shape=(nx,)))
nn2.add(tf.keras.layers.Dropout(.2))
nn2.add(tf.keras.layers.Dense(units=2,activation='softmax'))

#compile model
nn2.compile(loss='binary_crossentropy',optimizer='rmsprop',metrics=['accuracy'])

nhist2 = nn2.fit(Xtrain,ytrain_OHE,verbose=0,epochs=nepoch,batch_size=50,validation_da

### plot training by epoch
yhatnn2 = nn2.predict(Xtrain)
```

```
yprednn2 = nn2.predict(Xtest)

trL2 = np.array((nhist2.history['loss']))
teL2 = np.array((nhist2.history['val_loss']))
epind2 = range(1,len(trL2)+1)

plt.figure()
plt.plot(epind2,trL2,c='red',label='train')
plt.plot(epind2,teL2,c='blue',label='validation')
plt.xlabel('epoch'); plt.ylabel('binary_cross_entropy')
plt.legend()
minbce2 = teL2.min()
plt.title(f'NN2 Training, minimum BCE: {minbce2:0.2f}')

nn2_pred = nn2.predict(Xtest)
yhatr_nn2 = np.argmax(nn2_pred,axis=1)
print("ROC for NN2: ", roc_auc_score(ytest,yhatr_nn2))
```

```
ROC for NN1:  0.6246964545896065
ROC for NN2:  0.7187955318115591
```





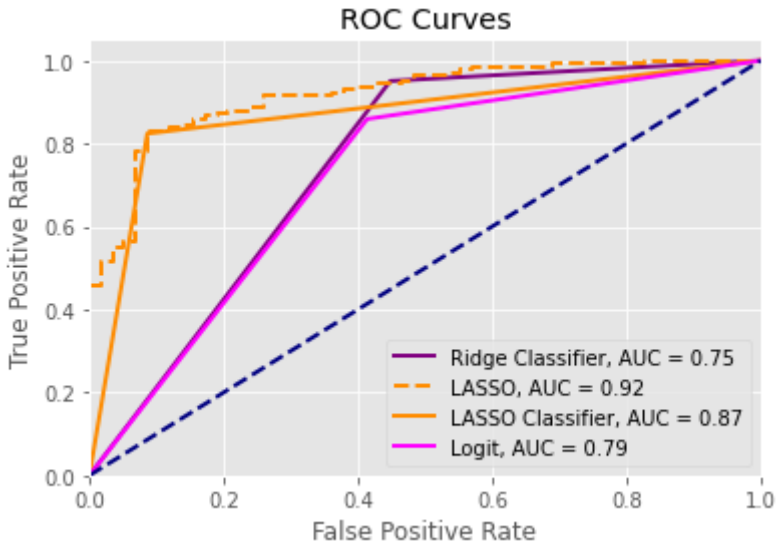

```
### Logit
```

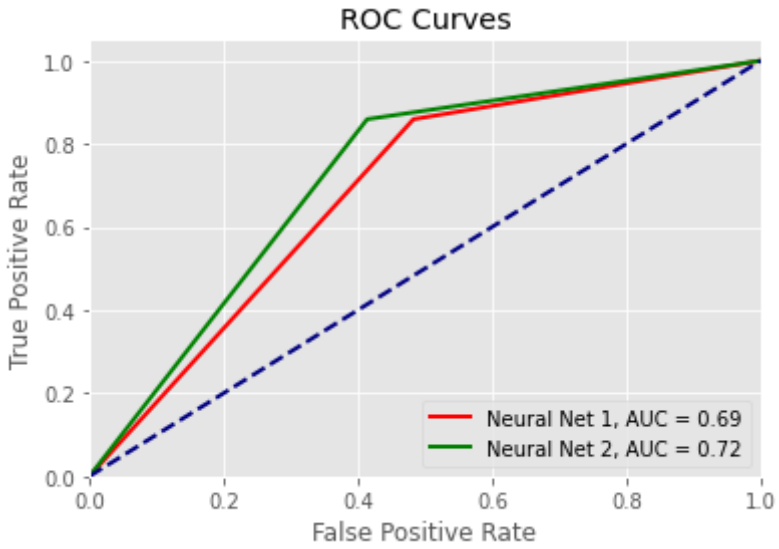
```
XX = sm.add_constant(Xtrain)
lfitM = sm.Logit(ytrain, XX).fit()
XXp = sm.add_constant(Xtest)
phlgt = lfitM.predict(XXp)
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.444838
         Iterations: 35
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:117: FutureWa
  x = pd.concat(x[::order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: Convergenc
  "Check mle_retvals", ConvergenceWarning)
```

```
#obtain roc_auc for each model
fpr_ridge, tpr_ridge, _ = roc_curve(ytest, yhatr_ridge)
roc_auc_ridge = auc(fpr_ridge, tpr_ridge)
fpr_lasso, tpr_lasso, _ = roc_curve(ytest, yhatr_lasso)
roc_auc_lasso = auc(fpr_lasso, tpr_lasso)
fpr_lasso_c, tpr_lasso_c, _ = roc_curve(ytest, yhatr_1_best)
roc_auc_lasso_c = auc(fpr_lasso_c, tpr_lasso_c)
fpr_nn1, tpr_nn1, _ = roc_curve(ytest, yhatr_nn1)
roc_auc_nn1 = auc(fpr_nn1, tpr_nn1)
fpr_nn2, tpr_nn2, _ = roc_curve(ytest, yhatr_nn2)
roc_auc_nn2 = auc(fpr_nn2, tpr_nn2)
fpr_log, tpr_log, _ = roc_curve(ytest, phlgt)
roc_auc_log = auc(fpr_log, tpr_log)

plt.figure()
lw = 2
plt.plot(fpr_ridge,tpr_ridge,color="purple",lw=lw,
         label="Ridge Classifier, AUC = %0.2f" % roc_auc_ridge,)
plt.plot(fpr_lasso,tpr_lasso,color="darkorange",lw=lw,
         label="LASSO, AUC = %0.2f" % roc_auc_lasso, linestyle="--",)
plt.plot(fpr_lasso_c,tpr_lasso_c,color="darkorange",lw=lw,
         label="LASSO Classifier, AUC = %0.2f" % roc_auc_lasso_c,)
plt.plot(fpr_nn2,tpr_nn2,color="magenta",lw=lw,
         label="Logit, AUC = %0.2f" % roc_auc_log,)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
legend = plt.legend(loc="lower right")
plt.setp(legend.get_texts(), color='k')
plt.show()

plt.figure()
lw = 2
plt.plot(fpr_nn1,tpr_nn1,color="red",lw=lw,
         label="Neural Net 1, AUC = %0.2f" % roc_auc_nn1,)
```

```
plt.plot(fpr_nn2,tpr_nn2,color="green",lw=lw,
         label="Neural Net 2, AUC = %0.2f" % roc_auc_nn2,)
plt.plot([0, 1], [0, 1], color="navy", lw=lw, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
legend = plt.legend(loc="lower right")
plt.setp(legend.get_texts(), color='k')
plt.show()
```





```
import numpy as np
import pandas as pd
from google.colab.data_table import DataTable
from sklearn.preprocessing import normalize, StandardScaler, MinMaxScaler

from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV

from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV



DataTable.max_columns = 62

#import data, remove unnecessary 'index' column
df = pd.read_csv("credit_data.txt")
df = df.drop('index', 1)
num_columns = ['A2','A5', 'A8', 'A11', 'A13', 'A16', 'A18']; #numerical features
cat_columns = ['A1', 'A3', 'A4', 'A6', 'A7', 'A9', 'A10', 'A12', 'A14', 'A15', 'A17',

#find the frequency of each option for each categorical feature
df_cat = df.apply(lambda x: x.value_counts() if x.name in cat_columns else x).T.stack(
df_cat[cat_columns]

#find the distribution of each numerical feature
df[num_columns].describe()

#change label to 0=good 1=bad
df['Label'] = -1*(df['Label']-2)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: FutureWarning:
```

```
df
```

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	...	A12	A13	A14	A15	A1
0	A11	6	A34	A43	1169	A65	A75	4	A93	A101	...	A121	67	A143	A152	
1	A12	48	A32	A43	5951	A61	A73	2	A92	A101	...	A121	22	A143	A152	
2	A14	12	A34	A46	2096	A61	A74	2	A93	A101	...	A121	49	A143	A152	
3	A11	42	A32	A42	7882	A61	A74	2	A93	A103	...	A122	45	A143	A153	
4	A11	24	A33	A40	4870	A61	A73	3	A93	A101	...	A124	53	A143	A153	

```
#One-Hot Encoding for categorical variables
data = pd.get_dummies(df, columns = cat_columns)
data.insert(len(data.columns)-1, 'Label', data.pop('Label')) #move Label to the end

#normalized data frame
data_n = data.copy()
data_n[num_columns] = MinMaxScaler().fit_transform(data_n[num_columns])
print(data_n)
```

```
           A2        A5        A8       A11       A13       A16  A18  A1_A11  \
0    0.029412  0.050567  1.000000  1.000000  0.857143  0.333333  0.0       1
1    0.647059  0.313690  0.333333  0.333333  0.053571  0.000000  0.0       0
2    0.117647  0.101574  0.333333  0.666667  0.535714  0.000000  1.0       0
3    0.558824  0.419941  0.333333  1.000000  0.464286  0.000000  1.0       1
4    0.294118  0.254209  0.666667  1.000000  0.607143  0.333333  1.0       1
..        ...       ...       ...       ...       ...       ...  ...     ...
995  0.117647  0.081765  0.666667  1.000000  0.214286  0.000000  0.0       0
996  0.382353  0.198470  1.000000  1.000000  0.375000  0.000000  0.0       1
997  0.117647  0.030483  1.000000  1.000000  0.339286  0.000000  0.0       0
998  0.602941  0.087763  1.000000  1.000000  0.071429  0.000000  0.0       1
999  0.602941  0.238032  0.666667  1.000000  0.142857  0.000000  0.0       0

     A1_A12  A1_A13  ...  A15_A153  A17_A171  A17_A172  A17_A173  A17_A174  \
0         0       0  ...         0         0         0         1         0
1         1       0  ...         0         0         0         1         0
2         0       0  ...         0         0         1         0         0
3         0       0  ...         1         0         0         1         0
4         0       0  ...         1         0         0         1         0
..      ...     ...  ...       ...       ...       ...       ...       ...
995       0       0  ...         0         0         1         0         0
996       0       0  ...         0         0         0         0         1
997       0       0  ...         0         0         0         1         0
998       0       0  ...         1         0         0         1         0
999       1       0  ...         0         0         0         1         0

     A19_A191  A19_A192  A20_A201  A20_A202  Label
0           0         1         1         0      1
1           1         0         1         0      0
2           1         0         1         0      1
3           1         0         1         0      1
4           1         0         1         0      0
..        ...       ...       ...       ...    ...
```

```
995          1          0          1          0       1
996          0          1          1          0       1
997          1          0          1          0       1
998          0          1          1          0       0
999          1          0          1          0       1

[1000 rows x 62 columns]
```

```
data
```

	A2	A5	A8	A11	A13	A16	A18	A1_A11	A1_A12	A1_A13	...	A15_A153	A17_A
0	6	1169	4	4	67	2	1	1	0	0	...	0	
1	48	5951	2	2	22	1	1	0	1	0	...	0	
2	12	2096	2	3	49	1	2	0	0	0	...	0	
3	42	7882	2	4	45	1	2	1	0	0	...	1	
4	24	4870	3	4	53	2	2	1	0	0	...	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
995	12	1736	3	4	31	1	1	0	0	0	...	0	
996	30	3857	4	4	40	1	1	1	0	0	...	0	
997	12	804	4	4	38	1	1	0	0	0	...	0	
998	45	1845	4	4	23	1	1	1	0	0	...	1	
999	45	4576	3	4	27	1	1	0	1	0	...	0	

1000 rows × 62 columns

```
#####################################################
### basic imports
import matplotlib.pyplot as plt
import numpy as np
import scipy
import pandas as pd
import math
import seaborn as sns; sns.set()

### sklearn model
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
```

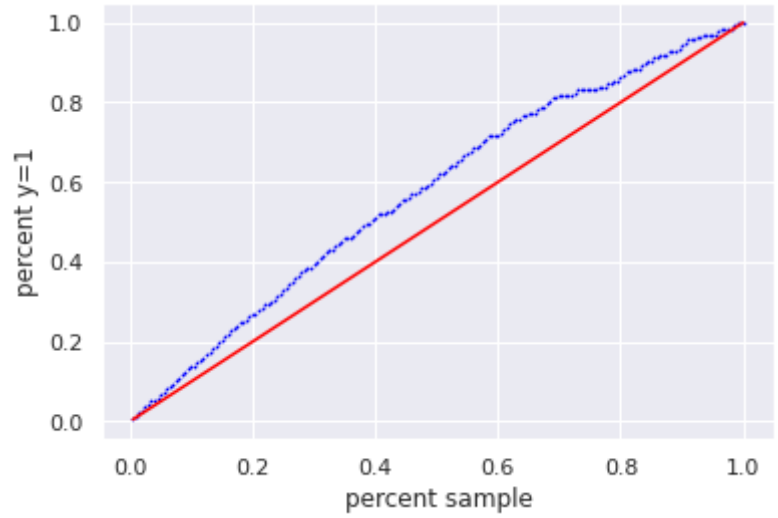
```
##roc/auc
from sklearn.metrics import roc_auc_score
## note online sklearn seemed to have plot_roc_curve, but I just found this one
##   I probably need to update my system!!
from sklearn.metrics import roc_curve

## fit simple logit
import statsmodels.api as sm

## lift curve
def mylift(y,p):
   """lift"""
   ii = np.argsort(p)[::-1]
   ps = np.cumsum(y[ii])/np.sum(y)
   return(ps)
```

```
#data split
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(data, test_size=0.2, random_state=49)

#train
ytrain = train_data.iloc[:,len(train_data.columns)-1].to_numpy(); xtrain = train_data.
ntrain = len(ytrain)
## test
ytest = test_data.iloc[:,len(test_data.columns)-1].to_numpy(); xtest = test_data.iloc[
ntest = len(ytest)
```

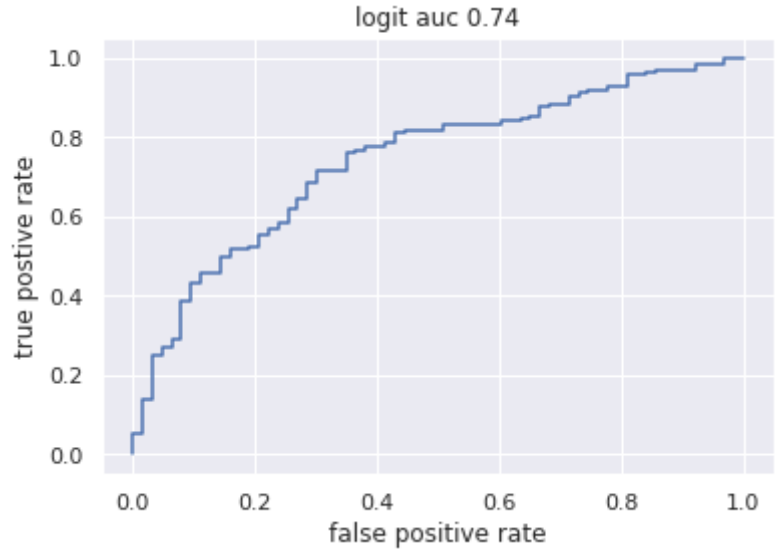
```
################################################
### logit
XX = sm.add_constant(xtrain)
lfitM = sm.Logit(ytrain, XX).fit()
XXp = sm.add_constant(xtest)
phlgt = lfitM.predict(XXp)

## out of sample lift
pvec = np.linspace(1,ntest,ntest)/ntest
plt.scatter(pvec,mylift(ytest,phlgt),s=.5,c='blue')
plt.xlabel('percent sample'); plt.ylabel('percent y=1')
plt.plot(pvec,pvec,c='red')
plt.show()

##auc
auclgt = roc_auc_score(ytest,phlgt)
print('auc for logit: ',auclgt)

##roc
roclgt = roc_curve(ytest,phlgt)
plt.plot(roclgt[0],roclgt[1])
plt.xlabel('false positive rate'); plt.ylabel('true postive rate')
```

```
plt.title('logit auc ' + str(np.round(auclgt,2)))
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:117: FutureWa
  x = pd.concat(x[::order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: Convergenc
  "Check mle_retvals", ConvergenceWarning)
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.426007
         Iterations: 35
```




```
auc for logit:  0.7425559031398447
```

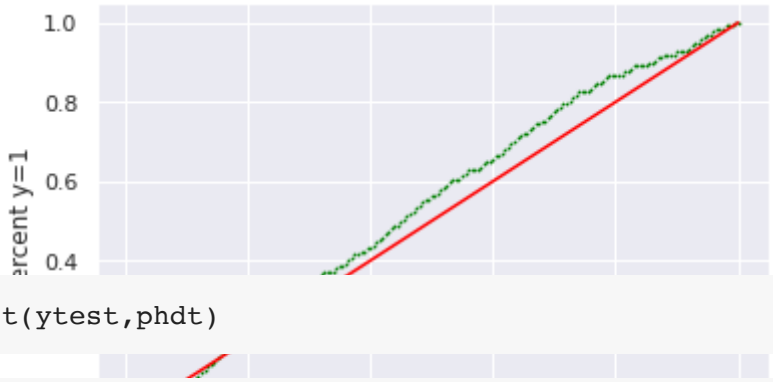



```
phlgt
```

```
318    0.874982
794    0.776014
718    0.969202
498    0.833577
181    0.456498
         ...
470    0.563561
334    0.029092
940    0.953643
```

```
76     0.285868
124    0.591217
Length: 200, dtype: float64
```

```
################################################
### decision tree
dtm = DecisionTreeClassifier(max_leaf_nodes=50)
dtm.fit(xtrain,ytrain)
phdt = dtm.predict_proba(xtest)[:,1]

## out of sample lift
plt.scatter(pvec,mylift(ytest,phdt),s=.5,c='green')
# plt.scatter(pvec,mylift(ytest,phlgt),s=.5,c='blue')
plt.xlabel('percent sample'); plt.ylabel('percent y=1')
plt.plot(pvec,pvec,c='red')
plt.show()

##auc
aucdt = roc_auc_score(ytest,phdt)
print('auc for tree: ',aucdt)

##roc
rocdt = roc_curve(ytest,phdt)
plt.plot(rocdt[0],rocdt[1])
plt.xlabel('false positive rate'); plt.ylabel('true postive rate')
plt.title('tree auc ' + str(np.round(aucdt,2)))
plt.show()
```




```
mylift(ytest,phdt)
```

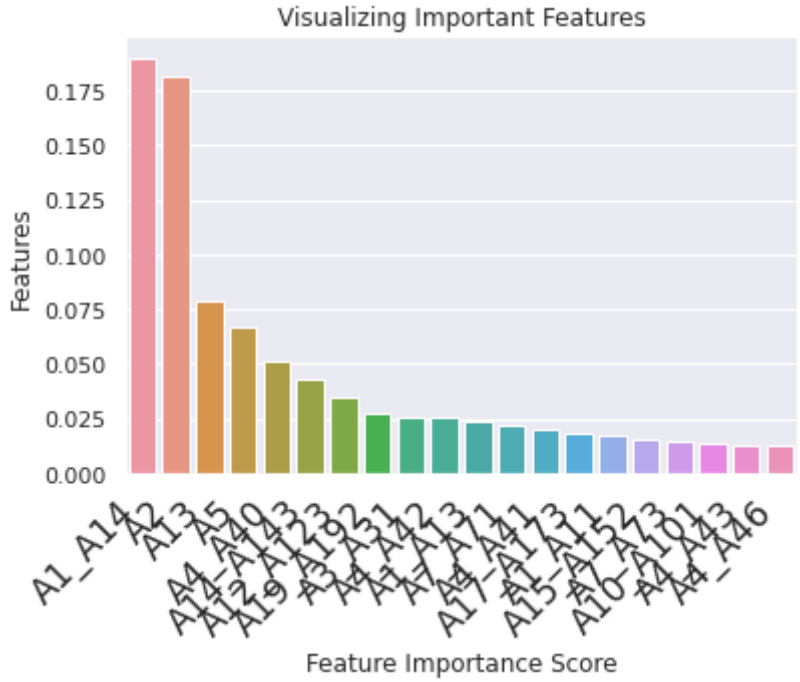
```
# check Important features
feature_importances_df3 = pd.DataFrame(
    {"feature": list(xtrain.columns), "importance": dtm.feature_importances_}
).sort_values("importance", ascending=False)

# Display
feature_importances_df3
```

	feature	importance
10	A1_A14	0.189929
0	A2	0.181062
4	A13	0.078393
1	A5	0.066541
16	A4_A40	0.051399
...	...	...
32	A7_A72	0.000000
34	A7_A74	0.000000
35	A7_A75	0.000000
36	A9_A91	0.000000
60	A20_A202	0.000000

61 rows × 2 columns

```
# visualize important featuers

# Creating a bar plot
sns.barplot(x=feature_importances_df3.feature[:20], y=feature_importances_df3.importar
# Add labels to your

plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Visualizing Important Features")
```

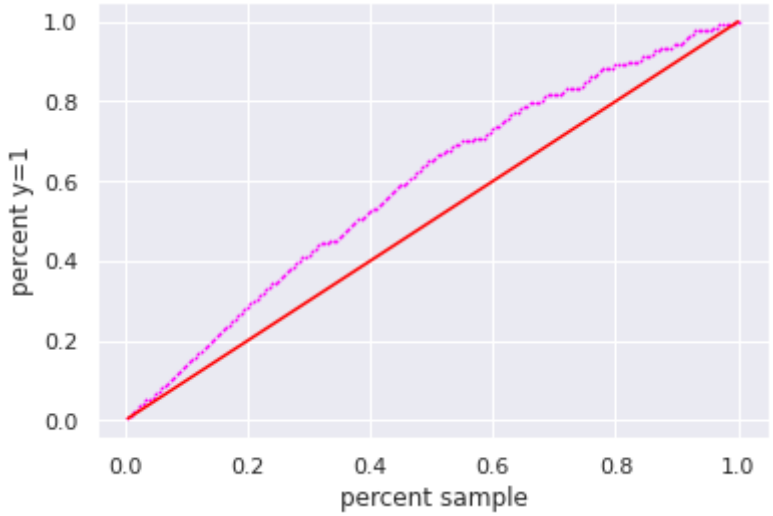
```
plt.xticks(
    rotation=45, horizontalalignment="right", fontweight="light", fontsize="x-large"
)
plt.show()
```



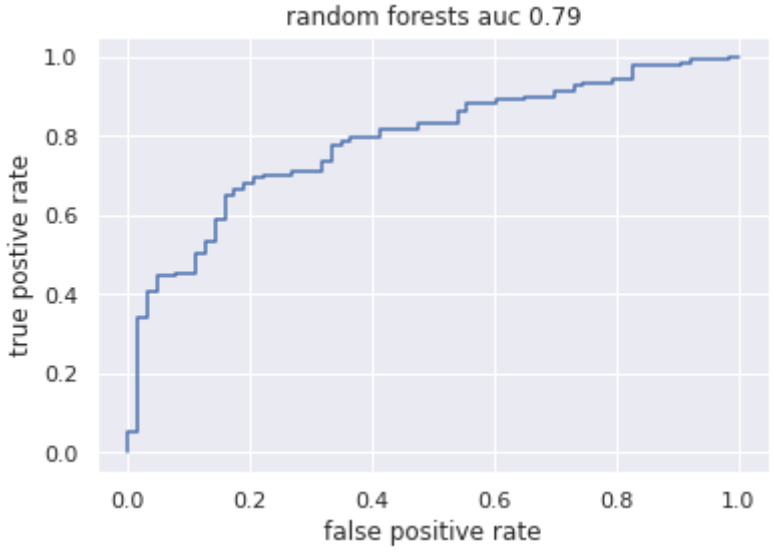

```
##################################################
### random forests
rfm = RandomForestClassifier(random_state=0,n_jobs=-1,n_estimators=500,max_features=2,
rfm.fit(xtrain,ytrain)
phrf = rfm.predict_proba(xtest)[:,1]

# the OOB score is computed as the number of correctly predicted rows from the out of
# not to useful in this application
print("oob score for Random Forests: ",rfm.oob_score_)

## out of sample lift
# plt.scatter(pvec,mylift(ytest,phdt),s=.5,c='green')
# plt.scatter(pvec,mylift(ytest,phlgt),s=.5,c='blue')
plt.scatter(pvec,mylift(ytest,phrf),s=.5,c='magenta')
plt.xlabel('percent sample'); plt.ylabel('percent y=1')
plt.plot(pvec,pvec,c='red')
plt.show()

##auc
aucrf = roc_auc_score(ytest,phrf)
print('auc for random forests: ',aucrf)

##roc
rocrf = roc_curve(ytest,phrf)
plt.plot(rocrf[0],rocrf[1])
plt.xlabel('false positive rate'); plt.ylabel('true postive rate')
```

```
plt.title('random forests auc ' + str(np.round(aucrf,2)))
plt.show()
```

```
oob score for Random Forests:  0.735
```



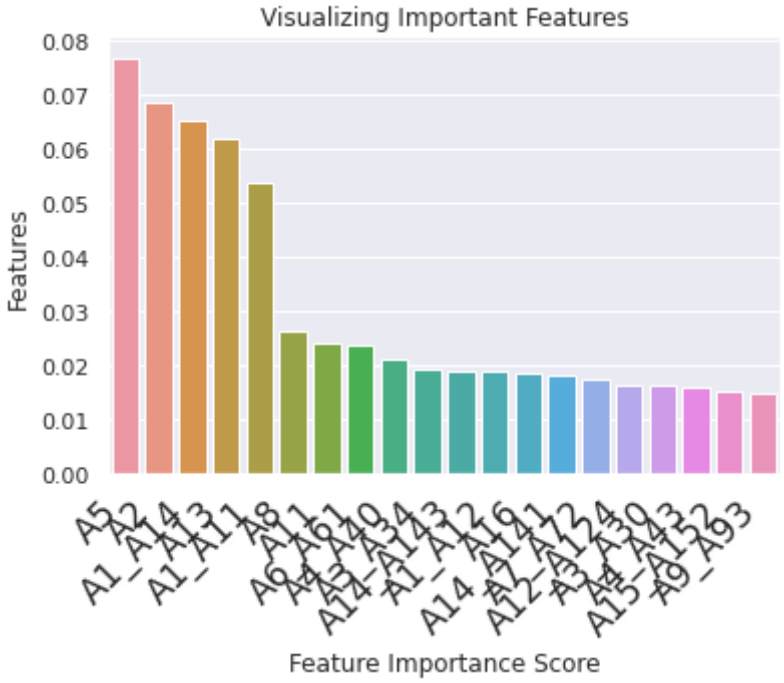

```
auc for random forests:  0.7870466921561813
```




```
# check Important features
feature_importances_df = pd.DataFrame(
    {"feature": list(xtrain.columns), "importance": rfm.feature_importances_}
).sort_values("importance", ascending=False)

# Display
feature_importances_df
```

	feature	importance
1	A5	0.076707
0	A2	0.068503
10	A1_A14	0.065258
4	A13	0.061940
7	A1_A11	0.053727
...	...	...
53	A17_A171	0.004750
22	A4_A45	0.003510

```
# visualize important featuers

# Creating a bar plot
sns.barplot(x=feature_importances_df.feature[:20], y=feature_importances_df.importance
# Add labels to your

plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Visualizing Important Features")
plt.xticks(
    rotation=45, horizontalalignment="right", fontweight="light", fontsize="x-large"
)
plt.show()
```



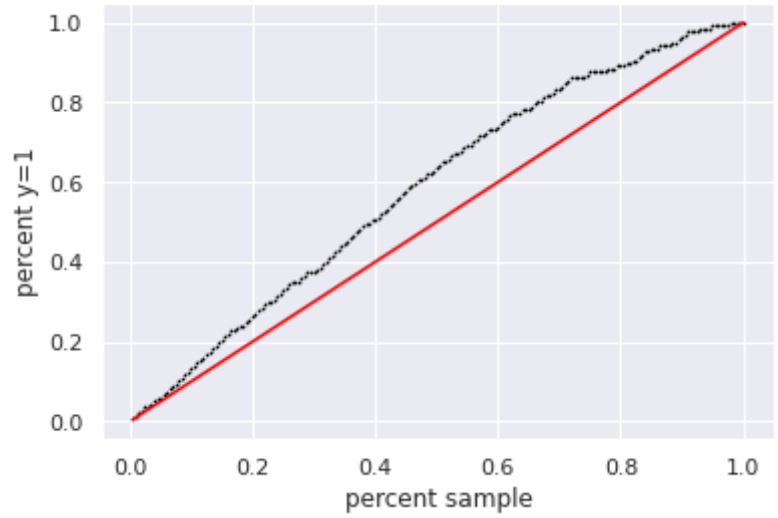

```
from sklearn.model_selection import cross_val_score
```

```
np.mean(cross_val_score(rfm, xtrain, ytrain, cv=10))
```

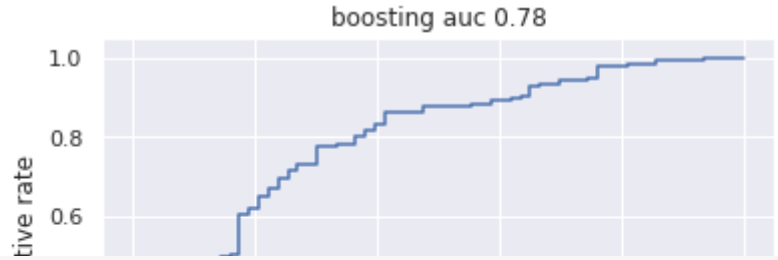
```
0.7337500000000001
```

```
# actual cross validation
```

```
#################################################
### gradient boosting

## let's try two different settings for boosting
# first setting
gbm = GradientBoostingClassifier(learning_rate=.01,n_estimators=1000,max_depth=4)
gbm.fit(xtrain,ytrain)
phgb = gbm.predict_proba(xtest)[:,1]

# second setting
gbm1 = GradientBoostingClassifier(learning_rate=.1,n_estimators=1000,max_depth=4)
gbm1.fit(xtrain,ytrain)
phgb1 = gbm1.predict_proba(xtest)[:,1]

## out of sample lift
# plt.scatter(pvec,mylift(ytest,phdt),s=.5,c='green')
# plt.scatter(pvec,mylift(ytest,phlgt),s=.5,c='blue')
# plt.scatter(pvec,mylift(ytest,phrf),s=.5,c='magenta')
plt.scatter(pvec,mylift(ytest,phgb),s=.5,c='black')
#plt.scatter(pvec,mylift(ytest,phgb1),s=.5,c='grey')
plt.xlabel('percent sample'); plt.ylabel('percent y=1')
plt.plot(pvec,pvec,c='red')
plt.show()

##auc
aucgb = roc_auc_score(ytest,phgb)
print('auc for tree: ',aucgb)

##roc
rocgb = roc_curve(ytest,phgb)
plt.plot(rocgb[0],rocgb[1])
plt.xlabel('false positive rate'); plt.ylabel('true postive rate')
plt.title('boosting auc ' + str(np.round(aucgb,2)))
plt.show()
```




```
auc for tree:  0.7759239949020972
```




```
# check Important features
feature_importances_df2 = pd.DataFrame(
    {"feature": list(xtrain.columns), "importance": gbm.feature_importances_}
).sort_values("importance", ascending=False)

# Display
feature_importances_df2
```

```
      feature  importance
```

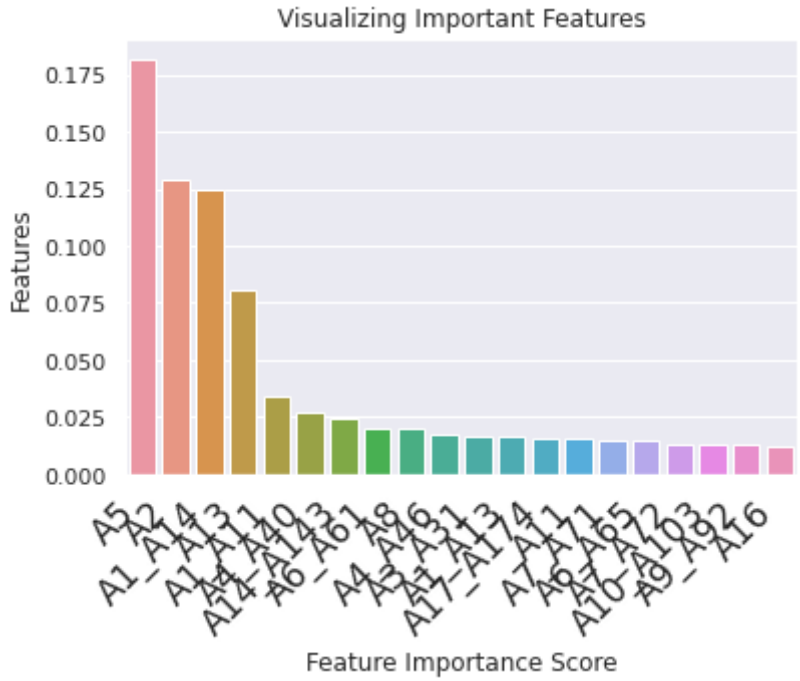
```
phgb
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-46a68b5e1330> in <module>()
----> 1 phgb

NameError: name 'phgb' is not defined
```

SEARCH STACK OVERFLOW

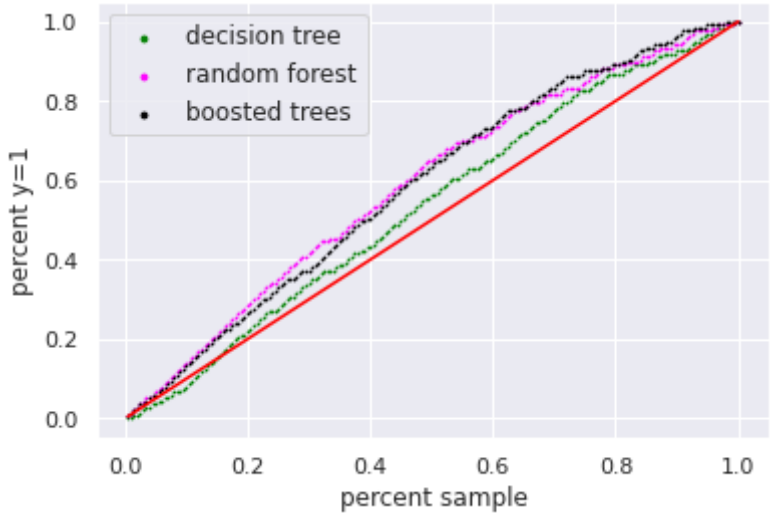
```
39     A9_A94       0.001441
```

```
# visualize important featuers

# Creating a bar plot
sns.barplot(x=feature_importances_df2.feature[:20], y=feature_importances_df2.importar
# Add labels to your

plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Visualizing Important Features")
plt.xticks(
    rotation=45, horizontalalignment="right", fontweight="light", fontsize="x-large"
)
plt.show()
```

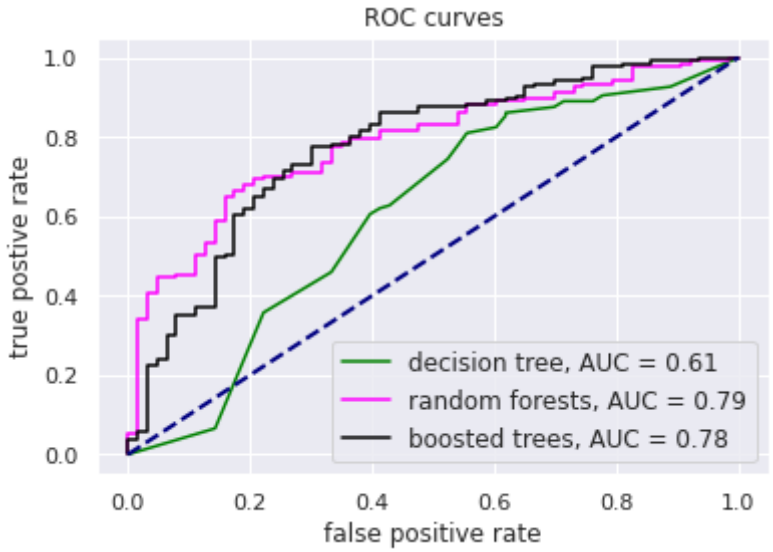



```
## comparison1
plt.scatter(pvec,mylift(ytest,phdt),s=.5,c='green', label= "decision tree")
plt.scatter(pvec,mylift(ytest,phrf),s=.5,c='magenta', label= "random forest")
```

```
plt.scatter(pvec,mylift(ytest,phgb),s=.5,c='black', label= "boosted trees")
plt.xlabel('percent sample'); plt.ylabel('percent y=1')
plt.plot(pvec,pvec,c='red')
plt.legend(fontsize='medium', markerscale=4)
plt.show()
```




```
#comparison2
plt.plot(rocdt[0],rocdt[1], label= 'decision tree, AUC = '+str(np.round(aucdt,2)), c='
plt.plot(rocrf[0],rocrf[1], label= 'random forests, AUC = '+str(np.round(aucrf,2)), c=
plt.plot(rocgb[0],rocgb[1], label= 'boosted trees, AUC = '+str(np.round(aucgb,2)), c='
plt.plot([0,1],[0,1], color='navy', lw=2, linestyle='--')
plt.xlabel('false positive rate'); plt.ylabel('true postive rate')
plt.title('ROC curves')
plt.legend(fontsize='medium', markerscale=4)
plt.show()
```




```
##################################################
### auc
```

```
aucv = np.array([auclgt,aucdt,aucrf,aucgb])
print('auc: (logit, tree, rf, gb)',np.round(aucv,2))
```

```
auc: (logit, tree, rf, gb) [0.74 0.61 0.79 0.78]
```