Machine Learning Nanodegree Capstone Project

Project Report

Sherif Ramadan Mohammed

Dec-2018

Definition

Project Overview

Defaulted loans is a problem that affects banks and could put them on risk if the number of defaulted loans is large.

In this project, I developed classification models that could support banks to decide if it is better to agree to give a loan for a new customer or not based on the new customer information like age, education and gender.

Why Machine learning?

I used machine learning to solve this project problem as machine learning increases understanding by showing which factors most affect specific outcomes.

References to academic papers and resources about similar cases:

http://www.iosrjournals.org/iosr-jce/papers/conf.15013/Volume%203/4.%2018-21.pdf

https://medium.com/@fenjiro/data-mining-for-banking-loan-approval-use-case-e7c2bc3ece3

Project Origin

This project is based on my completed capstone project of “Machine Learning with Python” course by IBM on Coursera.

Project data is an csv file containing loan information and this file was part of the mentioned capstone project.

Data file link:

https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/loan_train.csv

Problem Statement

The goal is to create a model to predict the loan pay off likelihood for a specific new customer.

The tasks involved are the following:

- Download the data.
- Clean the data.
- Explore the data.
- Implement the model.
- Tune model parameters to get best possible results.

- Evaluate and compare implemented models.

The final implemented model is expected to support the bank to better decide if it should give the loan or not.

Metrics

Accuracy is a common metric for binary classifiers. In this project, we will use Jaccard Similarity score to evaluate and compare our models.

The Jaccard similarity score function computes the average (default) or sum of Jaccard similarity coefficients, also called the Jaccard index, between pairs of label sets.

The Jaccard similarity coefficient of the i-th samples, with a ground truth label set yi and predicted label set y^i, is defined as:

$$J(y_i, \hat{y}_i) = \frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|}.$$

Why Jaccard Similarity score?

- I chose Jaccard similarity score as my problem is mainly a binary classification problem.
- My sample data is not very small.
- My sample data doesn't have missing observations.

References

https://www.statisticshowto.datasciencecentral.com/jaccard-index/

https://neo4j.com/docs/graph-algorithms/current/algorithms/similarity-jaccard/

http://ase.tufts.edu/chemistry/walt/sepa/Activities/jaccardPractice.pdf

Analysis

Data Exploration

This dataset is about past loans. The **Loan_train.csv** data set includes details of 346 customers whose loan are already paid off or defaulted. It includes following fields:

Field	Description
Loan Status	Paid Off or In Collection
Principal	Basic principal loan amount
Terms	Weekly, biweekly or monthly
Effective_date	When loan took effect
Due_date	One-time payoff schedule
Age	Age of applicant
Education	Education of applicant
Gender	The gender of applicant

Statistics about our data

- Age: (max=51, min=6, mean=30)
- Principal: (max=1000, min=300, mean=943)
- Gender: (male=294, female=52)
- Terms values: 7, 15, 30 (weekly, biweekly and monthly)
- Education:
    - High school or below: 151
    - College: 149
    - Bachelor: 44
    - Master or above: 2

How many classes in our dataset?

```
In [19]: df['loan_status'].value_counts()

Out[19]: PAIDOFF       260
         COLLECTION     86
         Name: loan_status, dtype: int64
```

260 people have paid off the loan on time while 86 have gone into collection

Loan Status per gender

```
In [15]: df.groupby(['Gender'])['loan_status'].value_counts(normalize=True)

Out[15]: Gender  loan_status
         female  PAIDOFF       0.865385
                 COLLECTION    0.134615
         male    PAIDOFF       0.731293
                 COLLECTION    0.268707
         Name: loan_status, dtype: float64
```

86 % of female pay there loans while only 73 % of males pay there loan

Loan Status per education

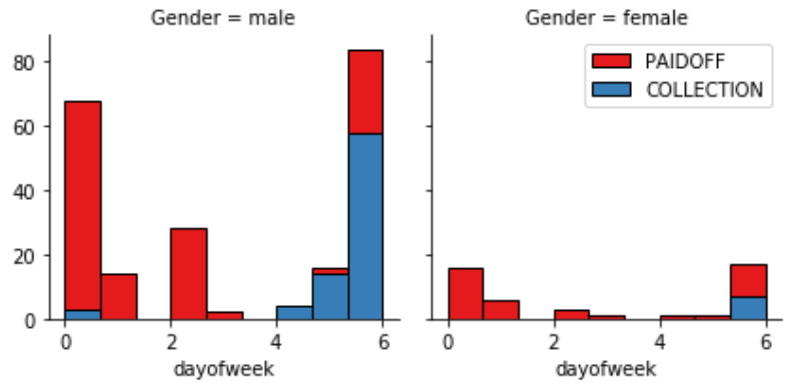
```
In [20]: df.groupby(['education'])['loan_status'].value_counts(normalize=True)

Out[20]: education             loan_status
         Bechalor              PAIDOFF        0.750000
                               COLLECTION     0.250000
         High School or Below  PAIDOFF        0.741722
                               COLLECTION     0.258278
         Master or Above       COLLECTION     0.500000
                               PAIDOFF        0.500000
         college               PAIDOFF        0.765101
                               COLLECTION     0.234899
         Name: loan_status, dtype: float64
```

Master or above education has the lowest paid off percentage(about 50%.

Exploratory Visualization

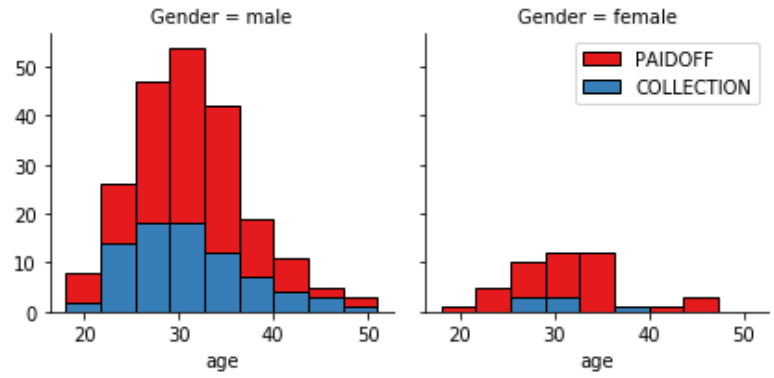
In the plots below, we visualize the relation ship between loan status and different features from the dataset like age, gender and day of week.

Loan status per gender and day of week



We see that people who get the loan at the end of the week dont pay it off.

Loan status per gender and age



Ages around 30 have significant amount of not paid off for both males and females.

Algorithms and Techniques

I used four classifiers to predict the loan status for a new applicant and I tuned the classifiers parameters to obtain the best results from each classifier.

As my problem actually turned to be a binary-classification problem, so that that I used the below classifiers to solve it.

The classifiers and parameters which I've used are:

KNN

KNN: (K Nearest Neighbor) is a simple algorithm that doesn't make any assumptions about underlying data and its structure determined from the data.

- In KNN, the output is a class membership and an object is classified by a majority vote of its neighbors.
- The nearer neighbors contribute more to the average than the more distant ones.
- The neighbors are taken from a set of objects for which the class is known.
- No training step is required.
- The training examples are vectors in a multidimensional feature space, each with a class label
- The best choice of k depends upon the data; generally, larger values of k reduces effect of the noise on the classification
    - Tuned Parameter(s): k value.

Decision tree

Decision tree is an easy to use algorithm that has a transparent nature and requires little effort from users regarding data preparation.

- A tree can be "learned" by splitting the source set into subsets based on an attribute value test.
- This process is repeated on each derived subset in a recursive manner called recursive partitioning.

- The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions.
- A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label.
    - Tuned parameter(s): max depth.

Support vector machine

- The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.
- Objective is to find a plane that has the maximum distance between data points of both classes.
- Hyperplanes are decision boundaries that help classify the data points
- Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane.
- There could be more than one hyper-plan and SVM target is selecting the best one (maximum margin hyper plan) to better separating of the classes.
    - Tuned parameter(s): kernel

Logistic regression

One of the most used Machine Learning algorithms for binary classification. It is a simple algorithm that you can use as a performance baseline, it is easy to implement, and it will do well enough in many tasks.

It uses a logistic function to model a binary dependent variable. In regression analysis, logistic is estimating the parameters of a logistic model; it is a form of binomial regression. Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail, win/lose, alive/dead or healthy/sick; these are represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value).

- Tuned parameter(s): C value

I used accuracy_score of sklearn.metrics to tune and test the parameters.

For each model(classifier), I trained it using the train set and test it using the test set, then I used Jaccard similarity score to evaluate the model.

References

https://en.wikipedia.org/wiki/Decision_tree_learning

https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1

https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch12.pdf

https://en.wikipedia.org/wiki/Logistic_regression#Logistic_model

Benchmark

We will use a naïve model as a benchmark for our classifiers then we will compare the benchmark accuracy with our final model to see how good our model does.

In our project, we implemented "Logistic Regression" model without any parameter tuning and then we trained it on the training set and tested it on a test set subset of the data.

The benchmark accuracy is: 0.71 and we expect our final model to outperform this benchmark.

In [114]:
```
#import required libs [Sherif Ramadan]
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
#Implement logistic regression as a benchmark
model_benchmark = LogisticRegression()
model_benchmark.fit(X_train,y_train)
benchmark_pred = model_benchmark.predict(X_test)
score = metrics.accuracy_score(benchmark_pred,y_test)
print("Score of benchmark model is: %.2f" % score)
```

```
Score of benchmark model is: 0.71
```

Methodology

Data Preprocessing

Data preprocessing done for this project includes the following:

- Calculating day of the week from "effective date"
- Get "Weekends" from "Day of the week".
- Convert categorical features to numerical values (Gender).
- Use one hot encoding to convert categorical values to binary values (Education).
- Data normalization, to give data zero mean and unit variance.

Implementation

The implementation done in two main stages:

- Implementing the four classifiers and tune their parameters.
- Computing and comparing the accuracy scores of the four models.

Scikit-learn was used for both defining and training the models and for measuring the accuracy and getting the scores.

Implementing the four classifiers and tune their parameters

KNN Implementation and parameter tuning

In [21]:
```
k=[1,2,3,4,5,6,7,8,9]
best_k = 1
best_score = 0
for x in k:
    model = KNeighborsClassifier(n_neighbors=x)
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    score = metrics.accuracy_score(y_pred,y_test)
    if score > best_score:
        best_score = score
        best_k = x
    print("K: ", x, " Score: ",score)
print("-----------")
```

```
K:  1  Score:  0.671428571429
K:  2  Score:  0.657142857143
K:  3  Score:  0.714285714286
K:  4  Score:  0.685714285714
K:  5  Score:  0.757142857143
K:  6  Score:  0.714285714286
K:  7  Score:  0.785714285714
K:  8  Score:  0.757142857143
K:  9  Score:  0.757142857143
-----------
```

In [22]:
```
print("Best Score:",best_score, "Best K:",best_k)
#train a model with best parameters
model_kn = KNeighborsClassifier(n_neighbors=best_k)
model_kn.fit(X_train,y_train)
```

```
Best Score: 0.785714285714 Best K: 7
```

Out[22]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=7, p=2,
                     weights='uniform')
```

Decision Tree Implementation and parameter tuning

In [43]:
```
from sklearn.tree import DecisionTreeClassifier
depths = [3,4,5,6,7,8,9,10,11]
best_depth = 1
best_score = 0
for x in depths:
    model2 = DecisionTreeClassifier(criterion="entropy", max_depth = x)
    model2.fit(X_train,y_train)
    y_pred = model2.predict(X_test)
    score2= metrics.accuracy_score(y_pred,y_test)
    if score2 > best_score:
        best_score = score2
        best_depth = x
    print("Depth:",x," Score:" ,score2)
print("---------------")
print("Best depth:", best_depth, " Best Score",best_score)
#train a model with best parameters
model_decesion_tree = DecisionTreeClassifier(criterion="entropy", max_depth = best_depth)
model_decesion_tree.fit(X_train,y_train)
```

```
Depth: 3  Score: 0.614285714286
Depth: 4  Score: 0.614285714286
Depth: 5  Score: 0.642857142857
Depth: 6  Score: 0.771428571429
Depth: 7  Score: 0.757142857143
Depth: 8  Score: 0.757142857143
Depth: 9  Score: 0.657142857143
Depth: 10  Score: 0.7
Depth: 11  Score: 0.685714285714
---------------
Best depth: 6  Best Score 0.771428571429
```

Support vector machine implementation and parameter tuning

In [52]:
```
from sklearn import svm
kernels = ['rbf','linear']
best_kernel = 'rbf'
best_score = 0
for x in kernels:
    model3 = svm.SVC(kernel=x)
    model3.fit(X_train, y_train)
    y_pred = model3.predict(X_test)
    score3 = metrics.accuracy_score(y_pred,y_test)
    if score3 > best_score:
        best_score = score3
        best_kernel = x
    print("With Kernel:",x, " Score is:", score3)
print("---------")
print("Best Kernel:",best_kernel, " Best Score:", best_score)
#train a model with best parameters
model_svm = svm.SVC(kernel=best_kernel)
model_svm.fit(X_train,y_train)
```

```
With Kernel: rbf  Score is: 0.757142857143
With Kernel: linear  Score is: 0.785714285714
---------
Best Kernel: linear  Best Score: 0.785714285714
```

Logistic regression implementation and parameter tuning

In [47]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
Cs = [0.01,0.1,1,10]
best_c = 0.01
best_score = 0
for c in Cs:
    model4 = LogisticRegression(C=c, solver='liblinear')
    model4.fit(X_train,y_train)
    y_pred = model4.predict(X_test)
    score4 = metrics.accuracy_score(y_pred,y_test)
    if score4 > best_score:
        best_score = score4
        best_c = c
    print(" C=",c," Score is:",score4)
print("-----------")
print("Best C:" , best_c, " Best Score:", best_score)
#train a model with best parameters
model_reg = LogisticRegression(C=best_c, solver='liblinear')
model_reg.fit(X_train,y_train)
```

```
 C= 0.01  Score is: 0.685714285714
 C= 0.1  Score is: 0.742857142857
 C= 1  Score is: 0.714285714286
 C= 10  Score is: 0.714285714286
-----------
Best C: 0.1  Best Score: 0.742857142857
```

Compute accuracy scores for the four parameters using the test set

KNN scores

In [55]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
```

In [45]:
```
y2 = test_df['loan_status'].values
y2[0:5]
```

In [59]:
```
y_pred_kn = model_kn.predict(X2)
print("Jaccard score: %.2f"% jaccard_similarity_score(y2,y_pred_kn))
print("F1-score: %.2f" % f1_score(y2,y_pred_kn,average='weighted'))
```

```
Jaccard score: 0.65
F1-score: 0.63
```

Decision Tree Scores

```
In [52]: y_pred_dt = model_dt.predict(X2)
         print("Jaccard score: %.2f"% jaccard_similarity_score(y2,y_pred_dt))
         print("F1-score: %.2f" % f1_score(y2,y_pred_dt,average='weighted'))

         Jaccard score: 0.72
         F1-score: 0.74
```

SVM Scores

```
In [53]: y_pred_svm = model_svm.predict(X2)
         print("Jaccard score: %.2f"% jaccard_similarity_score(y2,y_pred_svm))
         print("F1-score: %.2f" % f1_score(y2,y_pred_svm,average='weighted'))

         Jaccard score: 0.80
         F1-score: 0.76
```

Logistic Regression Scores

```
In [58]: y_pred_reg = model_reg.predict(X2)
         y_pred_prop = model_reg.predict_proba(X2)
         print("Jaccard score: %.2f"% jaccard_similarity_score(y2,y_pred_reg))
         print("F1-score: %.2f" % f1_score(y2,y_pred_reg,average='weighted'))
         print("LogLoss score: %.2f" % log_loss(y2, y_pred_prop))

         Jaccard score: 0.76
         F1-score: 0.71
         LogLoss score: 0.57
```

Refinement

As mentioned and shown in the implementation section, I have tuned the model parameters to get the best possible result.

I have tuned “k” for KNN, “max-depth” for decision tree, “kernel” for svm and “C” for logistic regression.

I have tuned them manually through putting different values for each parameter in a list and looping on them.

While in loop, I record the result (Score) of each parameter value.

By the end of the loop, I have the best score and best parameter value corresponding to it.

In this regard, I found the best result for each model by the using the following values for model parameters:

- For KNN, best “K” was 7.
- For decision tree, best “max depth” was: 6
- For SVM, best “kernel” was: linear
- For logistic regress, best “C” was: 0.1

Results

Model Evaluation and Validation

The model's final parameters values were chosen as they output the best results.

As we declared in "Refinement" section:

- For KNN, best "K" was 7.
- For decision tree, best "max depth" was: 6
- For SVM, best "kernel" was: linear
- For logistic regress, best "C" was: 0.1

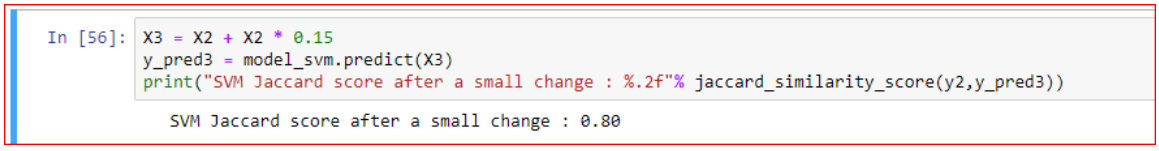
To verify the robustness of the models, they have been tested against separate data set (test set) to measure their performance and accuracy.

- KNN Jaccard score was 0.65
- Decision tree Jaccard score was: 0.72
- SVM Jaccard score was: 80
- Logistic regression Jaccard score was: 0.76

From these scores, we could detect that "SVM" with "linear" kernel achieved the best results from all the four models used (80 %).

As the model scored "80 %" when tested on un-seen data, it is considered a good model and generalizes well and we can trust it.

The model is not sensitive to small changes in data. To verify this, I have changed the test set by multiplying it by 0.15 and re-evaluate the model and get the same score (80%).



```
In [56]: X3 = X2 + X2 * 0.15
         y_pred3 = model_svm.predict(X3)
         print("SVM Jaccard score after a small change : %.2f"% jaccard_similarity_score(y2,y_pred3))

            SVM Jaccard score after a small change : 0.80
```

Justification

The SVM model with:" linear" kernel outperforms our benchmark values that were depending on the accuracy of the model against the "Training" set.

The following snippet shows the values of both our benchmark (training set) and our model (test set) and it clearly shows that the model is doing and generalizing well.

```
In [129]: y_pred_test = model_svm.predict(X2)

          print("Benchmark model Jaccard Score  is: %.2f" % jaccard_similarity_score(benchmark_pred,y_test))
          print("SVM Jaccard Score for test (our target) is: %.2f" % jaccard_similarity_score(y2,y_pred_test))

             Benchmark model Jaccard Score  is: 0.71
             SVM Jaccard Score for test (our target) is: 0.80
```

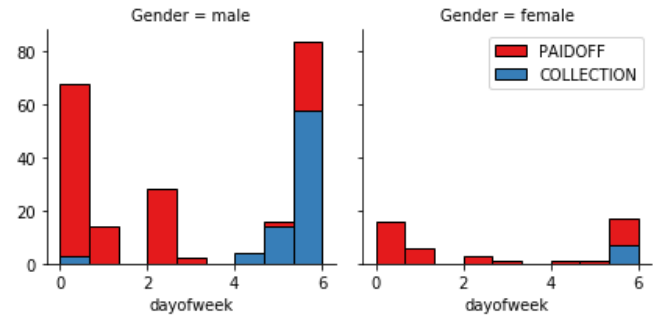
The final model seems to be good enough to solve the problem as of its good score and even after applying small change – as mentioned in above section – to the testing data the model still generalizes and performs well.

Conclusion

Free-Form Visualization

The plot below shows the "loan status" alongside the day of the week the loan was got.

```
In [101]: df['dayofweek'] = df['effective_date'].dt.dayofweek
          bins = np.linspace(df.dayofweek.min(), df.dayofweek.max(), 10)
          g = sns.FacetGrid(df, col="Gender", hue="loan_status", palette="Set1", col_wrap=2)
          g.map(plt.hist, 'dayofweek', bins=bins, ec="k")
          g.axes[-1].legend()
          plt.show()
```



As we see, people who get the loan at the end of the week don't pay it.

The plot also shows that males are more likely not to pay their loans taken at the end of the week more than females.

Reflection

The process used for this project can be summarized as follows:

1- Download and load the data.
2- Explore the data and get information about its main statistics:
    a. Mean, max, min, standard deviation of each numerical field.
    b. Show distribution of "loan-status" per gender, principal and week day.
    c. Show distribution of "loan status" per education level.
    d. Discover "loan status" taken at the end of the week.
3- Perform data cleaning and pre-processing
    a. Convert categorical data to binary (Gender).

    b. Perform one hot encoding on features.
4- Implement the benchmark classifier without parameter tuning.
5- Log the accuracy of the benchmark classifier.
6- Implement KNN algorithm and tune its “K” parameter.
7- Log the best score and best “K” for KNN.
8- Implement decision tree algorithm and tune its “max-depth” parameter.
9- Log the best score and best “max-depth” value.
10- Implement the SVM and tune its “kernel” parameter.
11- Log the bests core and best kernel.
12- Implement the logistic regression algorithm and tune its parameter “C”
13- Log the best score and best value for “C”.
14- Compute the scores of the implemented algorithms using Jaccard similarity by testing the classifier against test set.
15- Compare the scores of the implemented algorithms and pick the best one.
16- Compare the score of selected algorithm and the benchmark.

I found the part of parameter tuning the most interesting and challenging part. I felt really exited when reaching the best parameter values for the models.

Improvement

In this project, we have implemented four classifiers and tuned their parameters, then we compute and compare their accuracy to get the best classifier.

We could improve our solution by trying the following activities:

1- Apply more tuning on the model’s parameters.
2- Obtaining larger data set.
3- Trying other classifiers such as: LightGBM, xgBoost.

References

https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7

https://machinelearning-blog.com/2018/04/23/logistic-regression-101/

https://stats.stackexchange.com/questions/24437/advantages-and-disadvantages-of-svm