TERM PROJECT: PHASE III

PROJECT REPORT

Eric Jiang, Akhsitha Ajayan, Safa Shaikh

ABSTRACT

Machine Learning has a significant impact on fields in need of accurate predictions. This report demonstrates such beneficial use of Machine Learning methods and algorithms with three different datasets. The first dataset had the goal of detecting credit card fraud. The frauds were detected using K-Nearest Neighbor, Logistic Regression, and Support Vector Machine. The second dataset was used to identify images of fruits. The fruits were classified via Support Vector Machine, Logistic Regression, and Classification Tree. The third dataset required prediction of temperature in Szeged, Hungary. This was solved using regression methods: Least Squares, Ridge, and Regression Tree. All methods were successfully processed with very high accuracy rates. Realistically, these findings can be implemented into actual scenarios to help with weather forecasts, sorting fruit, and detecting fraud.

Eric Jiang

445 ECE: ML For ENG

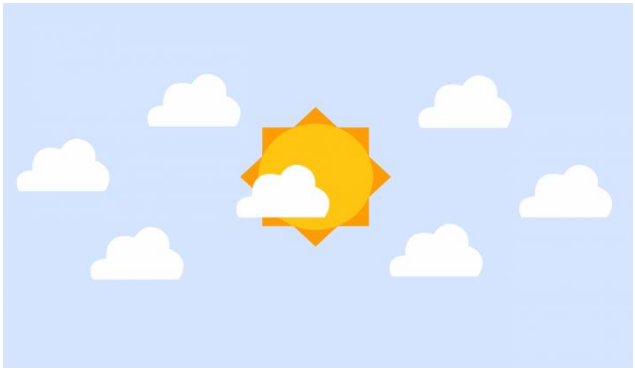
Table of Contents

Introduction:

For this project, processing for two of the three basic Machine Learning categories classification, regression, and clustering were used. Since each project group had three members, each member was responsible for researching a dataset of different modality for processing. These datasets are **credit card fraud with Eric** (https://www.kaggle.com/mlg-ulb/creditcardfraud), **fruits 360 with Safa** (https://www.kaggle.com/moltean/fruits), and **Weather in Szeged 2006-2016 with Akshitha** (https://www.kaggle.com/budincsevity/szeged-weather).

Each member was responsible for the overall quality of their assigned dataset processing including initial CSV loading, feature engineering via PCA, code comments, notebook markups, and combining all the approaches used for the dataset.

In addition, each member was responsible for processing the dataset with an approach totaling three approaches per set. For the credit card fraud dataset classification was used with k-Nearest Neighbor Classifier (Eric), Logistic Regression Classifier (Akshitha), and Support Vector Machine Classifier (Safa). The fruits 360 dataset was identified with classification using Support Vector Machine Classifier (Eric), Logistic Regression Classifier (Akshitha), and Classification Tree (Safa). The Weather in Szeged was predicted using regression via Least Squares Regression (Akshitha), Ridge Regression (Eric), and Regression Tree (Safa). The approaches were analyzed and produced accurate results. The approaches and results are discussed in the next analysis section.



Analysis – Credit Card Fraud:

Context:

The purpose of this dataset is to help credit card companies to recognize fraudulent credit card transactions to alert and benefit customers.

Dataset Overview:

The dataset for credit card fraud was taken from transactions made between two days in September 2013 with credit cards in Europe. Due to confidentiality reasons, the original data has been transformed into 28 principal components via PCA (V1, V2, … V28). The only known features that have not been transformed are 'Time' and 'Amount'. Time indicates the seconds elapsed from initializing the transaction to completion of the final transaction. Amount is the total amount of money involved in the transaction. The class feature is the output label representing the response variable with {1} for fraud and {0} for non-fraud.

The CSV Header for Credit Card Fraud:

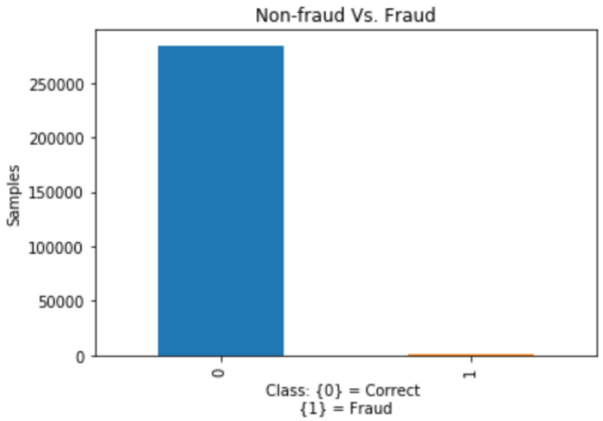
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

Dataset Imbalance:

After loading the values from the CSV, it was discovered that the majority of the 285,000 samples were non-fraud. Only 492 samples were actual fraud which was only 0.17% of the entire dataset. This meant that the dataset is highly imbalanced. This would cause a lot of errors by overfitting algorithms. The models would assume that most transactions are non-fraud due to the 99.83% of non-fraud in the set. To solve this problem, the dataset was under sampled into a subset with 50% fraud and 50% non-fraud.

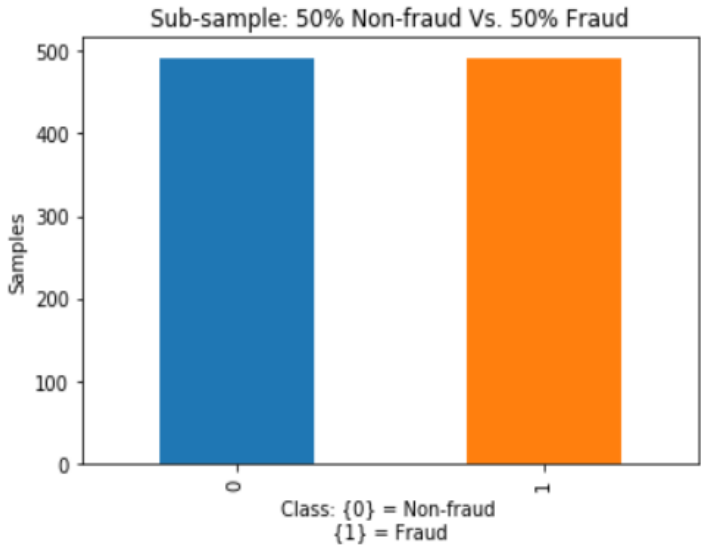
Bar graph of the entire dataset

```
#Portray the frequency of fraud
count_classes = pd.value_counts(dfo['Class'], sort = True).sort_index()
count_classes.plot(kind = 'bar')
plt.title("Non-fraud Vs. Fraud")
plt.xlabel("Class: {0} = Correct \n{1} = Fraud")
plt.ylabel("Samples")
plt.show()

correct = count_classes.tolist()[0]
fraud = count_classes.tolist()[1]
print(B+"Number of Correct Samples:"+E, correct)
print(B+"Number of Fraud Samples:"+E, fraud)
print(B+"Ratio of Fraud to Correct:"+E, fraud/correct)
```




```
Number of Correct Samples: 284315
Number of Fraud Samples: 492
Ratio of Fraud to Correct: 0.0017304750013189597
```

```
New undersampled sub-sample
Fraudulent samples: 492
Non-fraud samples: 492
```



To create the 50/50 subset, first all samples were fitted and transformed by scaling the amount and time with mean/standard deviation. Next, all 492 samples of fraud were taken and a random choice of 492 non-frauds were appended to the fraudulent list. Having a sub-sample like this would help solve the overfitting issue by allowing frauds to be detected more accurately. In addition, having a 50/50 sample would help identify the correlations of the unknown V features on the impact of classifying fraud.

Principal Component Analysis:

To do PCA on the data, the dataset was split into training and test sets. In this instance, the algorithm used 15% of the data for testing and 85% for training. To factor in randomization, the dataset was first scrambled in random order.

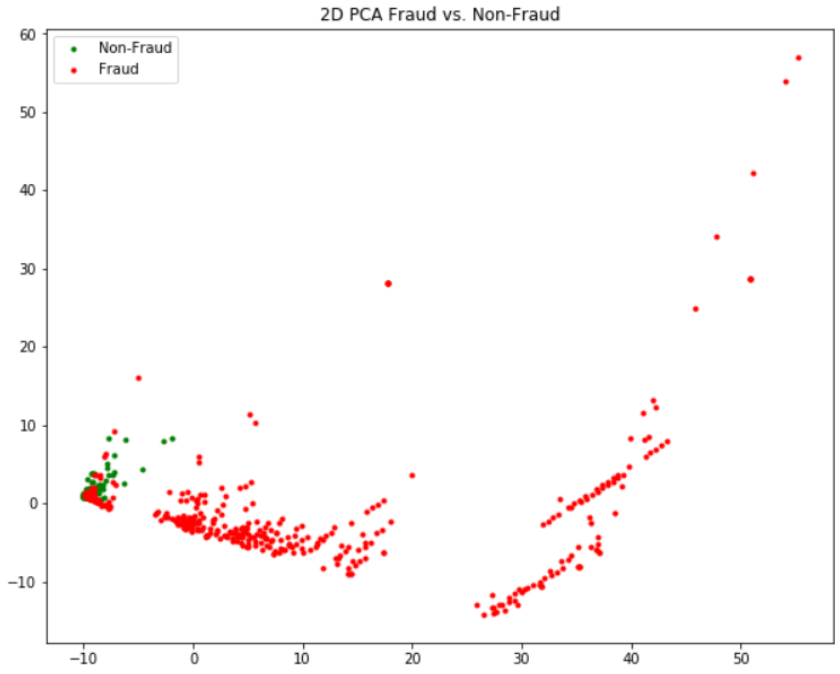
Scrambling the dataset:

```
# Scramble the dataset so fraud isn't ordered first
usd = usd.sample(frac=1)

# Undersampled X data and Y labels via Class
Xu = usd.loc[:, usd.columns != 'Class']
Yu = usd.loc[:, usd.columns == 'Class'] # Yu == 1 then fraud otherwise 0
```

Creating test and train data/labels:

```
#Optimize the train/test lengths (this uses 15% of data as test)
test_percent = .15
test_len = (np.floor(test_percent*len(Xu))).astype(int)
train_len = (len(Xu)-test_len).astype(int)

#Create Training Data
training_data = Xu[:train_len]
training_labels = Yu[:train_len]

#Create Test Data
test_data = Xu[train_len:]
test_labels = Yu[train_len:]
```

From the 2D PCA plot to the right, it is shown that half of the fraud (red) can be easily separated. However, a good portion of the fraud is still within the same region as the green non-fraudulent points. This would mean that errors are likely to happen in the overlapping region.

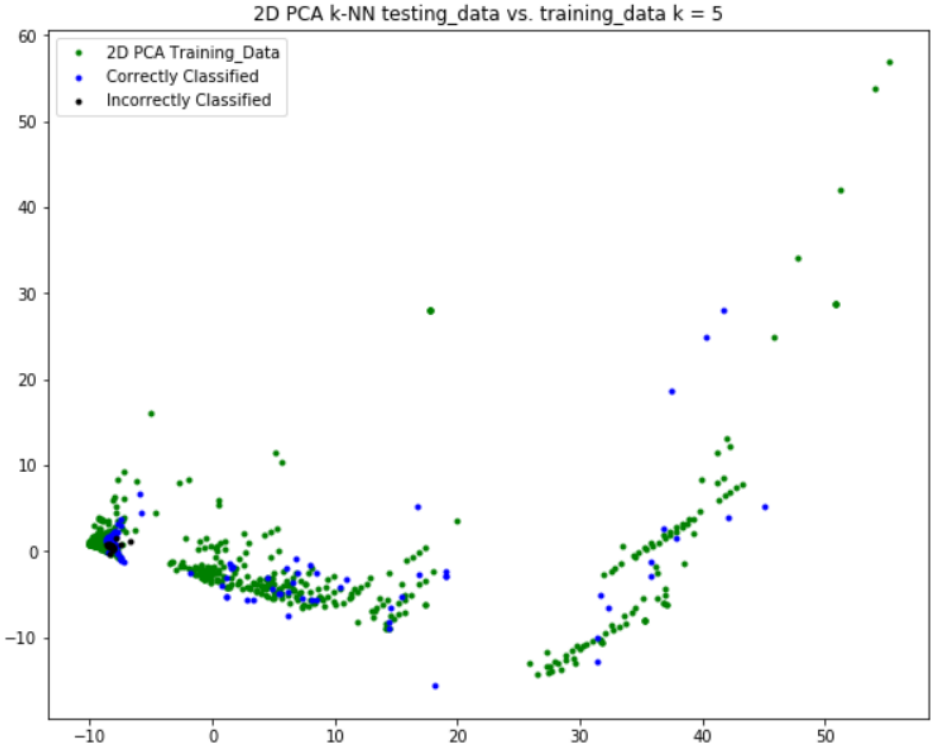


K-Nearest Neighbor:

k-Nearest Neighbor Algorithm

After implementing k-NN algorithm with k = 5. It was shown that there indeed was an error of 14%. The error varied depending on the given random non-fraud subsets along with the scrambling before creating test-folds. Generally, the error only occurred in the region to the left with overlapping of non-fraud/fraud. Whereas, the right side classified the points correctly was shown in the blue.

```
#Create k-NN function to calculate the norms to the desired 'point'
#Using closest 'k' # of points to classify labels
def knn(point,k):
    #Create a matrix to store all the norm distances w/ labels
    norm_mat = []

    #Use the train_len pt of the training_data set
    for i in range (0,train_len):
        #Use X_tilde the 2D PCA features of training_data
        #Find norm between point and training set
        norm = np.linalg.norm(point-X_tilde.T[i])

        #append each (norm, label) using class imagelabel(norm,label)
        if training_labels.values[i] == 0:
            norm_mat.append(imagelabel(norm,0))
        else:
            norm_mat.append(imagelabel(norm,1))

    #sort based on image's closest norms
    nmsort = sorted(norm_mat, key=lambda pt: pt.image)

    #Create counters to determine labels for k closest pts.
    counter0 = 0
    counter1 = 0

    #loop through k closest points
    for i in range (0,k):
        nms = nmsort[i]
        if nms.label == 0: #if label = 0 counter0++
            counter0+=1
        else:
            counter1+=1 #else label = 1 counter1++

    #output 0,1 label corresponding to highest counter
    if counter0 >= counter1:
        output = 0
    else:
        output = 1

    #return norm matrix, sorted norm matrix in order of closest to furthest pts, 0-1 output
    return norm_mat, nmsort, output
```




```
Error with k = 5: 0.14285714285714285
```

K-Nearest Neighbor (cont.):

For later use, the optimal k value was found on the full dataset. This was determined with sklearn's KNeighborsClassifier as the custom algorithm could not process the entire 285,000 samples. It was determined that the optimal value of k was 3 at the best accuracy score of 93.9%.

Implementing KNeighborsClassifier on entire dataset:

```
from sklearn.neighbors import KNeighborsClassifier
import operator

## KNeighborsClassifier(n_neighbors)
# Creates a classifier with parameters n_neighbors = k

## cross_val_score(estimator,X,y,cv,scoring)
# estimator -> classifier
# X = data | Y = labels | cv = folds | scoring = type of metric to estimate
# returns a set of scoring metrics unless specified via 'scoring param'

# search for an optimal value of K for KNN

# Retrieve Y values in 1D form
Yf = np.array(Yu).T[0]

# list of reasonable k-integers to test
k_range = range(1, 40)

# list of scores from k_range
k_scores = []

#loop through reasonable values of k
for k in k_range:
    #  run KNeighborsClassifier with k neighbours
    knn = KNeighborsClassifier(n_neighbors=k)
    # obtain cross_val_score for KNeighborsClassifier with k neighbours
    scores = cross_val_score(knn, Xu, Yf, cv=10, scoring='accuracy')
    # append mean of scores for k neighbors to k_scores list
    k_scores.append(scores.mean())

index, value = max(enumerate(k_scores), key=operator.itemgetter(1))

print(B+"Optimal value of k =", index+1)
print("Has accuracy score of:", value, "%"+E)
```

```
Optimal value of k = 3
Has accuracy score of: 0.9391020408163264 %
```

Confusion Matrix Metrics:

To minimize error rate, binary confusion matrices were implemented. The confusion matrix metrics such as recall, precision, and accuracy were analyzed in depth.

The formulas for recall, precision, and accuracy are given below:

Recall = True pos/(True pos + False neg)

Recall = True fraud/(True fraud + False non-fraud)

Precision = True pos/(True pos + False pos)

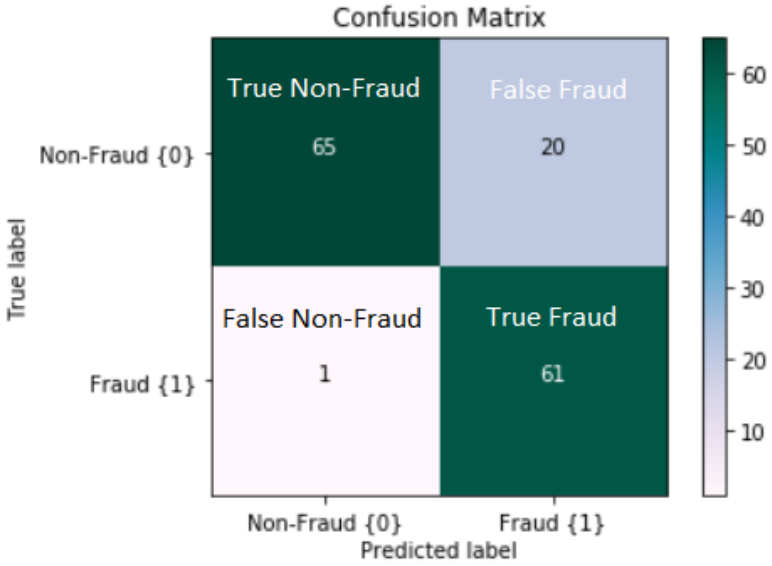
Precision = True fraud/(True fraud + False Fraud)

Accuracy = (True neg + True pos)/(True neg + False pos + False neg + True Pos)

Accuracy = (True non + True fraud)/(True non + True fraud + True non + True fraud)

- The recall metric was prioritized as it would determine the amount of false non-frauds. False non-frauds are the frauds that go undetected.
    - Higher recall would reduce the amount of undetected frauds.
- The precision metric was also desired but not mandatory as it determined the amount of false frauds. False frauds would correlate to the amount of non-frauds that were mistakenly identified as frauds.
    - Higher precision would lead to less false fraud alerts.
- The accuracy metric was the balance between max recall and max precision

Confusion Matrix for k-NN sub-sample:



As shown, k-NN had a high recall rate. This was a success as only 1 sample was undetected as fraud. However, it had lower precision which was shown with the 20 false fraud alerts.

```
Matrix Recall: 98.38709677419355 %
Matrix Precision: 75.30864197530865 %
```

Logistic Regression Classifier:

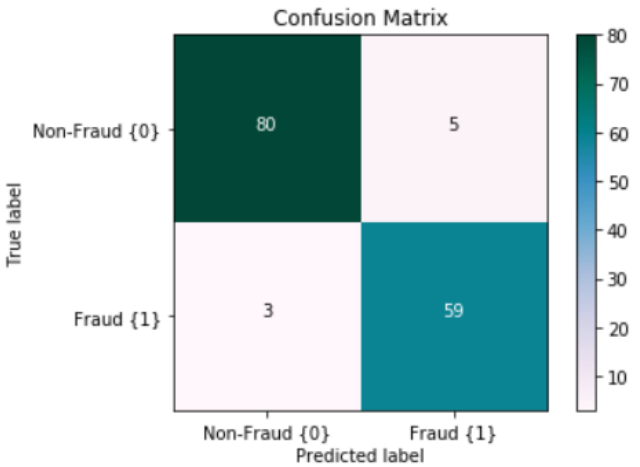
Several different C parameters [0.01, 0.1, 1, 10, 100] for Logistic Regression was tested to find the optimal C value. This was done through K-fold cross validation using 3-folds. This essentially divides the sub-sample into three folds of equal length using one-fold for testing and the remaining for training. Each fold was used for testing once, meaning there would be 3 iterations of testing for the 3 total folds.

The prioritized recall score was calculated per fold iteration and averaged out to determine the average recall for the corresponding parameter. Ultimately it was determined that the C-parameter of 0.01 was optimal for the best mean recall score.

As shown in the Logistic Regression Confusion Matrix to the right, there was a high recall and precision score. This showed that Logistic Regression was a good approach for credit card fraud classification.

Finding optimal C with k-fold CV:

```
Parameter for C = 0.01

Iteration 1 : recall score =  0.959731543624161
Iteration 2 : recall score =  0.9927007299270073
Iteration 3 : recall score =  0.9652777777777778

Mean recall score  0.9725700171096486
_____________________________________________________
Parameter for C = 0.1

Iteration 1 : recall score =  0.8791946308724832
Iteration 2 : recall score =  0.927007299270073
Iteration 3 : recall score =  0.9027777777777778

Mean recall score  0.9029932359734446
_____________________________________________________
Parameter for C = 1

Iteration 1 : recall score =  0.8657718120805369
Iteration 2 : recall score =  0.927007299270073
Iteration 3 : recall score =  0.9236111111111112

Mean recall score  0.9054634074872404
_____________________________________________________
Parameter for C = 10

Iteration 1 : recall score =  0.8590604026845637
Iteration 2 : recall score =  0.927007299270073
Iteration 3 : recall score =  0.9375

Mean recall score  0.9078559006515455
_____________________________________________________
Parameter for C = 100

Iteration 1 : recall score =  0.8590604026845637
Iteration 2 : recall score =  0.927007299270073
Iteration 3 : recall score =  0.9375

Mean recall score  0.9078559006515455
_____________________________________________________
Optimal C parameter =  0.01
```




```
Matrix Recall: 95.16129032258065 %
Matrix Precision: 92.1875 %
```

Support Vector Machine:

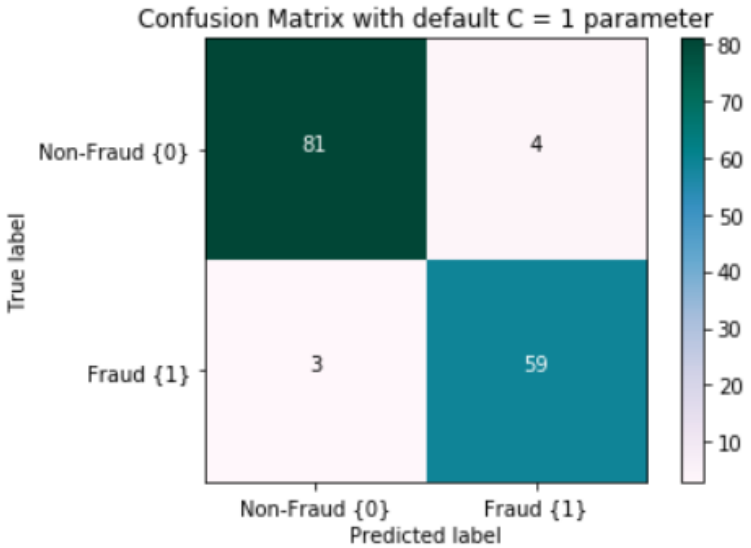
Originally the default C value of 1 was used. The optimal value of C was found by using GridSearchCV (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html). Different C parameters of [1, 10, 100, 1000] was input, along with the choice of linear or rbf kernel, and the corresponding gamma value. The scoring value to determine the optimal parameter was recall.

Implementation of GridSearchCV

```
from sklearn.model_selection import GridSearchCV

# Use GridSearchCV to estimate optimal parameter

parameters = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']}, {'C': [1, 10, 100, 1000], 'kernel': ['rbf'
], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}]

grid_search = GridSearchCV(estimator = classifier, param_grid = parameters, scoring = 'recall', cv = 10)

grid_search = grid_search.fit(training_data, training_labels.values.ravel())

best_parameters = grid_search.best_params_
print("Using optimal parameters:", best_parameters)
```

```
Using optimal parameters: {'C': 1, 'gamma': 0.8, 'kernel': 'rbf'}
```

After implementing GridSearchCV, it was discovered that the default C = 1 was the optimal parameter along with a non-linear Radial Basis Function kernel and Gamma of 0.8. Gamma represents the free variable for RBF, a lower gamma would mean lower variance. Lower variance corresponds to the support vectors having a larger influence on the model. From the confusion matrix shown below, SVM was a good approach as the recall and precision were high.




```
Matrix Recall: 95.16129032258065 %
Matrix Precision: 93.65079365079364 %
```

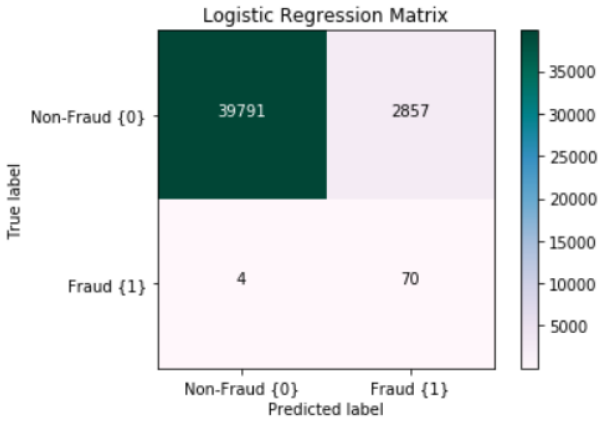
Discussion:

In conclusion:

1. The models were all very high in recall
- The recalls varied slightly and were different for each different scrambled sub - set.
2. It was noticeable that k-NN had consistently highest precision, meaning it was the best model for fraud detection along with least false alarms.
- This was is due to the focus on the accuracy metric vs. recall metric when finding the optimal k value.
- Accuracy is a balance between recall and precision.
3. Given already an almost capped recall, an increase of precision would drastically help reduce the number of false frauds detected.

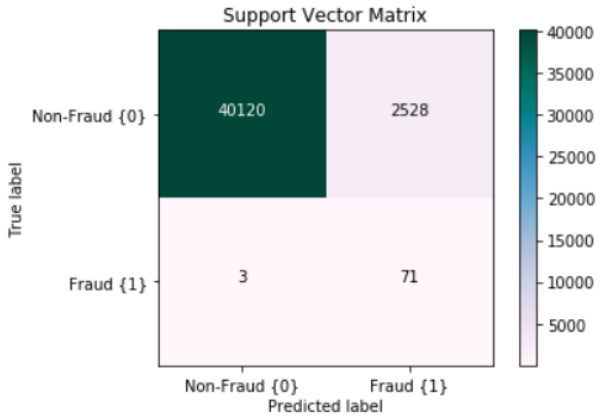
Although the recall scores were successfully high, the precision scores were all quite low. This is due to the fact that the entire dataset was very imbalanced on non-fraud. This would lead to a lot of non-frauds mistakenly identified as fraud.

Logistic Regression Classifier



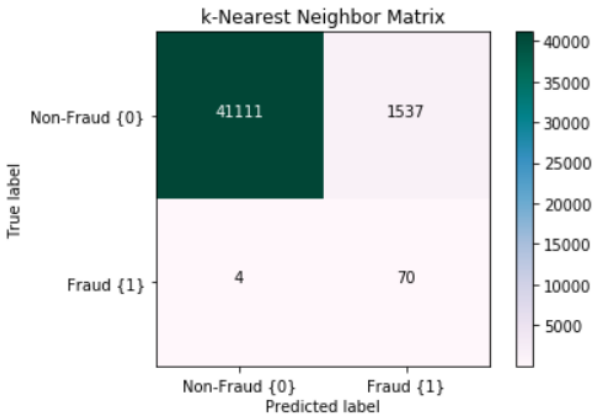
Matrix Recall: 94.5945945945946 %
Matrix Precision: 2.3915271609156132 %

SVM Classifier



Matrix Recall: 95.94594594594594 %
Matrix Precision: 2.7318199307425934 %

kNN Classifier



Matrix Recall: 94.5945945945946 %
Matrix Precision: 4.355942750466708 %

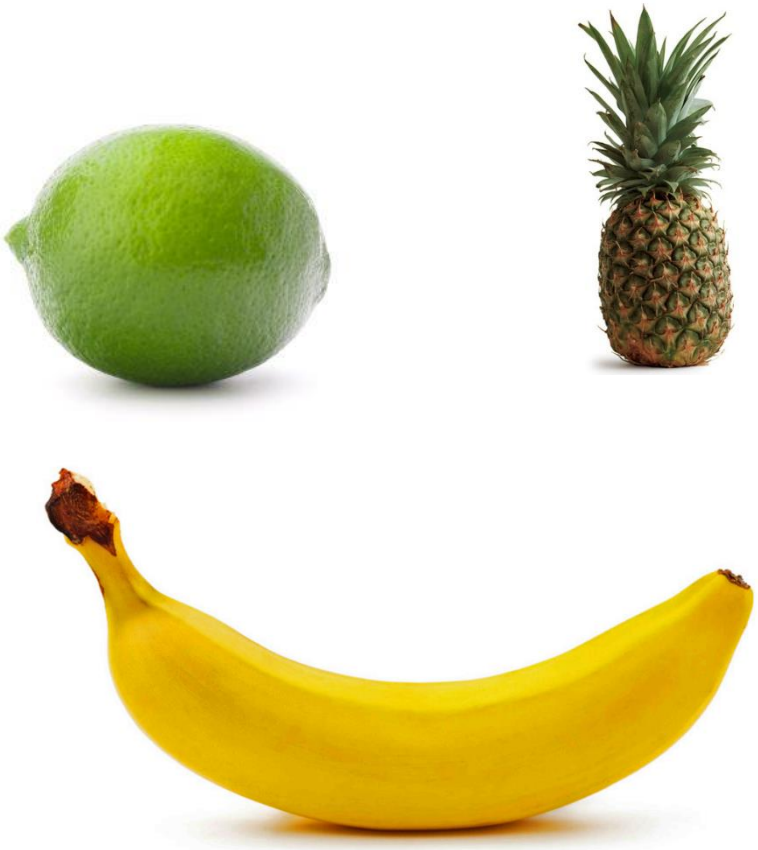
Analysis - Fruits 360°:

Context:

The purpose of this dataset is to classify various fruits through using their entire 360 degrees images.

Dataset Overview:

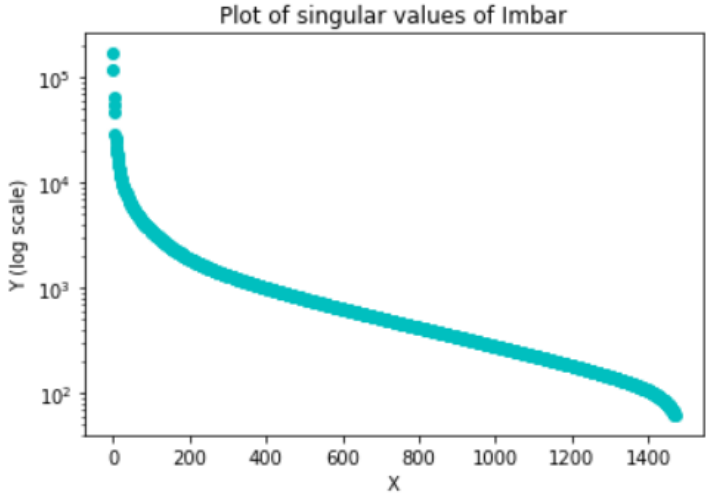
The dataset contains image files of various fruits obtained with a Logitech C920 camera. The fruits were placed behind a background of white sheet paper, meaning white pixels were identified as the background. The images were originally 100x100, but were resized to 45 x 45 x 3 numpy arrays for processing. There were also 88 different fruit classes, so the project chose three distinct fruits: Banana, Limes, and Pineapples for convenience. These fruits were labeled with numeric ID's Banana = 0, Limes = 1, Pineapple = 2.

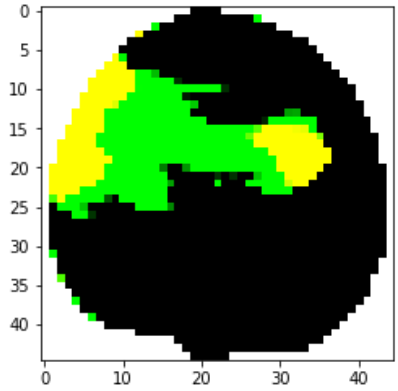


Principal Component Analysis:

Each image was flattened by combining the array set of 45x45x3 images into a one-dimensional array for input. These images were then fitted and transformed with mean centering for standardized PCA processing. A simple algorithm was used to find the smallest number of features that encompassed 90% of the data. It was determined that K = 51 had 90% of the data variation.

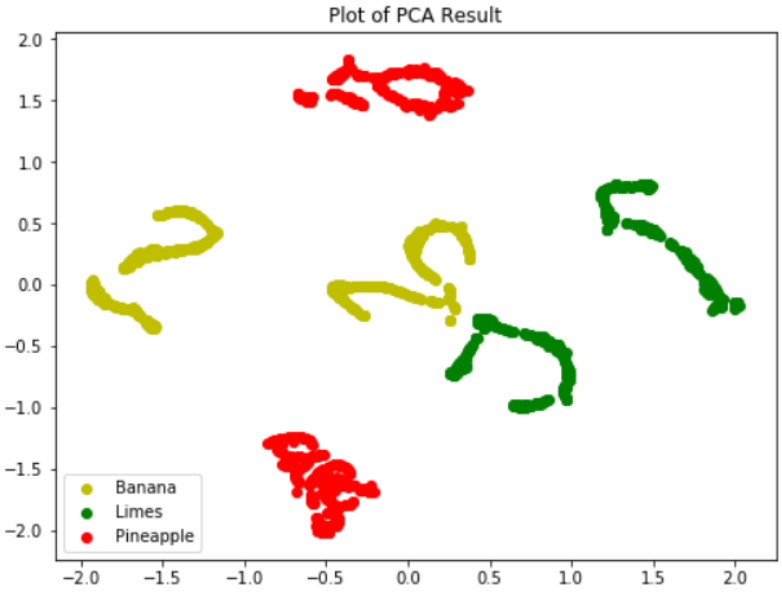
Log Plot showing the fruit image features X vs. data variation Y:





Lime with k = 51 PCA

As shown in the PCA plot to the right, the three classes Banana, Limes, and Pineapple could be easily separated.



Support Vector Machine:

Linear SVM forms hyperplanes to split classes via $f(x) = w^T x+b$. However, as shown in the PCA plot, the three classes are placed in very different positions resulting in bad linear boundaries. Using the non-linear radial basis function would be optimal in creating boundaries for these displaced classes. After using rbf SVM, the training set's best accuracy score was predicted resulting in a precision score of 95% and recall score of 94%.

The important parameters for sklearn.svm.SVC are:

C – the penalty parameter affecting the error score (usually [1, 10, 100, 1000]).

Kernel – The type of kernel used in the algorithm (can be 'linear', 'poly', 'rbf', or 'sigmoid'.

Gamma: The kernel coefficient for non-linear kernels that also determines the error score.

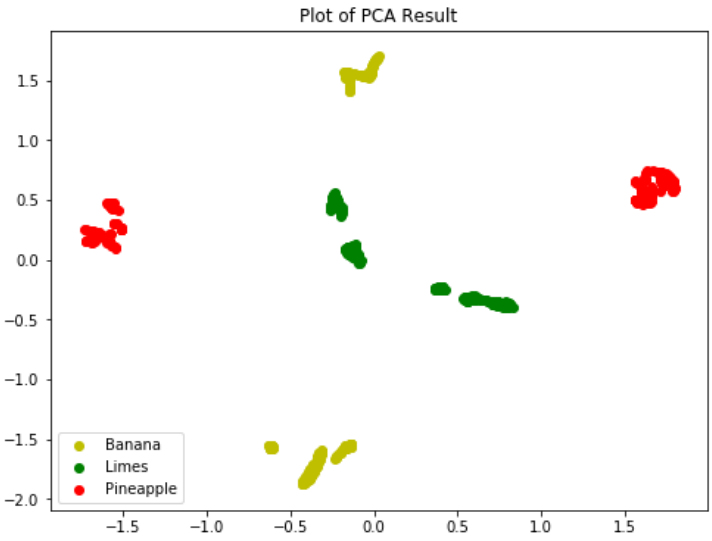
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

3x3 Confusion Matrix and Metrics of Fruits:

	precision	recall	f1-score	support
Banana	0.83	1.00	0.91	103
Limes	1.00	1.00	1.00	117
Pineapple	1.00	0.86	0.92	148
avg / total	0.95	0.94	0.94	368

```
[[103   0   0]
 [  0 117   0]
 [ 21   0 127]]
```

PCA of the validation set:



After processing SVM on the Test set and comparing to the validation set, it was discovered that SVM only had a validation accuracy of 64%. For future implementation, this could be improved with cross validation and averaging the accuracies.

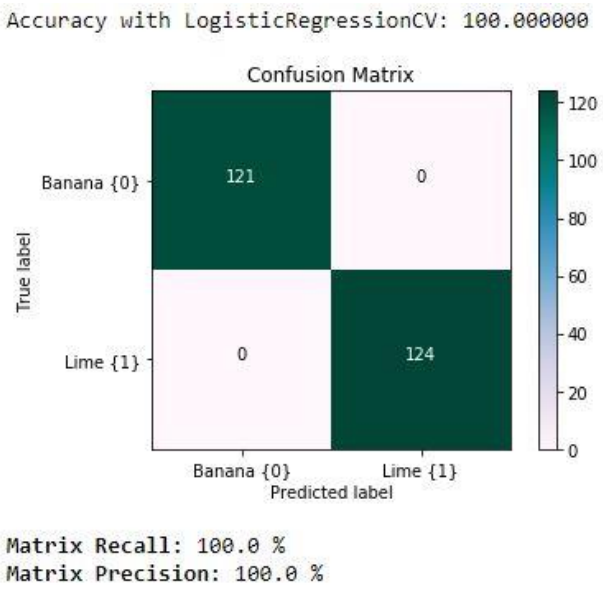
Logistic Regression Classifier:

LogisticRegressionCV was used:

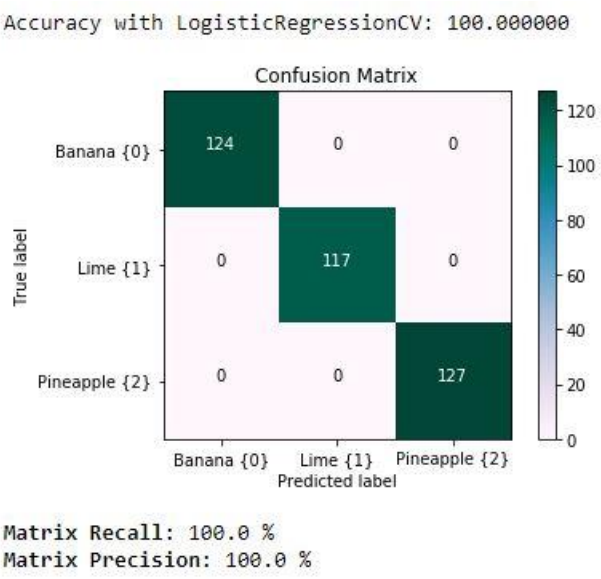
- **Cs:** list of C parameters to determine the desired error score
- **Fit_intercept:** Includes bias or intercept into the decision function
- **CV:** The amount of folds for k-fold cross validation
- **Penalty:** The norm used for penalization (can be 'l1' or 'l2'
- **Random State:** If set to different values will produce different results, it's custom to set it as a constant value to reproduce the same results.

As shown below, binary confusion matrix was created for banana and lime. In addition, all three classes were compared with a 3x3 confusion matrix. In both instances, the recall and precision scores were 100% meaning Logistic Regression is a good classifier.

Binary Matrix for Banana vs. Lime



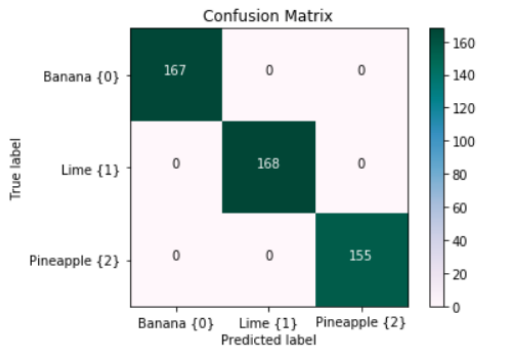
3x3 Confusion Matrix



Classification Tree:

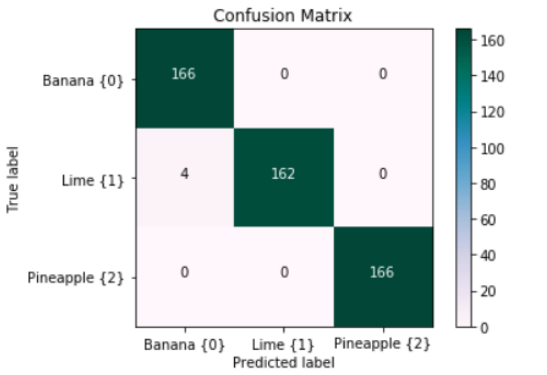
For classification tree, the Gini index was used as a metric for measuring variance. The gini index explains error for each class region. The model was fit with the training data to determine the label with maximum probability. After K-fold cross validation, the optimal training set was obtained. This led to a 99.86% accuracy with a 99% validation accuracy as shown in the corresponding matrices.

Accuracy with Classification Tree: 100.000000



	precision	recall	f1-score	support
Banana	1.00	1.00	1.00	167
Limes	1.00	1.00	1.00	168
Pineapple	1.00	1.00	1.00	155
avg / total	1.00	1.00	1.00	490

Validation Accuracy with Classification Trees: 99.196787



	precision	recall	f1-score	support
Banana	1.00	0.98	0.99	170
Limes	0.98	1.00	0.99	162
Pineapple	1.00	1.00	1.00	166
avg / total	0.99	0.99	0.99	498

Discussion:

It was determined that SVM performed the worst with 94% accuracy and 64% validation accuracy. Both logistic regression and classification trees performed much better with 100% accuracy and 99.2% validation accuracy. Thus, Logistic Regression and Classification trees should be preferred for identifying fruit.

Weather in Szeged Dataset:

Improper Madrid Air-Quality Set:

The initial dataset chosen was supposed to be Air Quality in Madrid 2018. This csv file contained parameters for chemicals in the air measured at every weather station in Madrid. The data was collected every 24 hours from January to April. Unfortunately, many of the parameters contained null values with the exception of Nitrogen Oxide compounds. This made the data incomplete for proper processing and instead Weather in Szeged was used.

Madrid Air-Quality header:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	29.0	31.0	NaN	NaN	NaN	2.0	NaN	NaN	28079004
1	2018-03-01 01:00:00	0.5	1.39	0.3	0.2	0.02	6.0	40.0	49.0	52.0	5.0	4.0	3.0	1.41	0.8	28079008
2	2018-03-01 01:00:00	0.4	NaN	NaN	0.2	NaN	4.0	41.0	47.0	NaN	NaN	NaN	NaN	NaN	1.1	28079011
3	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	35.0	37.0	54.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2018-03-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	27.0	29.0	49.0	NaN	NaN	3.0	NaN	NaN	28079017

Context:

The purpose of this dataset was to analyze past weather data in Szeged, Hungary and accurately predict future temperature forecasts for the city.

Dataset Overview:

The dataset contains hourly weather features in Szeged, Hungary between 2006 and 2016. The features extracted from the dataset are apparent temperature (C), humidity, wind speed (km/h), wind bearing (degrees), and visibility (km). The target variable was the temperature (C).

Weather in Szeged header:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Daily Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout the day.
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout the day.
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout the day.
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout the day.
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Partly cloudy throughout the day.

Least Squares and Ridge Regression:

For regression problems, the y label is estimated via y = X*w.

For least squares regression: The variable w is determined using $\mathbf{w = (XX^T)^{-1}X^Ty}$

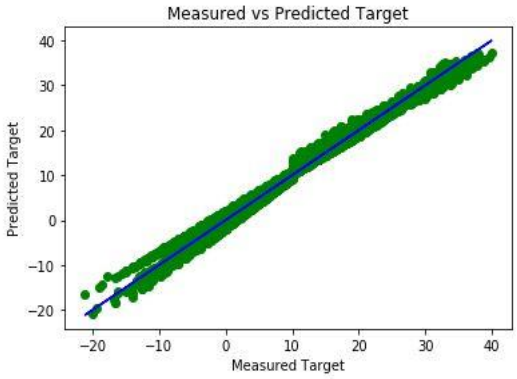
However, least squares regression requires the invertibility of XX^T. This means that the features + 1 must be greater than the data samples. Although in this case, the samples are plentiful and Least Squares can be implemented. Though a solution to this problem is Ridge Regression. This factors in the lambda penalizing parameter onto the norm of w. Thus, for Ridge regression, the variable w is determined using: $\mathbf{w = (XX^TX+\lambda I)^{-1}X^Ty}$.

K-Fold Cross Validation with 5 folds was also implemented to find the optimal parameter for minimizing average prediction error.

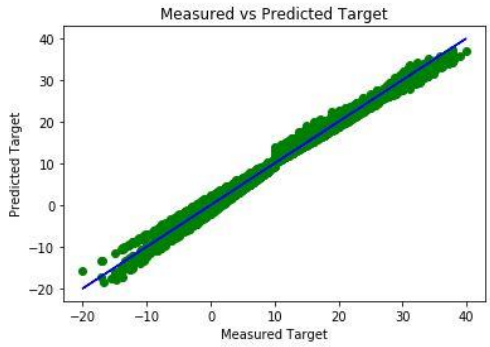
In the end, lambda = 4 was found for optimal Ridge Regression. Both Least Squares (bottom left) and Ridge Regression (bottom right) had similar error of ~90% as shown below.

Least Squares and Ridge Regression temperature plots:

```
0.8958770119825288
0.9019150108263143
0.9043085354334768
0.89913904063337
0.9053742030703918
The average prediction error from 5-fold CV:  0.9013227603892163
```




```
The average prediction error from 5-fold CV:  0.9013913970931414
From index:  3
Thus, the best lambda value: 0.3333333333333333
```

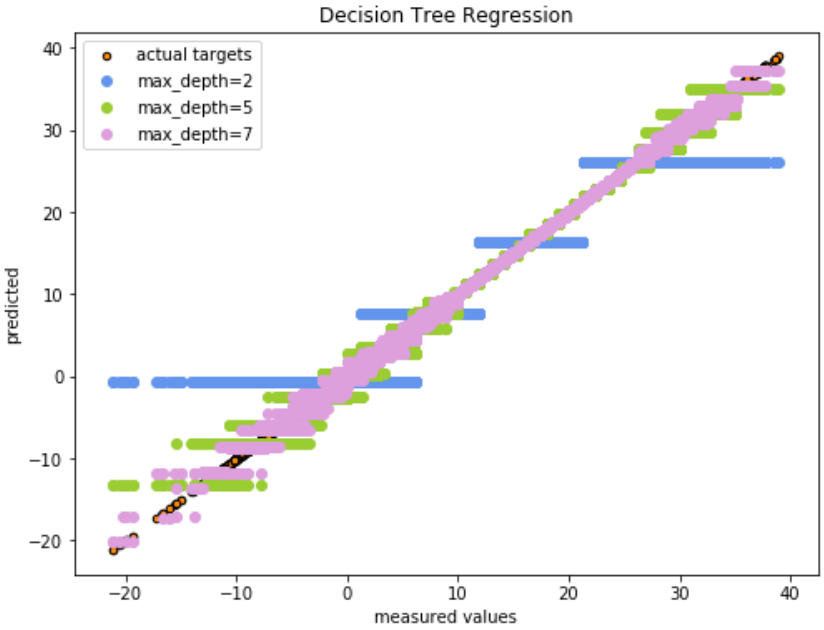


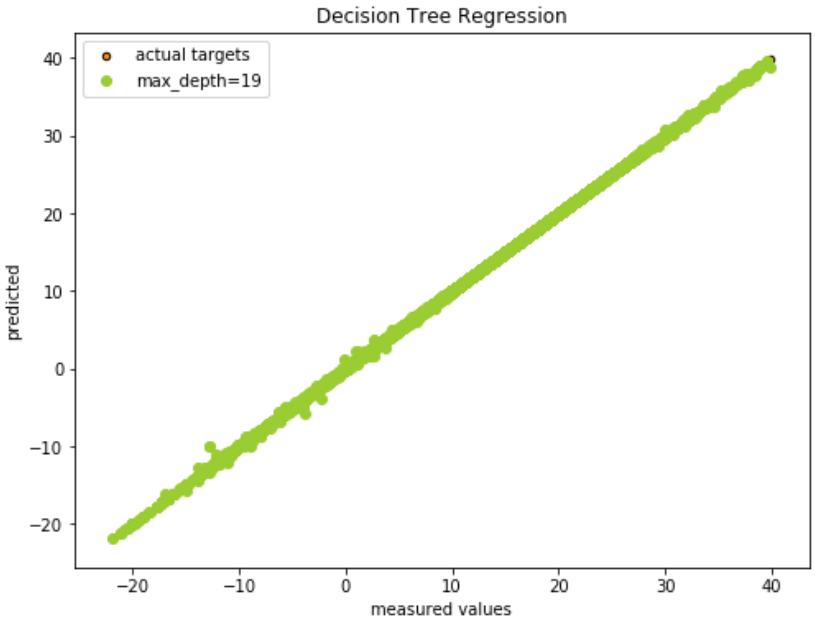
Regression Tree:

Due to the high errors in LS and RR, regression tree was implemented. Regression trees split data in a greedy fashion much like a binary tree. Each partition would result into two regions R1 and R2 based on the conditions:

$R1(j,s) = \{x: x_j<s\}$
$R2(j,s) = \{x: x_j >= s\}$ (R1's complement)
Where j = axis being split on, and s = value distance being split on

The data is continuously split until the number of samples per region are dropped below a threshold. For this implementation of sklearn's DecisionTreeRegressor, the max_depth parameter was set in place of threshold. Depths of 2, 5, and 7 were first tested (as displayed right top). Depth of 2 had an error of 100%, depth 5 had error of 72%, and depth 7 had an error of 17.5%. After using cross validation, the max depth of 19 was found to be optimal at 0.5% error (as displayed bottom right).





Discussion:

It was shown that least squares and ridge regression could be implemented for the weather data. Despite mostly following the regression line, their error rates were significantly high at 90%. Regression Trees became a much better model especially after optimizing with cross validation. This resulted in a depth length of 19 and an error rate of 0.5%.

Conclusion:

Several approaches of modeling were used to solve various prediction problems. These problems included detecting credit card fraud, determining fruits based on images, and forecasting the weather temperature for Szeged Hungary. Classification was used for credit card fraud and fruit images while regression was used to predict Szeged's weather.

For credit card fraud k-Nearest Neighbor, Logistic Regression, and Support Vector Machine was implemented. Unfortunately, the dataset was imbalanced with only 0.17% of the data classified as fraud. To solve this issue, under sampling with a subset of 50% fraud and 50% non-fraud was used. In addition, confusion matrices were used to analyze performance metrics in reducing specific error rates. In this case, Recall was prioritized as it determined the number of frauds that went undetected while Precision was second in priority as it determined the number of fake fraud alerts. After processing the three approaches, Support Vector Machine and Logistic Regression seemed to have slightly higher recall. K-Nearest Neighbor still had a relatively high recall and a much higher precision.

For fruits, SVM, Logistic Regression, and Classification tree was used. Given a dataset of 88 fruits, three very different fruits Lime, Pineapple, and Banana were extracted for processing. Support Vector Machines had the worst performance with 94% accuracy and 64% validation accuracy. Logistic Regression and Classification Trees outperformed at 100% accuracy and 99.2% validation accuracy. This was only used for 3 very different fruits, so having more fruits that overlap in boundaries may have much lower accuracy results.

In the Szeged weather dataset Least Squares, Ridge Regression, and Regression Tree was used to predict temperature. Least Squares and Ridge Regression both relatively fit the regression line but had high error rates of 90%. Regression Tree became the best model as it produced a low error of 0.5%.

Despite running into several obstacles, all three datasets were successfully processed with positive results in at least one model. The project helped understand various types of algorithms for very different datasets. The goal of using Machine Learning algorithms to solve realistic problems had been achieved and the knowledge can be applied to future data analyst projects.

Bibliography:

1. *Machine Learning Group. "Credit Card Fraud Detection." Kaggle, 23 Mar. 2018, www.kaggle.com/mlg-ulb/creditcardfraud.*

2. *"Fruits 360 dataset." Kaggle, Mihai Oltean, https://www.kaggle.com/moltean/fruits/home*

3. *"Air Quality in Madrid (2001-2018)." Kaggle, Decide Soluciones, www.kaggle.com/decide-soluciones/air-quality-madrid/home.*

4. *Norbert Budincsevity. "Weather in Szeged 2006-2016." Kaggle, Norbert Budincsevity, 8 Jan. 2017, www.kaggle.com/budincsevity/szeged-weather.*

5. *"Sklearn.model_selection.GridSearchCV." 1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation, scikit-learn.org/stable/modules/generated/sklearn.model_selection. GridSearchCV.html.*

6. *"Sklearn.svm.SVC." 1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation, scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.*