CS 229 Project Report:

Machine learning to deliver blood more reliably: The Iron Man(drone) of Rwanda.

Parikshit Deshpande (parikshd) [SU ID: 06122663] and Abhishek Akkur (abhakk01) [SU ID: 06325002]

(CS 229 Project, Autumn 2018)

This project is to predict if a flight will fail based on the historical data generated by the parts that will be used in the flight. Zipline drone consists of a battery, body and wings. Once the flight is completed it generates a lot of signals which are captured, analyzed and stored on Amazon AWS as S3 buckets.

This work deals with extracting the data from Amazon S3 which is stored as yaml files, converting data to csv with merging the same features, finding the correlation between the features and also between the feature and the output label, performing feature reduction and running supervised machine learning algorithms on the data to predict the flight failure based on previous telemetry signals reported by the parts to be used for that flight.

INTRODUCTION

Our CS229 Machine Learning project is collaborated with a company (Zipline – http://www.flyzipline.com) Some introduction about Zipline: Zipline delivers blood and supplies to remote, hard to reach hospitals and health centers via drone. Zip (drone) is a small fixed wing aircraft launching from a distribution center, flying low over a hospital and dropping a cake box with a small paper parachute. Inside the box is blood or medical supplies the hospital has ordered. Zipline currently does about 30 flights a day in Rwanda, delivering > 20% of the national blood supply.

Our project aims to help Zipline with predictive maintenance of the parts used in the flight before the flight takes off. With this effort they the attempt is to prevent failures that can be detected to achieve more delivery accuracy. Since the data includes previous flight records and analysis along with the labels this would be treated as a supervised learning problem.

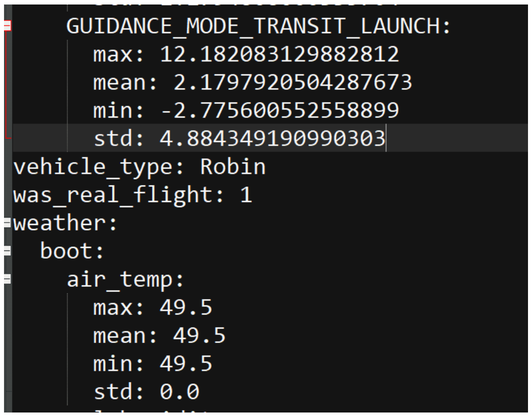
RELATED WORK

Problem of predictive maintenance has been one of the major concerns for most industries using moving parts. As discussed in [1] the ability to for see malfunctions is critical to reduce unanticipated outcomes. They also have adopted a ridge regression classifier to touch base on adopting machine learning for predictive maintenance.

[2] has a good description on the performance of SVM classification for Machine condition diagnosis. They have also provided promising result for the ability of SVM for future fault diagnosis.

DATASET AND FEATURES

Once a flight has completed its delivery, the battery is plugged in to a board which generates and collects information related to the flight. Such information can include things like power usage, weather conditions, total trip details, and other telemetry signals which are the input features for our machine learning algorithm. This data is analyzed by a set of scripts maintained by zipline developers which post the analysis as a yaml file describing its features. Below is a small snippet of a yaml file that is stored as Amazon S3 buckets.



```
    GUIDANCE_MODE_TRANSIT_LAUNCH:
      max: 12.182083129882812
      mean: 2.1797920504287673
      min: -2.775600552558899
      std: 4.884349190990303
vehicle_type: Robin
was_real_flight: 1
weather:
  boot:
    air_temp:
      max: 49.5
      mean: 49.5
      min: 49.5
      std: 0.0
```

(figa: Snippet of raw data)

The image shows a very small subset of how the flight.yaml file looks for a file. This file is generated after running analysis script on the telemetry signals generated by the battery after a flight. Each flight has an associated with flight.yaml file with 1400+ such features stored as nested objects in the yaml file.

As of today, we have data for 3199 flights, out of which 430 flights are labelled as failed flights. So, our value of m=3199. n (number of features is variable based on the flight label but is generally of the range 1200~1500.) Out of 430 flights 29 flights were flight crashes. So we had 3 labels namely: Success, Mission Failed (where flight returns back to origin) and flight fail (where flight deploys parachute).

As previously mentioned this data is stored into amazon s3 buckets along with all the raw telemetry data and other log files. On amazon s3 the data is stored as: flight_logs/DD/MM/YYY/nest_1_<flight_id>/ Our work included using the amazon s3 boto client to parse through these files and download the relevant data for our purpose. This required us to write a s3 util script to only query the data relevant to run machine learning algorithms. This data is stored in ap-south region servers in amazon, and it took substantial amount of time to query only the relevant data that we need.

One more challenge in getting the data was to fetch good enough number of failure flights. This was achieved by querying through a document storage database. We used the fields inside the yaml file to query for failure and passed cases, and then invoked our s3 util to fetch the relevant yaml files.

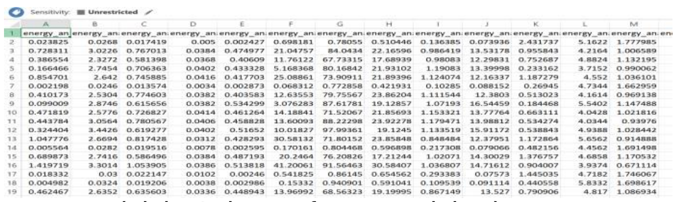
We started with ~3000 flight yaml files (so m=3000) with average of 1200 features for each flight (n=1200). We used python pandas to load the yaml file and then concatenate based on the columns. We generated a csv file mxn fields. This gave us a first real parse able input structure.

FEATURE REDUCTION & CORRELATION:

Our first work on this input included, normalizing the data. We used the standard scaler from sklearn which essentially subtracts the mean and divides by standard deviation. After doing this we spent some time in removing features that had None, or Nan values for most of the flights

Lot of the earlier flights recorded did not capture all the features. We used zero values for those missing features, in order to preserve the flight information.

Features also consisted of fields like flight_id, commit_id, description and some other string fields which we needed to clean. We wrote several utility functions to do this part. Small snippet of how data looks after the above steps

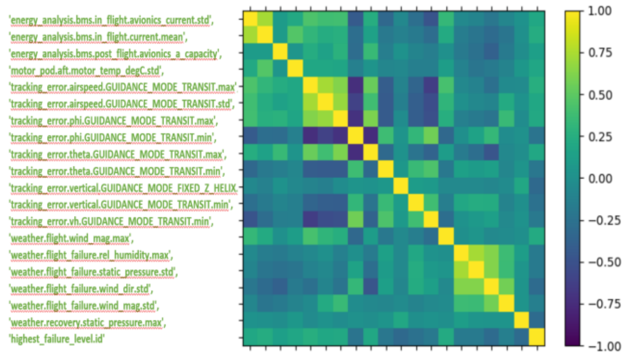

Sensitivity: Unrestricted

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	energy_an	energy_an	energy_an	energy_an	energy_an	energy_an	energy_an	energy_an	energy_an	energy_an	energy_an	energy_an	energy_an
2	0.023825	0.0268	0.017419	0.005	0.002427	0.698181	0.78055	0.510446	0.136385	0.073936	2.431737	5.1622	1.777985
3	0.728311	3.0226	0.767013	0.0384	0.474977	21.04757	84.0434	22.16596	0.986419	13.53178	0.955843	4.2164	1.006589
4	0.386554	2.3272	0.581398	0.0368	0.40609	11.76122	67.73315	17.68939	0.98083	12.29831	0.752687	4.8824	1.132195
5	0.166466	2.7454	0.706363	0.0402	0.433328	5.168368	80.16842	21.93102	1.19083	13.39998	0.233162	3.7152	0.990062
6	0.854701	2.642	0.745885	0.0416	0.417703	25.08861	73.90911	21.89396	1.124074	12.16337	1.187279	4.552	1.036101
7	0.002198	0.0246	0.013574	0.0034	0.002873	0.068312	0.772858	0.421931	0.10285	0.088152	0.26945	4.7344	1.662959
8	0.410173	2.5304	0.774603	0.0382	0.403583	12.63553	79.75567	23.86204	1.111544	12.3803	0.513023	4.1614	0.969138
9	0.099009	2.8746	0.615656	0.0382	0.534299	3.076283	87.61781	19.12857	1.07193	16.54459	0.184468	5.5402	1.147488
10	0.471819	2.5776	0.726827	0.0414	0.461264	14.18841	71.52067	21.85693	1.153321	13.77764	0.663111	4.0428	1.021816
11	0.443784	3.0564	0.780567	0.0406	0.458828	13.60093	88.22298	23.92278	1.179471	13.98812	0.534274	4.0344	0.93976
12	0.324404	3.4426	0.619277	0.0402	0.51652	10.01827	97.99361	19.1245	1.133519	15.91172	0.538843	4.9388	1.028442
13	1.047776	2.6694	0.817428	0.0312	0.428293	30.58132	71.80152	23.85848	0.848484	12.37951	1.172864	5.6562	0.914888
14	0.005564	0.0282	0.019516	0.0078	0.002595	0.170161	0.804468	0.596898	0.217308	0.079066	0.482156	4.4562	1.691498
15	0.689873	2.7416	0.586496	0.0384	0.487193	20.2464	76.20826	17.21244	1.02071	14.30029	1.376757	4.6858	1.170532
16	1.419719	3.3014	1.053905	0.0386	0.513818	41.20061	91.56463	30.58407	1.036807	14.71612	0.904007	3.9374	0.671114
17	0.018332	0.03	0.022147	0.0102	0.00246	0.541825	0.86145	0.654562	0.293383	0.07573	1.445035	4.7182	1.746067
18	0.004982	0.0324	0.019206	0.0038	0.002986	0.15332	0.940901	0.591041	0.109539	0.091114	0.440558	5.8332	1.698617
19	0.462467	2.6352	0.635603	0.0336	0.448943	13.96992	68.56323	19.19995	0.867149	13.527	0.790906	4.817	1.086934

(Figb: Snippet of extracted data)

After cleansing the data, we were left with ~700 features. Looking at the number of features and the values in most of them we decided to go ahead with feature reduction before starting with any classification algorithms. For feature reduction we generated a correlation matrix between all the features and the input features.

We used this matrix to remove the features based on the following rules. If the correlation between the features was too high, i.e. if two features were too closely correlated then we removed one of them. We also removed the features which were very highly uncorrelated with the output label. Below is the plot of the correlation matrix which we got after feature reduction.



Figc: Correlation plot for 18 features

After feature reduction using the above methods, we were left with 18 features so our initial (m, n) of (3000,1400) was reduced to (m, n) of (3000,18).

Evaluation metric that we used was to compare the predicted label (success, mission fail, flight fail) with the actual value of highest_failure_level captured by the flight analysis. The error reported is in percentage of prediction accuracy.

MODELS AND RESULTS

We chose the complete 3000 examples as our train set. Our test data were real time flight logs collected in recent time. We started with a test on 250 flights and kept adding daily flights.

Normal Equations: First set of Machine learning we applied was logistic regression using Normal equations. As per normal equations we got the parameters using:

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$

The output was calculated using sigmoid function:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}},$$

After several rounds of trial and error we selected a threshold of 0.65 to classify the flight as failure (i.e. if p(x=1) > 0.65. This gave us the following results on train and validation sets:

Data Set	Accuracy
Train	81.89%
Test	89.55%

The next algorithm we implemented was the used was the gradient descent algorithm.

Gradient Descent Algorithm: As per the gradient descent algorithm we chose to update parameters by minimizing the cost function:

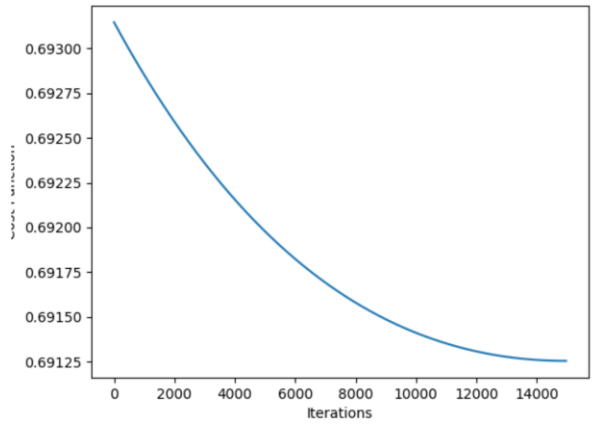
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)})).$$

We chose the following parameters for our gradient descent algorithm. We tried this algorithm with/without L1/L2 regularization

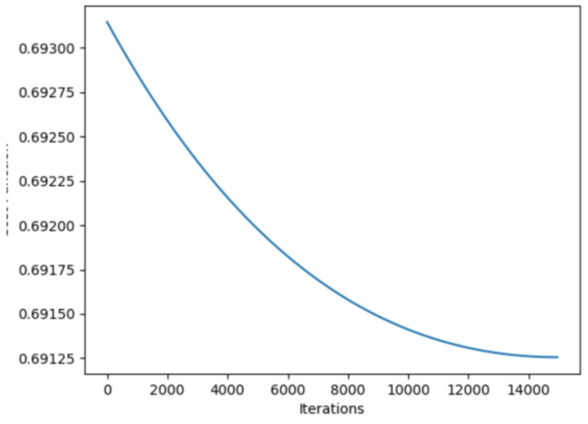
Parameter	Value
α (learning rate)	4.85e-5
λ (regularization values)	100,1

With gradient descent we achieved the following results:

Method & Dataset	Accuracy
Gradient descent with L1 reg. on train set	79.42%
Gradient descent with L1 reg. on test set	87.39%
Gradient descent with L2 reg. on train set	81.73%
Gradient descent with L2 reg. on test set	88.82%



FigD: Loss function against iterations for Gradient descent with L1 regularization.



FigE; Loss function against iterations for Gradient descent with L2 regularization.

Locally Weighted Linear Regression: As per the locally weighted linear regression algorithm we fit our parameters to minimize:

$$\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2.$$

And then used sigmoid function as mentioned above. Locally weighted linear regression gave us poor results as compared to the other two methods mentioned above. Even with varying the values of tau we used the maximum accuracy we reached with this method was **71.69%.**

Trees and Forests: Decision trees employs a top-down, greedy search through the space of possible branches with no backtracking. We have used Classification tree as the target is to classify the flight as one of the three classes, mission-success, mission-fail or flight fail.

Maximum depth of the Decision tree controls the tradeoff between error due to bias and variance, in our model we have optimized it to give the best bias/variance combination with a max_depth=5.

Ensemble method which combines several Independent Base classifiers to construct one classifier class is Random forest. Each base classifier is trained on a set sampled (sample_size) with replacement from the original training set, guaranteeing independence (Bagging or Bootstrap aggregation). Additional randomness is introduced by detecting the best split feature from a random subset of available features (feature size = sqrt(n))

Results with Decision trees and forests:

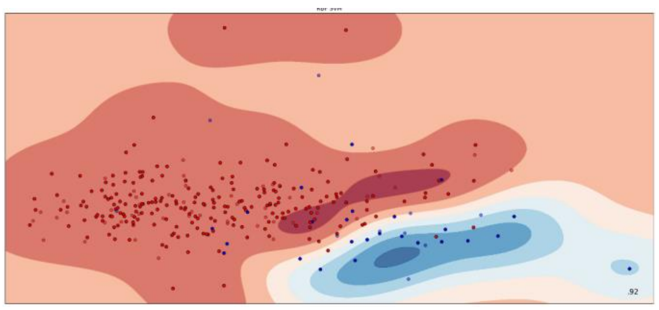
Trees & Forests	Accuracy
Decision tree on Train	91.68%
Decision tree on Test	90.12%
Random Forest on Train	86.09%
Random Forest on Test	86.45%

In order to visualize the data in 2-dimensions we decided to run PCA on it.

Principal component analysis: PCA is used to reduce the dimensions of data to two major components, before applying more sophisticated data analysis methods such as non-linear classification algorithms and decision trees.

Analysis on test and train set with two principal components was performed on Logistic Regression, LWR, Decision trees, Random Forests and SVM with Linear Kernel and Radial Basis Kernel (RBF). Logistic regression classifiers accuracy was not very good. This probably means that the decision boundary is nonlinear, hence SVM with RBF kernel was the best choice out of the above.

We trained and tested an SVM with RBF and Linear Kernels and optimize the hyperparameters (gamma). Apply PCA and see how the number of principal components influence the accuracy. Below contour shows the accuracy of SVM with RBF kernel, with two principal components as 82%.



FigF: Contour of SVM RBF after running PCA.

Support Vector Machines (SVM) Linear/RBF Kernel for classification:

The hyperparameter (gamma) obtained by running SVM on 2 principal components are used as starting point on the complete feature set SVM. If the value of gamma is too large, then the model can overfit and be prone to low bias/high variance.

As, gamma of Linear/RBF kernel controls the tradeoff between error due to bias and variance, in our model we have optimized it to give the best bias/variance combination.

$$f(\mathbf{x}) = \sum_{i=1}^{N_S} \alpha_i y_i \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x}) + b = \sum_{i=1}^{N_S} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}) + b$$

Linear Kernel:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2$$

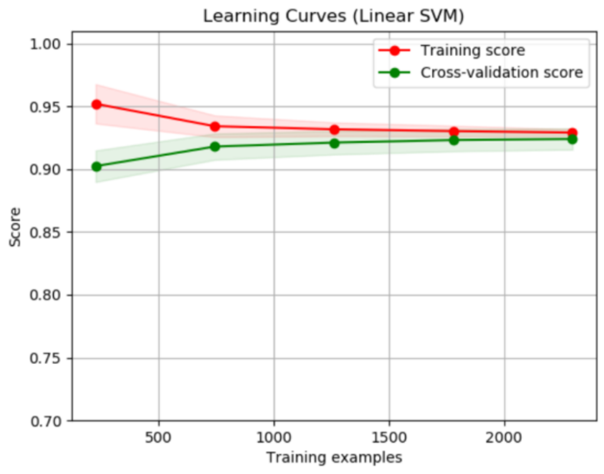
RBF Kernel:

$$k(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$$

Below are the results with Linear kernel SVM

SVM Linear Kernel	Accuracy
Train	86.09%
Test	86.45%

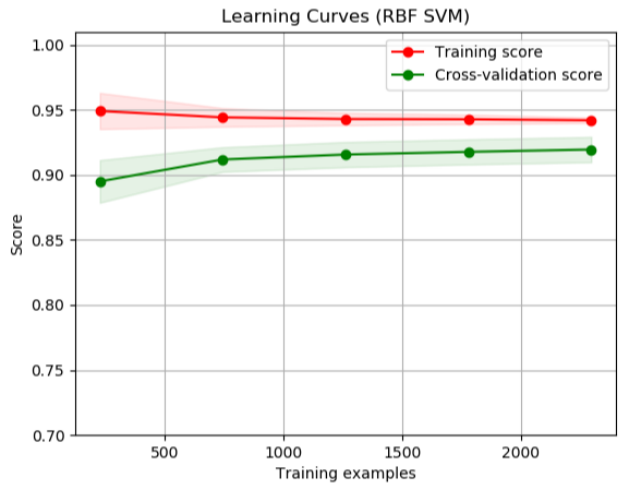
We got the following Learning curve with Linear SVM Kernel:



FigG: Learning curve on train and test with SVM Linear Kernel

Below are the results with RBF Kernel SVM

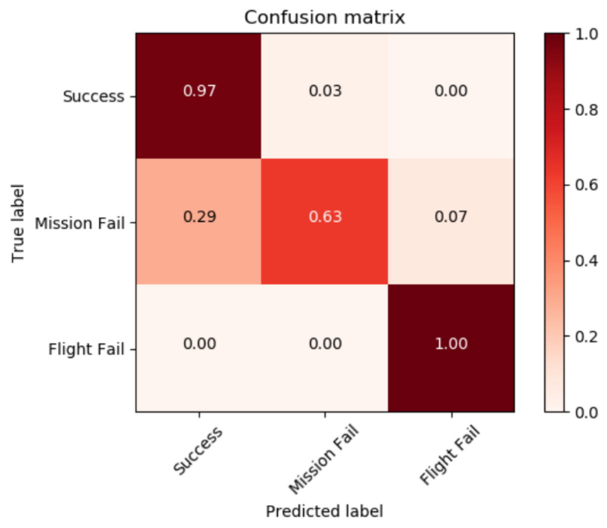
SVM RBF Kernel	Accuracy
Train	93.04%
Test	92.18%



FigH: Learning curve on train and test with SVM RBF Kernel

As you can see in both the learning curves, the distance between the training and cross validation scores narrows down with the number of examples.

We achieved good results with SVM on RBF Kernel. Below is the confusion matrix for the same:



FigI: Confusion matrix for results of SVM with RBF Kernel

Our model got 100% accuracy in predicting flight failures, 97% accuracy in predicting success, and 63% accuracy in predicting mission failure cases. Missing features in the earlier flight, makes some of the mission failure cases to be categorized as success. What this model tells us is that if the flight is categorized as mission failure or flight failure, with the previous run being successful, then it might need maintenance.

Our model will be serialized and used in the analysis scripts of Zipline. The predicted probabilities of each label will be added along with the flight data and used for predictive maintenance.

CONCLUSION & FUTURE WORK

This sums up the work we have carried out till now. To summarize, first we spent quite a lot of effort to extract the data from Amazon, clean it, perform feature reduction. After doing that we ran the above three algorithms that we have listed and got some results. From the results the method of applying SVM with RBF Kernel is getting us good results.

As a future work we can implement RUL – Remaining Useful life to predict the number of flights left in all the parts. This will help plan the maintenance cycles in a better way and increase the confidence with which a flight takes off. Another approach can be to have an unsupervised model to detect anomalies in the telemetry numbers reported.

TEAM CONTRIBUTION BREAKDOWN

Both the team members contributed equally for the project till now. Below are the distributions of tasks:

Parikshit Deshpande: Work on Data gathering, cleaning, feature reduction initial models & report compilation
Abhishek Akkur: Work on feature reduction, PCA, Trees and Forests, SVM linear and RBF, Plot generation & report compilation.
Pair programming: To debug and solve issues and poster preparation.

REFERENCES AND GITHUB LINK:

[1]: Susto, Gian Antonio, et al. "Machine learning for predictive maintenance: A multiple classifier approach."
[2]: Widodo, Achmad, and Bo-Suk Yang. "Support vector machine in machine condition monitoring and fault diagnosis."
https://scikit-learn.org/stable/

GitHub Repo for code:
https://github.com/parikshd/cs229-zipline

Contact info Zipline:
Emma Schott (emma@flyzipline.com)
Matt Fay (matt@flyzipline.com)