Classification of Objects in 3D Point Cloud Data

Stephan Roth

[sroth44489@gmail.com](mailto:sroth44489@gmail.com)

March 8, 2017

# Definition

## Project Overview

Images of things are no longer restricted to 2D pictures. Three dimensional images are becoming more available because of low cost commercial and industrial sensors. Sensors like the Microsoft Kinect allow anyone to create a 3D image of anything [Figure 1].

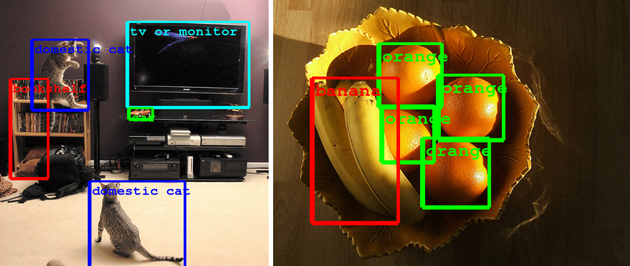


Figure 1: (left)3D Point cloud collected using a Microsoft Kinect (ref. <http://123kinect.com/wp-content/uploads/2012/08/Kinect-1-point-cloud-example.jpg>) (right) Google’s object recognition algorithm (ref. https://www.engadget.com/2014/09/08/google-details-object-recognition-tech/)

Just like with 2D images, it is now possible to do object recognition on 3D structures. Google, Microsoft and other tech companies have long had the ability to take a picture of a scene and identify the objects it contains [Figure 1]. It is now possible to do the same thing with these new 3D images. This has applications to things like gesture recognition for next generation user interfaces or industrial applications such as identifying structures from 3D survey data.

One application of 3D object recognition is the automatic identification of structures in a building survey. Surveys are no longer limited to measuring distances to single points on a job site. It is now possible to capture highly accurate 3D structural data [Figure 2]. The data collected during one of these surveys is what is termed a point cloud. A structure is represented as a dense collection of points in (x,y,z) possibly with brightness and color information. However, just as with images, there is no label attached to each point. The point cloud by itself does not identify walls, chairs, roofs, etc. It would be very useful to have these objects automatically identified within a point cloud. That is the goal of this project, to train a classifier to recognize a 3-dimensional object within a point cloud.

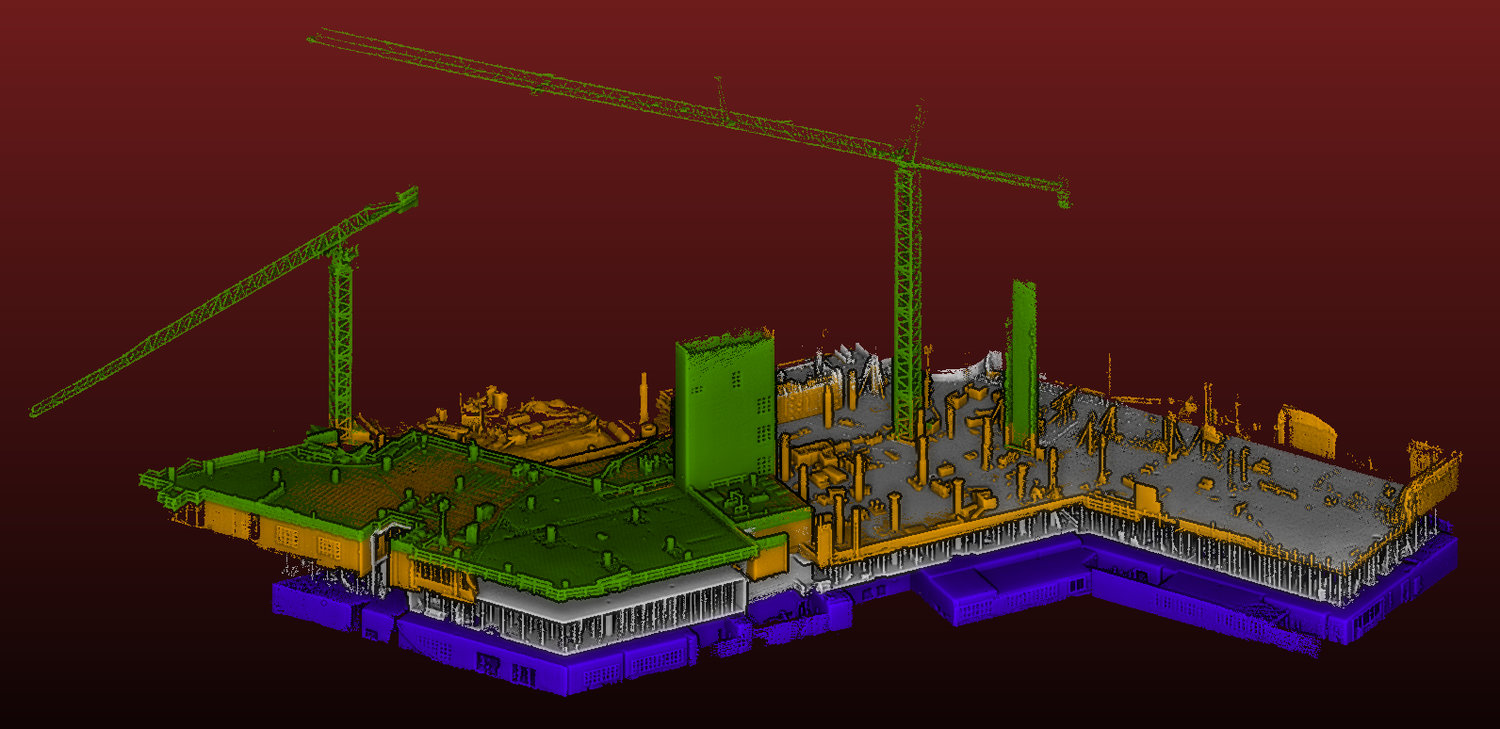


Figure 2: 3D Survey of a building under construction. The data is a 3 dimensional point cloud. In a point cloud, the building is represented by a dense cloud of (x,y,z) points, possibly with color or brightness information (Ref. kaarta.com/gallery)

## Problem Statement

The purpose of this project is to create a supervised learning classifier to identify and locate an object within a 3D point cloud. Because this project uses supervised learning, a hand labeled data set must be created. Each point in the hand labeled point cloud shall be classified as either belonging to the target object or not.

This project evaluated an array of supervised learning algorithms, including: Support Vector Classifier, Ada Boost, K Nearest Neighbors and Gaussian Naïve Bayes. These algorithms represent a wide array of learning techniques from simpler linear models to models with arbitrarily complex decision boundaries.

To improve the efficiency or predictive ability of the learning algorithm, it is sometimes possible to transform the data. Reducing the dimensionality of the feature vector can improve efficiency and other transforms, to give the data a more gaussian distribution, can improve predictive performance. One can reduce the dimensionality of the feature vector using PCA. PCA can help mitigate the curse of dimensionality. All of the training data is squeezed into a denser, lower dimension space. However, reducing the dimensionality can also reduce the discriminating power of the classifier, so its positive or negative impact must be measured. Some classifiers assume that he data is normally distributed with zero mean and standard deviation of 1. This can give all features equal weight. For instance, if the mean value of 1 feature is 10x larger than another, its importance may seem to be 10x as great. This could force the classifier to consider that feature to have more importance than another even when it does not. Therefore, some classifier performance is improved if the data is transformed to a normal distribution e.g. Again, the positive or negative impact of such a transform needs to be measured.

The overall process used is the following:

1. Create point clouds of a prototypical scene
2. Hand label a training set
3. Define a set of features to train on. The features will describe the point cloud geometry local to each point in the cloud
4. Investigate the data to ensure the defined features have the discriminating power necessary to separate the target object from others
5. Using cross validation, split the data in to a test and train set
6. Evaluate a variety of supervised learning algorithms
7. Investigate the effects of feature vector transforms such as PCA and normalizing the data
8. Select the best performing algorithm, ranking them according to their F1 score
9. Tune the selected algorithm using grid search to optimize its parameters
10. Test the optimized algorithm on other point clouds to see if it can detect the object in other untrained data

## Metrics

To measure the performance of the classifier, the F1 score is used. The classifier we are going to create is binary. A set of features either represents the target object or it does not. The F1 score measures the accuracy of a test with binary results. With a test, there are four possible outcomes: true positive, false positive, true negative and false negative. Given the rate of true positives, true negatives, etc. the precision, recall and F1 score of a test are defined as:

Figure 3: Possible test results are True positive, false positive, false negative and true negative

True Positive

Test Result

False Negative

True Result

True

False

False Positive

True Negative

True

False

The F1 score has a maximum value of 1 if the test results match the true results and a minimum value of 0 if the test is always wrong.

# Analysis

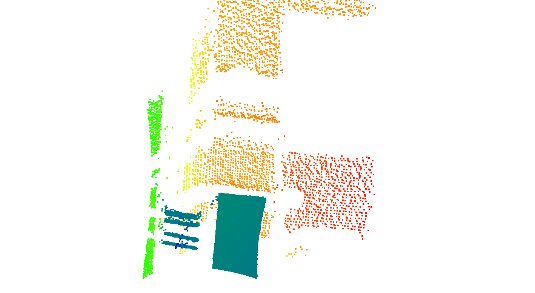
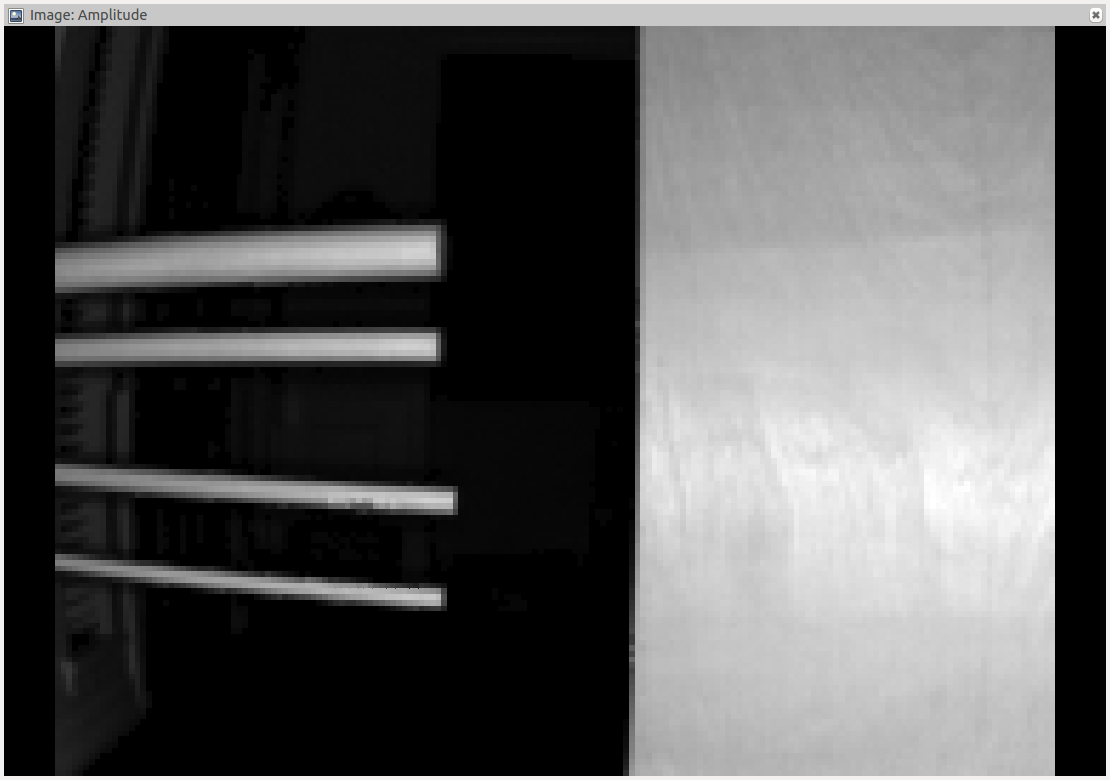
*(approx. 2-4 pages)*

## Data Exploration

The purpose of this project is to create a classifier which can identify pipe-like structures within a 3D point cloud. Towards this end, I created a prototypical scene with a series of pipes, surrounded by walls and other objects [Figure 4]. The 3D scene was imaged by an IFM 03D303 flash lidar from several different angles. The flash lidar generates a 3D point cloud of the scene. Each pixel in the 3D point cloud has a value (x ,y, z, intensity) [Figure 5]. This raw data cannot be used by the classifier because a single point does not contain enough information to describe what it is a point of. Instead, the features describing a point must come from a region of the point cloud. Given the size of object being classified and the resolution of the flash lidar, a 6cm cube is an appropriate volume. However, because points within the point cloud are not evenly distributed, the size of the cube must be adapted especially in areas of low point density (Zakhor, 2011). In the case where insufficient points are located in a 6cm cube, the size of the cube is increased until a minimum number of points (15) are included. This is to ensure that a sufficient number of points are used to generate accurate statistics of the volume.

Figure 4: 3D Scene with pipes, walls and other objects

Figure 5: Flash Lidar Output. (a) Intensity image showing pipes and a planar surface, (b), (c) point cloud as viewed from different angles. Pipes and planar surface are in green. Points colored by distance from the sensor



(a)

(b)

(c)

Once a representative group of points is gathered, features describing the local geometry are calculated. The features used are common in analysis of point clouds. They use the eigen values and vectors of the covariance matrix in the region around a point. Given the eigen values of the covariance matrix, the geometric features are, . These geometric features represent point-ness, surface-ness and linear-ness of the region. In addition, the algorithm contains directional features using the local tangent and normal vectors. The tangent and normal vectors are estimated using the eigen vectors of the largest and smallest eigen values. The sine and cosine of these vectors with respect to the horizontal plane are used, giving a total of 4 directional features. To estimate the confidence in these features, the features are scaled according to the strengths of their corresponding eigen values:. The complete feature vector concatenates the 3 geometric features and 4 directional features for a resulting 7D feature vector.

To train the classifier, a dataset was collected and segmented by hand. The points corresponding to the pipes were identified and labeled as class 1. The points corresponding to everything else are labeled as class 0. For the training set, there are 14,794 points in class 0 and 1,427 points in class 1. The statistics for both classes are shown in the following table:

Table 1: Class 0 (not pipe) feature statistics

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Pointness (x10-4)** | **Surfaceness (x10-4)** | **Linearness (x10-4)** | **Cos tangent** | **Sin tangent** | **Cos normal** | **Sin normal** |
| **Mean** | 1.09 | 6.14 | 4.71 | 0.222 | 0.15 | 0.865 | 0.011 |
| **Std** | 3.02 | 13.9 | 28.1 | 0.303 | 0.34 | 0.248 | 0.200 |
| **Min** | 0.00368 | 0.14 | 0.00843 | 0.0004 | -0.99 | 0.00186 | -0.999 |
| **25%** | 0.0772 | 4.43 | 0.392 | 0.0392 | 0.01 | 0.888 | -0.047 |
| **50%** | 0.0938 | 4.96 | 0.737 | 0.0837 | 0.05 | 0.993 | 0.064 |
| **75%** | 0.591 | 5.81 | 4.29 | 0.217 | 0.15 | 0.997 | 0.112 |
| **Max** | 200 | 1207 | 1641 | 1.00 | 0.99 | 1.00 | 0.989 |

Table 2: Class 1 (pipe) feature statistics

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Pointness (x10-4)** | **Surfaceness (x10-4)** | **Linearness (x10-4)** | **Cos tangent** | **Sin tangent** | **Cos normal** | **Sin normal** |
| **Mean** | 1.40 | 1.93 | 7.20 | 0.894 | 0.011 | 0.299 | -0.000697 |
| **Std** | 0.600 | 2.07 | 3.29 | 0.255 | 0.153 | 0.314 | 0.102 |
| **Min** | 0.270 | 0.13 | 0.24 | 0.00576 | -0.930 | 0.0114 | -0.859 |
| **25%** | 0.600 | 0.64 | 4.64 | 0.998 | -0.0284 | 0.0720 | -0.00349 |
| **50%** | 0.940 | 1.23 | 7.39 | 0.999 | 0.00119 | 0.189 | 0.00188 |
| **75%** | 1.37 | 2.01 | 10.4 | 0.999 | 0.0309 | 0.357 | 0.0117 |
| **Max** | 4.16 | 10.02 | 12.5 | 1.00 | 0.980 | 1.00 | 0.578 |

Table 3: Class 0 (not object of interest) Feature Statistics.

Comparing the data between classes, class 1 does seem to occupy a distinct region. In particular, because the pipes are very linear in shape, one would expect the “Linearness” feature to stand out, and it does. For class 0, the mean is 4.71 and std is 28.1. For class 1, the mean is 7.20 and std is 3.29. For class 1, the mean linearness is larger and standard deviation is smaller This indicates that the class 1 linearness forms a tight group higher in value than the class 0 linearness values.

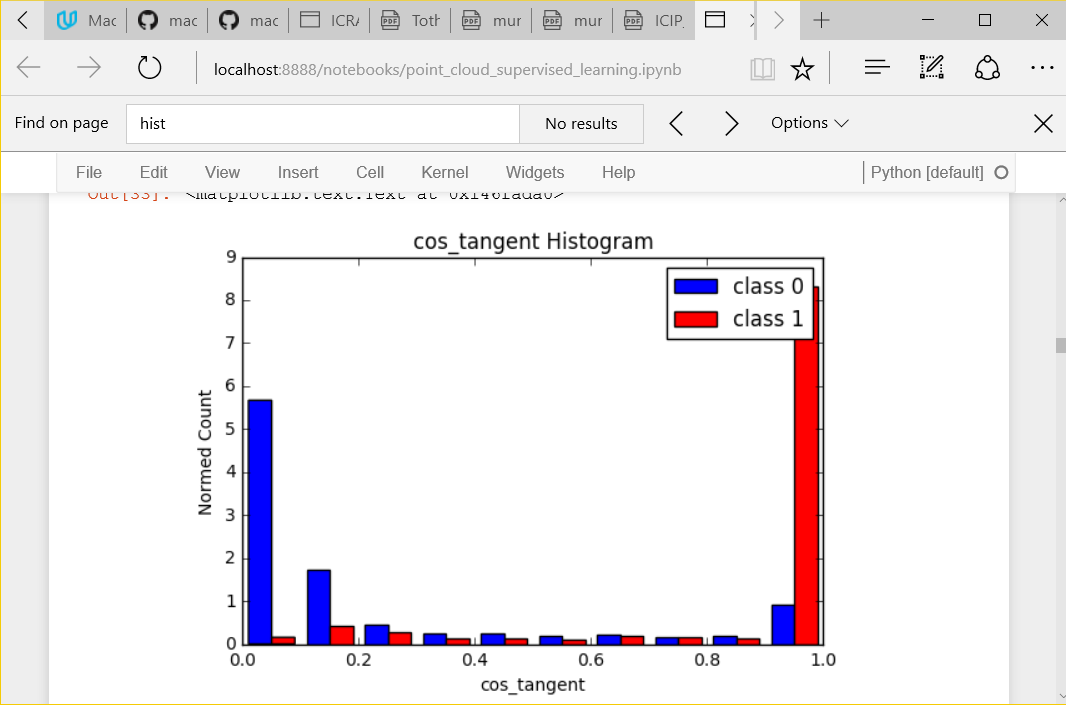
The tangent data should also clearly identify the pipe objects because they were oriented horizontally. This shows up in the statistics. For a horizontal line, the cosine of the tanget should be approximately 1 and the sine approximately 0. This holds true in the data. The cosine for class 1 has a mean of 0.894 and std 0.255, indicating horizontall. The sine data for class 1 has mean 0.011 and std 0.153, indicating horizontal. Both of these values are tightly grouped and distinct when compared to the class 0 data. The cosine of the tanget for class 0 has mean 0.222 and std 0.303. This data tends to be lower in value than the class 1 data. The sine of the tangent for class 0 has mean 0.15 and std 0.34. This data tends to be higher in value and more spread out than the class 1 data.

The distinctiveness of the class 1 data holds across the other features as well. For pointness, class 0 75% < class 1 25%. For surfaceness, class 1 75% < class 0 25%. For cos normal, class 1 75% < class 0 25%.

## Exploratory Visualization

To verify that the class 0 and 1 objects form distinct clusters, histograms were made of the linearness and cosine tangent features. The results are shown in [Figure 6] and [Figure 7]. As expected, the class 0 and 1 distributions are distinct and agree with the data from the previous tables. For the cosine tangent feature, the class 1 data is clustered primarily around a value of 1.0, indicating horizontalness. The class 0 data is primarily clustered around 0. This is probably due to the fact that most of the other point cloud data is from vertical walls. For the linearness feature, the class 1 values tend to be larger than the class 0 values. This is as expected because the pipes are very linear in appearance. Again, the class 0 points will be primarily from planar surfaces, so the linearness feature should be lower.

Figure 6: Histogram of the cosine tangent feature for class 0 and 1.



In addition 2 dimensional scatter plots were created for different features [Figure 8]. In the plot of surfaceness vs linearness, the class 1 points are clustered in the upper left hand corner, indicating high linearness and low surfaceness. The class 0 points are less linear and more surface-like. In the plot of linearness vs cos\_tangent, the class 1 points are mainly to the right indicating horizontal linearity. The class 0 points are primarily on the left. Again, the class 1 objects are clustered together and are somewhat distinct from the class 0 objects.

Figure 7: Histogram of linearness feature for class 0 and 1.

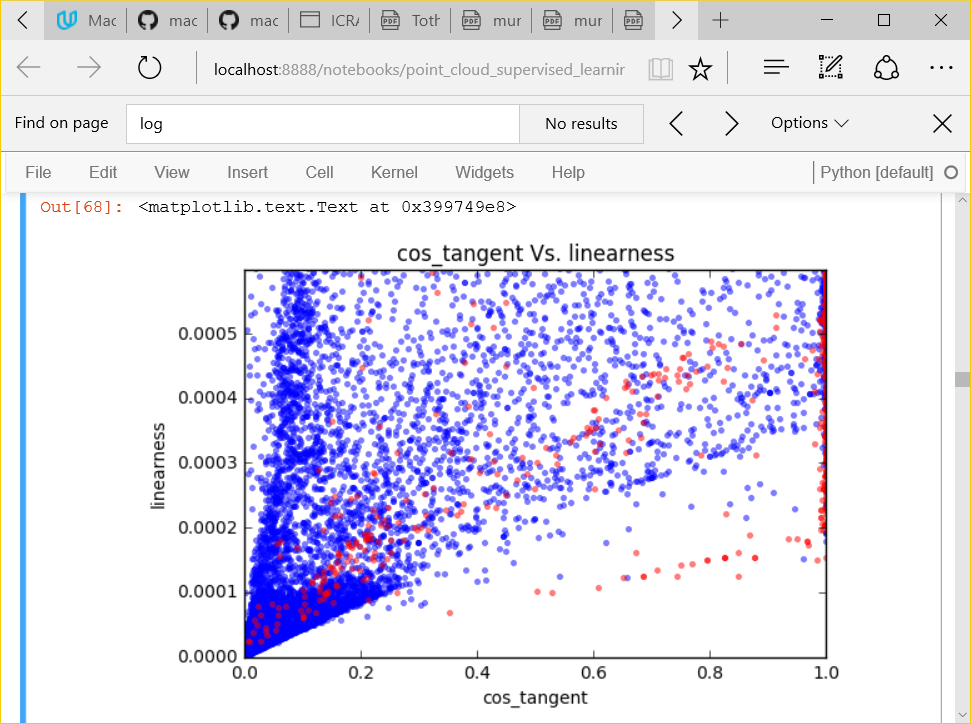
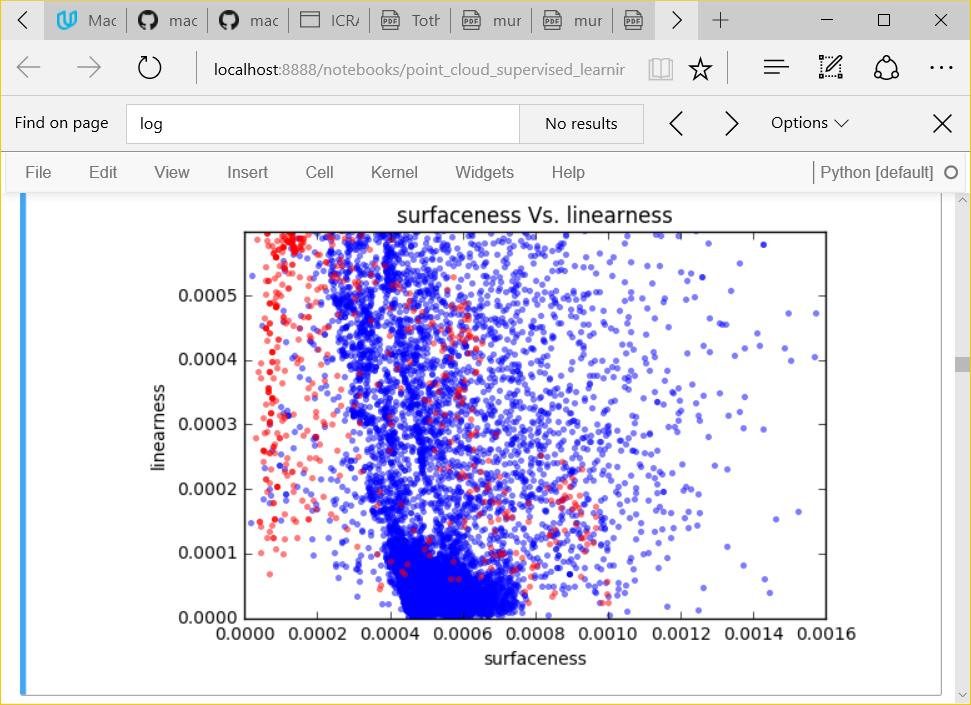
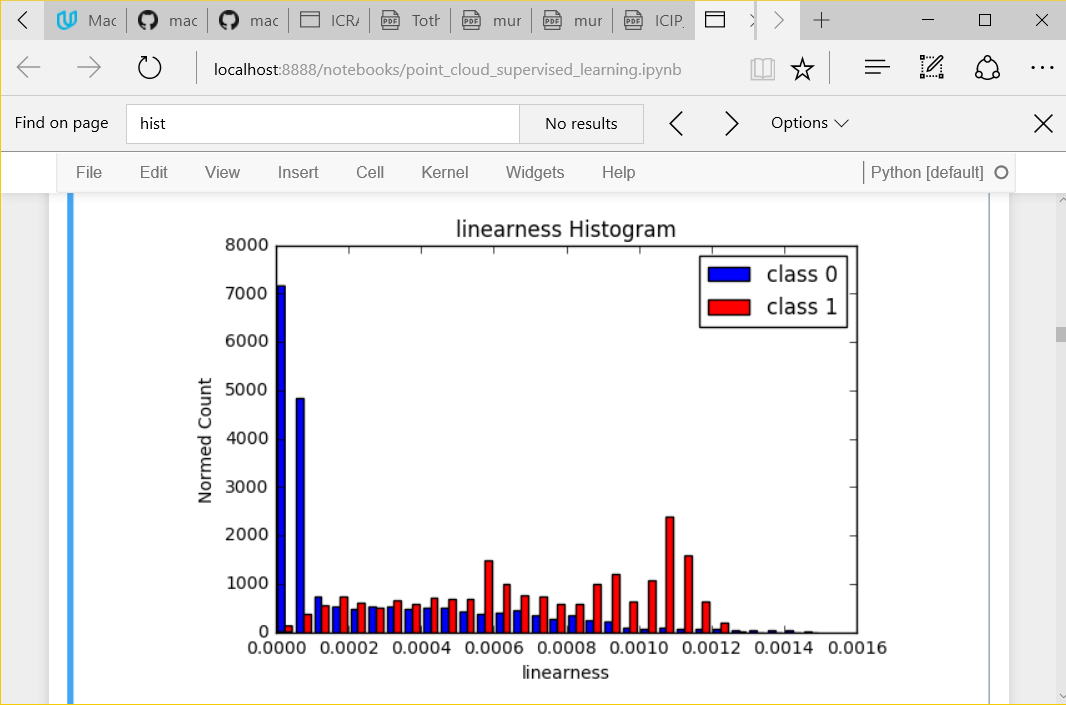


Figure 8: Scatter plots showing the class 0 (blue) and class 1 (red) feature distributions. The two classes mostly fall into two distinct clusters

## Algorithms and Techniques

In the sciKit Learn documentation (scikit-learn, 2016), in choosing the right estimator, it is suggested for problems with < 100K samples to use the linear SVC and if this fails to move onto K Nearest neighbors and Ensemble classifiers. In addition, after analyzing the data, there was no obvious simple boundary separating the different classes, so a classifier that could handle complex decision boundaries may perform better. For this project, I evaluated the following supervised learning algorithms: Support Vector Classifier, Ada Boost, K Nearest Neighbors and Gaussian Naïve Bayes.

Looking at the data we are using for training, there are 14,794 points in class 0 and 1,427 points in class 1. The feature vector used has 7 dimensions. With only 15,000 training points, the 7 dimensional feature space will be sparsely populated. For each classifier, a training subset of calculated features and hand labels will be fit. To evaluate each classifier, a subset of test data features will be used to predict a class ID. The predicted class ID will be compared to the hand labeled class ID and scored according to its F1 score.

The first classifier evaluated was the Support Vector Classifier. Support vector classifiers have been used in analyzing the solvency of companies by banks / lenders (Laura Auria, 2008). Balance sheet data from the companies is used as features and the classifier is used to predict the insolvency of a business within 3 years. The strength of the support vector classifier lies in its ability to use kernels to create non-linear boundaries between classes.

The second classifier evaluated was AdaBoost. In industry, AdaBoost has been used in predicting energy consumption in the steel industry (Rui Hu, 2013). The authors are trying to predict the energy consumption in the steel industry. Because energy consumption in the steel industry is tied to current global demand for steel but also government incentives for steel production, traditional modeling methods fail. The authors use a neural network based Ada Boost algorithm.

The strength of the AdaBoost classifier is that it is one of the best "out of the box" classifiers, and tends to be less susceptible to over fitting. It is also computationally efficient. A disadvantage of AdaBoost is that it can be susceptible to outliers. The algorithm can spend too much time trying to fit a data point that is really just noise.

AdaBoost is a good candidate for this problem due to its versatility and computational efficiency.

K nearest neighbors has been used in fault detection in industrial plants (Fellipe do Prado Arruda e. a., 2014). The strength of the K nearest neighbor is that the training time is efficient. The algorithm simply needs to store all of the training data. The algorithm can also learn arbitrarily complex classification boundaries. However, a weakness is the classification time is costly because the algorithm needs to find the k nearest neighbors by sifting through the data. Also, because all of the training data needs to be stored, the memory requirements increase as the size of the learning set increases.

K nearest neighbors is a good candidate for this problem because of its ability to handle arbitrarily complex classification boundaries.

Naive Bayes classifier has been used in text classification (Andrew McCallum, 1998). Here, an algorithm is trying to classify a document based on an analysis of the text. For example, it may classify a document as being about tennis, physics, sports, etc. The naive bayes classifier is trained using a dictionary of words found to be highly discriminative of these categories.

The strength of the Naive Bayes classifier is that it can is very fast to learn and classify data. It has also been shown to work on a variety of machine learning problems. The speed at which it can learn and classify data comes from the assumption that probabilities are conditionally independent within the model. This is both a strength and weakness. For classification problems where things are conditionally dependent, the Naive Bayes algorithm will fail to correctly classify items. For example, a Naive Bayes classifier cannot learn XOR(x1, x2). Naive Bayes is a good candidate for this problem due to its computational efficiency.

## Benchmark

To compare the performance of the classifier, it was benchmarked with a simple logistic regression model. Logistic regression tries to fit a linear model to the classified data. With logistic regression, the probability of the classification outcome is modeled using a sigmoid function of the input features. It is a simple model with fast evaluation time.

As mentioned before, the benchmark was scored according to its F1 score on the testing data. The benchmark was fit using a training subset of the hand labeled data. The benchmark was then scored according to its F1 score on its predictions on the hand labeled test data. The training and testing data are independent subsets of the hand labeled data.

Table 4: Benchmark Score

|  |  |  |
| --- | --- | --- |
| **Classifier** | **Training F1 Score** | **Testing F1 Score** |
| LogisticRegression | 0.751 | 0.731 |

# Methodology

*(approx. 3-5 pages)*

## Data Preprocessing

No preprocessing of the data was necessary. All features of the model are numeric, so no text labels or other types of data needed to be transformed into integer values, etc. Also, the noise in the data is due to the sensor. There were no obvious outliers that needed to be removed from the training data.

I did attempt to transform the data using PCA and log transforms. Using PCA, lower dimensionality data adversely impacted the classifier’s performance, so was abandoned. Similarly, using a log transform or scaling the data to 0 mean and 1 standard deviation to achieve a more gaussian distribution of data reduced the classifier’s performance. It improved SVC’s performance, but in the end its performance was less than AdaBoost, the classifier used in the end. None of these data transforms were used in the final algorithm.

## Implementation

### Jupyter Notebook

The bulk of the work in this project occurs in the Jupyter Notebook point\_cloud\_supervised\_learning.ipynb. This notebook is used to load the point cloud data, calculate the features, transform the data, train the classifier and display all the results [Figure 9]. The workflow is as follows:

Figure 9: Workflow

Load the Training Data

Calculate Feature Statistics for all Classes

Visualize Data for all Classes, Looking for Patterns

Try Data Transforms (PCA and Scaling)

Benchmark Classifier

Try Different Classifiers

Tune the Best Classifier

Display the Trained Results

Split Training and Testing Data

The notebook first loads the training data. This involves loading the point cloud data and the hand segmented data for each file in the training set. Each point in the point cloud is given a class ID label. If the point cloud point is in the hand segmented data, it is given a class ID of 1. If not, it is given a class ID of 0. Additionally, for each point in the point cloud, a feature vector is calculated. The feature vector and class ID together form the training data to be sent to the classifier.

Next, for each class ID, the notebook calculates and displays the statistics. This gives an idea of the distribution of features between the two classes.

A better way to look for patterns in the data is to plot some subsets of the data. The notebook plots histograms of the linearness, surfaceness, etc. features for the two classes. In this way, you begin to see that the two classes create two distinct distributions of the features.

The next step in the process is to try different data transformations. These include PCA for dimensionality reduction and scaling to normalize the data.

Next, data is split into training and testing data, and the different classifiers are tried. First, the benchmark classifier, LogisticRegression is trained and optimized. Then, an array of other algorithms are tried including AdaBoostClassifier, KneighborsClassifier, SVC and GaussianNB. The classifier with the best performance, AdaBoostClassifier, is then tuned to optimize its performance.

Finally, the results of the AdaBoostClassifier are shown. The classified point cloud is plotted with the different class points shown in different colors. This shows that the performance of the classifier is pretty reasonable.

### Point Cloud Classification

In processPly.py, I have implemented files for loading training and testing data, and calculating feature vectors.

The most important function, calculate\_features, takes a point cloud and calculates the feature vector associated with each point in the point cloud. The feature of a point are defined using the eigen values and eigen vectors of the covariance matrix of all points within a 6cm cube. To find these neighborhood points, a kdtree is used. The kdtree provides a fast way to look up points within a neighborhood in a k dimensional space. Once the neighborhood points are found, the eigen vectors and eigen values of the covariance matrix are used to calculate the point cloud features.

def calculate\_features(point\_cloud\_points, point\_cloud\_intensities, kdtree, pca\_min\_radius=0.06):

"""

This function calculates the geometric and directional features of a point cloud.

This function calculates the geometric and directional features of a point cloud.

Each pixel in the 3D point cloud has a value (x ,y, z, intensity).

This raw data cannot be used by the classifier because a single point does not contain

enough information to describe what it is a point of.

Instead, the features describing a point must come from a region of the point cloud.

Given the size of object being classified and the resolution of the flash lidar,

a 6cm cube is an appropriate volume. However, because points within the point cloud

are not evenly distributed, the size of the cube must be adapted especially in areas

of low point density (Zakhor, 2011). In the case where insufficient points are located

in a 6cm cube, the size of the cube is increased until a minimum number of points (15)

are included. This is to ensure that a sufficient number of points are used to generate

accurate statistics of the volume.

Once a representative group of points is gathered, features describing the local geometry

are calculated. The features used are common in analysis of point clouds. They use the eigen

values and vectors of the covariance matrix in the region around a point. Given the eigen

values w\_3<w\_2<w\_1 of the covariance matrix, the geometric features are

{pointness = w\_1, surfaceness = w\_2-2\_1, linearness = w\_3-w\_2} These geometric features

represent point-ness, surface-ness and linear-ness of the region. In addition, the algorithm

contains directional features using the local tangent and normal vectors. The tangent and

normal vectors are estimated using the eigen vectors of the largest and smallest eigen

values. The sine and cosine of these vectors {v\_t,v\_n} with respect to the horizontal

plane are used, giving a total of 4 directional features. To estimate the confidence

in these features, the features are scaled according to the strengths of their

corresponding eigen values:

scale{v\_t,v\_n}={linearness,surfaceness }/(max(linearness,pointness,surfaceness)).

The complete feature vector concatenates the 3 geometric features and 4 directional

features for a resulting 7D feature vector.

Parameters

-------------------

point\_cloud\_points: np.array

2D array of point cloud points. Each row contains an (x,y,z) point

point\_cloud\_intensities: np.array

1D array of point cloud intensities.

kdtree: spatial.cKDTree

kd tree used to find points in a region

pca\_min\_radius: float

the radius of the neighborhood to use when calculating geometric and

directional features

Return Values

-------------------

features: pandas.DataFrame

The first 4 columns are the x,y,z,intensity data from the point cloud.

The remaining columns are the geometric and directional features for that point.

"""

# To prevent areas of great density from bogging down the algorithm, set a max

# number of points to use in pca.

pca\_max\_points = 200

# To prevent areas of low density from impacting statistical significance of pca,

# put a floor on the number of points used in pca.

pca\_min\_points = 15

# Define the names of the features found

feature\_names = ["x", "y", "z", "intensity", "pointness", "surfaceness", "linearness",

"cos\_tangent", "sin\_tangent", "cos\_normal", "sin\_normal"]

# Features is a 2D matrix. Each row is the features of a point in the point cloud

features = np.empty([point\_cloud\_points.shape[0], len(feature\_names)])

# For each point in the point cloud, calculate a feature

feature\_index = 0

for query\_point in point\_cloud\_points:

points\_distance, points\_index = kdtree.query(query\_point, k=pca\_max\_points,

distance\_upper\_bound=pca\_min\_radius, p=np.inf)

# if we don't get enough points to do pca, expand the radius to

# ensure you get a minimum number of points

num\_points = np.count\_nonzero(np.isfinite(points\_distance))

if num\_points < pca\_min\_points:

points\_distance, points\_index = kdtree.query(query\_point, k=pca\_min\_points)

# Get a list of the points (with dist < inf)

points = [point\_cloud\_points[index]

for dist, index in zip(points\_distance, points\_index) if np.isfinite(dist)]

points = np.transpose(np.array(points))

# points is now a matrix with columns of (x,y,z) triplets

# Find the covariance of the points

cov = np.cov(points)

# Find the eigen values(ascending), vectors of the covariance matrix

w, v = np.linalg.eigh(cov)

# Spectral features from Munoz icra 2009

pointness = w[0]

surfaceness = w[1] - w[0]

linearness = w[2] - w[1]

max\_spectral\_feature = max([pointness, surfaceness, linearness])

# Directional Features from Munoz icra 2009

tangent = v[2]

normal = v[0]

# Find the sine and cosine of the tangent line w.r.t. horizontal (x,y) plane

adjacent = np.sqrt(tangent[0]\*\*2 + tangent[1]\*\*2)

opposite = tangent[2]

hypotenuse = np.sqrt(adjacent\*\*2 + opposite\*\*2)

cos\_tangent = adjacent / hypotenuse

sin\_tangent = opposite / hypotenuse

# scale the values based on strength of the extracted directions

scale = linearness / max\_spectral\_feature

cos\_tangent \*= scale

sin\_tangent \*= scale

# Find the sine and cosine of the normal line w.r.t. horizontal (x,y) plane

adjacent = np.sqrt(normal[0]\*\*2 + normal[1]\*\*2)

opposite = normal[2]

hypotenuse = np.sqrt(adjacent\*\*2 + opposite\*\*2)

cos\_normal = adjacent / hypotenuse

sin\_normal = opposite / hypotenuse

# scale the values based on strength of the extracted directions

scale = surfaceness / max\_spectral\_feature

cos\_normal \*= scale

sin\_normal \*= scale

# create the new feature vector

new\_feature = [points[0][0], points[1][0], points[2][0],

point\_cloud\_intensities[points\_index[0]],

pointness, surfaceness, linearness, cos\_tangent,

sin\_tangent, cos\_normal, sin\_normal]

features[feature\_index, :] = new\_feature

feature\_index += 1

data = pd.DataFrame(data=features, columns=feature\_names)

return data

The remaining functions are used to load in the training data.

### Utilities

In ply\_file.py, I have implemented file utilities for reading and writing .ply files. Ply, or polygon file format, is designed to store 3D data, and is a simple file format.

## Refinement

During the work on this project, many different ways of improving the algorithm’s performance were tried. First, several feature vectors were tried. Different kinds of features included using the covariance matrix values directly and including statistics of the intensity values in the neighborhood. Different transforms on the data were also tried. PCA to reduce the dimensionality of the feature vector negatively impacted the algorithm’s performance. Log and other scaling transforms to make the data more normally distributed also negatively impacted the performance or had no impact.

In the end, the primary way to improve performance was to look at different learning algorithms and improve its performance using grid search. AdaBoostClassifier was found to be the best performing algorithm of all those tried. Once it was chosen, grid search was used to find the best n\_estimators parameter and algorithm. The n\_estimators parameter sets the maximum number of estimators used when boosting. The following values were tried: [10, 20, 30, 40, 49, 50, 51, 60, 70, 80, 90, 100]. The parameter algorithm sets the boosting algorithm and the values tried were {‘SAMME’, ‘SAMME.R’}.

Table 5 shows some of the progressions in performance improvement. In the end, the benchmark classifier, LogisticRegression has a testing F1 score of 0.73 and the tuned AdaBoostClassifier has an F1 score of 0.90. The final AdaBoostClassifier beats the performance of the benchmark.

Table 5: Performance improvements

|  |  |  |
| --- | --- | --- |
| **Classifier Description** | **Training F1 Score** | **Testing F1 Score** |
| LogisticRegression (Benchmark) | 0.75 | 0.73 |
| SVC (no scaling) | 0.0 | 0.0 |
| SVC (scale (mean = 0, variance =1)) | 0.85 | 0.77 |
| GaussianNB | 0.83 | 0.69 |
| KneighborsClassifier | 0.78 | 0.73 |
| AdaBoostClassifier (PCA) | 1.0 | 0.7 |
| AdaBoostClassifier (scaled) | 1.0 | 0.85 |
| AdaBoostClassifier (no scale) | 1.0 | 0.85 |
| AdaBoostClassifier (Tuned) | 0.91 | 0.90 |

# Results

*(approx. 2-3 pages)*

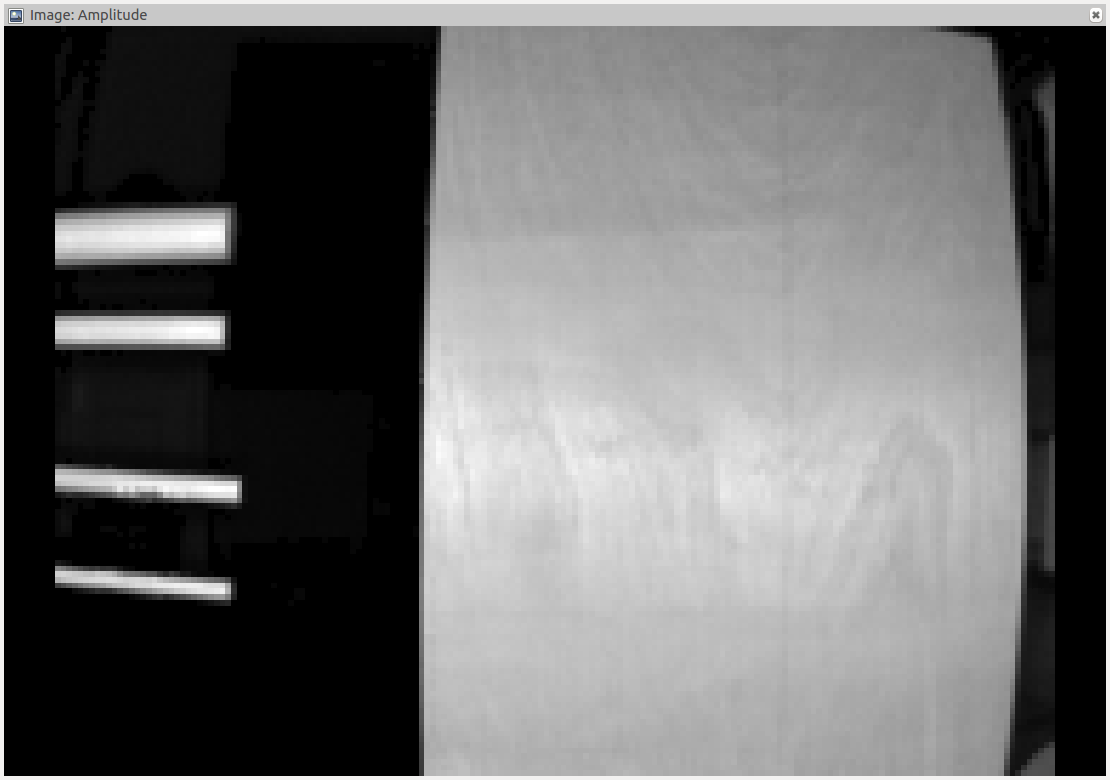
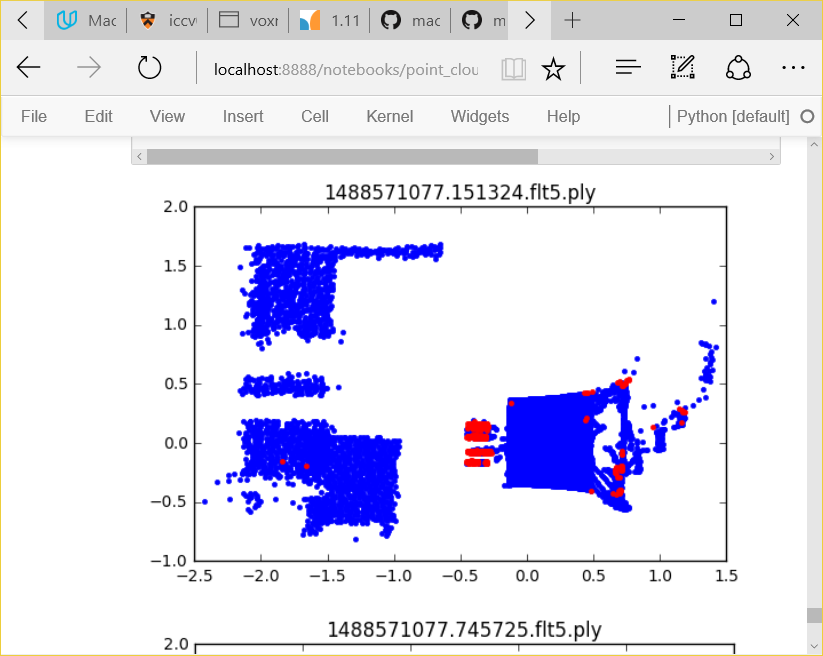
## Model Evaluation and Validation

The final classifier used is the AdaBoostClassifier. AdaBoost is an ensemble learner where a series of weak learners are combined to create a classifier more powerful than its parts. The core of the algorithm uses an array of weak learners, each of which is trained, focusing on data that the previous learner was failing on. The predictions from all of the learners are then combined by a weighted vote. For this project, the AdaBoostClassifier uses DecisionTrees as its base classifier.

This classifier was chosen because it had the best performance of the classifiers evaluated. Decision trees are capable of classifying data whose boundaries are arbitrarily complex. This makes them very flexible at the cost of added computational burden. The results from the training and testing data show an F1 score of 0.9. The classifier is performing well on the training and testing data.

To measure the sensitivity of the classifier, various amounts of noise were added to the hand labeled point cloud data. The point cloud features were calculated on this noisy data and passed to the optimized classifier from above. The noise was added to the x,y,z point cloud values and was normally distributed with 0 mean. The standard deviation of the noise was varied from 0 to 2cm. As a reference point, the noise of the IFM 03D3xx sensor used is 0.3cm. This was measured by imaging a planar surface and measuring the standard deviation of the point distance from a fitted plane. [REF] shows the resulting plot of F1 score vs. noise level. As can be seen, the F1 score remains relatively stable until the noise level reaches 0.5cm at which point it drops quickly. Because the noise level of the IFM is 0.3cm, its noise level could double before having a significant effect on the classifier’s F1 score. This verifies the classifier’s resistance ot the effects of sensor noise.

To verify that the model generalizes to unseen data, the classifier was given point clouds different from the training data. For example, in [REF], a smaller section of the pipe was imaged from a different angle. The classifier was still able to detect the pipes with a small number of false detections. The classifier responds well to data that was not seen during the training process.



## Justification

Compared to the benchmark LogisticRegression, the optimized AdaBoostClassifier has better performance. From [REF], the LogisticRegression classifier has a F1 score of 0.73 on the testing data set. The AdaBoostClassifier has an F1 score of 0.90.

In this section, your model’s final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

* *Are the final results found stronger than the benchmark result reported earlier?*
* *Have you thoroughly analyzed and discussed the final solution?*
* *Is the final solution significant enough to have solved the problem?*

# Conclusion

*(approx. 1-2 pages)*

## Free-Form Visualization

Foo

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

* *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
* *Is the visualization thoroughly analyzed and discussed?*
* *If a plot is provided, are the axes, title, and datum clearly defined?*

## Reflection

Foo

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

* *Have you thoroughly summarized the entire process you used for this project?*
* *Were there any interesting aspects of the project?*
* *Were there any difficult aspects of the project?*
* *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

## Improvement

Although this classifier does a good job of identifying objects in a point cloud; however, it can be improved. Incremental improvements to the algorithm would include gathering more hand labeled training data or doing a more through grid search to further optimize performance.

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

* *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
* *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
* *If you used your final solution as the new benchmark, do you think an even better solution exists?*

**Before submitting, ask yourself. . .**

* Does the project report you’ve written follow a well-organized structure similar to that of the project template?
* Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
* Would the intended audience of your project be able to understand your analysis, methods, and results?
* Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
* Are all the resources used for this project correctly cited and referenced?
* Is the code that implements your solution easily readable and properly commented?
* Does the code execute without error and produce results similar to those reported?

# References

Andrew McCallum, K. N. (1998). *A Comparison of Event Models for Naive Byes Text Classification.* AAAI.

Fellipe do Prado Arruda, e. a. (2014). Fault Detection in Industrial Plant Using K-Nearest Neighbors with Random Subspace Method. *Proceedings on the International Conference on Artificial Intellicgence (ICAI). .* WorldComp.

Fellipe do Prado Arruda, V. d. (n.d.).

Laura Auria, R. A. (2008). Support Vector Machines (SVM) as a Technique for Solvency Analysis. *Discussion Papers of DIW Berlin 811* (p. 16). Berlin: DIW Berlin, German Institute for Economic Research.

Rui Hu, Q. (2013). Prediction of Energy Consumption in Steel Enterprises based on BP Adaboost Algorithm. *of the Sixth International Conference on Management Science and Engineering Management. Lecture Notes in Electrical Engineering, vol 185* (pp. 411-419). London: Springer.

scikit-learn. (2016). *Choosing the Right Estimator*. Retrieved from SciKit Learn: http://scikit-learn.org/stable/tutorial/machine\_learning\_map/index.html

Zakhor, X. S. (2011). Fast approximation for geometric classification of LiDAR returns. *18th IEEE International Conference on Image Processing* (pp. 2925-2928). Brussels: IEEE.