Classification of Objects in 3D Point Cloud Data

Stephan Roth
sroth44489@gmail.com

April 6, 2017

1 Definition

1.1 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].

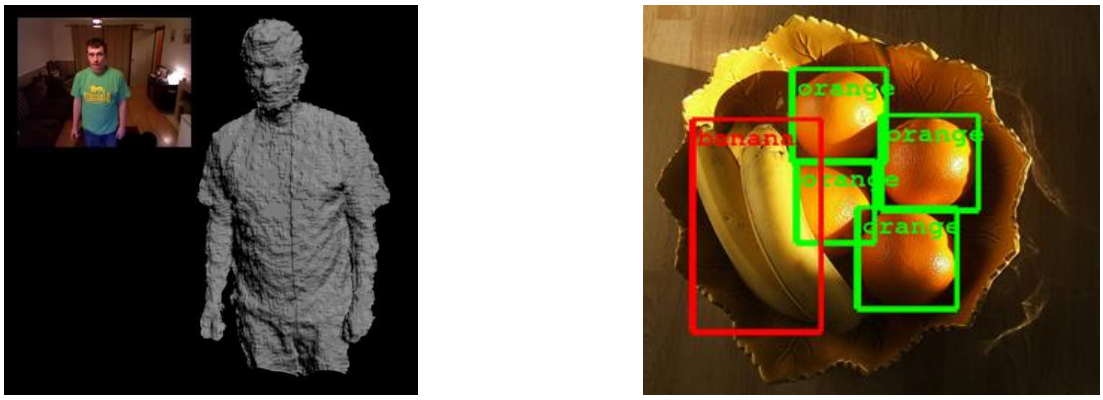


Figure 1: (left) 3D Point cloud collected using a Microsoft Kinect (ref. http://123kinect.com/wp-content/uploads/2012/08/Kinect-1-point-cloud-example.jpg) (right) Google's object recognition algorithm (ref. https://www.engadget.com/2014/09/08/google-details-object-recognition-tech/)

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 2Figure 1]. It is now possible to do the same thing with these new 3D images. This has applications in areas 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.

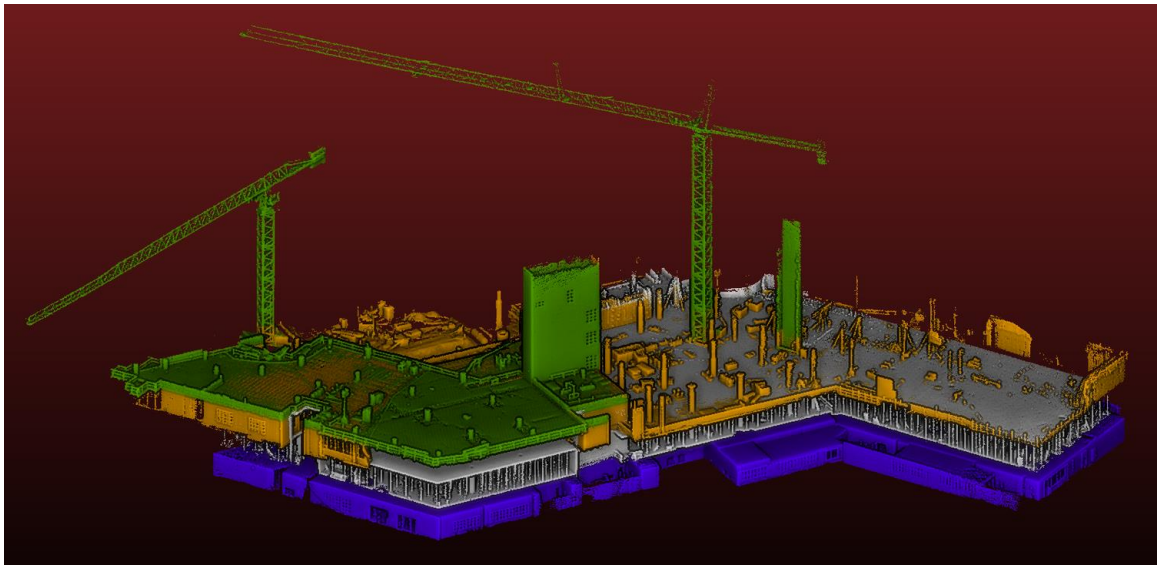


Figure 2: 3D Survey of a building under construction. The data is a 3 dimensional point cloud. In a point cloud, the building is represented by a dense cloud of (x,y,z) points, possibly with color or brightness information (Ref. kaarta.com/gallery)

1.2 Problem Statement

The purpose of this project is to create a supervised learning classifier to identify and locate pipe-like objects (target 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. 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

1.3 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 performance of a test with binary results. With a binary 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:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F_1 = 2\frac{precision * recall}{precision + recall}$$

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. The F1 score is appropriate because it considers both values of the "true results" and "test results", one is not valued over the other.

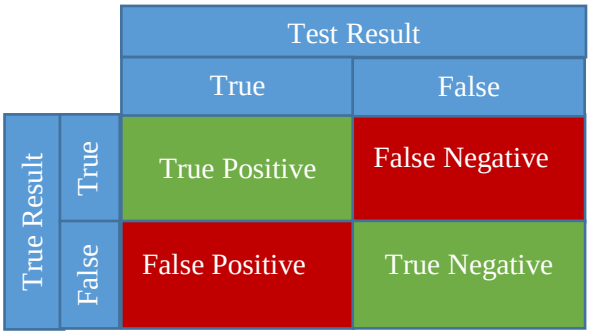


Figure 3: Possible test results are True positive, false positive, false negative and true negative

Accuracy is another metric I could have used. Accuracy is not a good candidate because there is not an equal number of each class in this project. An accuracy score would tend to reward a classification of the more numerous class. Also, precision and recall could have been used by themselves. Again, because the dataset does not have equal numbers of each class, recall would have concentrated too much on the points which are members of the class we are interested in, ignoring the other points' classifications. The precision score would have ignored false negative values.

2 Analysis

2.1 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

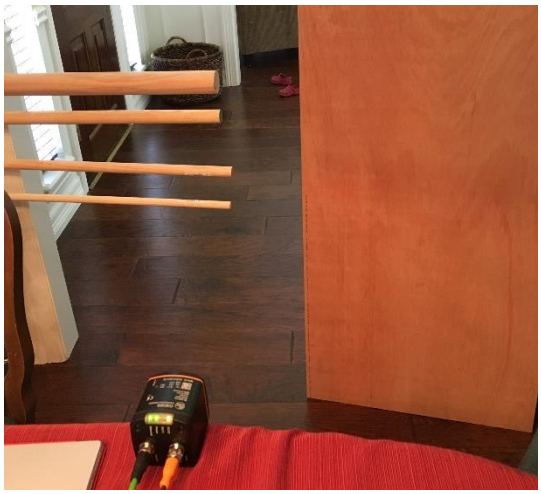
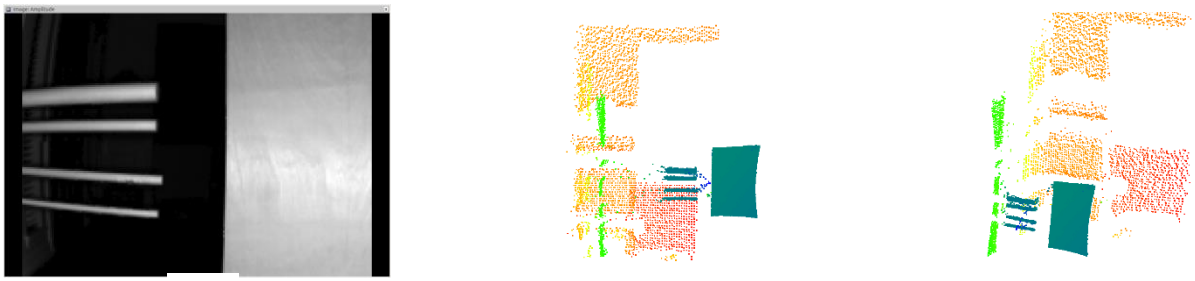


Figure 4: 3D Scene with pipes, walls and other objects

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 of the region around a point. Given the eigen values $\lambda_3 < \lambda_2 < \lambda_1$ of the covariance matrix, the geometric features are$\{\sigma_p = \lambda_1,\ \sigma_s = \lambda_2 - \lambda_1,\ \sigma_l = \lambda_3 - \lambda_2\}$. 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:$scale\{v_t, v_n\} = \frac{\{\sigma_l, \sigma_s\}}{\max(\sigma_l, \sigma_p, \sigma_s)}$. 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 class 0 points are much more numerous than the class 1 points, but that is just due to the nature of the problem. The pipes we are detecting will not make up an equal portion of an image as other items. The statistics for both classes



(a) (b) (c)

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. The other objects are walls and chairs in the background. The points are colored by distance from the sensor

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.

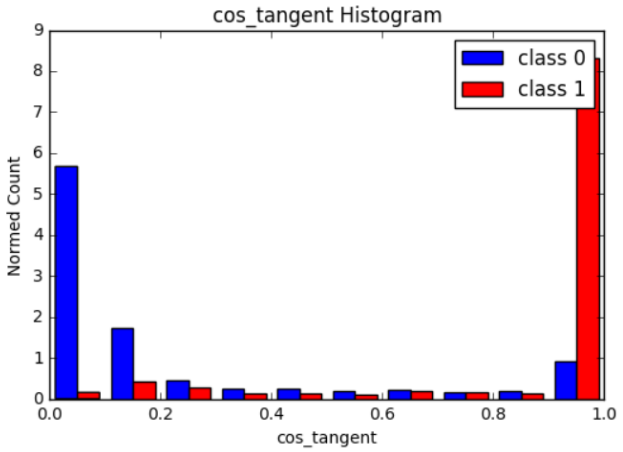
Comparing the data between classes, class 1 does seem to occupy a distinct region. In particular, because the pipes are very linear in shape, one would expect the "Linearness" feature to stand out, and it does. For class 0, the mean is $4.71x10^{-4}$ and std is $28.1x10^{-4}$. For class 1, the mean is $7.20x10^{-4}$ and std is $3.29x10^{-4}$. For class 1, the mean *l*inearness 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 horizontal. 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%.

2.2 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.

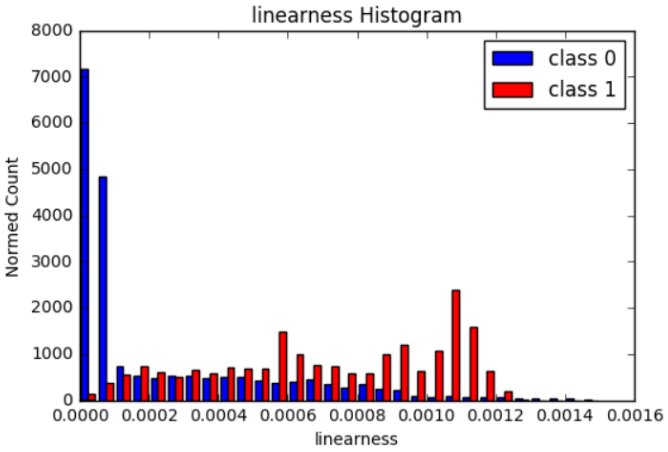


Figure 7: Histogram of linearness feature for class 0 and 1. Class 1 tends to be more linear than class 0

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.

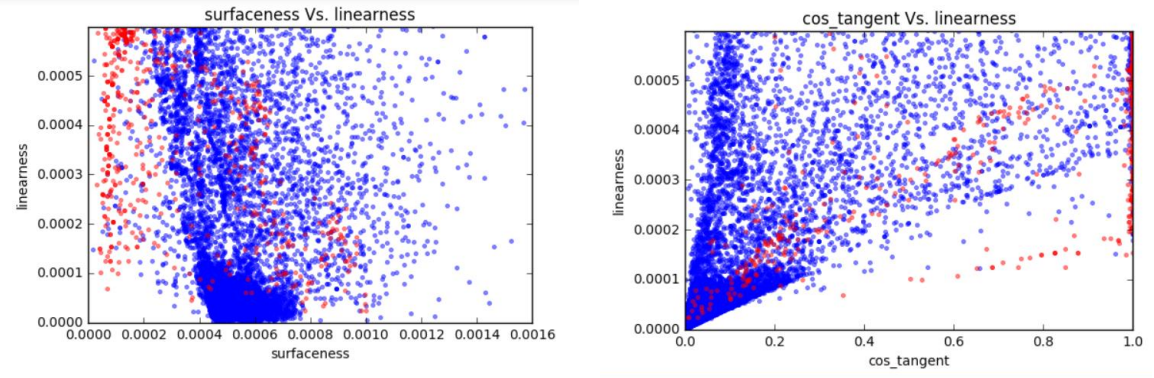


Figure 8: Scatter plots showing the class 0 (blue) and class 1 (red) feature distributions. The two classes mostly fall into two distinct clusters

2.3 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. These algorithms were chosen because they represent a wide array of learning techniques from simpler linear models to models with arbitrarily complex decision boundaries.

Looking at the hand labeled data set, there are 14,794 points in class 0 and 1,427 points in class 1. The feature vector used has 7 dimensions. With only 16,000 training points, the 7 dimensional feature space will be sparsely populated (only leaving 4 data points per dimension (4^7=16384)). PCA can be used to reduce the dimensionality of the search space, and thereby increase the density of the training data.

To train each classifier, the hand labeled data set will be split into an independent training and testing set. To effectively increase the size of the training data, cross validation is used when training each

classifier. 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. Once the best classifier has been found, its parameters will be optimized with grid search.

The first classifier evaluated was the Support Vector Classifier. Support Vectors are used for regression and classification. With linear models, Support Vectors Classifiers divide the two classes using a hyper plane. The hyper plane is located so as to maximize the distance between the plane and the points in the two classes closest to that plane (the margin). 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. AdaBoost is an ensemble learner (AdaBoostClassifier, n.d.). An array of weak learners are combined in such a way that the overall classifier is stronger than its individual parts. 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 complexly tied to current global demand for steel and government incentives for steel production, traditional modeling methods fail. The authors use a neural network based AdaBoost 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 is an instance based learner. (Nearest Neighbors, n.d.) It does not create a model of how the features are related to the classes. Instead, it stores all features and classes from the learning data. When classifying a new feature, it takes a weighted vote of the k closest learned features to the new feature. 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.

The Gaussian Naïve Bayes algorithm assumes that the likelihood of the features is gaussian (Gaussian Naive Bayes, n.d.). The mean and standard deviation of the distributions are estimated using maximum likelihood. 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.

2.4 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.

The benchmark was scored according to its F1 score on the testing data. The benchmark was fit using cross validation on the training data set and optimized using grid search. 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

3 Methodology

3.1 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.

It is sometimes possible to transform the input data to improve the efficiency or predictive ability of the learning algorithm. Reducing the dimensionality of the feature vector can improve efficiency. Other transforms, which 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. 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. These results are more fully detailed in 3.3 Refinement.

Some classifiers assume that the data is normally distributed with zero mean and standard deviation of 1, giving 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 standardized normal distribution. Again, the positive or negative impact of such a transform needs to be measured.

In this project, 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. These results are more fully detailed in 3.3 Refinement.

3.2 Implementation

3.2.1 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 1Figure 9]. The workflow is as follows:

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.

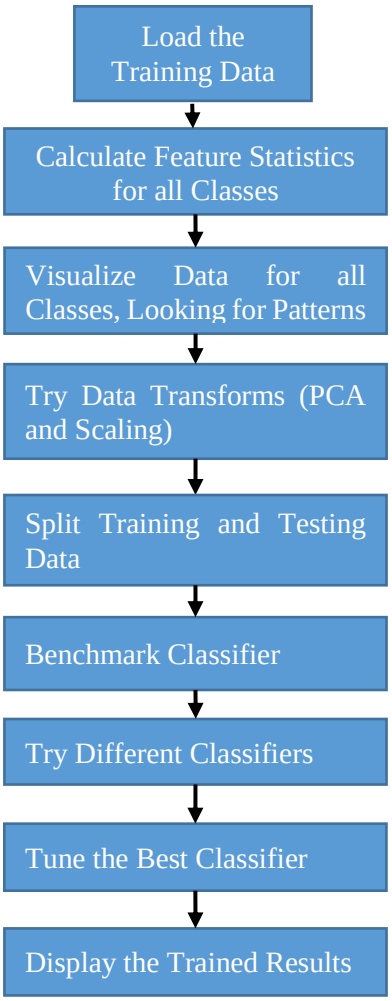


Figure 9: Jupyter Notebook Workflow

3.2.2 Point Cloud Classification Python File

In processPly.py, I have implemented functions 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 features 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 covariance matrix is calculated. 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 point cloud and hand labeled data. The most important of these is load_all_training_data(data_file_path). It takes as input the path to the training data directory. It loads all of the point cloud data, calculates the feature vectors for all points and attaches the hand labeled class ID to each point. 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.

In implementing this code, one challenge was to efficiently find the neighborhood of points when calculating the feature vector. Because this had to happen for each point in the point cloud, a brute force search was much too slow. The kdtree, or k-dimensional tree, was used to speed this up. The kdtree is a

way to organize data in a k dimensional space for things like nearest neighbor searches. The kdtree speeds the finding of nearest neighbors, but its construction is still slow. There is another data structure called an octree which is optimized for 3 dimensional data such as the point cloud. Using one may increase the processing speed, but is left for future work.

3.3 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. Using 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 and algorithm parameters. 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. The final, tuned AdaBoostClassifier had parameters n_estimators=50, learning_rate=1.0, algorithm = 'SAMME.R'

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

4 Results

4.1 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. Figure 10 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.

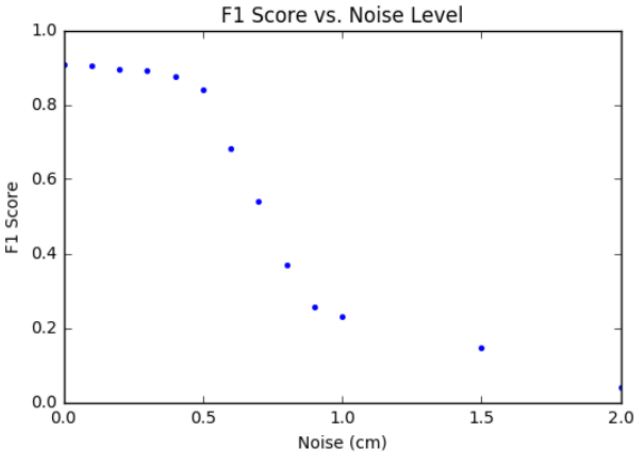


Figure 10: Classifier's F1 Score vs Noise. The noise value is the standard deviation of normally distributed error applied to the x,y,z values in the point cloud.

To verify that the model generalizes to unseen data, the classifier was given point clouds different from the training data. For example, in Figure 11, a smaller section of the pipe was imaged from a different

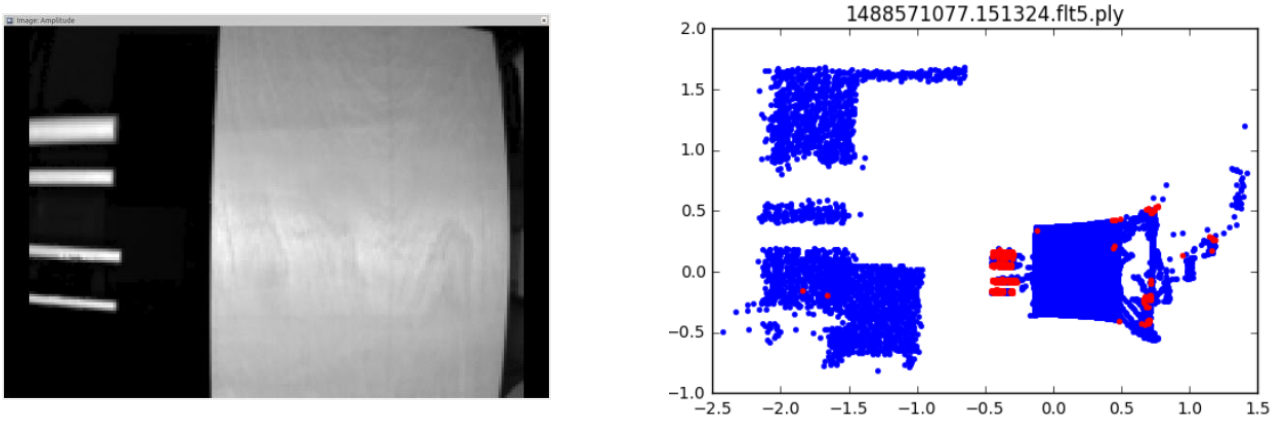


Figure 11: Classifier result on untrained data. The left shows the intensity image from the flash lidar. The right shows the classified point cloud (class 0 in blue, class 1 in red). The classifier successfully detects the four pipes. Additional false detections are mostly isolated points.

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.

4.2 Justification

Compared to the benchmark LogisticRegression, the optimized AdaBoostClassifier has better performance. From Table 5: Performance improvementsTable 5, the LogisticRegression classifier has a F1 score of 0.73 on the testing data set. The AdaBoostClassifier has an F1 score of 0.90. Although the AdaBoostClassifier outperforms the benchmark by a significant amount, there is still room for improvement. This improvement will probably need to come through an increase in the amount of training data. However, it is not clear how much additional training will improve performance. Figure 12 shows a plot of F1 score vs. training size. The F1 score seems to level off at about 0.9. Significant improvement would need to come from a different algorithm altogether or some additional processing of the data after the AdaBoostClassifier. These items are discussed in section 5.3 Improvement.

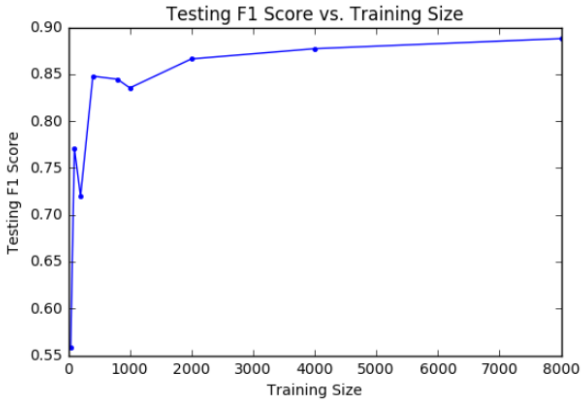


Figure 12: AdaBoostClassifier F1 score vs. training set size. The F1 score levels off at 0.9. Additional training data may not increase the final F1 score.

5 Conclusion

5.1 Free-Form Visualization

The following video shows an animation of the classified point cloud https://youtu.be/_cHg8JF91zc . The pipe points are colored red and the other points are colored blue. This animation shows the true 3-D nature of the point cloud and the classification. What this also shows is the relative isolation of the misclassified points. They tend to be individual points of red within a cloud of blue points. The exception is the upper right hand corner of the planar object. There is a tight cluster of points that are misclassified as pipe. It may be that the edge of the planar object looks very linear and that is why the classifier fails there.

5.2 Reflection

The basic process used in this project, and in general the process used for supervised learning, was:

1) Gather data
2) Classify the data by hand
3) Define a set of features for the classifier
4) Look at the distribution of features
5) Train a classifier
6) Measure its performance

The two areas that were most challenging were classifying the data by hand and defining a set of features for the classifier. Before classifying the data by hand, I tried to use unsupervised learning to find clusters of features to identify the pipes in the training data. This did not work as well as I was hoping, so I ended up hand labeling the data, hoping this would lead to a more accurate classifier. One downside to this is that, due to the labor involved, hand labeling the data leads to a smaller dataset to use for learning.

The second challenge was defining the features to use for the classifier. I tried using statistics on the intensity information, thinking that similarly bright areas might belong to the same object. However, I ended up rejecting that idea because intensity is related to distance from the sensor. If an object was moved closer or further from the sensor, this intensity feature would be changing. That would mean that, to train on that

feature, an object would need to be seen at a range of distances. This would add an extra layer of complexity to the training process without adding to the classifier's performance.

5.3 Improvement

Although this classifier does a good job of identifying objects in a point cloud, 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. A more significant improvement would be to add the ability to classify a point based on point cloud features and to incorporate the classification of neighboring points.

It is reasonable to expect that within a point cloud, points that are close to each other should have the same classification. Looking at the classification results above, many of the mis-classified points are isolated. They are single points surrounded by other correctly classified points. To counter this, the classification of a point could be a weighted vote of the classification based on point cloud features of all the points in a neighborhood. The weights would be based on distance from a given point. In this way, isolated points would be reclassified as the class of the majority of its closest neighbors.

There are other modern algorithms that use 3D convolutional neural networks (CNNs) (Maturana, 2015). Instead of using hand crafted features like we have used in this project, VoxNet uses point cloud volumetric data as input to a convolutional neural network. The volumetric data are simplified representations of the point cloud. The features and classifiers are then jointly learned from the data. This is an application of Deep Learning to the problem of 3D point cloud classification, and has shown great promise, outperforming state of the art classifiers.

6 References

AdaBoostClassifier. (n.d.). Retrieved from Scikit Learn: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

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.).

Gaussian Naive Bayes. (n.d.). Retrieved from Scikit Learn: http://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes

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.

Maturana, D. a. (2015). VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. *IROS.*

Nearest Neighbors. (n.d.). Retrieved from Scitkit learn: http://scikit-learn.org/stable/modules/neighbors.html#classification

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.