Using Supervised Learning for Mushroom Classification

Prepared for: Udacity Nanodegree Program
Prepared by: Džanan Ganić
2 December 2016

I. DEFINITION

Project Overview

“A fungus is any member of the group of eukaryotic organisms that includes unicellular microorganisms such as yeasts and molds, as well as multicellular fungi that produce familiar fruiting forms known as mushrooms.” (Source: Wikipedia) “According to records from 1991, there are 1.5 million fungi on the Earth. Fungal habitats include soil, water, and organisms that may harbour large numbers of understudied fungi, estimated to outnumber plants by at least 6 to 1.”(Source) Mushrooms are the fleshy, spore-bearing fruiting bodies of a fungi, typically produced above ground on soil or on its food source. In this project, we want to focus on mushrooms which can be easily found while taking a stroll in the forests (the most common ones).

There exist many edible, high quality mushrooms which are rich with vitamins and minerals and have great value at market. However, some of the mushrooms are toxic, which can cause different type of health problems if consumed, and a small number of them is even deadly. Throughout this project, we want to classify the mushrooms and find out which ones are edible, and which ones are toxic.

The dataset we will be working has the records drawn from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf. The donor is Jeff Schlimmer (jeffrey.Schlimmer@a.gp.cs.cmu.edu), and the date is 27 April 1987.

Figure 1.1 - Amanita caesarea - an example of an edible mushroom (Source: Google images)

Problem Statement

The goal is to create a machine learning model which will correctly classify the mushroom based on its traits; the tasks involved are the following:

1. Fetch the mushroom dataset
2. Explore the dataset in order to see what kind of records and traits are contained
3. Prepare/preprocess the data, check out the missing values etc.
4. Perform feature selection
5. Perform feature transformation (if needed)
6. Train a few supervised models and see which are performing the best
7. Tune the best one for the best performance

The final model is expected to have more than 95% success score.

Metrics

In this project we are tackling binary classification problem. In order to evaluate it, we will use accuracy score. The reason why accuracy is good choice is that in this problem is that there is not a huge imbalance in the data (there are 51.8% edible and 48.2% poisonous mushrooms). However, some types of misclassifications can be worse than the others. For example, it is less worse to misclassify edible mushroom as poisonous one, than poisonous as edible (as we can kill someone that way). We will be using confusion matrices in this project in order to observe the ratio of false positive, false negative, true positive and true negative classifications, and according to that, use cost sensitive learning if needed. The formula for accuracy score is the following

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

It is taking into account both true positives and true negatives with equal weight.

II. ANALYSIS

Data Exploration

The dataset we are using has a total number of 8124 instances, each having 22 features. There are 4208 edible mushrooms and 3916 poisonous mushrooms (51.8% are edible and 48.2% are poisonous). As noted before, we have a balanced dataset to work with, and because of that the accuracy score should perform the best. The dataset contains samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each record has the following features:

1. Cap shape (bell=b, conical=c, convex=x ,flat=f, knobbed=k, sunken=s)
2. Cap surface (fibrous=f, grooves=g, scaly=y, smooth=s)
3. Cap colour (brown=n, buff=b, cinnamon=c, grey=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y)
4. Bruises (bruises=t, no=f)
5. Odor (almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s)
6. Gill attachment (attached=a, descending=d, free=f, notched=n)
7. Gill spacing (close=c, crowded=w, distant=d)
8. Gill size (broad=b, narrow=n)
9. Gill colour (black=k, brown=n, buff=b, chocolate=h, grey=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y)
10. Stalk shape (enlarging=e, tapering=t)
11. Stalk root (bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r)
12. Stalk surface above ring (fibrous=f, scaly=y, silky=k, smooth=s)
13. Stalk surface below ring (fibrous=f, scaly=y, silky=k, smooth=s)
14. Stalk colour above ring (brown=n, buff=b, cinnamon=c, grey=g, orange=o, pink=p, red=e, white=w, yellow=y)
15. Stalk colour below ring (brown=n, buff=b, cinnamon=c, grey=g, orange=o, pink=p, red=e, white=w, yellow=y)
16. Veil type (partial=p, universal=u)
17. Veil colour (brown=n, orange=o, white=w, yellow=y)
18. Ring number (none=n, one=o, two=t)
19. Ring type (cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z)
20. Spore print colour (black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y)

21. Population (abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y)
22. Habitat (grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d)

There are 2480 missing values denoted by "?", all for stalk root feature. We will see that in 'Data Preprocessing' chapter the 'stalk-root' feature can be predicted based on the other features, thus being unnecessary.

After exploration we have obtained some interesting results which can be seen in the section 'Exploratory Visualisation'.

Exploratory Visualisation

The figure 2.1 shows the comparison of total number of poisonous and edible mushrooms. We can see that we are dealing with quite balanced dataset.

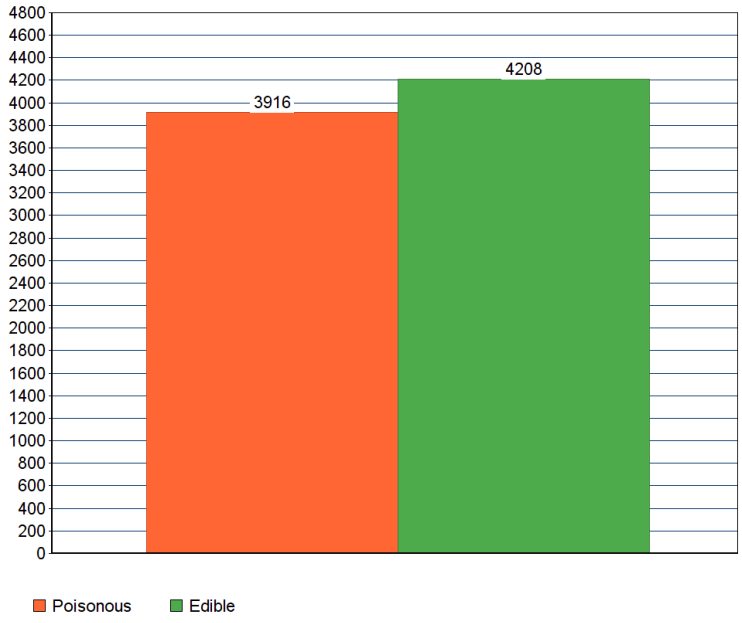

Figure 2.1 - Mushroom type comparison

The figure 2.2 shows the number of mushrooms at certain locations. We can see that most mushrooms are in the woods and grasses. At paths, we can see high domination of poisonous mushrooms, whereas in the woods and grasses, there are more edible ones. High domination of edible mushrooms is present in meadows.

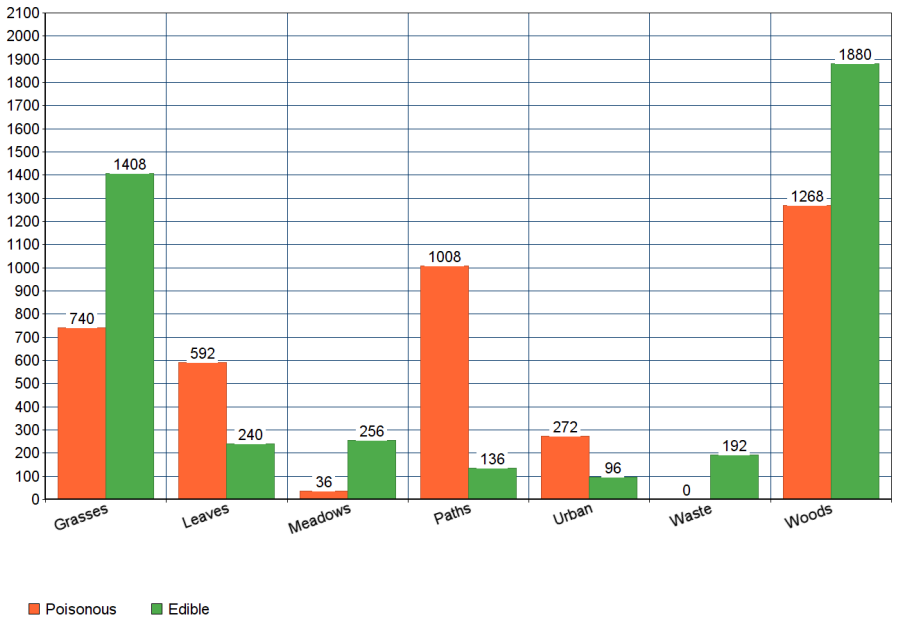

Figure 2.2 - Mushroom locations

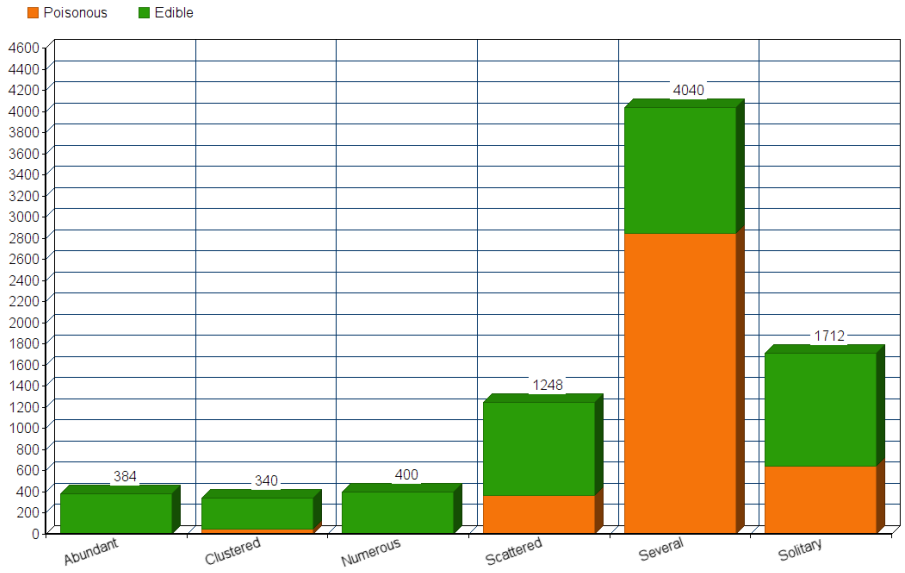

Figure 2.3 - Population Stats

In the Figure 2.3, we can see that edible mushrooms tend to have Abundant, Clustered and Numerous populations, while there are poisonous mushrooms with ‘Several’ population, and some of them can be solitary or scattered.

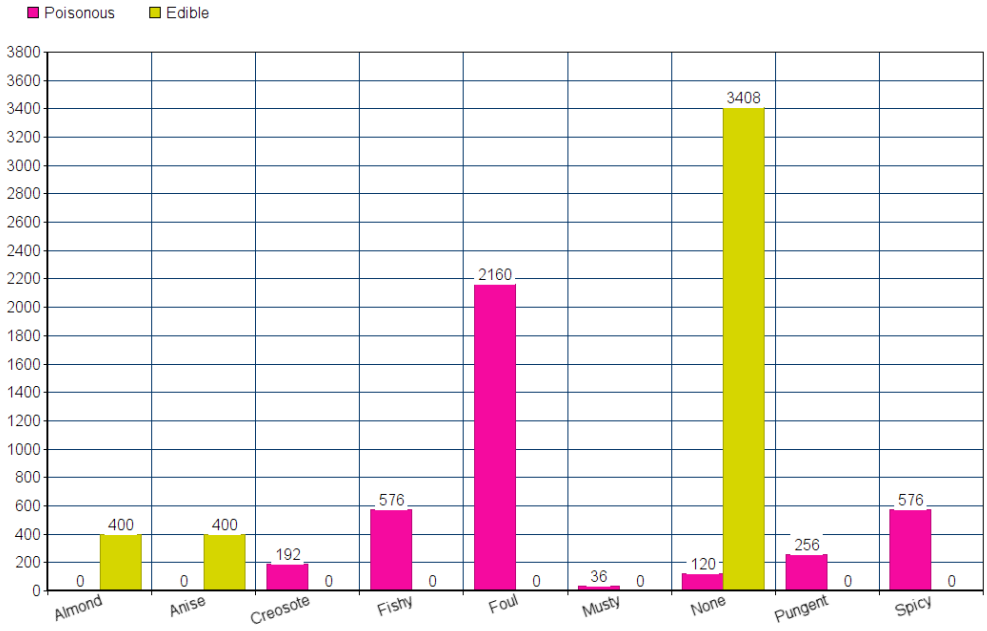

Figure 2.4 - Odor Stats

In the Figure 2.4, it is possible to notice that edible mushrooms do not have any odor, while poisonous ones have usually spicy, pungent, foul, creosote and fishy odors. When edible mushrooms have some odor, it is usually almond or anise.

Algorithms and Techniques

While tackling this problem, we will choose 3 different supervised learning models, fit our data, and compare their performances. We will pick the best one and proceed to fine tuning in order to achieve as good result as possible. Depending on results, we will compare confusion matrices and implement cost sensitive learning if needed. During the experiments, we will evaluate our models with accuracy score and produce tables which show the training set size, time, prediction time, accuracy score on training and accuracy score on testing set.

1. Support Vector Machines

Support Vector Machines perform best with binary classification, and that is exactly the problem we are dealing with now. The idea behind Support Vector Machines is to find the best way to separate the data. That means - to find the line which will separate the data with the biggest margin possible. Such line is called a separating hyperplane. We can find example scatter plot (Image from google) in the figure 2.5 which shows how the data is separated with the red line called hyperplane.

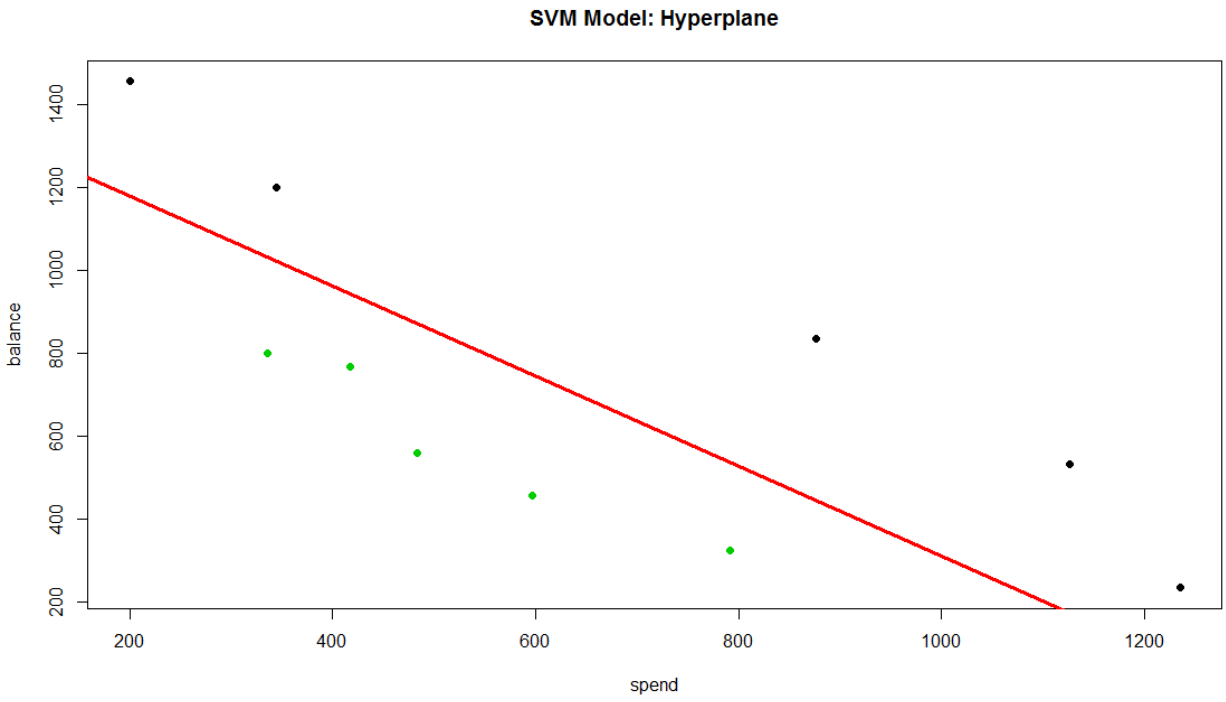

Figure 2.5 - An example of SVM hyperplane separating the data

In this example, the red line is called hyperplane, because we are talking about 2D. Hyperplane is a generalisation of a plane; in one dimension, it is called a point; in two dimensions, it is line; in three dimensions, it is a plane and in more dimensions we can call it hyperplane. Given a particular hyperplane, we can compute the distance between the hyperplane and the closest data point. That value is called the margin. We can see an example of margin in the figure 2.6.

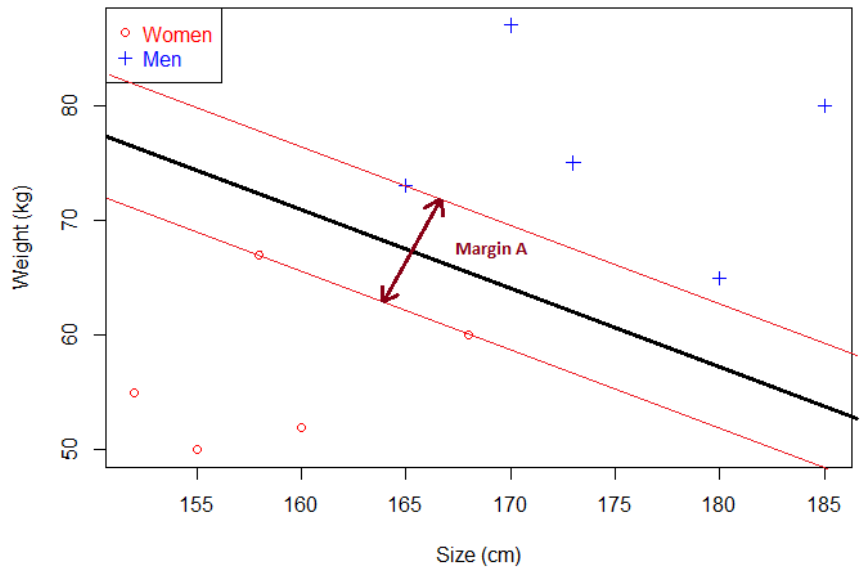

Figure 2.6 - An example of SVM hyperplane separating the data

We can see that some data points are irrelevant in computing the largest margin, as we are looking only at the closest ones in order to properly separate the data. The relevant data points are called support vectors.
The goal of SVMs is to find the optimal separating hyperplane which maximises the margin of the training data.

If the data cannot be linearly separable, the kernel trick is used to raise the dimension. In the figure 2.7, we can see how our data in 2D is not linearly separable, and we are mapping it to 3D in order to make it linearly separable. Now we can split it with a hyperplane. Any linear model can be turned into a non-linear by applying the kernel trick to the model (replacing its features (predictors) by a kernel function.

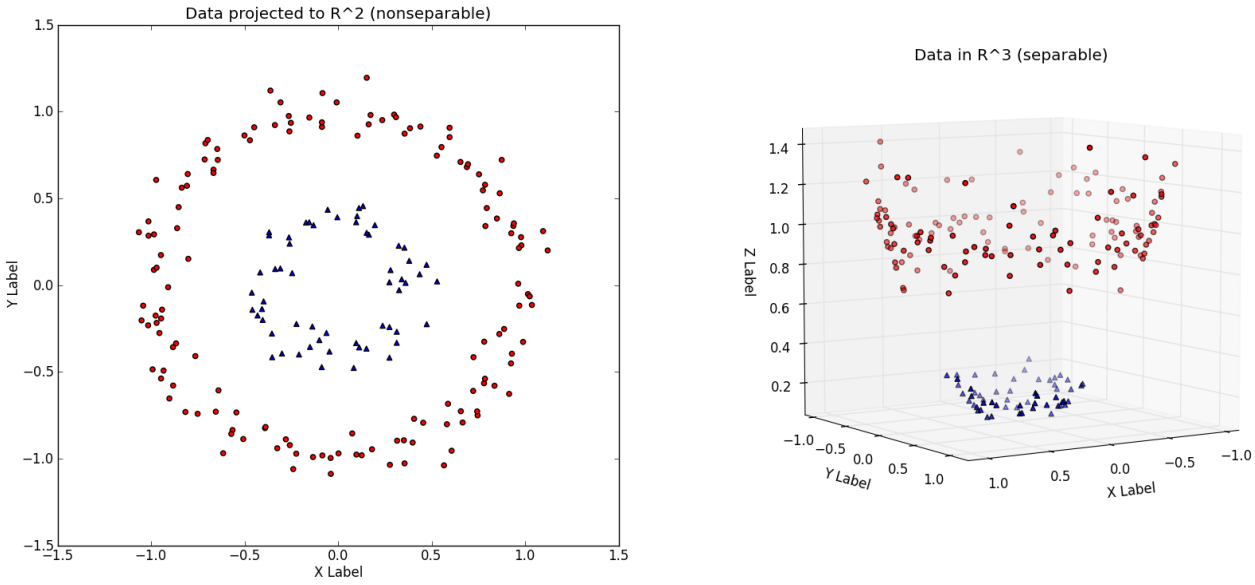

Figure 2.7 - An example of using kernel trick
to turn 2D data into 3D

Some of the popular kernels are:

- Fisher kernel
- Graph kernels
- Kernel smoother
- Polynomial kernel
- RBF kernel
- String kernels

SVM performs poorly with large datasets and with datasets with lots of noise. This is not the case with our problem, and we can expect good results. The most important parameters we can tune in this model are:
- Kernel (Specifies the kernel type to be used in the algorithm)

- Gamma (Kernel coefficient)

- Probability (Whether to enable probability estimates)

- Random State (The seed of the pseudo random number generator to use when shuffling the data for probability estimation.)

2. Stochastic Gradient Descent

Stochastic Gradient Descent is optimisation method for minimising an objective function that is written as a sum of differentiable functions. It can be used with neural networks and the real world application would be to use it on such problems as handwriting recognition.This model should not be a good fit for our type of problem, but we will test it to compare its results with the other two. The algorithm is based on estimating the gradient by computing the part of the gradient for a small sample of randomly chosen training inputs. By averaging over this small sample it turns out that we can quickly get a good estimate of the true gradient, and this helps speed up gradient descent, and thus learning. The important parameters for tuning are:

- Alpha (Constant that multiplies the regularisation term)

- Random State (The seed of the pseudo random number generator to use when shuffling the data)

- Learning rate (constant: eta=eta0, optimal: eta=1.0/(alpha*(t+t0)) or invscaling: eta=eta0/ pow(t,power_t))

- eta0 (The initial learning rate for the ‘constant or ‘invscaling’ schedules)

- Class weight (Weights associated with classes. If not given, all classes are supposed to have weight one.)

This is the formula for gradient descent algorithm:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Θj represents the next position
α is the step (usually small step) we are making
J is the direction of fastest increase

Stochastic gradient descent is a stochastic approximation of the gradient descent optimisation method. Stochastic approximation algorithms are recursive update rules that can be used, among other things, to solve optimisation problems and fixed point equations when the data is subject to noise. It is highly used with neural networks and backpropagation.

3. Random Forest

While reading papers regarding this problem we are tackling, we have seen that the decision trees can give very high accuracy scores while predicting the mushrooms as edible or poisonous. In this paper, we can see that it is possible to achieve 100% score with decision trees.

Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees (Source)

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. (Source)

In this case, we are using decision trees.

Each decision tree is constructed by using a random subset of the training data. After we have trained our forest, we can pass each test row through it, in order to output a prediction. This model should perform the best because it is based on decision trees, as decision trees are very good fit for this type of problem. The most important tuning parameters are:

- Max depth (The maximum depth of the tree)

- Min samples leaf (The minimum number of samples required to be at a leaf node)

- Max features (The number of features to consider when looking for the best split)

Benchmark

Dr. Włodzisław Duch of Nicolas Copernicus University in Poland has been able to perfectly classify this mushroom data by using a decision tree. The method Dr. Duch used is focused on finding and using the single criterion that has the fewest wrong classifications in order to separate the data, and then finding the next most effective one until he did completely classify the data. (Source)

In this project, we will analyse the data to get an insight in correlations, feature selection and transformation, and then test the different supervised learning algorithms with different parameters in

order to successfully classify the data, and then compare the results to the decision tree classifier Dr. Duch made.

III. METHODOLOGY

Data Preprocessing

Before applying supervised models on our data, it is necessary to do data preprocessing.

The first fact is that we have found 2480 missing values from the stalk-root feature, which will be unnecessary and removed later.

The second thing is to perform label encoding. Our dataset is composed of non-numerical data with which our supervised learning models will not work. That is why we need to convert our data into numeric by using label encoder which is a part of sklearn python library.

The next thing we did is to identify feature and target columns and split our data accordingly. We will have one target column (the type of mushroom we are trying to predict) and 22 other features. Since we have 22 features for every record in the dataset, we have analysed which features are irrelevant and can be removed. In order to perform such task, we have run simple decision tree classifier on every feature and report how good the prediction is. The figure 3.1 shows our results.

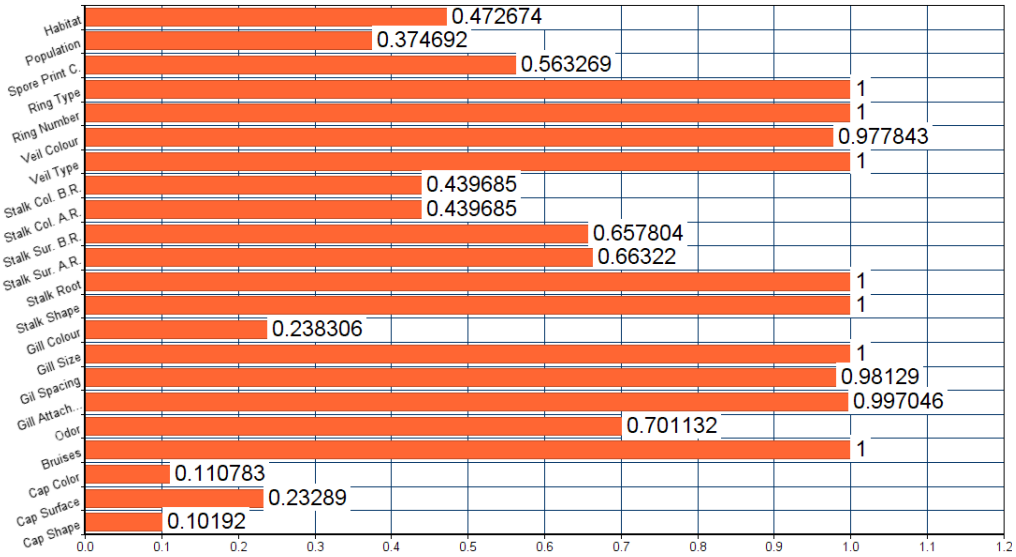

Figure 3.1 - Feature Prediction Scores

We can see that some features can be predicted easily based on the other traits. This makes them irrelevant because we can easily acquire that information if we need it. We will remove them at this point and reduce the number of features to 12.

We can use PCA for additional dimensionality reduction. It is possible to get 0.93 explained variance ratio with 8 components, but this would turn our feature values into continuous values. With that, we can perform additional clustering, but it will be hard to achieve 100% with this type of problem we are tackling. We will first try to perform traditional supervised learning, and then if there is need to speed up the training/testing time as well as reduce variance and bias, we can use PCA dimensionality reduction and reduce the number of features.

Implementation

After preprocessing our data, we can begin modeling, training and evaluating our models. We noted before that we will be using 3 different supervised learning models - Support Vector Machines, Stochastic Gradient Descent and Random Forests.

After fitting and predicting our data to each of those 3 models (without any parameter tuning), the results are the following:

Classifer 1 - SVC

Training Set Size	Training Time	Prediction Time (test)	Accuracy Score (train)	Accuracy Score (test)
100	0.0013	0.0005	1.0000	0.8922
200	0.0024	0.0012	0.9900	0.9222
300	0.0039	0.0019	0.9933	0.9473

Classifer 2 - Stochastic Gradient Descent Classifier

Training Set Size	Training Time	Prediction Time (test)	Accuracy Score (train)	Accuracy Score (test)
100	0.0006	0.0001	0.5900	0.5574
200	0.0006	0.0001	0.6650	0.6539
300	0.0006	0.0002	0.8500	0.8518

Classifer 3 - Random Forests Classifier

Training Set Size	Training Time	Prediction Time (test)	Accuracy Score (train)	Accuracy Score (test)
100	0.0237	0.0009	0.9900	0.9399
200	0.0228	0.0009	1.0000	0.9655
300	0.0227	0.0010	1.0000	0.9842

Figure 3.2 - Initial Model Scores

Confusion Matrices for Random Forests model using a training set size of 300 are:

Predicted Class \ Actual Class	Positive	Negative
Positive	159	0
Negative	0	141

Figure 3.3 - Confusion matrix for training (100% score)

Predicted Class \ Actual Class	Positive	Negative
Positive	1019	21
Negative	19	972

Figure 3.4 - Confusion matrix for testing (98.03% score)

Based on the experiments performed, we have confirmed our assumptions about each model with Figure 3.2 . Random Forests performs the best, as it is ensemble of decision trees. SVMs are performing well as we are dealing with the binary classification here, and Stochastic Gradient Descent is the fastest, but the accuracy score is very low compared to the other two models.

Refinement

Because the Random Forests model is giving us the best performance, we will proceed to addition tuning in order to increase the accuracy score as much as possible. We used GridSearchCV, which is a part of sklearn library in order to find the best parameter combination. We have run the grid search with the following parameters with values:

- 'max-depth' : [2,4,6,10,15]
- 'min_samples_leaf' : [2,5,10]
- 'max_features': [1,2,10]

After tuning our model, we have the following results:

Training set:

- Made predictions in 0.0044 seconds
- Tuned model has a training accuracy score of 1.0

Testing set:

- Made predictions in 0.0020 seconds
- Tuned model has a testing accuracy score of 1.0

Figures 3.5 and 3.6 show us the confusion matrices for training and testing respectively.

Predicted Class	Actual Class: Positive	Actual Class: Negative
Positive	3168	0
Negative	0	2925

Figure 3.5 - Training Confusion Matrix

Predicted Class	Actual Class: Positive	Actual Class: Negative
Positive	1040	0
Negative	0	991

Figure 3.6 - Testing Confusion Matrix

After tuning our Random Forest model, we have achieved exceptional results. We have successfully obtained 100% accuracy score on both training and testing sets with satisfiable prediction time of 0.0044 and 0.0020 seconds. Since we managed to to successfully classify 100% of mushrooms both in training and testing set, the additional dimensionality reduction with PCA is not necessary. Random Forests model is very quick and accurate.

IV. RESULTS

Free-Form Visualisation

In the figures 4.1 and 4.2, we can see the comparison between the initial models with the final model in terms of speed and accuracy with training set size of 300.

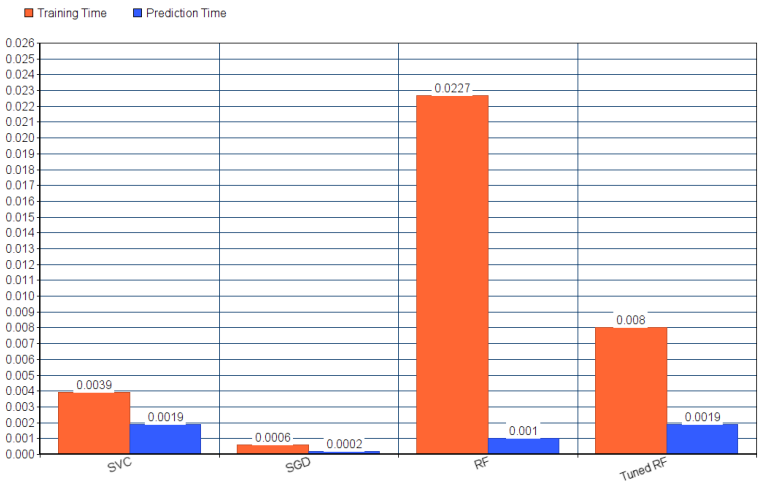

Figure 4.1 - Time Comparison

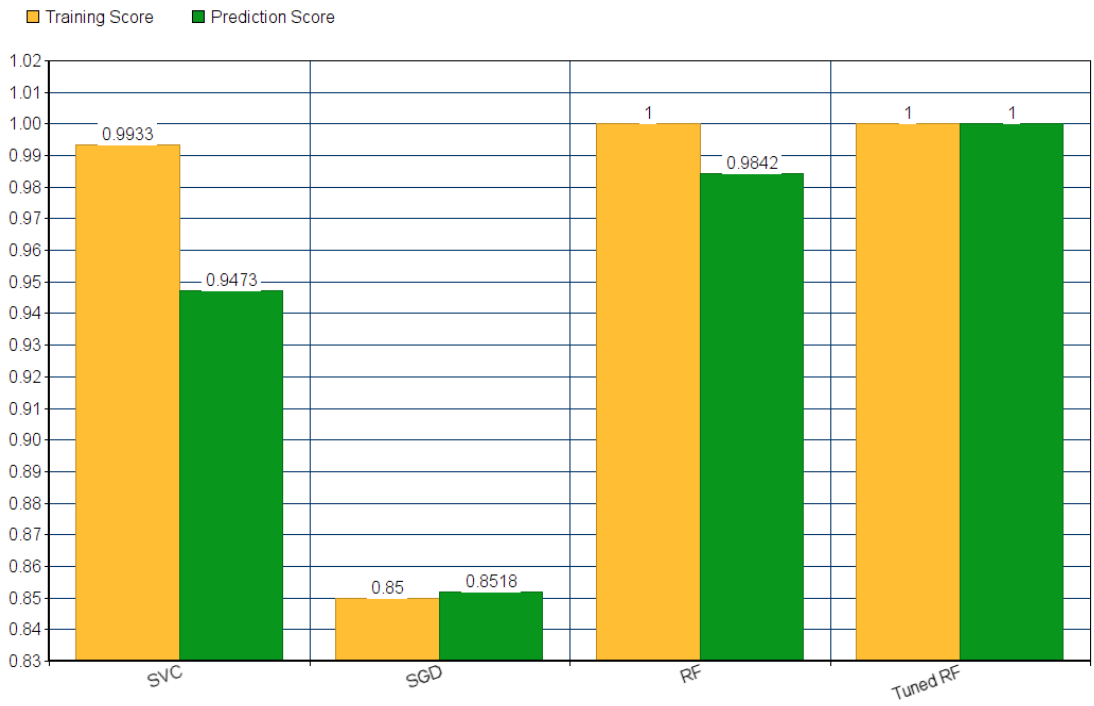

Figure 4.2 - Score Comparison

Model Evaluation and Validation

The obtained results with Random Forests classifier exceeded our expectations. Although we were aiming for high 95%, we have managed to successfully classify 100% of the instances from both training and testing sets. That means that we can declare our final model as reasonable and aligning with solution expectations. We have tested the model with different train sizes in order to test its robustness, and it consistently gives 100% accuracy score. Thus, we can conclude that we have developed a model which can be trusted.

The final model is Random Forest classifier with the following parameters:

- 'max_depth' : 15,
- 'min_samples_leaf' : 2,
- 'max_features' : 10

Max depth parameter represents the maximum depth of the tree. If none, then nodes are expanded untill all leaves are pure or until all leaves contain less than min_samples_split samples. If we limit the max depth to low value, our tree might be shallow and not be able to predict successfully due to

high bias (model is not complex enough to capture the underlying relationships). For example, if we set ‘max_depth’ parameter to 2, we will get worse results (Figure 4.3).

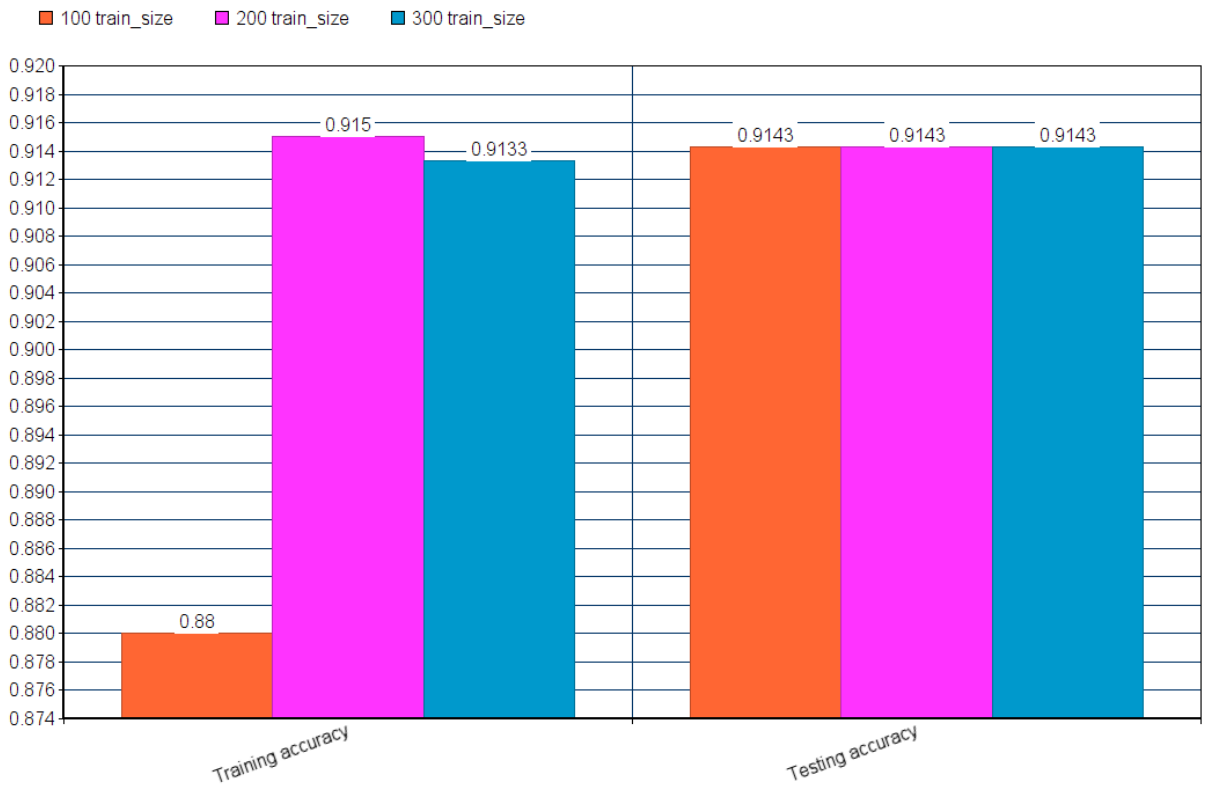

Figure 4.3 - Score Comparison
with limited max_depth to 2

Min samples leaf parameter is the minimum number of samples required to split an internal node. If the integer is provided, then min_samples_split is considered as the minimum number. If float is provided, then min_samples_split is a percentage and ceil (min_samples_split * n_samples) are the minimum number of samples for each split. If we choose small leaf size, it can take us more splits to reach it. And too deep tree means overfitting (model is unable to generalise its predictions to the larger population). On the contrary, if we choose too large leaf size, the tree will stop growing very soon and it will give us poor predictive performance. In the figure 4.4, we can see the score comparison with min_samples_leaf parameter set to 600.

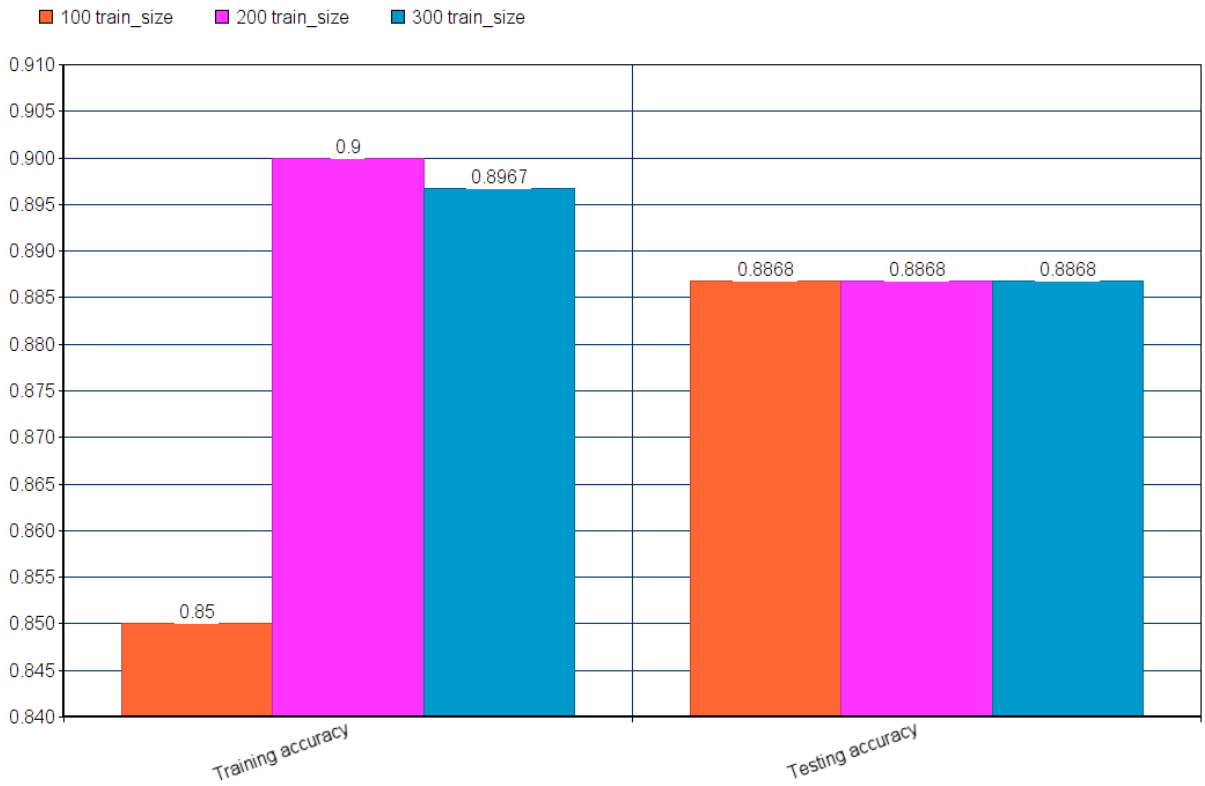

Figure 4.4 - Score Comparison with min_samples_leaf set to 600

Max features parameter is the number of features to consider when looking for the best split. When we are making split in decision tree, we are considering the features and trying to see which feature should we pick to get us the best split results (with which feature will we get the biggest information gain). Information gain is the amount of information that we want to gain by picking particular attribute. We can determine the criteria by which we define information gain with parameter 'criterion'. It can be either "gini" or "entropy". Entropy is defined as following:

$$H = -\sum p(x) \log p(x)$$

It is the sum of all the possible values that we might see times -1. As a result we get a value between 0 and 1, 0 meaning that there is no entropy and 1 meaning that there is full entropy.

If we limit max_features parameter to something low, it means that small amount of features will be considered, meaning that the model will not be able to understand the underlying relationships. It will result in high bias. In the Figure 4.5, we can see the results with 'max_features' parameter set to 1 (Note: results are the same for 100, 200 and 300 train set size, and we are getting pretty high scores because the other parameters are tuned and we are not dealing with large dataset).

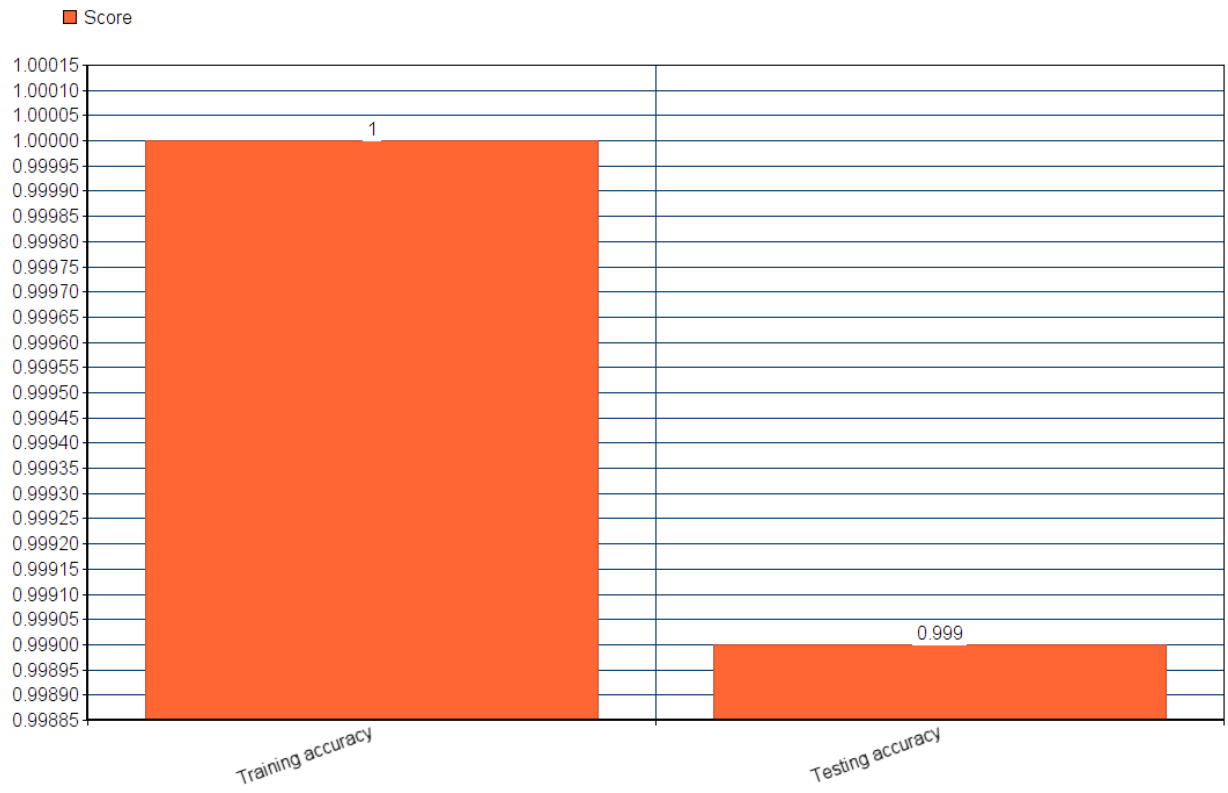

Figure 4.5 - Score Comparison with
'max_features' set to 1

Justification

Dr. Włodzisław Duch of Nicolas Copernicus University in Poland has perfectly classified the mushroom data by using a decision tree. We have approached the problem with Random Forests classifier which is an ensemble of decision trees, and managed to get the same scores. Thus, our results are as good as the benchmark results.

However, our model is useful in a limited domain. We will discuss how to solve the bigger problem (classifying real world mushrooms in real time) in 'Improvement' section.

V. CONCLUSION

Reflection

The process used for this project can be summarised using the following steps:

1. We have found the initial problem and the dataset needed for solving it
2. The dataset is fetched and preprocessed
3. We have found a benchmark for our classifiers
4. Three different classifiers (SVMs, SGD and Random Forests) were trained using the data with different training set sizes
5. Random Forests classifier was declared as the best performing
6. Chosen classifier was additionally tuned with Grid Search
7. The tuned classifier was tested on different training set sizes and it successfully gave perfect results

The most interesting aspects of this project were exploring the data and testing it on different classifiers. I have found this idea incredibly interesting especially because it gives a lot of room for improvement and additional experimenting, which will be discussed in the 'Improvement' section below.

Improvement

This project can be expanded to find a real world application. The next steps would be to create bigger and more relevant datasets by using different mushroom books and guides. Then to train Convolutional Neural Networks to extract information from the images through the Android/iOS app, and classify the mushroom accordingly. Misc information could be provided as well, such as the mushroom name, confidence in classification, as well as the market value and short description. A Pokemon GO-like application could be created for the mushrooms which would encourage people to take walks in the woods, collect and explore different types of mushrooms.

VI. REFERENCES

- https://en.wikipedia.org/wiki/Fungus
- https://en.wikipedia.org/wiki/Mushroom
- https://en.wikipedia.org/wiki/Mushroom_poisoning
- www.amjbot.org/content/98/3/426.full
- homepages.cae.wisc.edu/~ece539/fall13/project/Yan_rpt.pdf
- http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
- http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- https://en.wikipedia.org/wiki/Random_forest
- https://en.wikipedia.org/wiki/Ensemble_learning
- https://en.wikipedia.org/wiki/Kernel_method
- http://www.svm-tutorial.com/2015/06/svm-understanding-math-part-3/
- https://en.wikipedia.org/wiki/Stochastic_approximation
- https://en.wikipedia.org/wiki/Stochastic_gradient_descent