**Naïve Bayes Algorithm**

Strengths

* Easy to implement
* Deals with lot of features set

Weakness

* Can break in case of phrases comprising multiple words

**Support Vector Machine**

Classifier that attempts to maximize the margin of separation from nearest point across multiple classes.

If linear separation is not possible then the hyper-plane can be constructed by transforming the inputs.

Strengths

* Work well with complicated

Weakness

* Training time can be high
* Prone to overfitting in case of noise

SVM C parameter -> lower value of C param avoids overfitting to the test data. Decision boundary becomes more complex when a higher value of C is used i.e. classifier ends up overfitting to the existing data.

**Decision Trees**

**Entropy:** degree of randomness. Alsodegree of impurity in the sample

**Entropy =** Sum[-p \* log p] p = fraction of each input in a class ; log is to base 2

**Information Gain =** entropy(parent) – [weighted average] \* entropy(children)

**St:**

* Easy to use

**We:**

* Prone to over-fitting

**Bias-Variance Trade-off**

**High Bias:** Is not able to learn from data. Results from few features in the dataset.

**High Variance:** Fits perfectly to the test data. Results from too many features in the dataset leading to overfitting

**Ensemble Methods:** Meta and many classifiers built from(usually) decision trees

**Adaboost:**

An AdaBoost [1] classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

**Outliers**

**Unsupervised Learnings**

* **Clustering (k-means algorithm)**
  + Hill climbing algorithm and may result in a local minima

**Features Scaling**

* Outliers can mess up as the Xmax & Xmin may be skewed
* Feature scaling = (X-Xmin)/(Xmax-Xmin)
* SVM and K-means clustering algorithms are affected by features scaling

**Text Learning**

* Low information words: also known as stop words
* **NLTK (Natural Language Toolkit)**
* **Text Representation**

1. **Bag of Words**
2. **Stemming**
3. **TfIdf (Term Frequency , Inverse Document Frequency)**
   * Term Frequency(Tf) -> similar to bag of words
   * Inverse Document Frequency (Idf) -> weighted by how often the word occurs in the corpus

**Feature Selection**

* **K% percentile selection**
* **K feature**

**Regularization Regression**

**Lasso Regression**

**Method of penalizing extra features**

Minimize SSE + Lambda \* Co-efficient of regression

**Principal Component Analysis**

* PCA works only by rotation and translation
* Objective is to find the axis such that information loss is minimum and the axis is along the line that have maximum variance
* Projection of data points along the axis of maximum variance will minimize the information loss
* Principal components (PCs) are directions in the data that maximizes variance when the data points are projected on these directions
* More variance along a principal component, higher the component is ranked

**Machine Learning Approach**

**Train/test split -🡪 PCA -🡪 SVM/Decision Tree/ Naïve Bayes**

1. pca.fit(training\_features) -> extract principal components
2. pca.transform(training\_features) ->
3. classifier.fit(training\_features\_transform)
4. pca.transform(test\_features)
5. classifier.predict(test\_features\_transform)

**Cross Validation**

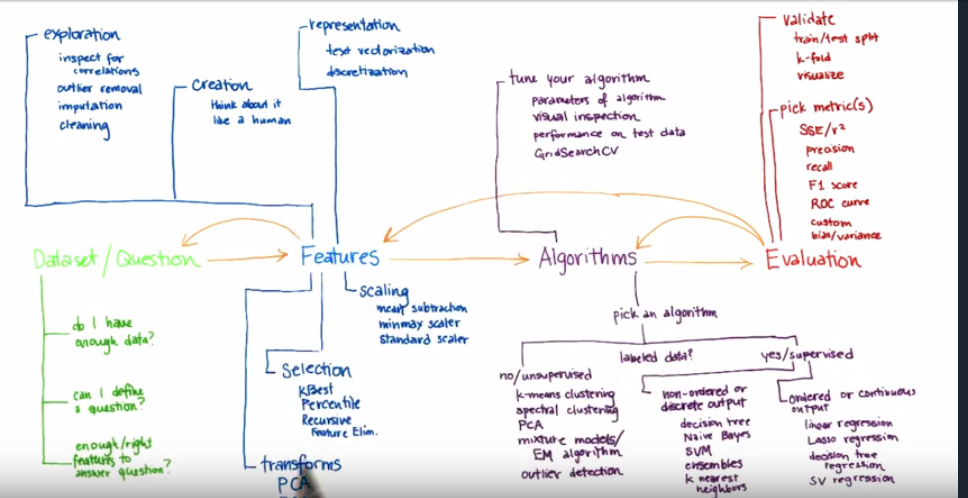
GridSearchCV

GridSearchCV is a way of systematically working through multiple combinations of parameter tunes, cross-validating as it goes to determine which tune gives the best performance. The beauty is that it can work through many combinations in only a couple extra lines of code.

Here's an example from the sklearn [**documentation**](http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html):

parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}  
svr = svm.SVC()  
clf = grid\_search.GridSearchCV(svr, parameters)  
clf.fit(iris.data, iris.target)

**Machine Learning Snapshot**



**Deep Learning**

**Multinomial Logistic Classification**

**Input -> Logit Scores(wx + b) -> Softmax probabilities -> Cross-entropy -> Hot Labels for classes**

**Gradient Descent**

* Slow

**Stochastic Gradient Descent**

* Fast
* Requires inputs to have mean = 0 and equal variance
* Initial weights should be randomized
* Momentum -> keep running average of the gradients calculated

Rectified Linear Units(RELU)

**L2 Norm: least square**

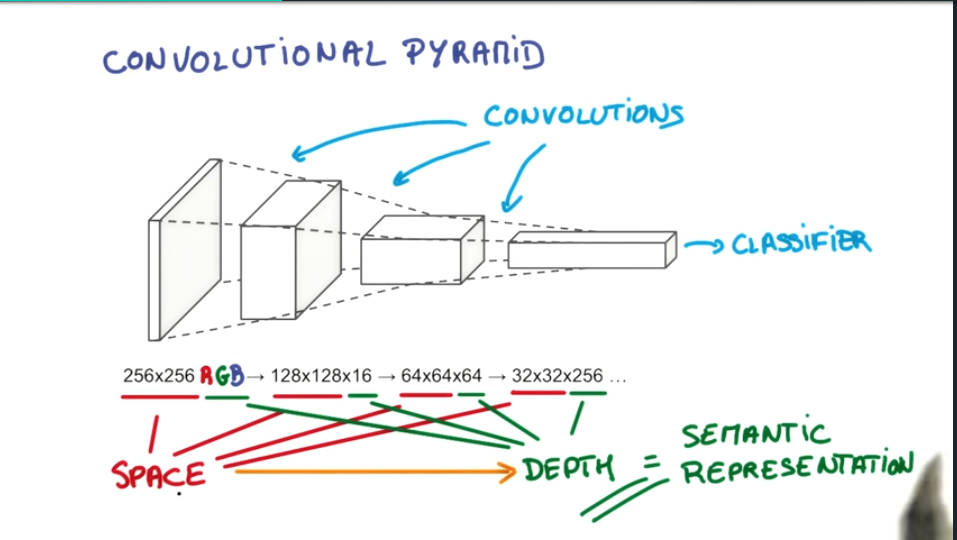
**Used in L2 Regularization**

New Loss = Loss + Beta \* 0.5 \* Weight^2

**Dropout technique** is used to avoid over-fitting. With Dropout technique, network is trained to be redundant as any weights can be set to 0 during an iteration. Setting a weight of node to 0 implies that the node is not active and is called dropout.

**Statistical invariant are objects that don’t change on average in time or space.**

**Convolutional Neural Network**



**Advanced Convo-net**

1. Pooling
   1. Max Pooling
2. 1\*1 Convolutions
3. Inceptions

**Tensor Flow**

A TensorFlow computation, represented as a dataflow graph.

A Graph contains a set of Operation objects, which represent units of computation; and Tensor objects, which represent the units of data that flow between operations.

**Accuracy Metrics**

1. **Accuracy** = Corrected Prediction for a class/ Total samples in the class
2. **Recall = (True Positive )/(True Positives + False Negatives)**
3. **Precision= (True Positive )/(True Positives + False Positive)**

**Python Libraries**

1. **numpy ->** numeric function
2. **scipy ->** science function
3. **sklearn ->** machine learning
   1. feature\_selection
   2. feature\_extraction
   3. decomposition for PCA
4. **nltk ->** natural language tool kit
   1. **Stemmer**
5. **re ->** regular expression
6. **pandas ->** dataframe
7. **pickle**