Platforms for competitions:

* Kaggle
* Driven data
* Crowd analityx
* Codalab
* datascienceChallenge.net
* Datascience.net
* single -competition sites (like KDD, VizDooM)
* analytics vidhya

People can also host their competitions on kaggle for free and invite people to participate.

**Families of ml algorithms:**

* Linear: logistic, SVM.

Implementation in scikit learn and wovpal wobbit

* Tree based: VERY USEFUL. Makes boxes

Decision trees: dividing the space into boxes and getting smaller and smaller like divide and conquor.

Most famous: Random forest, gradient boosting decision trees (GBDT).

Implementation: scikit learn. Better: XGBoost, MICROSOFT/LightGBM

* Knn
* Neural nets: implementation: tensorflow, keras, mxnet, pytorch (better to use this for faster computation and simplicity. Said by many to be better than TF), lasagne.

[**Explanation of random forest.**](https://www.datasciencecentral.com/profiles/blogs/random-forests-explained-intuitively)

Random forest is a collection of many decision trees. Instead of relying on a single decision tree, you build many decision trees say 100 of them. And you know what a collection of trees is called - a **forest**.

Why is it called **random** then?

Say our dataset has 1,000 rows and 30 columns. There are two levels of randomness in this algorithm:

At row level: Each of these decision trees gets a random sample of the training data (say 10%) i.e. each of these trees will be trained independently on 100 randomly chosen rows out of 1,000 rows of data. Keep in mind that each of these decision trees is getting trained on 100 randomly chosen rows from the dataset i.e they are different from each other in terms of predictions.

At column level: The second level of randomness is introduced at the column level. Not all the columns are passed into training each of the decision trees. Say we want only 10% of columns to be sent to each tree. This means a randomly selected 3 column will be sent to each tree. So for the first decision tree, may be column C1, C2 and C4 were chosen. The next DT will have C4, C5, C10 as chosen columns and so on.

Something similar happens in the random forest as well. The results from each of the tree are taken and the final result is declared accordingly. Voting and averaging is used to predict in case of classification and regression respectively.

### **When is a random forest a poor choice relative to other algorithms?**

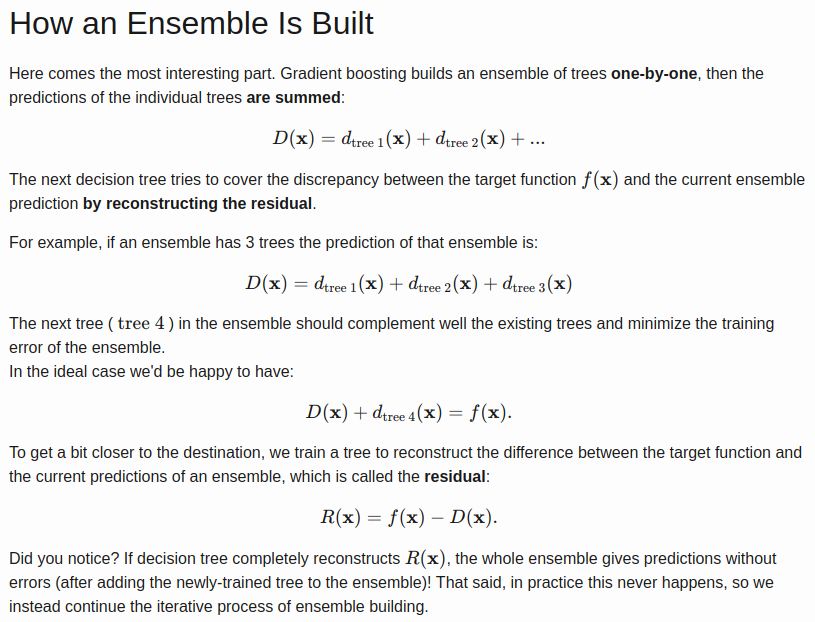
1. **Random forests don't train well on smaller datasets** as it fails to pick on the pattern. To simplify, say we know that 1 pen costs INR 1, 2 pens cost INR 2, 3 pens cost INR 6. In this case, linear regression will easily estimate the cost of 4 pens but random forests will fail to come up with a good estimate.
2. **There is a problem of interpretability with random forest.** You can't see or understand the relationship between the response and the independent variables. Understand that random forest is a predictive tool and not a descriptive tool. You get variable importance but this may not suffice in many analysis of interests where the objective might be to see the relationship between response and the independent features.
3. The **time taken to train random forests** may sometimes be too huge as you train multiple decision trees. Also, in the case of a categorical variable, the time complexity increases exponentially. For a categorical column with n levels, RF tries split at 2^n -1 points to find the maximal splitting point. However, with the power of H2O we can now train random forests pretty fast. You may want to read about H2O at [H2O in R explained](http://manishbarnwal.com/blog/2017/03/28/h2o_with_r/).
4. In the case of a regression problem, **the range of values response variable can take** is determined by the values already available in the training dataset. **Unlike linear regression, decision trees and hence random forest can't take values outside the training data.**

### **What are the advantages of using random forest?**

1. **Since we are using multiple decision trees, the bias remains same as that of a single decision tree. However, the variance decreases and thus we decrease the chances of overfitting. I have explained bias and variance intuitively at** [**The curse of bias and variance**](http://manishbarnwal.com/blog/2017/02/08/the_curse_of_bias_and_variance/)**.**
2. When all you care about is the predictions and **want a quick and dirty way out**, random forest comes to the rescue. You don't have to worry much about the assumptions of the model or linearity in the dataset.

# [Gradient Boosting](http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html)

The above gives a nice visualization tutorial as to how having multiple decision trees can help to create the desired function.



So we have parameters like

depth of a tree - how complex function a single tree can handle

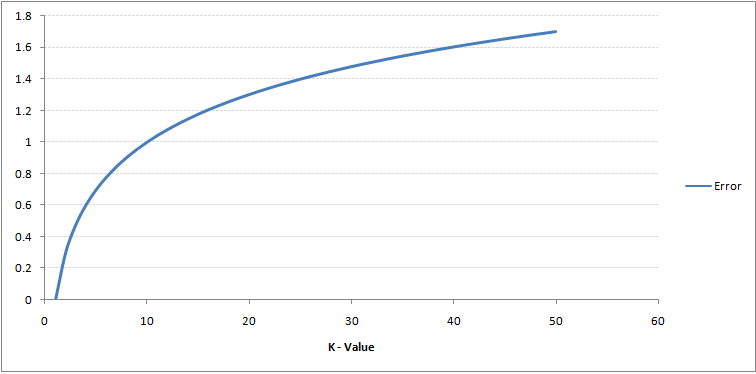
Number of trees - how many trees we want to ensemble to get the final result.

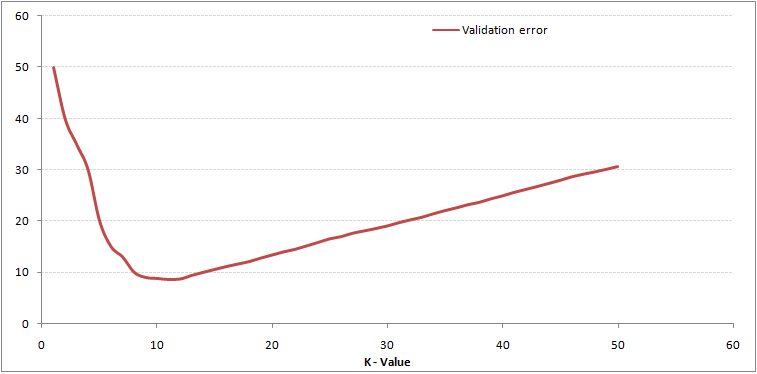
* before a tree is added to an ensemble, it's predictions are multiplied by some factor
* this factor (called \eta*η* or learning rate) is an important parameter of GB ( \eta = 0.3*η*=0.3 in this demo)
* if we set number of trees to 10 and vary the depth: we notice that as the depth gets higher, the residual gets smaller, but it's also more noisy
* discontinuities ('spikes') appear at those points where a decision tree split
* the larger the learning rate — the larger the 'step' made by a single decision tree — and the larger the discontinuity
* **in practice GBDT is used with small learning rate ( 0.01 < \eta < 0.1 , 0.01<*η*<0.1 ) and large number of trees to get the best results.**

KNN

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry.

The training error rate and the validation error rate are two parameters we need to access on different K-value. Following is the curve for the training error rate with varying value of K :

As you can see, the error rate at K=1 is always zero for the training sample. This is because the closest point to any training data point is itself.Hence the prediction is always accurate with K=1. If validation error curve would have been similar, our choice of K would have been 1. Following is the validation error curve with varying value of K:

This makes the story more clear. At K=1, we were overfitting the boundaries. Hence, error rate initially decreases and reaches a minima. After the minima point, it then increase with increasing K. To get the optimal value of K, you can segregate the training and validation from the initial dataset. Now plot the validation error curve to get the optimal value of K. This value of K should be used for all predictions.

**Random forest vs Extra trees**

The main difference between random forests and extra trees (usually called extreme random forests) lies in the fact that, instead of computing the locally optimal feature/split combination (for the random forest), for each feature under consideration, a random value is selected for the split (for the extra trees).

This leads to more diversified trees and less splitters to evaluate when training an extremely random forest.

Important question:

Consider two cases:

1. We fit two RandomForestClassifiers 500 trees each and average their predicted probabilities on the test set.
2. We fit a RandomForestClassifier with 1000 trees and use it to get test set probabilities.

All hyperparameters except number of trees are the same for all models.

The quality of predictions in the *case 1* will be roughly the same as the quality of the predictions in the *case 2*

# Overview of methods

* [Scikit-Learn (or sklearn) library](http://scikit-learn.org/)
* [Overview of k-NN](http://scikit-learn.org/stable/modules/neighbors.html) (sklearn's documentation)
* [Overview of Linear Models](http://scikit-learn.org/stable/modules/linear_model.html) (sklearn's documentation)
* [Overview of Decision Trees](http://scikit-learn.org/stable/modules/tree.html) (sklearn's documentation)
* Overview of algorithms and parameters in [H2O documentation](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science.html)

# [Additional Tools](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science.html)

* [Vowpal Wabbit](https://github.com/JohnLangford/vowpal_wabbit) repository
* [XGBoost](https://github.com/dmlc/xgboost) repository
* [LightGBM](https://github.com/Microsoft/LightGBM) repository
* [Interactive demo](http://playground.tensorflow.org/) of simple feed-forward Neural Net
* Frameworks for Neural Nets: [Keras](https://keras.io/)[,](about:blank)[PyTorch](http://pytorch.org/)[,](about:blank)[TensorFlow](https://www.tensorflow.org/)[,](about:blank)[MXNet](http://mxnet.io/), [Lasagne](http://lasagne.readthedocs.io/)
* [Example from sklearn with different decision surfaces](http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html)
* [Arbitrary order factorization machines](https://github.com/geffy/tffm)

GBDT NOTEBOOK:

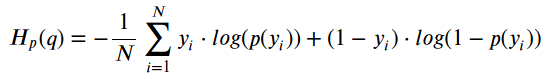
On a trained model .score() for accuracy given params test and train split.

clf.score(X\_test, y\_test)

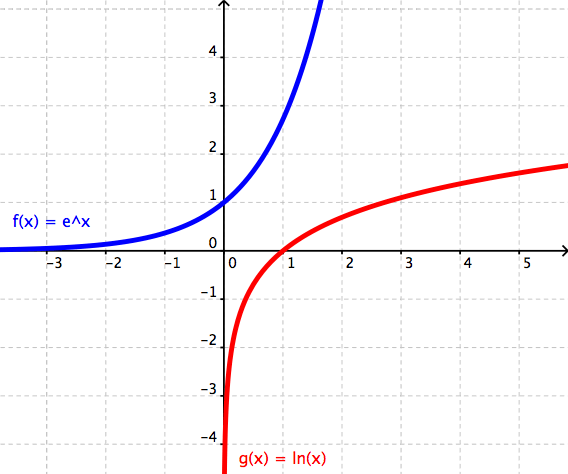
Log\_loss

[Link](https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a) tutorial link for log loss

[Visual tutorial by chris](http://colah.github.io/posts/2015-09-Visual-Information/) for information theory.!



Y and 1-y given to switch between classes. Log is used as the metric to define the magnitude of error. Meant for only two class distinction or binary classification.



log(x) and e^x

Also called **binary cross-entropy.**

# Overview of methods

* [**Scikit-Learn (or sklearn) library**](http://scikit-learn.org/)
* [**Overview of k-NN**](http://scikit-learn.org/stable/modules/neighbors.html) **(sklearn's documentation)**
* [**Overview of Linear Models**](http://scikit-learn.org/stable/modules/linear_model.html) **(sklearn's documentation)**
* [**Overview of Decision Trees**](http://scikit-learn.org/stable/modules/tree.html) **(sklearn's documentation)**
* **Overview of algorithms and parameters in** [**H2O documentation**](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science.html)

# [Additional Tools](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science.html)

* [**Vowpal Wabbit**](https://github.com/JohnLangford/vowpal_wabbit) **repository**
* [**XGBoost**](https://github.com/dmlc/xgboost) **repository**
* [**LightGBM**](https://github.com/Microsoft/LightGBM) **repository**
* [**Interactive demo**](http://playground.tensorflow.org/) **of simple feed-forward Neural Net**
* **Frameworks for Neural Nets:** [**Keras**](https://keras.io/)[**,**](about:blank)[**PyTorch**](http://pytorch.org/)[**,**](about:blank)[**TensorFlow**](https://www.tensorflow.org/)[**,**](about:blank)[**MXNet**](http://mxnet.io/)**,** [**Lasagne**](http://lasagne.readthedocs.io/)
* [**Example from sklearn with different decision surfaces**](http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html)
* [**Arbitrary order factorization machines**](https://github.com/geffy/tffm)

[**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**](https://github.com/geffy/tffm)

## StandCloud Computing:

* [AWS](https://aws.amazon.com/), [Google Cloud](https://cloud.google.com/), [Microsoft Azure](https://azure.microsoft.com/)

## [AWS spot option:](https://azure.microsoft.com/)

* [Overview of Spot mechanism](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html)
* [Spot Setup Guide](http://www.datasciencebowl.com/aws_guide/)

## [Stack and packages:](http://www.datasciencebowl.com/aws_guide/)

* [Basic SciPy stack (ipython, numpy, pandas, matplotlib)](https://www.scipy.org/)
* [Jupyter Notebook](http://jupyter.org/)
* [Stand-alone python tSNE package](https://github.com/danielfrg/tsne)
* [Libraries to work with sparse CTR-like data:](https://github.com/danielfrg/tsne) [LibFM](http://www.libfm.org/), [LibFFM](https://www.csie.ntu.edu.tw/~cjlin/libffm/)
* [Another tree-based method: RGF (](https://www.csie.ntu.edu.tw/~cjlin/libffm/)[implemetation](https://github.com/baidu/fast_rgf), [paper](https://arxiv.org/pdf/1109.0887.pdf))
* Python distribution with all-included packages: [Anaconda](https://www.continuum.io/what-is-anaconda)
* [Blog "datas-frame" (contains posts about effective Pandas usage)](https://tomaugspurger.github.io/)

**Feature preprocessing and generation**

**Numeric feature**

**Categorical features**

**Id – unique determining**

**Date features**

**Text features**

Categorical feature: One hot encoding for model to better learn the state change of a categorical model rather than taking its magnitude and creating some inbuilt relation to it.

Random forest doesn’t need this one hot encoding due to decision splits inside trees.

Things like when days are given, to add a column for week, for the model to easily understand the data and its linear trend.

**Numeric feature:**

Model types:

Tree based

Non tree based

Decision trees: tries to find best split. No change seen in model if data scaling is done.

K Nearest neighbor, linear models and neural nets are dependent on data scaling and that’s why normalization is done to make sure all the features are on same scale or else this scaling can be used to give weighted prioritization. Like the bigger the feature, more important it is for KNN.

Gradient descent methods works good with proper scaling.

Sklearn.preprocessing.MinMaxScalar

(x-x.min())/(x.max() – x.min()) converts to the range of [0, 1]

(x-x.mean())/x.std() converts mean to 0 and std to 1

This way non tree based methods can be corrected. Also we can

Outlier: even a single outlier can change the prediction values dramatically.

To solve this we can clip values to lower and upper bounds and is called Winsorization.

**Rank transformation:**

Given an array lets say sorted. Rank transformation will put indices in place of the numbers. As the first element is the smallest and so its rank is 0 then second smallest is given rank 1 and so on. Used in models if we don’t have time for outlier removal.

Scipy.stats.rankdata

On non tree models: using log transforms or raising to the power of the feature can help the model to learn the model better and increase the performance and result significantly.

**Feature generation**

If we lets say have squared area and price. Add the column of price per squared area for model to better understand the data.

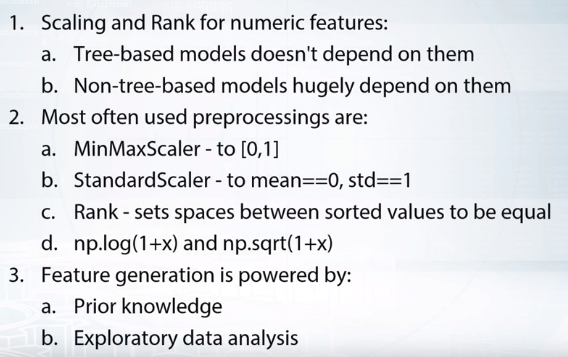
Rather than model figuring out in the long run. Give these relations prior itself.

Even though GBDT performs well it faces difficulty with multiplication and division. So adding addition features like this can help a lot.

Another example can be separating and specifically mentioning in a new column only the fractional value of the price for the item.

Giving this increases a feature focusing on how the fractional values only depend on the target value and so adds to better performance.

**SUMMARY:**



**CATEGORICAL AND ORDINAL FEATURES**

**label encoding:** map unique category values to different numbers.

this works good for trees as they take it as category of numbers

non-tree based methods on the other hand takes it like magnitude

which created problem.

todo this we have:

sklearn.preprocessing.LabelEncoder

Pandas.factorize

**Frequency encoding:** if a column has 3 categories. then percentage

of categories is there new label [a, b, c] -> [0.7, 0.1, 0.2] in

the whole column.

This also helps non-tree based methods also if

the target is dependent on the frequency of occurence in column.

This can be a bit of a problem if two classes have the same

distribution in the data.

**label encoding for non-tree based method is: one-hot encoding.**

in which for each category we create a new column.

note:

if a lot of unique values are there in a categorical feature

this can lead to slowing down of tree based methods. also the

columns will have hardly any non zaro values. To store this

we can have sparse matrices, which directly stores non zero

values.

if you think two features categories can clash in one hot

encoding. Just append some string to one to make sure both

the categories one hot encoding is done.

label and frequency encoding are often used for tree based methods.

one hot is used for non tree based models.

**Date time and coordinates:**

DATETIME

1) add features like sec, min, hour

2) time since:

1) a reference time - column independent

2) sum from previous rows and to keep adding.

difference between dates

3) subtracting columns for new data. like predicting churn

The duration since last purchase.

COORDINATES

1) add distance column via coordinates.

2) center of cluster

3) aggregated stats.

**Handling missing values:**

To find out missing values you can plot hist and see if some values just lie the fartherst and can

Be considered as replaced inplace of missing values.

**Missing values:**

1. Replace by arbitrary value of -1 999 etc
2. Replace by mean or median: beneficial for simple linear models and neural nets. Not for trees
3. Trying to reconstruct value somehow.

Using 1) or 2) above can lead to huge error as putting -999 and later replacing these by mean of category will lend to take the whole graph to the left as linear model takes it as magnitude. Trying to put mean values can also lead to huge errors during. Same is true for replacing by mean or median.

Solution is to simply ignore missing values during calculations.

We can also put outliers as missing values.

Also putting missing values in the test rather than train can also help.

XGBoost can handle missing numbers directly.

# Question 1

## Suppose we have a feature with all the values between 0 and 1 except few outliers larger than 1. What can help us to decrease outliers' influence on non-tree models?

Correct answers:

* Apply rank transform to the features. Yes, because after applying rank distance between all adjacent objects in a sorted array is 1, outliers now will be very close to other samples.
* Apply np.log1p(x) transform to the data. This transformation is non-linear and will move outliers relatively closer to other samples.
* Apply np.sqrt(x) transform to the data. This transformation is non-linear and will move outliers relatively closer to other samples.
* Winsorization. The main purpose of winsorization is to remove outliers by clipping feature's values.

Incorrect answers:

* StandardScaler. No, despite feature will be scaled, relative distances between outliers and other values still will be huge.
* MinMaxScaler. No, despite feature will be scaled, relative distances between outliers and other values still will be huge.

# Question 2

## Suppose we fit a tree-based model. In which cases label encoding can be better to use than one-hot encoding?

Correct answers:

* When categorical feature is ordinal. Correct! Label encoding can lead to better quality if it preserves correct order of values. In this case a split made by a tree will divide the feature to values 'lower' and 'higher' that the value chosen for this split.
* When we can come up with label encoder, that assigns close labels to similar (in terms of target) categories. Correct! First, in this case tree will achieve the same quality with less amount of splits, and second, this encoding will help to treat rare categories.
* When the number of categorical features in the dataset is huge. One-hot encoding a categorical feature with huge number of values can lead to (1) high memory consumption and (2) the case when non-categorical features are rarely used by model. You can deal with the 1st case if you employ sparse matrices. The 2nd case can occur if you build a tree using only a subset of features. For example, if you have 9 numeric features and 1 categorical with 100 unique values and you one-hot-encoded that categorical feature, you will get 109 features. If a tree is built with only a subset of features, initial 9 numeric features will rarely be used. In this case, you can increase the parameter controlling size of this subset. In xgboost it is called *colsample\_bytree,* in sklearn's Random Forest *max\_features.*

Incorrect answers: None

# Question 3

## Suppose we fit a tree-based model on several categorical features. In which cases applying one-hot encoding can be better to use than label-encoding?

Correct answers:

* If target dependence on the label encoded feature is very non-linear, i.e. values that are close to each other in the label encode feature correspond to target values that aren't close. Correct! If this feature is important, a tree would try to make a lot of splits and select each feature' value in a category on its own. But because tree is build in a greedy way, it can be hard to select one important value in label encoded vector. This won't be the problem if you use OHE.

Incorrect answers:

* When the feature have only two unique values. Incorrect. In this case both one-hot encoding and label encoding will produce similar columns.

# Question 4

## Suppose we have a categorical feature and a linear model. We need to somehow encode this feature. Which of the following statements are true?

Correct answers:

* Depending on the dataset either of label encoder or one-hot encoder could be better. Correct! Although one-hot-encoding is usually gives better results in this case, we can come up with examples when one-hot-encoded feature will not lead to a better performance of a linear model.

Incorrect answers:

* Label encoding is always better than one-hot encoding. Incorrect. Usually the dependence between the feature and the target is non-linear. In this case a linear model will not be able to utilize Label Encoded feature efficiently.
* One-hot encoding is always better than label encoding. Incorrect. Consider the toy example when the label encoded feature and the target are equal. In this case a linear model on this feature will have the perfect quality.

**Additional material:**

## Feature preprocessing

* [Preprocessing in Sklearn](https://www.coursera.org/learn/machine-learning/lecture/xx3Da/gradient-descent-in-practice-i-feature-scaling)
* [Feature Scaling and the effect of standardization for machine learning algorithms](http://sebastianraschka.com/Articles/2014_about_feature_scaling.html)

[------------------------------------------------------------------------------------------------------------](http://sebastianraschka.com/Articles/2014_about_feature_scaling.html)

**Feature extraction from texts and images**

**TEXT**

**PREPROCESS TEXT:**

**LOWERCASE:** Make sure to lower case text to avoid, same words taken differently. Count vectorizer does this by default.

**Stemming**: usually refers to a heuristic process that chops off ending of words and thus unite duration of related words like democracy, democratic, and democratization, producing something like, democr, for each of these words.

**Lemmatization:** on the hand, usually means that you have want to do this carefully using knowledge or vocabulary, and morphological analogies of force,returning democracy for each of the words below.

**Stopwords: max\_df in countVectorizer uses this to consider the number with this frequency as stopwords.**

NLTK has a stop word library.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Bag of words:** count of unique word is taken in columns.

sklearn.feature\_extraction.text.CountVectorizer.

Also make sure to normalize data at the end. You can simply take the:

**term frequency for this: Word\_count / (total\_number of words in the column)**

**Inverse document freq: idf = np.log(x.shape[0] / (x>0).sum(0))**

To supress common words in the data and give imp to rara ones.

**sklearn.feature\_extraction.text.TfidfVectorizer**

You can use a variant of tf-idf with n-grams. Uni bi tri nd all.

sklearn.feature\_extraction.text.CountVectorizer: Ngram\_range, analyzer

**Embeddings :**

Words with same context will be close to each other.

Also arithematic operation is possible. King + woman - man = queen

You can use pretrained models.

**Words: word2vec, glove, fastText**

**For Sentences: doc2vec**

Bag of words give large vectors where as embeddings gives smaller ones.

Both bag of words and embedding approach can be used.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**IMAGES**

Fine tuning: using pre-trained models and fitting them. U should use models trained on similar category and it can help a lot when u have very less data.

Fine tuning by adding our fully connected layer should be trained on learning rate a thousand times less than the initial learning rate of the pret-erained model.

**Image augmentation:**

Rotation of images.

Adding patches

And other ways find on the internet.

Quiz

## Select true statements about n-grams.

Correct answers:

* N-grams can help utilize local context around each word. Correct, because ngrams encode sequences of words.
* N-grams features are typically sparse. Correct. Ngrams deal with counts of words occurrences, and not every word can be found in a document. For example, if we count occurrences of words from an english dictionary in our everyday speech, a lot of words won't be there, and that is sparsity.

Incorrect answers:

* N-grams always help increase significance of important words. No, ngrams deals with words occurrences and not their importance.
* Levenshteining should always be applied before computing n-grams. Although, there is Levenshtein distance, there is no such thing as Levenshteining.

# Question 2

## Select true statements.

Correct answers:

* Bag of words usually produces longer vectors than Word2vec. Correct! Number of features in Bag of words approach is usually equal to number of unique words, while number of features in w2v is restricted to a constant, like 300 or so.
* Semantically similar words usually have similar word2vec embeddings. Correct. This is one of the main benefits of w2v in competitions.

Incorrect answers:

* Meaning of each value in BOW matrix is unknown. Incorrect. Meaning of a value in BOW matrix is the number of a word's occurrences in a document.
* You do not need bag of words features in a competition if you have word2vec features. Incorrect. Both approaches are valuable and you should try to utilize both of them.

# Question 3

## Suppose in a new competition we are given a dataset of 2D medical images. We want to extract image descriptors from a hidden layer of a neural network pretrained on the ImageNet dataset. We will then use extracted descriptors to train a simple logistic regression model to classify images from our dataset.

## We consider to use two networks: ResNet-50 with imagenet accuracy of X and VGG-16 with imageNet accuracy of Y (X < Y). Select true statements.

Correct answers:

* It is not clear what descriptors are better on our dataset. We should evaluate both. Correct! This depends on the a specific dataset and a specific task, so you should evaluate both!

Incorrect answers:

* With one pretrained CNN model you can get only one vector of descriptors for an image. Incorrect. With one CNN you can get different descriptors from different layers.
* Descriptors from ResNet 50 will always be better than the ones from VGG-16 in our pipeline. Incorrect. Although, ResNet50 shows better performance on Imagenet, this depends on the a specific dataset and a specific task.
* For any image descriptors from the last hidden layer of ResNet-50 are the same as the descriptors from the last hidden layer of VGG-16. Incorrect in general. Moreover it is hard to come up with an image that will have the same descriptors in both networks.
* Descriptors from ResNet-50 and from VGG-16 are always very similar in cosine distance. Incorrect. This depends on the a specific dataset and a specific task.

# Question 4

## Data augmentation can be used at (1) train time (2) test time

Correct answer:

True, True. Data augmentation can be used (1) to increase the amount of training data and (2) to average predictions for one augmented sample.

## Feature extraction from text

### Bag of words

* [Feature extraction from text with Sklearn](http://scikit-learn.org/stable/modules/feature_extraction.html)
* [More examples of using Sklearn](https://andhint.github.io/machine-learning/nlp/Feature-Extraction-From-Text/)

### [Word2vec](https://andhint.github.io/machine-learning/nlp/Feature-Extraction-From-Text/)

* [Tutorial to Word2vec](https://www.tensorflow.org/tutorials/word2vec)
* [Tutorial to word2vec usage](https://rare-technologies.com/word2vec-tutorial/)
* [Text Classification With Word2Vec](http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/)
* [Introduction to Word Embedding Models with Word2Vec](https://taylorwhitten.github.io/blog/word2vec)

[NLP Libraries](https://taylorwhitten.github.io/blog/word2vec)

* [NLTK](http://www.nltk.org/)
* [TextBlob](https://github.com/sloria/TextBlob)

## [Feature extraction from images](https://github.com/sloria/TextBlob)

### Pretrained models

* [Using pretrained models in Keras](https://keras.io/applications/)
* [Image classification with a pre-trained deep neural network](https://www.kernix.com/blog/image-classification-with-a-pre-trained-deep-neural-network_p11)

### [Finetuning](https://www.kernix.com/blog/image-classification-with-a-pre-trained-deep-neural-network_p11)

* [How to Retrain Inception's Final Layer for New Categories in Tensorflow](https://www.tensorflow.org/tutorials/image_retraining)
* [Fine-tuning Deep Learning Models in Keras](https://flyyufelix.github.io/2016/10/08/fine-tuning-in-keras-part2.html)