Capstone Project

Udacity Machine Learning Engineer Nanodegree

Git repo:
https://github.com/joshnewnham/udacity_machine_learning_engineer_nanodegree_capstone

Define

This project is based on the paper **How Do Humans Sketch Objects?** published in journal ACM Transactions on Graphics (Proc. SIGGRAPH 2012) by **Mathias Eitz, James Hays and Marc Alexa**. Research available: http://cybertron.cg.tu-berlin.de/eitz/projects/classifysketch/

The paper was comparing sketch recognition performance of people and their recognition system, citing people accurately identified 73% of the object category from sketches while in comparison, their system achieved 56% accuracy. The complete crowd-sourced dataset of sketches to the community was released under Creative Commons.

This project follows their agenda (and general approach) of trying to **accurately classify a user's sketch to a trained category** using a subset of their data. The motivation for choosing this was due to its close relationship to my work (Design and Machine Learning/Intelligent Interfaces).*

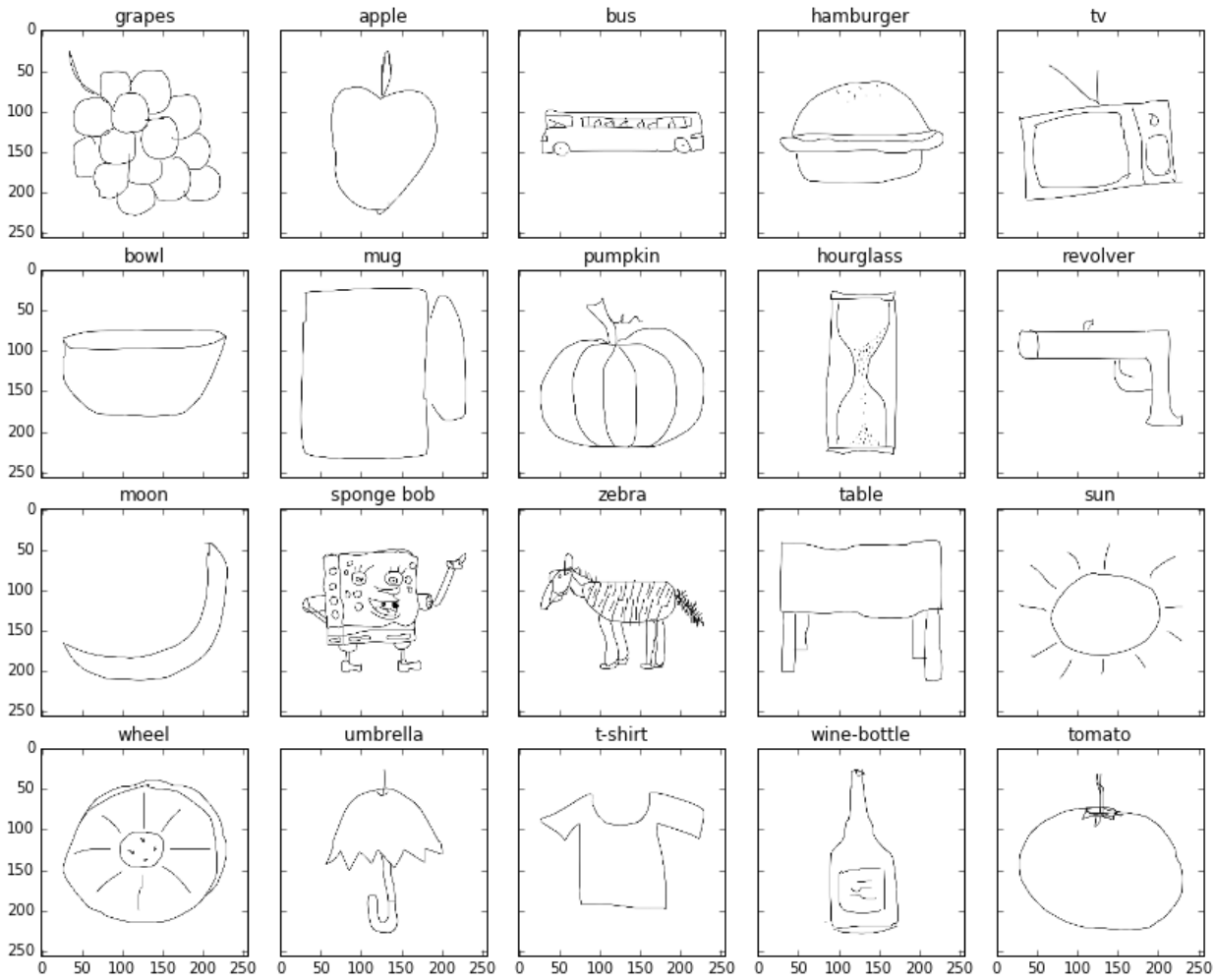
Analyze

Dataset

The dataset is available http://cybertron.cg.tu-berlin.de/eitz/projects/classifysketch/sketches_png.zip and released publicly under the Creative Commons license.

The **full dataset consist of 250 categories** each containing 80 samples (20000). SVG and PNG versions are made available, where each sample (PNG) consists of a 1111x1111 sketch of the relevant category object, files organised into their categories within sub-directories.

Because of computing demands a **subset of 113 categories** were selected, this list can be found in the accompanying subset_labels.csv file. For illustrative purposes, below shows 2 images from categories.



Test Set

A randomly generated test set was created to be used for evaluation throughout the process. This test set is defined in the accompanied file test_set.json, containing a dictionary structure for each category and corresponding file names. 8 (10%) of the images were selected from each category to form this test set.

Evaluation

Performance of the features and model are based on:

Log Loss (Logarithmic Loss):
Log Loss quantifies the accuracy of a classifier by penalising false classifications i.e. minimising the Log Loss is basically equivalent to maximising the accuracy.

Log Loss heavily penalises classifiers that are confident about an incorrect classification e.g. if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large.
More details can be found: https://www.kaggle.com/wiki/LogarithmicLoss

Accuracy score:
The fraction of correct predictions.

Accuracy of the top 10:
For each k, how accurate is the model i.e. when k is set to 2, then consider a correct prediction if the predicted probability includes the correct prediction within the top 2 predictions.

Implement

As described above, the goal of this project is to be able to correctly assign a learn't category from a given user's sketch, the training set includes 113 categories, for each 72 available images for learning their features. In this notebook I introduce these features and the extraction process along with some exploration of (what I have called) feature tuning i.e. tuning the features to increase performance of classification.

Image pre-processing

Prior to extracting features from the images, images were first pre-processed - the original images were 1111x1111 PNG's, with varying scales. To mitigate variance in scale, each images padding was cropped out and the image uniformly rescaled to 256x256.

Feature Extraction - Bag of Visual Words

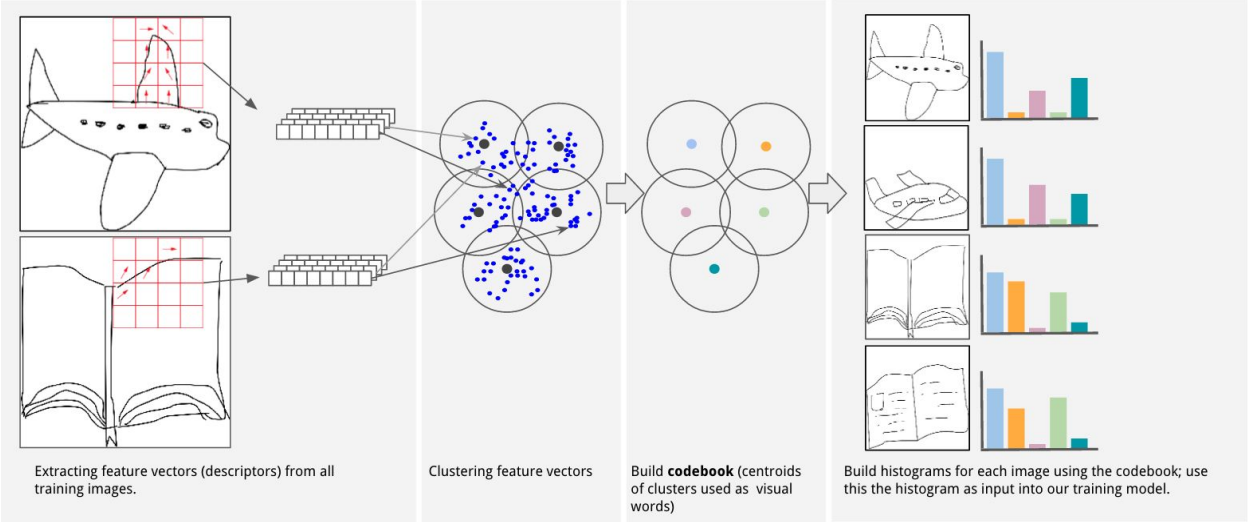
In text analysis, an effective approach of classifying a document is by using the concept of Bag of Words (BoWs) where a global dictionary is created including all relevant domain words (also known as tokens and features) then building a vector feature for each document consisting of a count for each word in the document (aka Bag of Words). A model is then trained to identify the frequency of words most applicable for each category e.g. technology related news may consist of a higher count for the words 'Software' and 'Pivot' than other news categories.

A similar approach can be taken when classifying images but instead of words being your tokens, you use patch descriptors from your image ie the image is decomposed into a feature vectors for each patch. A global vocabulary is constructed using all the feature vectors extracted from all images then reduced using clustering. The centroids of these clusters become your **visual words** (or **code labels**) of your vocabulary, this vocabulary is known as a **codebook**.

The **codebook** was created using a batch K-Means clustering algorithm (sklearn.cluster.MiniBatchKMeans) - the motivation behind using this (batches) as opposed to KMeans was convergence time due to the high number of clusters (and probably my laptop's computing resources). KMeans method was set to **k-means++** (selects initial cluster centers for k-mean clustering in a smart way to speed up convergence) and batch size (size of the mini batches) was set to 100.

The **codebook** (collection of visual words) is then used to build a visual word histograms (**bag of visual words**) as their feature vector for each image (process called vector quantization), it is of the premise that this histograms can uniquely describe a category type through a trained classifier.

The following figure illustrates the above workflow.



Feature Vectors (/Descriptors)

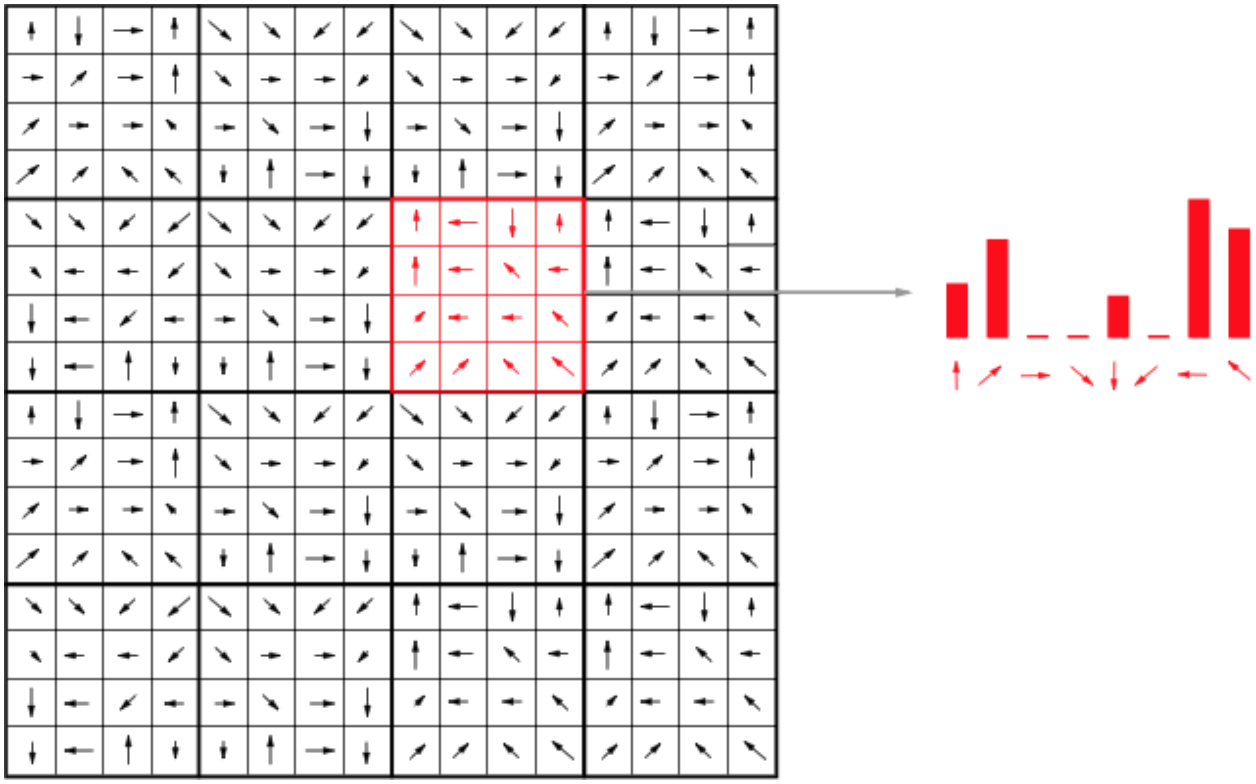
Feature Vectors (or, as commonly referred to in Computer Vision, Descriptors) are encodings from a patch of an image used to describe some characteristic of the patch (and collectively the image) i.e. you could describe a patch of a image from its average RGB (Colour) channels.

Because sketches are generally sparse in terms of detail, I create a grid of **keypoints**, this to ensure I extract all meaningful amounts of information from the image.

SIFT (Scale-invariant feature transform) was used to extract the feature vectors (descriptors) from each of the keypoints. SIFT was created by David Lowe and described fully in his original paper at http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.

The feature vector is built by using a 16x16 patch (centered around each keypoint), this patch is decomposed into 4x4 pixel tiles (x16 tiles for each patch) where for each tile is described as a gradient orientations and magnitude histogram (computed across 8 bins, at 45 degrees). The final result is a **128 bit feature vector** for each keypoint (16x8).

The following figure illustrates the decomposing of a patch into a 8-bin gradient orientation histogram.



Feature engineering (exploration)

In this section I experiment by varying properties used to describe the image - the classification model[1] is held constant (along with the other properties) to observe the influence adjusting each property has on the performance.

Admittedly properties are interdependent but given the constraints of time, computing power and patience, this factor was ignored (to some extend).

Properties

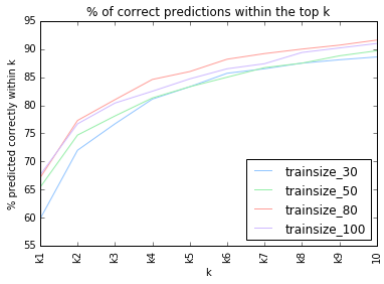
- Influence of training size i.e. is there a point where no more information is gained through samples.

[1] Linear SVM; selected as it provided the better performance during initial training and evaluation on.

- Number of cluster (aka visual words) i.e. how many visual words are required to effectively capture all the required information.
- Window resolution and Window Overlay both have to do with the density of the grid of keyboards i.e. how dense does this grid need to be to capture all necessary information.

Training set size

Exploring what influence the training size has on the performance of our model.

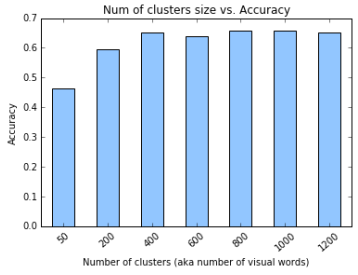


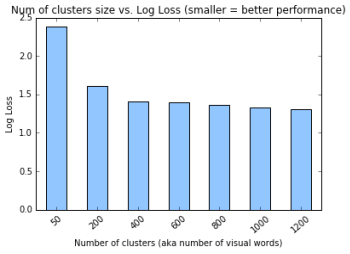
Accuracy | Cluster Size | K Accuracy %

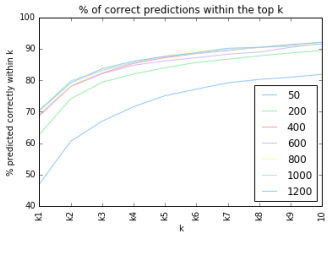
NB: all subsequent tests are performed on 50% of the training set to increase responsiveness whilst experimenting with the properties.

Number of clusters

Clusters in this context is the number of visual words we create for each image feature vector to be assigned to. Similar to compression, the lower the number of clusters the more detail we lose, but possibly gain in generalisation/pattern identification. In this section we vary the number of clusters and compare the performance.



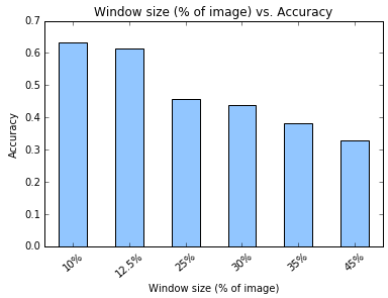


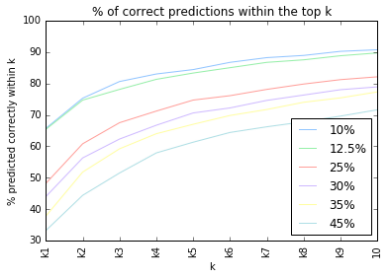


Accuracy | Cluster Size | K Accuracy %

Window Resolution

SIFT (and all considered feature extractors) can be confined to a window such to describe segments of the sketches. Here we vary the window size to determine what it's influence has on the performance of our model.

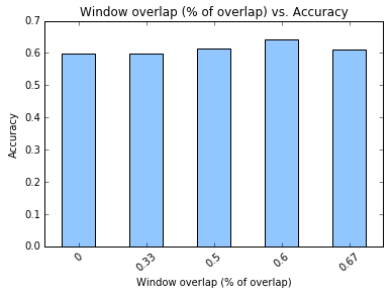


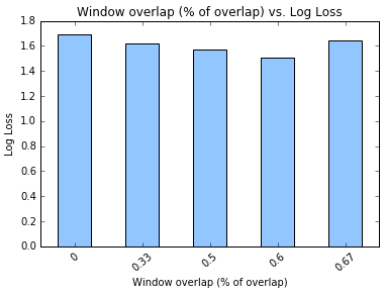


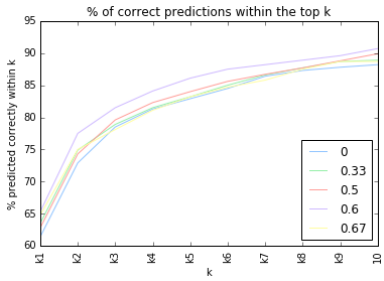
Accuracy | Cluster Size | K Accuracy %

Window overlap

This parameter determines how much overlap there is for each window (or step size 1-1/N where N is the overlap) i.e. N=2 would mean there is a 1/2 overlap between all windows as the window is slide along.







Accuracy | Cluster Size | K Accuracy %

Summary

In regards to **training size**, as expected, the more training data available increases the performance, up to a point - appears to plateau around 80% - this could mean the additional

extracted features don't add any more information or that the other parameters of our training are limiting the models ability to learn these additional features.

The number of **clusters** (visual words) beyond 400 resulted in marginal performance gains, for this reason and to keep things as simple as possible, a visual word dictionary of approx. 500 will be chosen.

Increasing **window size** reduced performance significantly, possibly loosing to much detail of the sketches. A window size of around 10-12.5% appears to give optimal results but obviously incurs additional computational cost.

Our last property we explored for extracting features was **window overlap** i.e. what stride was taken for each iteration. The result have shown that an overlap of approx. 60% provide optimal results, similar to **window size**, this adds to the computing overhead.

Based on the above results, the feature properties moving forwards will be:

- Use the full training dataset during training
- 450 visual words when creating the codebook
- Window size of 12.5%
- Overlap of 60%

Results

In section above the focus was on feature selection and tuning, in this section I compare various Machine Learning Algorithms and parameters in respect to the performance metrics described below and finally conclude with the performance of the most performant algorithm.

Performance of the *features* and *model* are based on (also described above):

Log Loss (Logarithmic Loss):
Log Loss quantifies the accuracy of a classifier by penalising false classifications i.e. minimising the Log Loss is basically equivalent to maximising the accuracy.
Log Loss heavily penalises classifiers that are confident about an incorrect classification e.g. if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large.

More details can be found: https://www.kaggle.com/wiki/LogarithmicLoss

Accuracy score:
The fraction of correct predictions.

Accuracy of the top 10:

For each k, how accurate is the model i.e. when k is set to 2, then consider a correct prediction if the predicted probability includes the correct prediction within the top 2 predictions.

Models

The following is a list of algorithms and parameters used for sketch recognition.

Naive Bayes Classification

"Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features." - http://scikit-learn.org/stable/modules/naive_bayes.html

1. Multinomial Naive Bayes Classifier

"The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work." - http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB

2. Gaussian Naive Bayes Classifier

"GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian" - http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

K Nearest Neighbors Classification

"Classifier implementing the k-nearest neighbors vote." - http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

3. K Nearest Neighbors Classifier

A GridSearch was performed on the parameters (with ranges/values):

- k = 10
- algorithm = auto (attempt to decide the most appropriate algorithm based on the values passed during training)
- Distance metrics = ['minkowski', 'euclidean', 'manhattan']
- weights = ['uniform', 'distance'] (uniform; all points in each neighborhood are weighted equally, distance; weight points by the inverse of their distance)

Best performed parameters (based on the GridSearch):

- Distance metric = 'manhattan'

- weights = 'distance'

Support Vector Machines (SVMs)

SVMs find the "maximum-margin" line that separates the classes (line "straight in the middle"). If the data cannot be linearly separable, the SVM will project the datums into higher dimensions (hyper planes). This can be done effectively by using kernals (known as the 'kernal trick') - http://scikit-learn.org/stable/modules/svm.html

4. SVM

http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

A GridSearch was performed on the parameters (with ranges/values):

- kernals = ['linear', 'rbf', 'poly']
- C_range = np.logspace(-2, 10, 13)
- gamma_range = np.logspace(-9, 3, 13)

Best performed parameters (based on the GridSearch):

- kernal = RBF
- C = 10.0
- gamma = 10.0

5. LinearSVM

http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

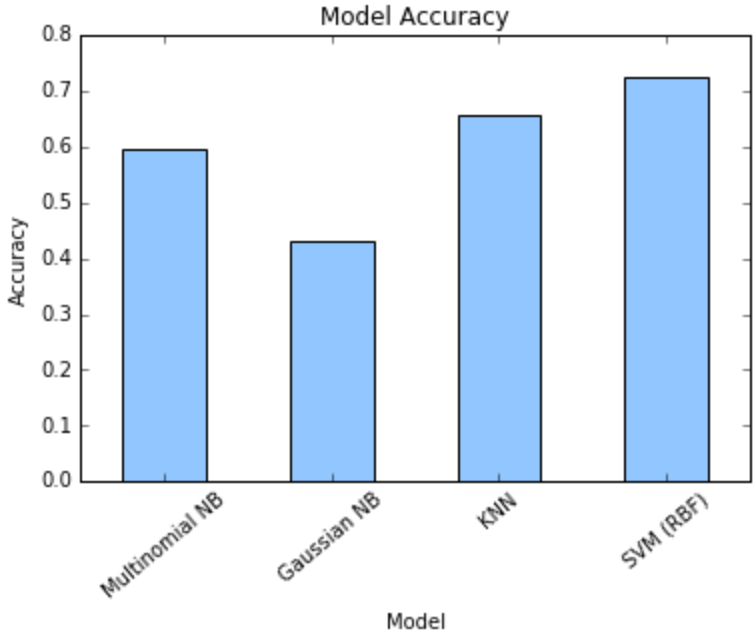
A GridSearch was performed on the parameters (with ranges/values):

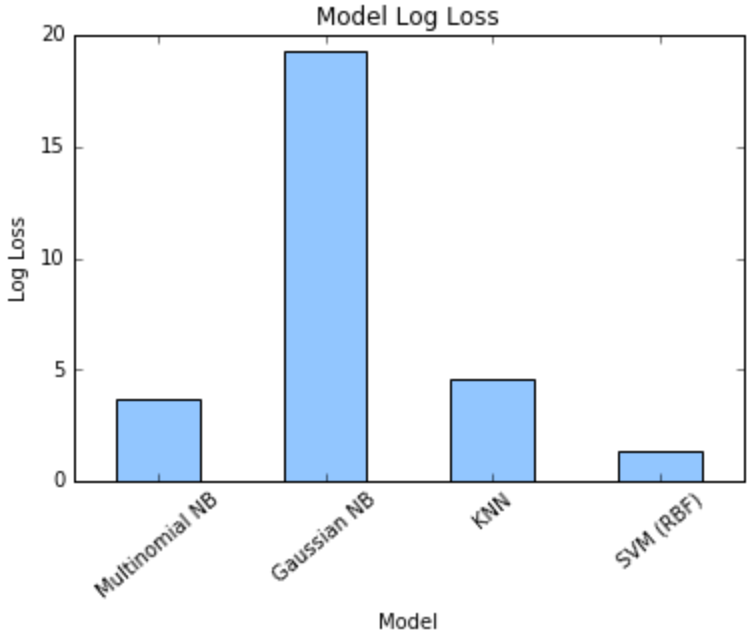
- C_range = np.logspace(-2, 10, 13)
- loss_functions = ['squared_hinge', 'hinge'] (squared_hinge; square of the hinge loss, hinge; standard SVM loss)

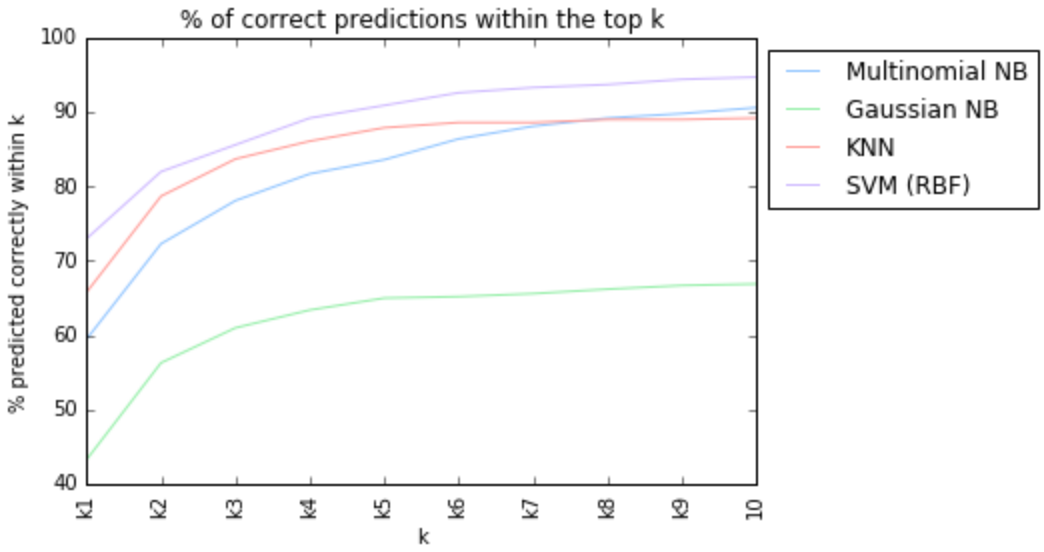
Best performed parameters (based on the GridSearch):

- C = 18
- loss = 'squared_hinge'

Model Evaluation





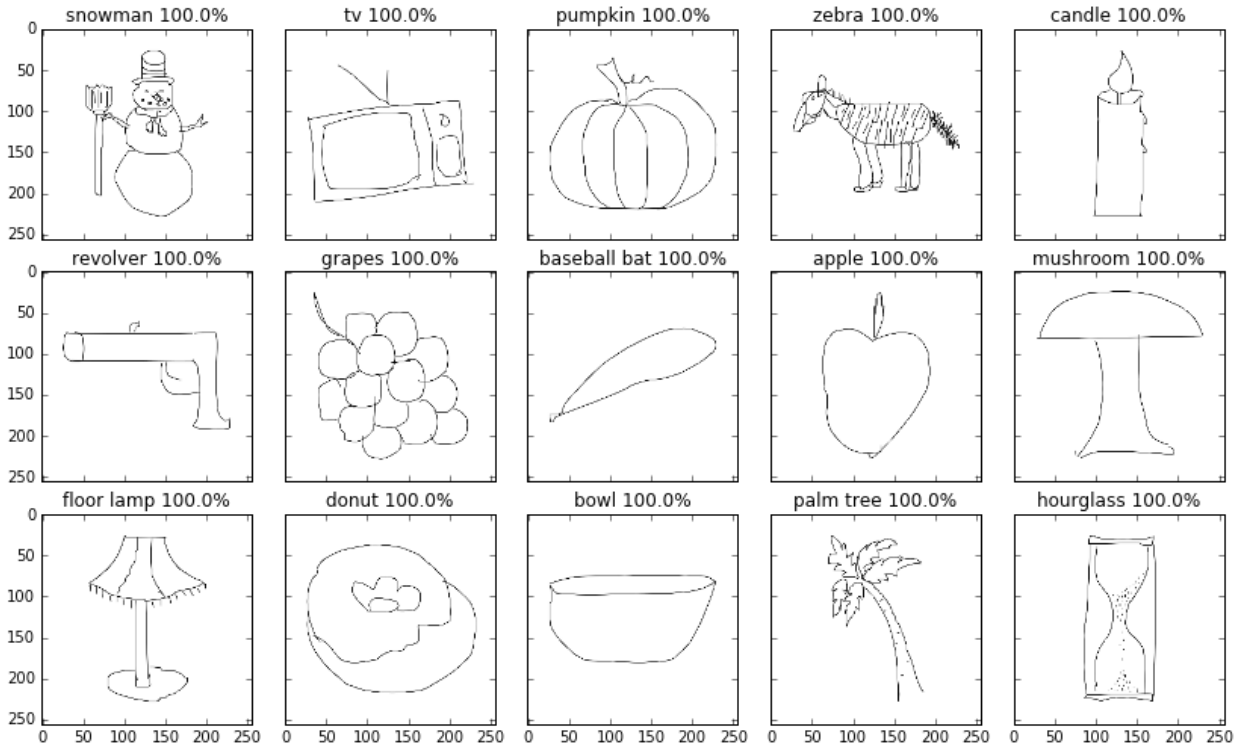


Model Selection

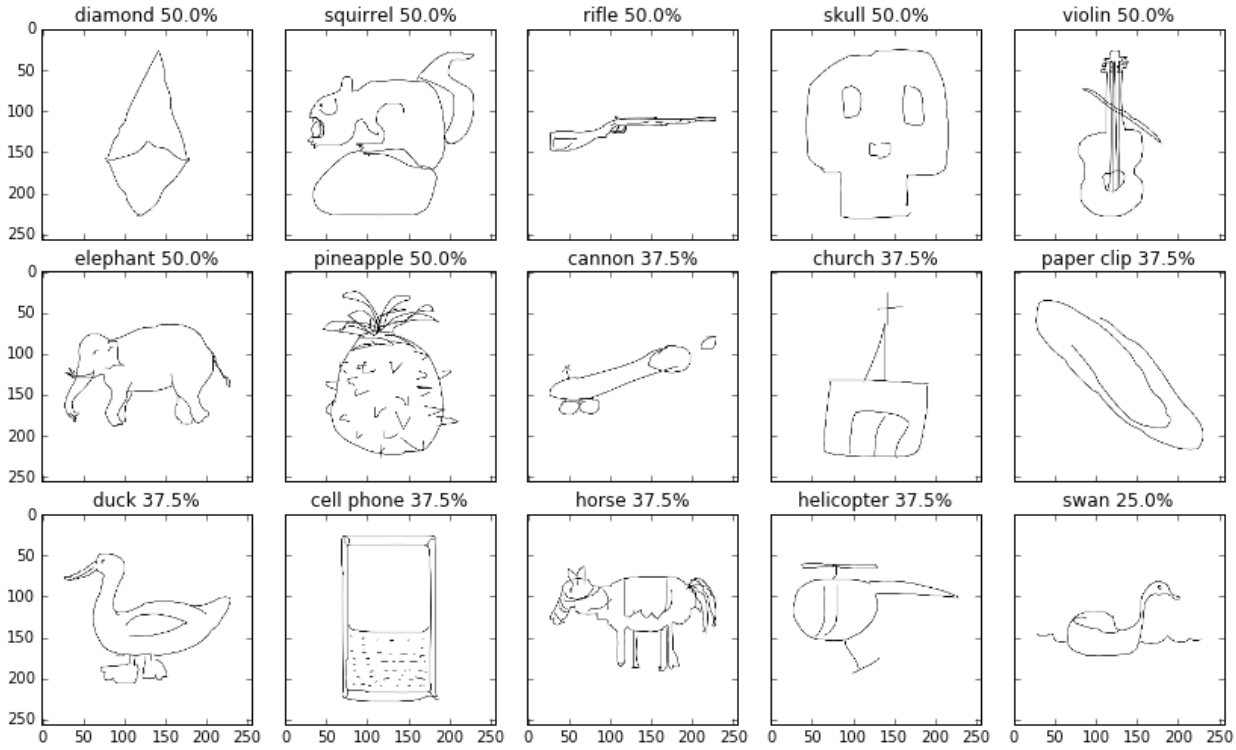
With respect to Accuracy, Log Loss, and 'K Accuracy'; SVM (using RBF) outperformed the other algorithms:

- Accuracy achieved was 0.73 (with an accuracy of 94.69 with k = 10)
- Log Loss achieved was 1.33

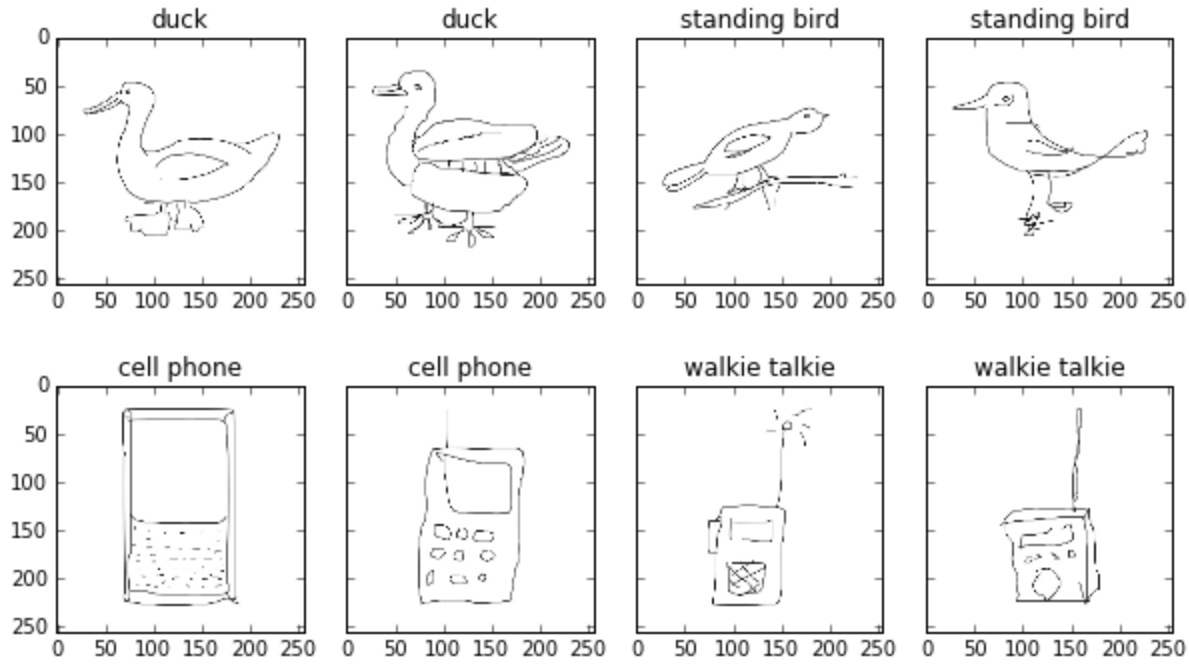
20 most accurate classified sketches

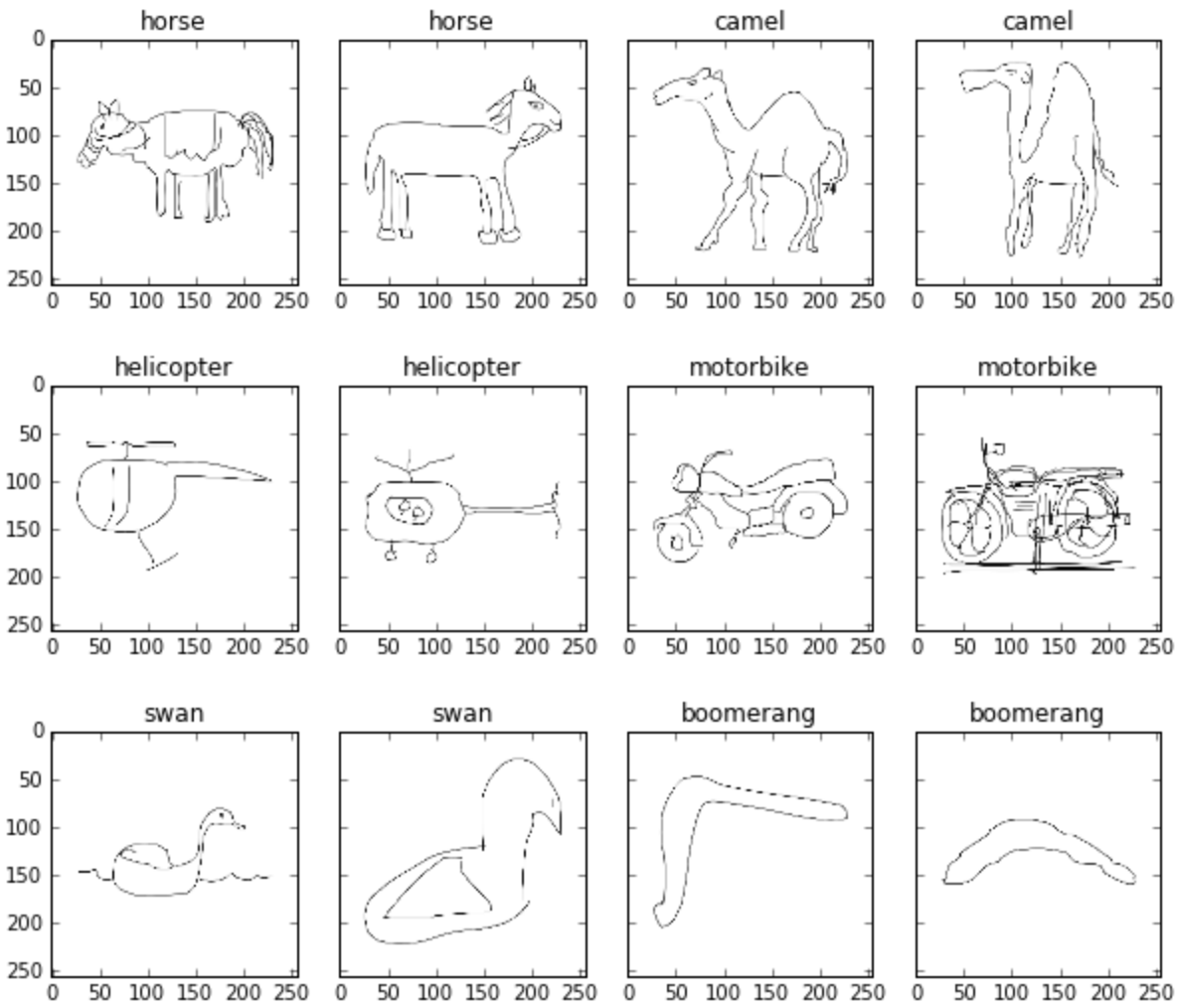


20 least accurate classified sketches



Most confused with





Conclusion

In this project we have explored how to extract features from images and how to efficiently describe them using a technique not too dissimilar to text classification (Visual Bag of Words).

We reviewed the effects of adjusting the level granularity of detail when describing the images as well as evaluated and tuned models to classify the sketches, whereby the winning model was created using SVM using a RBF kernel which achieved an accuracy of 73% and log loss of 1.33.

Whether this is sufficiently accurate enough or not is highly dependent on the application as well as the user interface i.e. if used in a search application, it would be required to display the top N to confirm the user's intent.

Given the advancements and success in Neural Networks (Deep Learning) it would make for an interesting project to compare the performance of this with such model (and one I hope to do in the near future).