# LSDA - Homework 2

*Silvan Adrian (zlp432)*

May 14, 2019

## Contents

**Fun with Runtimes** **2**
- Exercise 1 . . . 2
- Exercise 2 . . . 2
- Exercise 3 . . . 2
- Exercise 4 . . . 3

**Nearest Neighbor Search** **3**
- Nearest Neighbor Classifier . . . 3
- Early Stopping . . . 3
- Distance-Based Pruning . . . 4

**Sparse Least-Squares Support Vector Machines** **4**
- Dense Data Matrix . . . 4
- Getting started . . . 5
- Gradient-Based Optimization . . . 5

**Tree Ensembles** **6**
- Random Forests . . . 6
- Random Forests with Restricted Trees . . . 6

# Fun with Runtimes

## Exercise 1

for ranking the functions in the assignment we remember that to assess big-O domination, we can see that if $f(n)$ is dominated and $g(n)$ is dominating ($f(n) \in O(g(n))$ the following is satisfied:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \tag{1}$$

Having that in mind I end up on the following rankings:

$$f_4 \leq f_7 \leq f_8 \leq f_3 \leq f_6 \leq f_1 \leq f_5 \leq f_2 \tag{2}$$

## Exercise 2

Prove that: $2^{2n} \in \mathcal{O}(2 \cdot 4^n n^4)$
By definition we have: $2^{2n} = 4^n \leq c \cdot (2 \cdot 4^n - n^4)$ Solve for $c$:

$$c = \lim \frac{f(n)}{g(n)} < \infty \tag{3}$$

$$\frac{4^n}{2 \cdot 4^n - n^4} \leq c \tag{4}$$

Take limit as $n \to \infty$

$$\lim_{n\to\infty} \frac{4^n}{2 \cdot 4^n - n^4} = \frac{1}{2} \tag{5}$$

Since n can be bounded we proved that $2^{2n} \in \mathcal{O}(2 \cdot 4^n n^4)$

## Exercise 3

To prove $\sum_{i=1}^{n} (log_2 i)^2 \in \mathcal{O}(n(log_2 n)^2)$ there must exist a constant $c$ and an integer $n_0$ such that for all $n \geq n_0$ the inequality $\sum_{i=1}^{n} (log_2 i)^2 \leq c \cdot n(log_2 n)^2$ holds

$$\sum_{i=1}^{n} (log_2 i)^2 \leq \sum_{i=1}^{n} (log_2 n)^2 \tag{6}$$

This means that for $c = 1$ and $n_0 = 1$ the inequality holds by which we have proven that $\sum_{i=1}^{n} (log_2 i)^2 \in \mathcal{O}(n(log_2 n)^2)$ holds.

## Exercise 4

$T(1) = T(2) = T(3) = 1$ and for $n \geq 4$ we keep solving the Recurrence relations until the base case:

$$T(n) = 4 \cdot T\left(\frac{n}{4}\right) = 4 \cdot 4 \cdot T\left(\frac{n/4}{4}\right) \tag{7}$$

$$= 4^2 \cdot T\left(\frac{n}{4^2}\right) = 4^2\left(4 \cdot T\left(\frac{n}{4^3}\right)\right) \tag{8}$$

$$= 4^k \cdot T\left(\frac{n}{4^k}\right) \tag{9}$$

Base case: $\frac{n}{4^k} = 1$.

$$n = 4^k \iff \log_4(n) = \log_4(4^k) = k$$

So:

$$T(n) = 4{\cdot}T\left(\frac{n}{4}\right) = 4^k{\cdot}T\left(\frac{n}{4^k}\right) = 4^{\log_4(n)}{\cdot}T\left(\frac{n}{4^{\log_4(n)}}\right) = n{\cdot}T\left(\frac{n}{n}\right) = n{\cdot}T(1) = n$$

For a positive number $k \geq 1$, we see that

$$T(n) = n \leq k\, n \;\; \forall n > 0$$

Therefore we have $T(n) \in \mathcal{O}(n)$

# Nearest Neighbor Search

## Nearest Neighbor Classifier

**Runtime:** 21.51 (in seconds)
**Accuracy:** 0.9690721649484536

## Early Stopping

**Runtime:** 0.9243 (in seconds)
**Accuracy:** 0.9415807560137457
**Correctness:** The answers are still correct according to the accuracy we have

which is still high, but we do can get issues when we need to query more then 10 leafs (so the best answer would be in an other leaf then one of the 10 we queried). Which results in our case here only in a small reduction of the accuracy.
**Asymptotic runtime:** Since we do have a reduction of the running time we know that it should be less then the worst case for searching the whole tree which is $\mathcal{O}(n)$. So we know the search case for a single leaf which is $log_2(n)$ and we have a maxium of 10 leafs (L), so we get as a worst case runtime:

$$log_2(n) \cdot L \qquad \text{(Leafs)} \tag{10}$$

### Distance-Based Pruning

**Runtime:** 7.889789 (in seconds)
**Accuracy:** 0.9656357388316151
**Correctness:** We still have a relatively high accuracy which tells us that the answers are correct. Which means that our predictions on the model mostly returned the right answers, so we can expect the answers to be correct.
**Quality:** In this case we take the k-th nearest point which technically would be the worst out of points we find (1st,2nd,...,k-th). But we do only search around a smaller "radius" around the predicting point, so we still get quite good results (since we divide by $\alpha = 2.0$). Our Accuracy stays high as before.

## Sparse Least-Squares Support Vector Machines

### Dense Data Matrix

Each value has the size of `8 Bytes` (since double precision). We have $n = 71175$ and $d = 20707$. As well $X \in R^{n \times d}$. We have $71175 \times 20707$:

$$71175 \times 20707 = 1473820725 \qquad \text{(Values, rows times columns)} \tag{11}$$
$$1473820725 \times 8 = 11790565800 \qquad \text{(Bytes)} \tag{12}$$
$$11790565800/10^9 = 11.790 \qquad \text{(Gigabytes)} \tag{13}$$
$$= 11.79GB \tag{14}$$

Which means the data set stored as a dense data matrix (every value gets saved) would not fit into the 2GB memory.

## Getting started

```
print("Amount of non-zero entries:", X.count_nonzero())
```

**Non-zero elements stored in the Matrix X:** 3'652'855

## Gradient-Based Optimization

**Explanation:** Since I use the kernel trick 1, by which I do the calculation by Matrix times a vector first to reduce the overall memory use and performance of the calculation therefore the runtime stays fast. I also had to take care of python not evaluating something too early by splitting up all the calculations over many variables to put it together.
**Accuracy:** 0.9592582186007306

# Tree Ensembles

## Random Forests

By using the `RandomForestClassifier(n_estimators=10, max_features=None)` I end up on following accuracy score for the validation data:
**Accuracy score:** 0.75229
Predicting the test data and printing it as a image I ended up on the following image:

## Random Forests with Restricted Trees

**Implementation:** I implemented this exercise by using `BaggingClassifier`, on which then I say I want to have 10 estimators and as a base estimator I use `DecisionTreeClassifier()` as described in the homework. As a last step I set `max_features=2` which then assign 2 random features to each estimator and then takes the predictions of all of the estimators into account.
**Performance:** The accuracy decreases I suspect the usage of only 2 features per estimator is not sufficient enough to predict with higher accuracy, since from 10 features only 2 get used. So in our case with only 10 features it doesn't have any particular advantage to split the features across different "weaker learners" to expect a better outcome. So for this exercise the Random Forrest on the whole data set with all features does have the better performance/accuracy.
**Accuracy score:** 0.65293

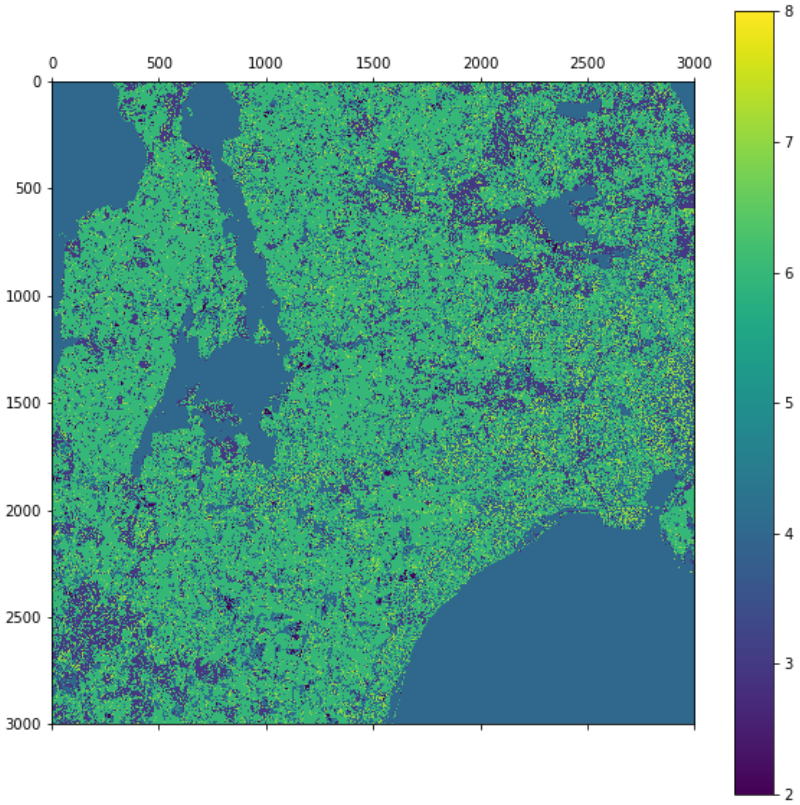


Figure 1: Landsat image