# Machine Learning A (2025)
# Home Assignment 1

Simon Henriksen, xwj436

## Contents

# 1 Make Your Own (10 points)

1. Variabels: 1.grades from prior courses, 2. current enrolled faculty, 3. level of study
   Sample space: 1.
   $$\mathcal{X} = \{-3, 00, 02, 4, 7, 10, 12\}$$
   , 2.
   $$\mathcal{X} = \{Law, Science, Humanities, SocialScience, HealthScience, Theology\}$$
   , 3.
   $$\mathcal{X} = \{bachelor, master\}$$
2. The label space would be the final character, having a sample space for Y being:
   $$\mathcal{Y} = \{-3, 00, 02, 4, 7, 10, 12\}$$
3. I would use the emperical loss function.
   $$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i))$$
4. I would define the distance using the euclidean distance formular
   $$\sqrt{(x_n - x'_n)^2}$$
5. We evaluate the performance of the algorithm by computing the empirical risk on a held-out test set, which is an unbiased estimator of the true risk
   $$R(h) = \mathbb{E}_{(x,y)\sim P}\big[\, \ell(y, h(x)) \,\big].$$
6. We would assume the chosen hypothesis with lowest risk to be overoptimistic. Generalize poorly on unseen data. The overoptimistic performance can be seen in the light of Jensens inequality. The hypothesis related to the mean error of the hypothesis space of empirical risk would be appropriate.

# 2 Digits Classification with $K$ Nearest Neighbors (40 points)

## 2.1 Task #1

Auxiliary functions.

```
def _euclidean_dists(train, test):
    A = train.T
    B = test.T
    A2 = np.sum(A**2, axis=1, keepdims=True)
    B2 = np.sum(B**2, axis=1, keepdims=True).T
    return np.sqrt(A2 + B2 - 2 * (np.dot(A, B.T)))

def map_labels(labels, classes=None):
    labels = np.asarray(labels).reshape(-1)
    if classes is None:
        classes = np.unique(labels)
        if len(classes) != 2:
            raise ValueError("Kr ver pr cis to klasser, eller angiv
   classes=...")
    # map classes[0] -> -1, classes[1] -> +1
    return np.where(labels == classes[0], -1, 1).astype(int)
```

The K-NN function.

```
def knn(training_points, training_labels, test_points, test_labels):

    classes = np.unique(np.asarray(training_labels).reshape(-1))
    if len(classes) != 2:
        raise ValueError("Tr ningslabels skal have pr cis to klasser.")
    # Map labels to -1 and 1
    y_train = map_labels(training_labels, classes=classes)
    y_test  = map_labels(test_labels,    classes=classes)
    # Compute distances between X_train and Y_test
    dists = _euclidean_dists(np.asarray(training_points), np.asarray(
   test_points))
    # Compare and sort the points which are closest
    order = np.argsort(dists, axis=0)
    # Get the labels of the nearest neighbors (1 or -1)
    neigh_labels = y_train[order]
    # Cumulative sum of votes for each K
    votes_cum = np.cumsum(neigh_labels, axis=0)
    # Predictions for all K (either positive (1) or negative (-1))
    preds_allK = np.sign(votes_cum)
    # Handle the case of a tie (0 votes) by predicting randomly
    preds_allK[preds_allK == 0] = np.random.choice([-1, 1], size=np.sum(
   preds_allK == 0))

    errors = (preds_allK != y_test[None, :]).mean(axis=1)

    return errors
```

1. Plots of the cross-validation error for the KNN function.

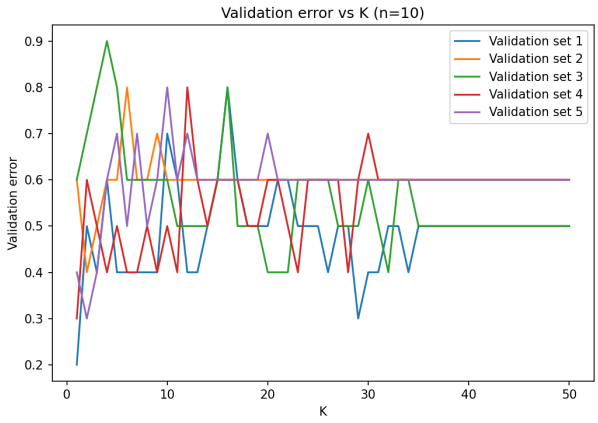


Figure 1: Error for n=10

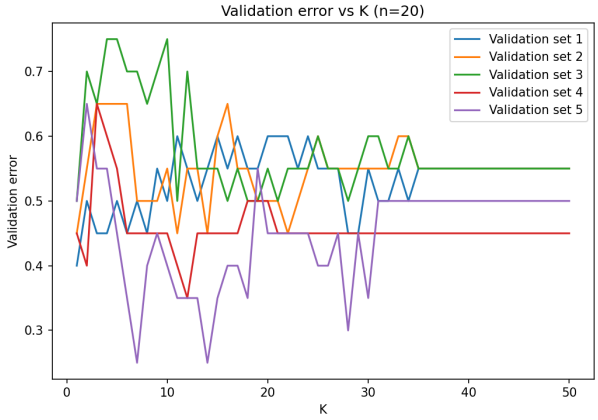


Figure 2: Error for n=20

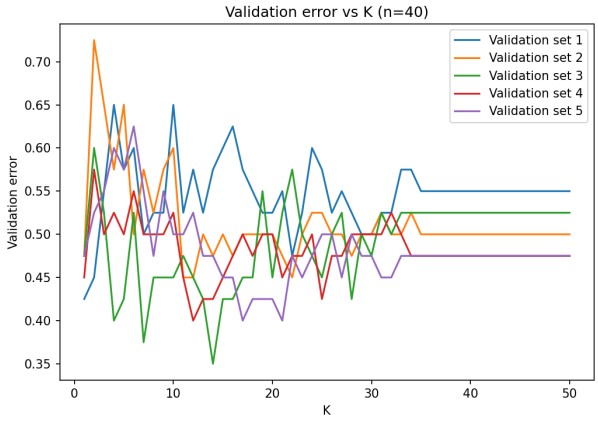


Figure 3: Error for n=40


Figure 4: Error for n=80

2. Plot of the variance error.

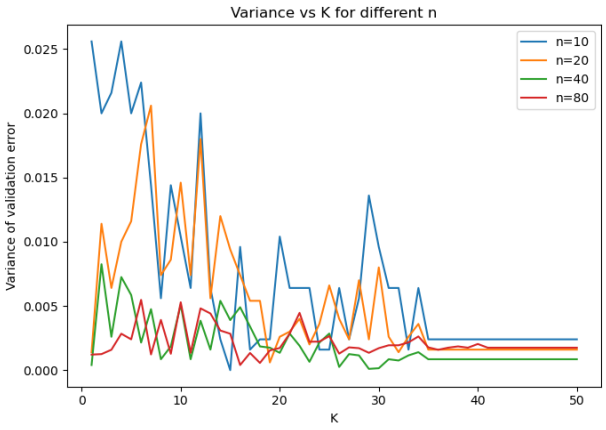


Figure 5: Variance of the validation sets error

3. Error interpretation:

   First, the size of the validation sets $n$ has a strong influence on the stability of the error curves: for small $n$ (e.g. $n = 10$) the validation error fluctuates heavily between the five sets, while larger $n$ (e.g. $n = 80$) yields much more consistent curves. As errors in smaller sets like $n = 10$ or $n = 20$ yields a larger fraction of the total error, the error will also be reflected more dominant than errors with $n = 40$ and $n = 80$. It can also be related to the law of large numbers.

   Second, the number of neighbors $K$ also affects the behaviour. For small $K$, the validation error is unstable and varies considerably. Smaller K values are expected to catch more nuances and therefore more fluctuations which is to to be seen in the plots. This results in very sensitive model with higher error.

As $K$ increases, the classifier will predict the dominant value every time ignoring local nuances. The error curves flatten and converge, reflecting a stabile error. This could be interpreted as a good sign, but is due to the reason mentioned above and isnt a good representation of the error. Very much bias.

Variance of error interpretation:

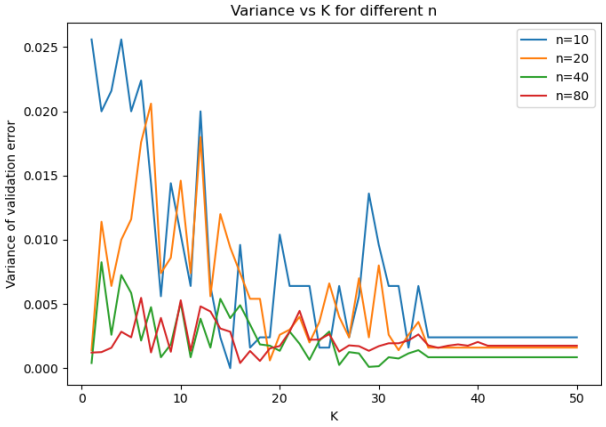


Figure 6: Variance of validation error

There's a general pattern that $n = 10$ and $n = 20$ have more significantly higher variance for all $K$ values which relates well to my previous interpretations of small $n$ values in the error plots.

$n = 40$ and $n = 80$ perform more stabile with lower variance compared to $n = 10$ and $n = 20$, also proved more stabile as $K$ increases.

It is clear that the higher $n$ values has much more consistent and stabile error which can therefore be interpreted as a more representative solution than the lower $n$ sets.

## 2.2 Task #2 (optional, not for submission)

# 3 Regression (50 points)

## 3.1 Task 1 (5 points)

```
def linear_regression(X, y):
    X_intercept = np.c_[np.ones((X.shape[0], 1)), X]
    theta_optimized = np.linalg.inv(X_intercept.T.dot(X_intercept)).dot(
   X_intercept.T).dot(y)
    return theta_optimized
```

## 3.2 Task 2 (10 points)

```
linear = linear_regression(X, y)
b_lin, a_lin = linear[0], linear[1]
y_pred_lin = a_lin * X.reshape(-1) + b_lin
MSE_lin = np.mean((y - y_pred_lin) ** 2)
print("Linear MSE =", MSE_lin)

Output:
Linear MSE = 24.80106431657058
[-1.45194395  1.55777052]
```

```
y_log = np.log(y)

theta_raw = linear_regression(X.reshape(-1, 1), y_log)
b, a = theta_raw[0], theta_raw[1]  # y = a * x + b

def h(x_in):
    x_in = np.asarray(x_in, dtype=float).reshape(-1)
    return np.exp(a * x_in + b)

y_pred = h(X)
MSE = np.mean((y - y_pred) ** 2)

print("a =", a, "b =", b)
print("MSE =", MSE)

Output:
a = 0.2591282395640714 b = 0.031472469714475926
MSE = 34.83556116722034
```

## 3.3 Task 3 (10 points)

We consider the dataset

$$S = \{(x_1, y_1), (x_2, y_2)\}, \quad x_1 = x_2 = 0,\ y_1 = 1,\ y_2 = 9.$$

**Without log-transformation:** The model is $h(0) = \exp(b)$. The optimal value minimizes

$$\sum_{i=1}^{2} (y_i - \exp(b))^2,$$

which yields

$$\exp(b) = \tfrac{1}{2}(y_1 + y_2) = 5.$$

**With log-transformation:** We regress on $(x, \ln y)$ and obtain

$$b = \tfrac{1}{2}(\ln y_1 + \ln y_2) = \tfrac{1}{2}(0 + 2.197) \approx 1.099.$$

Transforming back gives

$$h(0) = \exp(b) = \exp(1.099) \approx 3.$$

**Conclusion:** Without log-transformation the best value is 5 (arithmetic mean), while with log-transformation we obtain 3 (geometric mean). Therefore,

$$\arg\min_{a,b} \sum (y_i - h(x_i))^2 \neq \arg\min_{a,b} \sum (\ln y_i - (ax_i + b))^2.$$

## 3.4 Task 4 (5 points)

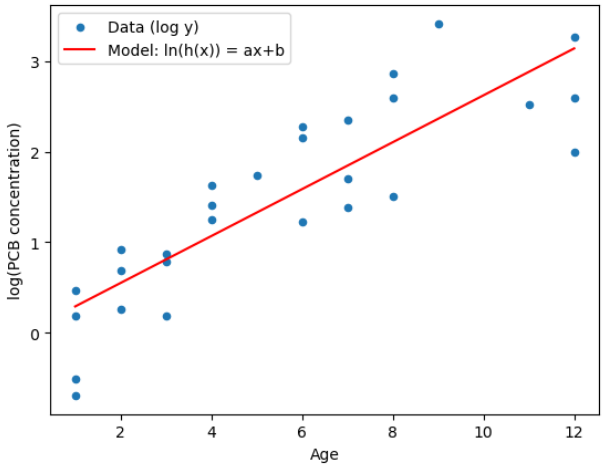


Figure 7: (X, log(y))

```
plt.scatter(x, y_log, label="Data (log y)", s=20)
plt.plot(x_line, y_model_log, color="red", label="Model: ln(h(x)) = ax+b"
    )
plt.xlabel("Age")
plt.ylabel("log(PCB concentration)")
plt.legend()
plt.show()
```

## 3.5 Task 5 (10 points)

```
r2 = 1 - (np.sum((y - y_pred) ** 2) / np.sum((y - np.mean(y)) ** 2))
print("R^2 =", r2)

output:
R^2 = 0.3570135731609867
```

The value 0.36 says that the model explain the varaiance quite poorly. It's above 0, which would be the same as taking the mean and not -1 which would imply no explanation of variance at all. It's not close to 1 which would be perfect explanation of the variance, but it explains some.

## 3.6 Task 6 (10 points)

```
Non-linear MSE = 100.78096247836443
R^2 (log) = 0.7313915439623445
```

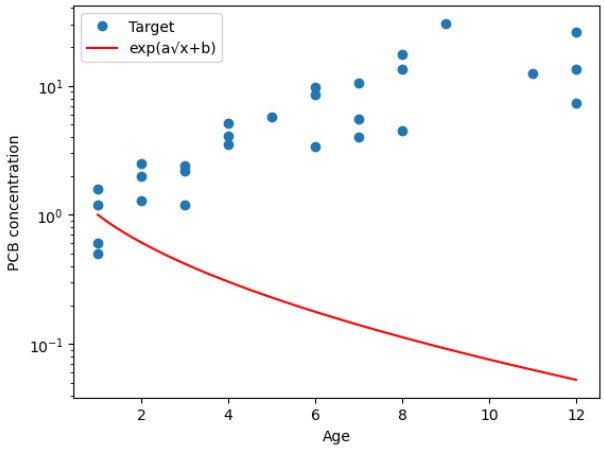


Figure 8: exp sqrt

The MSE score shows that the predicted non linear values are far from the original $y$ values. The plot proves that the log transformation has poor explanation on the original data. On the contrary, the explanation degree, R2, for $y_log$ and $y_predon_linear$ it scores 0.73 which must be considered a good score, close to 1, but still with the required noise.