MAD 2019/2020 Exam

Bulat Ibragimov, Fabian Gieseke, Kim Steenstrup Pedersen

13.01.2020 – 20.1.2020

You have to submit your individual solution of the exam electronically via the **Digital Exam**[1] system. The submission deadline is **20 January 2020 at 10:00**. The exam must be solved **individually**, i.e., you are **not allowed to work in teams or to discuss the exam questions with other students**. Your solution should consist of

1. a pdf file `answers.pdf` containing your answers, and
2. a `code.zip` file containing the associated code (python files and / or Python Jupyter notebooks).

The correction will be done in an anonymous fashion; **hence you should not mention your name somewhere in the answers or code**. Instead, please **add your exam number** at the beginning of your `answers.pdf`.

WARNING: The goal of this exam is to evaluate your individual skills. We have to report any suspicion of cheating, in particular collaboration with other students, to the head of studies. Note that, if proven guilty, you may be expelled from the university. Please: Do not put yourself and your fellow students at this risk.

You are allowed to ask questions via the Discussions board in Absalon, but make sure that you do not reveal any significant parts of the solution. In doubt, just approach Kim, Bulat or Fabian directly via e-mail! Any additional hint given by us will be made available to all of you via Absalon. Some further comments:

1. *You* ***are allowed*** *to reuse the code that was made available to you via Absalon as well as the code you have developed in the course of the assignments. If you reuse code from the lectures or from the assignments, make sure to put a reference to this in your code, and if your code was developed as part of an assignment, in collaboration with a fellow student, add a corresponding comment to your answers, although keeping anonymity (i.e., just mention which parts stem from team work). In case you reuse code snippets you have found on the internet, please make sure that you provide a reference to this external source as well.*
2. *In case you notice any inaccuracies in the problem descriptions below, please let us know. If needed, we will provide updates and additional comments via Absalon. Thus, make sure that you check Absalon for announcements and discussions regularly!*
3. *For the coding tasks, you are given Python files/Jupyter notebook templates. You are supposed to complete these files and notebooks. Note that you are allowed to import additional Python packages and to make use of the functions provided by, e.g., the Numpy package. However, you should not use built-in functions for the task at hand (e.g., a single `kmeans` function from some other package that implements the K-means clustering approach). If in doubt, please ask us via e-mail!*
4. *All templates and data can be found in the files `code.zip` and `data.zip`.*
5. *The deadline is hard (late submissions are not allowed), so make sure to submit in good time before the deadline. Up until the deadline it is possible to upload new versions of your solution. In the unlikely event that the Digital Exam system fails when you submit just at the deadline, then immediately send an e- mail to `uddannelse@diku.dk` and `kimstp@di.ku.dk` with an explanation of what happened and attach your solution (the pdf and zip files) to the exam. Your solution will be assessed if you have a valid excuse and submitted on time.*
6. *Good luck! :-)*

[1] `https://eksamen.ku.dk/`

Exercise 1 (Principal Component Analysis, 3 points). The aim of this task is to test your understanding of principal component analysis algorithm. You are given the following set of $N = 8$ 2-dimensional points:

Points								
coordinate x	0.1	0.5	1.1	-0.5	1.3	0.2	-0.1	1
coordinate y	1	1	2	0.2	-0.1	-0.1	-1.5	-2.5

Your task is to compute the principal component decomposition for these points. You can use either N or $N - 1$ in the denominator during calculation of the covariance matrix.

Comments: Please either manually type your calculations using latex/word or implement them in Python. If you decide to use Python, note that you are not allowed to use any specific functions for calculating covariance matrix, eigenvalues or eigenvectors. You should be able to solve the assignment using arithmetical operations. If you have doubts, please contact us.

Deliverables. All your manual calculation or source code used to solve this exercise (as a Jupyter notebook or Python script file(s)). Your report should include mathematical derivations of or essential code snippets for the following:

1. Mean point of the data.
2. Two eigenvalues.
3. Two eigenvectors.

Exercise 2 (Classification. Random Forests, 2 points). The aim of this task is to test your understanding of some of the math behind the random forest (RF) classification algorithm. Let's say you are implementing a binary decision tree for classification of some data into 3 classes: class1, class2, and class3. You are given a training set of 28 samples containing 7 samples of class1, 8 samples of class2 and 13 samples of class3. Some feature λ and its optimal threshold were selected to separate the initial 28 samples into the following two subsets:

Proposed separation of the original set T_{orig} of 28 samples			
	class1	class2	class3
First subset of samples T_{first}	6	1	10
Second subset of samples T_{second}	1	7	3

The task is to numerically evaluate the quality of this separation by computing the information gain and Gini impurity index. Just as a reminder, the information gain is computed as follows:

$$Gain(T_{orig}, \lambda) = H(T_{orig}) - \frac{|T_{first}|}{|T_{orig}|} H(T_{first}) - \frac{|T_{second}|}{|T_{orig}|} H(T_{second}),$$

where $T_{orig}, T_{first}, T_{second}$ are the original set of samples, and first and second subsets, respectively, while $H(T)$ corresponds to the entropy of the set. Please check lecture 10 for more details. The Gini impurity index is computed as follows:

$$Gini(T_{orig}, \lambda) = \frac{|T_{first}|}{|T_{orig}|} gini(T_{first}) + \frac{|T_{second}|}{|T_{orig}|} gini(T_{second}),$$

where $gini(T_{first})$ or $gini(T_{second})$ are computed as follows:

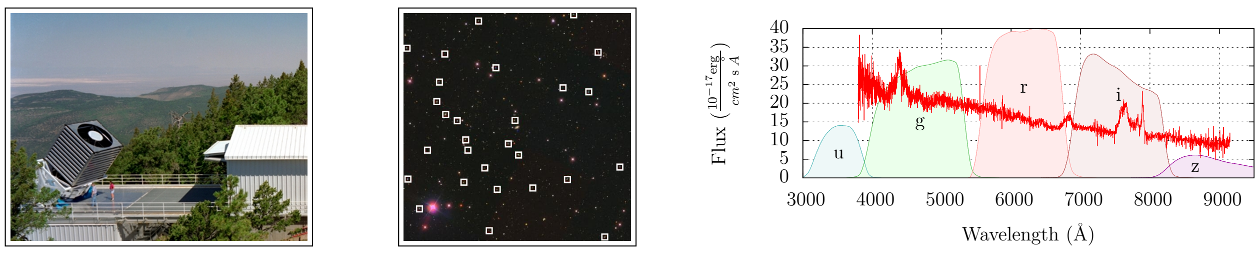
$$gini(T_{first}) = 1 - \sum_{i \in \{class1, class2, class3\}} \left(\frac{|t_i^{T_{first}}|}{|T_{first}|} \right)^2,$$

where $t_i^{T_{first}}$ is the number of samples from $class\_i$ in the T_{first} set.

Deliverables. Your report should include manual derivation of values for:

1. The information gain $Gain(T_{orig}, \lambda)$.
2. Gini impurity index $Gini(T_{orig}, \lambda)$.

Exercise 3 (Classification. Support vector machines, 4 points). This task aims to test your understanding of training/validation/testing data separation for optimal classifier parameter tuning. You are asked



(a) Observatory (b) Image Data (c) Spectra

Figure 1: The Apache Point Observatory shown in Figure (a) gathers both images and spectra. The image data are given in terms of multi-spectral images that are based on five filters covering different wavelength ranges, called the `u`, `g`, `r`, `i`, and `z` bands, see Figure (c). In Figure (b) an RGB image is shown that is based on such images of a particular region. For a small subset of detected objects (white squares), detailed follow-up observations in terms of spectra are available, see Figure (c). The dataset provided to you contains, for each such follow-up observations, the true redshift (obtained via the corresponding spectrum) as well as 10 features, which correspond to features extracted from the image data (so-called magnitudes). Your task is to build a regression model that can predict the redshift for all remaining objects in the image data.

to implement a Python class for selecting the optimal parameter of SVM by performing the five-fold cross-validation method. This task is accompanied by two files `dataSVM.txt` and `Exam_SVM_SV.ipynb`. The file `dataSVM.txt` contains 1000 samples, where each row represents a sample in the form of two features and the sample label. Please extend the Jupyter notebook `Exam_SVM_SV.ipynb` by implementing a Python class for n-fold cross-validation for SVM. Note that it should be sufficient to use only the imports already coded in the notebook `Exam_SVM_SV.ipynb`. You are, therefore, not allowed to import new libraries, for example `cross_validate` from `sklearn`.

Deliverables. All your source code used to solve this exercise (as a Jupyter notebook or Python script file(s)). Add to your report:

1. The performance of the cross-validation, i.e. the results of `print(accuracyList)` line in the `Exam_SVM_SV.ipynb`.
2. A figure showing the decision boundary for the optimal svm classifier, i.e. the results of `plot_decision_boundary(clf, testingFeatures, testingLabels)` line in `Exam_SVM_SV.ipynb`.

Exercise 4 (**Regression, 6 points**). The aim of this task is to test your understanding of methods for regression. An important problem in astrophysics is to estimate the distance of objects to our Earth. Since one cannot directly measure these distances, one resorts to the light emitted by the objects and that reaches Earth. In particular, one considers the so-called *redshift* z of an object: The larger the redshift, the larger the distance is from an object to Earth. This redshift can be measured very accurately in case one has access to the detailed spectrum of the light for an object at hand. Unfortuntely, obtaining such spectra is very time-consuming and are, hence, only available for a small subset of the detected objects, see Figure 1. Instead, one has access to image data and one usually aims at estimating the redshift based on the image data only. **Note that the aforementioned physical details are not important for solving this exercise, but are provided to motivate the corresponding machine learning task.**

The task of computing the redshifts can be formalized as a regression problem for which you have access to a training set $\{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$ and a corresponding test set, which are stored in `galaxies_train.csv` and `galaxies_test.csv`, respectively. Each file contains, for each object (row), a target value $t_n \in \mathbb{R}$ (first column) and a vector $\mathbf{x}_n \in \mathbb{R}^{10}$ of $D = 10$ attributes (second to eleventh column). Your task is to build regression models that can be used to predict the target (redshift) for new, unseen objects!

1. *Nearest Neighbor Regression (4 points):* As discussed in L9, one can use nearest neighbours in the context of regression scenarios. More precisely, for a vector $\mathbf{x}$, the prediction is given via

$$f(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_n \in N_k(\mathbf{x})} t_n$$

where $N_k(\mathbf{x})$ denotes the set of the $k \geq 1$ nearest neighbors of $\mathbf{x}$ in the set $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of training points.

   (a) Extend the Jupyter notebook `Galaxies_kNN.ipynb` and implement nearest neighbor regression in Python for arbitrary numbers $k \geq 1$ of nearest neighbors; make use of the Eulidean distance $d(\mathbf{p}, \mathbf{q}) =$

$\sqrt{\sum_{i=1}^{D}(p_i - q_i)^2}$ for data points $\mathbf{p}, \mathbf{q} \in \mathbb{R}^D$. Use the training set to train the model using $k = 3$ neighbors. Afterwards, apply the model to the instances given in the test set. Compute the RMSE between the predictions made by the model for the test instances and the corresponding test labels. Also, visualize the model quality via a scatter plot ('true redshift' vs. 'predicted redshift').

(b) As discussed during the lecture, one is not restricted to the Eulcidean distance only. For instance, one can make use of

$$d(\mathbf{p}, \mathbf{q}) = (\mathbf{p} - \mathbf{q})^T \mathbf{M}(\mathbf{p} - \mathbf{q})$$

for points $\mathbf{p}, \mathbf{q} \in \mathbb{R}^D$, where $M \in \mathbb{R}^{D \times D}$ is a positive semidefinite matrix. Implement this variant for arbitrary positive semidefinite matrices $\mathbf{M}$. Afterwards, consider the following concrete example $\mathbf{M} = diag(c_i)$ with $c_i = 0.00001$ for $i = 1, \ldots, 8$ and $c_i = 1.0$ for $i = 9, 10$ (i.e., a diagonal matrix with values c_i on the diagonal). Again, train the model using $k = 3$ and apply it to the test instances afterwards. Generate another scatter plot and report the RMSE. Can you explain what this particular matrix M does?

2. *Non-Linear Least-Squares via Neighbors (2 points):* We have seen how one can obtain non-linear least-squares regression models by extending the data matrix. An alternative to this approach works as follows: Given a training set $T = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\} \subset \mathbb{R}^D \times \mathbb{R}$ of points $\mathbf{x}_n$ with associated targets t_n, one can compute, for **each** new point $\bar{\mathbf{x}} \in \mathbb{R}^D$, a prediction via the following three steps:

   - Step 1: Compute the k nearest neighbors $N_k(\bar{\mathbf{x}})$ of $\bar{\mathbf{x}}$ in T (make use of the Euclidean distance).
   - Step 2: Fit a regularized linear least-squares regression model using these k nearest neighbors and the associated targets, i.e., compute:

   $$\hat{\mathbf{w}}(\bar{\mathbf{x}}) = \text{argmin}_{\mathbf{w}} \frac{1}{k} \sum_{\mathbf{x}_n \in N_k(\bar{\mathbf{x}})} \left(f(\mathbf{x}_n; \mathbf{w}) - t_n\right)^2 + \lambda \sum_{i=0}^{D} w_i^2$$

     Here, $f(\mathbf{z}; \mathbf{w}) = w_0 + w_1 z_1 + \ldots + w_D z_D$ depicts the linear model and $\lambda > 0$ a regularisation parameter.
   - Step 3: Use this linear model to obtain a prediction $f(\bar{\mathbf{x}}; \hat{\mathbf{w}}(\bar{\mathbf{x}}))$ for $\bar{\mathbf{x}}$.

   Extend the Jupyter notebook `Galaxies_LinRegNeighbors.ipynb` and implement this regression model in Python. Similarly to the the nearest neighbor model, make use of the training set to fit the model and apply it to the test set. Use $k = 15$ nearest neighbors and $\lambda = 1.0$. Again, visualize the model quality via a scatter plot. What is the induced RMSE value?

Comments: You are supposed to implement both parts on your own, i.e., you are not allowed to use, e.g., corresponding class/functions from the *Scikit-Learn* package. Note that you are allowed to make use of the *Numpy* package and the code provided in MAD2019_L3_code.zip. In doubt, please get in touch with us.

Deliverables. All your source code used to solve this exercise (as a Jupyter notebook or Python script file(s)). Add to your report:

1. (a) RMSE value and scatter plot and (b) RMSE value, scatter plot, short explanation for what M does (2-3 lines).
2. RMSE value and scatter plot

Exercise 5 (Predicting Sunspots Using Sampling, 8 points). The aim of this task is to test your understanding of sampling methods and Bayesian inference. In this task, we will use sampling methods to perform Bayesian regression and predict new unseen values (i.e. by performing Bayesian inference using the Metropolis-Hastings sampling algorithm).

Background

The goal of the task is to predict the average number of sunspots. We consider data based on the yearly sunspot number data provided by the Sunspot Index Data Center (SIDC), see `http://sidc.oma.be`. On `https://en.wikipedia.org/wiki/Sunspot` we find a short introduction to sunspots summarized below:

> Sunspots are extended regions on the Sun with a strong magnetic field. They have a lower temperature (3500-4500 K) than the surrounding photosphere (5800 K). The sunspots radiate less energy than the undisturbed photosphere of the Sun and are therefore visible as dark spots on the surface of the Sun. Sunspots are observed with some regularity since 1700 and on a strict daily basis since 1849; the relative [...] number (defined as ten times the number of groups + the number of spots) shows an 11 year cycle detected by Schwabe in 1843. The sunspot number reflects the magnetic activity of the Sun, which has a large impact on the magnetosphere of the Earth and is responsible for e.g. magnetic storms and polar lights.

In this task, you are going to predict the average sunspot number in a year yr based on the average numbers in the years $yr-1$, $yr-2$, $yr-4$, $yr-8$, and $yr-16$.

The training data can be found in the file `sunspotsTrainStatML.dt` and the test data in `sunspotsTestStatML.dt`. Each row corresponds to a an observation for year yr where the first 5 columns corresponds to a 5-dimensional observation $\mathbf{x}$ (each representing the average sunspot number in $yr-16$, $yr-8$, $yr-4$, $yr-2$, and $yr-1$) and in the last column, the target variable denoted t (the average sunspot number in year yr). The training data consist of observations made in the years 1716–1915, and the test data consist of observations made in the years 1916–2011.

We provide a Python function for reading the dataset files called `readdataset` found in the `sunspot.py` script, which also provides some visualizations of the dataset.

The goal of our modeling is to find a mapping $f\colon \mathbb{R}^D \to \mathbb{R}$ for predicting the sunspot number based on previous observations.

In the following we will consider the following three selections of observation variables (i.e. 3 different models):

Selection 1: Let $\mathbf{x}$ consist of the data in column 5 (i.e. $yr-1$), hence the dimensionality of this subset is $D = 1$.

Selection 2: Let $\mathbf{x}$ consist of the data in columns 3-4 (i.e. $yr-4$ and $yr-2$), hence the dimensionality of this subset is $D = 2$.

Selection 3: Let $\mathbf{x}$ consist of the data in column 1-5 (i.e. using all past observations), hence the dimensionality of this subset is $D = 5$.

The model

Our dataset consists of N observations $t_1, \ldots, t_N$ and corresponding past observations $\mathbf{x}_1, \ldots, \mathbf{x}_N$, where each past observation is a vector $\mathbf{x} \in \mathbb{R}^D$. We will assume that the observed sunspot number T is a Poisson distributed random variable where the rate parameter depends on the past observations in a linear manner. The rate parameter of the Poisson distribution is given by the linear model

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D \tag{1}$$

with parameter vector $\mathbf{w} \in \mathbb{R}^{D+1}$. The likelihood for the n'th data is therefore given by

$$p(t_n|\mathbf{x}_n; \mathbf{w}) = \frac{f(\mathbf{x}_n; \mathbf{w})^{t_n} \exp\left(-f(\mathbf{x}_n; \mathbf{w})\right)}{t_n!} \quad \text{for } t_n \geq 0 \tag{2}$$

and $p(t_n|\mathbf{x}_n; \mathbf{w}) = 0$ for $t_n < 0$, where $t_n!$ denotes the factorial of t_n and we use the convention $0! = 1$. Notice that we must have that $f(\mathbf{x}; \mathbf{w}) > 0$ (for it to be a proper rate parameter of the Poisson distribution) and the expectation of this Poisson likelihood is

$$\mathbb{E}_{p(t|\mathbf{x};\mathbf{w})}[t] = f(\mathbf{x}; \mathbf{w}) \tag{3}$$

end the variance is

$$\mathrm{Var}[t] = f(\mathbf{x}; \mathbf{w}) \ . \tag{4}$$

Our choice of prior for the parameter vector $\mathbf{w}$ will be an isotropic multivariate Gaussian distribution with mean vector $\mu_0 \in \mathbb{R}^{D+1}$ and variance given by σ_0^2,

$$p(\mathbf{w}|\mu_0, \sigma_0^2) = \mathcal{N}(\mu_0; \sigma_0^2 \mathbf{I}) \tag{5}$$

where $\mathbf{I}$ denotes the $(D+1) \times (D+1)$ identity matrix.
Remark: This choice of prior is not without problems, since we have the constraint that $f(\mathbf{x}; \mathbf{w}) > 0$, not all values of $\mathbf{w}$ are allowed (e.g. w_0 cannot be negative). This is not enforced by this prior and sampling directly from the prior will lead to values for $\mathbf{w}$ that violates the constraint.

Using the standard notation for the dataset, $\mathbf{t} = (t_1, \ldots, t_N)^T$ and $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^T$, the posterior distribution for our Bayesian model is given by

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \mu_0, \sigma_0^2) = \frac{p(\mathbf{t}|\mathbf{X}; \mathbf{w}) p(\mathbf{w}|\mu_0, \sigma_0^2)}{p(\mathbf{t}|\mathbf{X})} \tag{6}$$

where the marginal likelihood is given by

$$p(\mathbf{t}|\mathbf{X}) = \int p(\mathbf{t}|\mathbf{X}; \mathbf{w}) p(\mathbf{w}|\mu_0, \sigma_0^2) \, d\mathbf{w} \ . \tag{7}$$

Because of our choice of likelihood and prior, which is not a conjugate prior-likelihood pair, we cannot perform analytical derivation of the posterior, the problem being the integral in (7). In order to be able to make predictions (inference) with our model we need to be able to evaluate expectations of the likelihood (2) with respect to the posterior (6), i.e. compute the predictive distribution

$$p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{t}, \mathbf{X}, \mu_0, \sigma_0^2) = \mathbb{E}_{p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \mu_0, \sigma_0^2)}[p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w})] \ . \tag{8}$$

In this task, we will use the Metropolis-Hastings sampling algorithm to produce samples from the posterior and approximate the predictive distribution. In order to make approximate predictions with our model, we can use the following approximation

$$t_{\text{new}} = \mathbb{E}_{p(t|\mathbf{x}_{\text{new}}, \mathbf{t}, \mathbf{X}, \mu_0, \sigma_0^2)}[t] \approx \frac{1}{N_s} \sum_{s=1}^{N_s} f(\mathbf{x}_{\text{new}}; \mathbf{w}_s) \ , \tag{9}$$

over N_s samples from the posterior (6).

Problems to solve:

1. (1 point) We will assume that t_n is identical and conditional independent given $\mathbf{x}_n$ (i.e. i.i.d.), based on (2), write the expression for the likelihood $p(\mathbf{t}|\mathbf{X}; \mathbf{w})$ for the dataset.

2. (1 point) Show that the log-posterior $\log p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \mu_0, \sigma_0^2)$ for the model given by the likelihood $p(\mathbf{t}|\mathbf{X}; \mathbf{w})$ from the previous question and the prior $p(\mathbf{w}|\mu_0, \sigma_0^2)$ from (5) can be written as

$$\begin{aligned}\log p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \mu_0, \sigma_0^2) &= \sum_{n=1}^{N} \left(t_n \log f(\mathbf{x}_n; \mathbf{w}) - f(\mathbf{x}_n; \mathbf{w}) - \log t_n! \right) \\ &\quad - \frac{1}{2\sigma_0^2} (\mathbf{w} - \mu_0)^T (\mathbf{w} - \mu_0) - \log \left(\left(\sqrt{2\pi\sigma_0^2} \right)^{D+1} \right) - \log p(\mathbf{t}|\mathbf{X}) \ .\end{aligned}$$

3. (3 point) Implement your own version of the Metropolis-Hastings sampling algorithm for sampling from our model posterior (6). Use a multivariate Gaussian distribution as proposal distribution such that the proposal distribution becomes symmetric and the acceptance ratio simplifies (see slide 38 in lecture L11 Sampling). In order to avoid numerical problems, use the log-acceptance ratio formulation of the algorithm. Also since we need to compute the factorial of large numbers in the likelihood you have to use Stirling's approximation which allows us to use this expression to rewrite the log-posterior from the previous question,

   $$\log t_n! \approx t_n \log t_n - t_n + \frac{1}{2} \log(2\pi t_n) \ \text{ for } t_n > 0 \ ,$$

   and for $t_n = 0$, we have $\log t_n! = \log 0! = 0$ using the convention that $0! = 1$. Use the prior parameter values $\mu_0 = (1, 1, ..., 1)^T \in \mathbb{R}^{D+1}$ and $\sigma_0^2 = 0.25$.

   Hint: Remember that if you add a first element of 1 to the input vector $\mathbf{x}$, the model from (1) can be written as a vector dot product, $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x}$. In `sunspot.py`, we provide functions for computing the log-factorial using Stirling's approximation (these functions also allows for non-integer input since in the sunspot dataset t_n is real valued). You will encounter numerical problems whenever you try to sample $\mathbf{w}_s$ values that break the constraint $f(\mathbf{x}; \mathbf{w}) > 0$ (see previous remark). One way to handle this problem is to make sure that the initial value $\mathbf{w}_1$ for the algorithm is not violating the constraint.

4. (1 point) Using the training dataset and model selection 1 described in the Background section, use your Metropolis-Hastings sampler to generate samples $\mathbf{w}_s$ from the posterior. In this case, the parameter vector is $\mathbf{w}_s \in \mathbb{R}^2$ and you can visualize these parameter samples as points in a 2D plot. Do your samples seem to be independent or are they correlated?

5. (2 point) Use your Metropolis-Hastings sampler to generate samples $\mathbf{w}_s$ from the posterior for the training dataset on the 3 model selections described in the Background section. For each of the test dataset use the 3 model selection sample sets to make predictions using (9). Report the RMSE on the test set for each of the 3 selections. For model selection 1, make a scatter plot of the test set and plot the predictions produced by the model. Which of the 3 model selections gives you the best prediction results?

Deliverables. All your source code used to solve this exercise (as a Jupyter notebook or Python script file(s)). Add to your report:

1. The expression for the likelihood and a short explanation (1-2 lines) of how you arrived at that.
2. Include the essential steps for deriving the expression.
3. Your source code for your implementation of the Metropolis-Hastings algorithm.
4. Include your visualization of the parameter samples in the report as well as your answer to the question.
5. Your source code for your implementation for making predictions. In the report include the computed RMSE values for at least 5 different sets of samples (runs of the sampling algorithm). Include the scatter plot and predictions plot. Also list and provide arguments for how many samples are needed in your implementation. How many burn-in steps do you discard? What do you do to ensure that your samples are independent? Include your judgement of the 3 different model selections.