Name of Candidate: .....

Student id: .....

Signature: .....

THE UNIVERSITY OF NEW SOUTH WALES

Term 2, 2020

COMP9417 Machine Learning and Data Mining – Final Examination

1. TIME ALLOWED — 24 HOURS
2. THIS EXAMINATION PAPER HAS 14 PAGES
3. TOTAL NUMBER OF QUESTIONS — 5
4. ANSWER **ALL 5 QUESTIONS**
5. TOTAL MARKS AVAILABLE — 100
6. OPEN BOOK EXAM - LECTURE NOTES, TUTORIALS, AND ONLINE RESOURCES ARE PERMITTED. PLEASE USE REFERENCES WHERE NECESSARY.
7. SUBMISSION — YOU WILL SUBMIT A SINGLE PDF FILE `answers.pdf` CONTAINING YOUR ANSWERS FOR ALL QUESTIONS ATTEMPTED. START EACH SUB-QUESTION ON A NEW PAGE. MARKS MAY BE DEDUCTED FOR UNCLEAR WORK. YOU MAY TYPE YOUR SOLUTIONS USING LaTeX, OR TAKE CLEAR PHOTOS OF HANDWRITTEN WORK, OR MAKE USE OF A DIGITAL TABLET. FOR QUESTIONS THAT REQUIRE CODING, YOU WILL SUBMIT A SINGLE PYTHON FILE `solutions.py` (SEE TEMPLATE) CONTAINING YOUR CODE, THOUGH ALL GENERATED PLOTS/TABLES **MUST** BE INCLUDED IN THE PDF FILE. CODE MUST **ONLY** BE SUBMITTED IN THE PYTHON FILE `solutions.py`.
8. DISCUSSION WITH OTHER STUDENTS IS STRICTLY PROHIBITED. CODE SUBMISSIONS WILL BE CHECKED FOR PLAGIARISM. CHEATING WILL RESULT IN A FAILING GRADE FOR THE COURSE AND POTENTIAL FURTHER

DISCIPLINARY ACTION.

9. IF NEEDED, YOU ARE PERMITTED TO SEEK CLARIFICATION ON THE EXAM WEBCMS FORUM. QUESTIONS SPECIFIC TO CONTENT WILL NOT BE ANSWERED AND MAY BE REMOVED. OFFENDERS MAY HAVE EXAM MARKS DEDUCTED IF INFORMATION IS DIVULGED ON THE FORUM DURING OR FOLLOWING THE EXAM.
10. ENSURE YOUR ANSWER FORMATTING IS CLEAR AND LEGIBLE. BEGIN EACH SUB-QUESTION ON A NEW PAGE WITH A CLEAR HEADING. HIGHLIGHT FINAL ANSWERS AND MAKE USE OF DOT POINTS FOR DISCUSSION QUESTIONS. MARKS MAY BE DEDUCTED FOR WORK THAT IS DIFFICULT TO READ OR UNDERSTAND.
11. SUBMIT YOUR EXAM ANSWERS AND SOLUTIONS VIA THE CSE GIVE SYSTEM. THE COMMAND TO TO SUBMIT WILL BE:

    ```
    give cs9417 exam answers.pdf solutions.py
    ```

    SUBMISSION WILL BE OPEN FROM 09:00 AUSTRALIAN EASTERN STANDARD TIME (AEST) FOR 24 HOURS.
12. BY STARTING THIS EXAM AS A STUDENT OF THE UNIVERSITY OF NEW SOUTH WALES, YOU DO SOLEMNLY AND SINCERELY DECLARE THAT YOU HAVE NOT SEEN ANY PART OF THIS SPECIFIC EXAMINATION PAPER FOR THE ABOVE COURSE PRIOR TO ATTEMPTING THIS EXAM, NOR HAVE ANY DETAILS OF THE EXAM'S CONTENTS BEEN COMMUNICATED TO YOU. IN ADDITION, YOU WILL NOT DISCLOSE TO ANY UNIVERSITY STUDENT ANY INFORMATION CONTAINED IN THE ABOVEMENTIONED EXAM, AND YOU WILL COMPLETE THE EXAM ON YOUR OWN WITHOUT ANY EXTERNAL HELP. VIOLATION OF THIS AGREEMENT IS CONSIDERED ACADEMIC MISCONDUCT AND PENALTIES MAY APPLY.

Question 1 [12 marks]

This question covers **Naive Bayes** and requires you to refer to the training data given in the table below, which shows the number of times each of four words A, B, C and D occurs in each of eight documents, four in the positive class, and four negative.

Document No.	A	B	C	D	Class
1	2	0	4	4	+
2	0	3	3	0	+
3	3	0	0	2	+
4	0	0	2	0	+
5	0	0	0	1	-
6	3	0	0	0	-
7	4	3	0	0	-
8	4	0	0	1	-

Consider a new document $x_\star$ with 1 occurrence of A, no occurrences of B, no occurrence of C, and 2 occurrences of D. Round all answers to four decimal places.

(a) [4 marks]

Recall that if $X = (X_1, X_2, X_3, X_4) \sim \text{Multinomial}(p_1, p_2, p_3, p_4)$, then

$$P(X = (x_1, x_2, x_3, x_4)) = \frac{(\sum_{i=1}^4 x_i)!}{\prod_{i=1}^4 x_i!} \prod_{i=1}^4 p_i^{x_i}.$$

Construct a Naive Bayes classifier for the problem of predicting whether a document should be classified as positive or negative using the multinomial distribution to model the probability of a word occurring or not in a class. Under your model, what is the value of $P(+|x_\star)$? Do not use smoothing.

(b) [2 marks]

Now, apply (add-1) smoothing to your probability estimates. Under the smoothed probabilities, what is $P(-|x_\star)$ under the multinomial model?

(c) [4 marks]

Recall that if $X \sim \text{Bernoulli}(p)$ then

$$P(X = x) = p^x (1-p)^{1-x}.$$

Construct a Naive Bayes classifier for the problem of predicting whether a document should be classified as positive or negative using the multivariate Bernoulli distribution to model the probability of a word occurring or not in a class. What is $p(+|x_\star)$ now? Do not use smoothing.

(d) [2 marks]

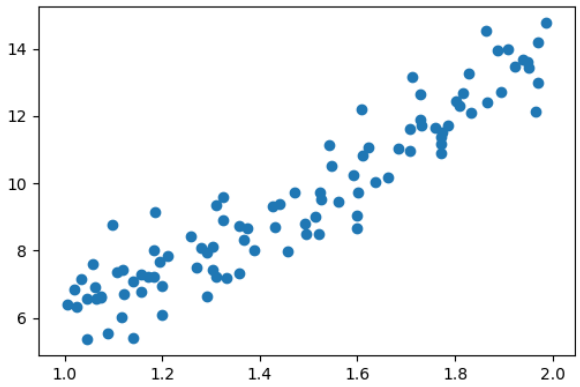
Now, apply (add-1) smoothing to your probability estimates. Under the smoothed probabilities, what is $P(-|x_\star)$ under the multivariate Bernoulli model?

Question 2 **[23 marks]** In this question, we will apply gradient descent to a simulated dataset. You may make use of numpy and matplotlib. You are not permitted to make use of any existing numpy implementations of gradient descent (if they exist). Generate data using the following Python code:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)        # make sure you run this line for consistency
x = np.random.uniform(1, 2, 100)
y = 1.2 + 2.9 * x + 1.8 * x**2 + np.random.normal(0, 0.9, 100)
plt.scatter(x,y)
plt.show()
```

Your data should look like the following:



Then, consider the loss function

$$\mathcal{L}_c(x, y) = \sqrt{\frac{1}{c^2}(x - y)^2 + 1} - 1,$$

where $c \in \mathbb{R}$ is a hyper-parameter.

(a) **[3 marks]** Consider the (simple) linear model $\hat{y}_i = w_0 + w_1 x_i$ for $i = 1, \ldots, n$. We can write this more succinctly by letting $w = (w_0, w_1)^T$ and $X_i = (1, x_i)^T$, so that $\hat{y}_i = w^T X_i$. The loss achieved by our model (w) on a given dataset of n observations is then

$$\mathcal{L}_c(y, \hat{y}) = \sum_{i=1}^{n} \mathcal{L}_c(y_i, \hat{y}_i) = \sum_{i=1}^{n} \left[\sqrt{\frac{1}{c^2}(y_i - w^T X_i)^2 + 1} - 1 \right],$$

Compute the following derivatives:

$$\frac{\partial \mathcal{L}_c(y_i, \hat{y}_i)}{\partial w_0} \qquad \text{and} \qquad \frac{\partial \mathcal{L}_c(y_i, \hat{y}_i)}{\partial w_1}.$$

If you are unable to figure this out, you may use Wolfram Alpha or a similar program to compute the derivatives in order to make use of them in later questions. Note that you must show your working for full marks, so solutions using a tool like Wolfram Alpha will receive a mark of zero.

(b) **[2 marks]** Using the gradients computed in (a), write down the gradient descent updates for w_0 and w_1 (using pseudocode), assuming a step size of α. Note that in gradient descent we consider the loss over the entire dataset, not just at a single observation. For simplicity, assume that your updates are always based on the values at the previous time step, even if you have access to the value at the current time step (i.e., when updating multiple parameters, the update for $w_1^{(t+1)}$ might depend on w_0, so you could use the new value of w_0, $w_0^{(t+1)}$, since it has already been computed, but here we assume that we just use the old value $w_0^{(t)}$).

(c) **[12 marks]** In this section, you will implement gradient descent from scratch on the generated dataset using the gradients computed in (a), and the pseudocode in (b). Initialise your weight vector to $w^{(0)} = [1, 1]^T$, and let $c = 2$. Consider step sizes $\alpha \in [1, 0.1, 0.01, 0.001, \ldots, 0.00000001]$ (a total of 9 step sizes). For each step-size, generate a plot of the loss achieved at each iteration of gradient descent. You should use 100 iterations in total. Generate a 3×3 grid of figures showing performance for each step-size. Add a screen shot of the Python code for this question in your answers PDF file (as well as pasting the code into your solutions.py file). The following code may help with plotting:

```
fig, ax = plt.subplots(3,3, figsize=(10,10))
alphas = [10e-1, 10e-2, 10e-3,10e-4,10e-5,10e-6,10e-7, 10e-8, 10e-9]
for i, ax in enumerate(ax.flat):
  # losses is a list of 9 elements. Each element is an array of length 100
   storing the loss at each iteration for
   # that particular step size
   ax.plot(losses[i])
   ax.set_title(f"step size: {alphas[i]}")  # plot titles
   plt.tight_layout()      # plot formatting
   plt.show()
```

(d) [**1 mark**] Comment on your results in part (c): what do you see happening for different step sizes? Why does this occur?

(e) [**3 marks**] To find an appropriate step-size, re-run the gradient descent to find the optimal model parameters. State this step-size, the final model, and generate two plots: the first for the weights w_0 and w_1 over the 100 iterations, and the second of the data with the final model super-imposed. What do you think of the final model? Are there any obvious ways to improve it? Be sure to include the plots in your answers PDF file, and the code used in your solutions.py file.

(f) [**2 marks**] Consider the following scenario: you re-run the analysis for various values of c and you notice that the optimal step-size varies across different values of c. Describe an approach to choosing an appropriate value of c.

Question 3 **[25 marks]** This question covers the (kernel) perceptron and requires you to refer to the following training data for parts (a)-(c). You are only permitted to make use of numpy and matplotlib. You are not permitted to make use of any existing numpy implementations of perceptrons (if they exist).

x_1	x_2	y
-0.8	1	1
3.9	0.4	-1
1.4	1	1
0.1	-3.3	-1
1.2	2.7	-1
-2.45	0.1	-1
-1.5	-0.5	1
1.2	-1.5	1

Table 1: Data for Question 3

(a) **[3 marks]** Plot the data in Table 1. Recall that the polynomial kernel is defined as $k(x, y) = (m + x^T y)^d$ for $m \in \{0, 1, 2, \dots\}$ and $d \in \{1, 2, \dots\}$. Each such kernel corresponds to a feature representation of the original data. Find the simplest polynomial kernel for which this data becomes linearly separable (**note:** simplest in this case is defined as the polynomial kernel with the smallest values of **both** m and d).

(b) **[2 marks]** The optimal kernel found in (a) induces a feature representation in $\mathbb{R}^p$ for some integer p determined by your choice of kernel. Choose a subset of two coordinates from this p-dimensional vector and plot the transformed data. For example, for vector $(w, x, y, z)^T \in \mathbb{R}^4$, a subset of two coordinates could be the first two coordinates (w, x), or the first and third coordinates (w, y), etc.). Is your transformed data linearly separable?

(c) **[10 marks]** Train a kernel perceptron on this data with initial weight vector $w^{(0)} = \mathbf{1}_p$ (the vector of ones in $\mathbb{R}^p$). Use a learning rate of $\eta = 0.2$. Note: this should be done in numpy. Provide a table outlining all updates of the weight vector, and the iteration number at which the update occured. State the final learned perceptron and the number of iterations until convergence. Demonstrate that your perceptron correctly classifies each of the points. You may use the following table as a template for presenting your results:

Iteration No.	w_0	w_1	...	w_p
0	1	1	...	1
first update iteration	$1+\delta_0$	$1+\delta_1$	...	$1+\delta_p$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

where δ_j is the update for w_j computed at the first iteration. To demonstrate that your perceptron classifies each point correctly, use the following table:

x_i	$\phi(x_i)$	$y_i\phi^T(x_i)w^*$
$[-0.8, 1]^T$	$\phi([-0.8, 1]^T)$	$r_1 > 0$
$\vdots$	$\vdots$	$\vdots$
$[1.2, -1.5]^T$	$\phi([1.2, -1.5]^T)$	$r_8 > 0$

where $r_i = y_i\phi^T(x_i)w^*$ should be positive if your perceptron has converged. Along with your table(s) of results, provide a screen shot of your code in your answers PDF file, and add the code to your solutions.py file.

(d) [5 marks] Let $x, y \in \mathbb{R}^2$ (i.e., x and y are two dimensional vectors), and consider the kernel

$$k(x, y) = (2x^T y + 3)^3.$$

Compute the feature vector $\phi(x)$ correpsonding to this kernel (in other words, the feature representation of x and y such that $\phi(x)^T\phi(y) = k(x, y)$).

(e) [5 marks]

Consider the following dataset. The positive examples (class = +1) are:

$$(2, 0),\ (0, 2),\ (1, 1),\ (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$$

and the negative examples (class = −1) are:

$$(-2, 0),\ (0, -2),\ (-1, -1),\ (-\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}).$$

Claim: An ordering of the examples in this dataset on which a perceptron (with learning rate $\eta = 1$) makes at least 5 mistakes during training cannot exist.

If you agree with this claim, provide a proof of why it must be true. Otherwise, provide an ordering of the examples such that the number of mistakes is achieved.

Question 4 [**20 marks**] This question covers the Support Vector Machine and requires you to refer to the following two-dimensional training data:

x_1	x_2	y
-3	9	-1
-2	4	-1
-1	1	-1
0	-3	1
1	1	1
2	4	-1
3	9	-1

(a) [**1 mark**] Plot the data (no need to provide any code if you do this in Python). Is the data linearly separable?

(b) [**11 marks**] In this section, we will build an SVM classifier by hand. Recall that the dual formulation of the SVM classifier for a dataset of size n is

$$\arg\max_{\alpha_1,\ldots,\alpha_n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j x_i \cdot x_j, \qquad \text{subject to} \quad \alpha_i \geq 0, \text{ for all } i, \quad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Solve for $(\alpha_1, \ldots, \alpha_7)$ for the provided dataset. (**Hint:** Using the plot in the previous section and your knowledge of SVMs, try to simplify the (dual) objective before doing any calculus.)

(c) [**3 marks**] For your SVM, compute the corresponding weight vector ($w \in \mathbb{R}^2$) and bias t. Superimpose a plot of the SVM onto the scatter in (a). What is the margin for this model?

(d) [**2 marks**] Discuss briefly the following claim: Linear classifiers are unable to represent non-linear functions.

(e) [**3 marks**] In your own words, explain what is meant by the 'kernel trick'. Why is this such an important technique for machine learning?

Question 5 **[20 marks]** In this question, we will consider the Scikit-learn implementations of the following classifiers:

- Decision Trees
- Random Forest
- AdaBoost
- Logistic Regression
- Multilayer Perceptron (Neural Network)
- Support Vector Machine

You are required to compare the performance of the above models on a binary classification task. The following code loads in these classifiers and defines a function to simulate a toy dataset:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import time
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification

def create_dataset():
    X, y = make_classification( n_samples=1250,
                                n_features=2,
                                n_redundant=0,
                                n_informative=2,
                                random_state=5,
                                n_clusters_per_class=1)
    rng = np.random.RandomState(2)
    X += 3 * rng.uniform(size = X.shape)
    linearly_separable = (X, y)
    X = StandardScaler().fit_transform(X)
    return X, y
```

(a) **[3 marks]** Generate a dataset. Then, randomly split the dataset into training set Xtrain and test set Xtest, with 80 examples for training and 20 for testing. Plot the decision boundaries of each of the classifiers on the test set. You may wish to use the following plotter function which plots the decision boundary and can work for any sklearn model.

```
def plotter(classifier, X, X_test, y_test, title, ax=None):
  # plot decision boundary for given classifier
  plot_step = 0.02
  x_min, x_max = X[:, 0].min() - 1, X[:,0].max() + 1
  y_min, y_max = X[:, 1].min() - 1, X[:,1].max() + 1
  xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                       np.arange(y_min, y_max, plot_step))
  Z = classifier.predict(np.c_[xx.ravel(),yy.ravel()])
  Z = Z.reshape(xx.shape)
  if ax:
      ax.contourf(xx, yy, Z, cmap = plt.cm.Paired)
      ax.scatter(X_test[:, 0], X_test[:, 1], c = y_test)
      ax.set_title(title)
  else:
      plt.contourf(xx, yy, Z, cmap = plt.cm.Paired)
      plt.scatter(X_test[:, 0], X_test[:, 1], c = y_test)
      plt.title(title)
```

Note that you can use the same general approach for plotting a grid as used in Question 2, and the plotter function supports an 'ax' argument. **Note that this plotter function differs slightly from the one in the sample final.** Be sure to include all your code in solutions.py, and any figures generated in your answers PDF file.

(b) **[10 marks]** Now, test how the performance of each of the classifiers varies as you increase the size of the training set. Fix your training and test sets from part (a). Then, starting from a random subset (chosen with replacement) of your training set of size 50, train your classification model, and compute the accuracy on the test set. Repeat this process for training set sizes of $[50, 100, 200, 300, \ldots, 1000]$. Repeat the experiment a total of 10 times for each classifier. Then, for each classifier, plot its average accuracy at each training set size. Compare the accuracy across different algorithms in a single figure, and in 5 lines or less, discuss your observations. Please use the following color code for your plots: [Decision Tree, Random Forest, AdaBoost, Logistic Regression, Neural Network, SVM]. Be sure to include all your code in solutions.py, and any figures generated in your answers PDF file.

(c) [**7 marks**] Using the time module, record the training time for each of the classifiers at each of the training set sizes. Plot the average training time over the 10 trials of each classifier as a function of the training size. You may add code for this section to your code in part (b). What do you observe? In 5 lines or less, discuss your observations. Use the same color scheme as in (b). Be sure to include your code in solutions.py, and any figures generated in your answers PDF file.

END OF PAPER