HW Assignment 1 (Due by 10:30am on Sep 26)

1 Theory (40 points)

1. [**Polynomial Curve Fitting**, 20 points]
Consider the problem of fitting a dataset of N points with a polynomial of degree M, by minimizing the sum-of-squares error:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(h_{\mathbf{w}}(\mathbf{x}_n) - t_n)^2 \tag{1}$$

where $h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=0}^{M} w_j x^j$. We have shown in class that the solution to minimizing $J(\mathbf{w})$ satisfies the following set of linear equations:

$$\sum_{j=0}^{M} A_{ij} w_j = T_i \tag{2}$$

$$\text{where } A_{ij} = \sum_{n=1}^{N} x_n^{i+j} \text{ and } T_i \;\; = \;\; \sum_{n=1}^{N} x_n^i t_n \tag{3}$$

Derive the solution for the regularized version of polynomial curve fitting, which minimizes the objective function below:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(h_{\mathbf{w}}(\mathbf{x}_n) - t_n)^2 + \frac{\lambda}{2}||\mathbf{w}||^2 \tag{4}$$

2. [**Probability Theory**, 20 points] *Exercise 1.3, page 58 in PRML.*
Suppose that we have three coloured boxes r (red), b (blue), and g (green). Box r contains 3 apples, 4 oranges, and 3 limes, box b contains 1 apple, 1 orange, and 0 limes, and box g contains 3 apples, 3 oranges, and 4 limes. If a box is chosen at random with probabilities $p(r) = 0.2$, $p(b) = 0.2$, $p(g) = 0.6$, and a piece of fruit is removed from the box (with equal probability of selecting any of the items in the box), then what is the probability of selecting an apple? If we observe that the selected fruit is in fact an orange, what is the probability that it came from the green box?

3. [**Ridge Regression (*)**, 20 points]
Consider the regularized linear regression objective shown below:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(h(x_n, \mathbf{w}) - t_n)^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$

   (a) Minimizing the L2 norm of $\mathbf{w}$ drives all parameters, including w_0, towards 0. Are there situations in which we do not want to constrain w_0 to be small? If yes, give an example, if not show why it is useful to constrain all the weights to be small, including w_0 .

   (b) We have seen in class how to compute the weights $\mathbf{w}$ that minimize $J(\mathbf{w})$. Assume now that we replace $||\mathbf{w}||$ in $J(\mathbf{w})$ with $||\mathbf{w}_{[\mathbf{1}:]}||$, where $\mathbf{w}_{[\mathbf{1}:]} = [w_1, w_2, ..., w_M]$. Derive the solution for $\mathbf{w}$ that minimizes this new objective function.

2 Implementation (80 points)

In this exercise, you are asked to run an experimental evaluation of linear regression on the Athens houses dataset, and on an artificial dataset with and without L2 regularization. The input data is available at `http://ace.cs.ohio.edu/~razvan/courses/ml4900/hw01.zip`. Make sure that you organize your code in folders as shown in the table below. Write code only in the Python files indicated in bold.

```
ml4900/
    hw01/
        code/
            univariate.py
            multivariate.py
            polyfit.py
            train_test_line.png
        data/
            univariate/
                train.txt, test.txt
            multivariate/
                train.txt, test.txt
            polyfit/
                train.txt, test.txt, devel.txt
```

1. [**Univariate Regression**, 20 points]
   Train a univariate linear regression model to predict house prices as a function of their floor size, based on the solution to the system with 2 linear equations discussed in class. Use the dataset from the folder `hw01/data/univariate`. Python3 skeleton code is provided in `univariate.py`. After training print the parameters and report the RMSE and the objective function values on the training and test data. Plot the training using the default blue circles and test examples using lime green triangles. On the same graph also plot the linear approximation.

2. [**Multivariate Regression**, 20 points]
   Train a univariate linear regression model to predict house prices as a function of their floor size, number of bedrooms, and year. Use the normal equations discussed in class, and evaluate on the dataset from the folder `hw01/data/multivariate`. Python3 skeleton code is provided in `multivariate.py`. After training print the parameters and report the RMSE and the objective function values on the training and test data. Compare the test RMSE with the one from the univariate case above.

3. [**Polynomial Curve Fitting**, 40 points]
   In this exercise, you are asked to run an experimental evaluation of a linear regression model, with and without regularization. Use the normal equations discussed in class, and evaluate on the dataset from the folder `hw01/data/polyfit`.

   (a) Select 30 values for $x \in [0, 1]$ uniformly spaced, and generate corresponding t values according to $t(x) = sin(2\pi x) + x(x + 1)/4 + \epsilon$, where $\epsilon = N(0, 0.005)$ is a

zero mean Gaussian with variance 0.005. Save and plot all the values. Done in `dataset.txt`.

(b) Split the 30 samples (x_n, t_n) in three sets: 10 samples for training, 10 samples for validation, and 10 samples for testing. Save and plot the 3 datasets separately. Done in `train.txt, test.txt, devel.txt`.

(c) Consider a linear regression model with polynomial basis functions, trained with the objective shown below:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(h(x_n, \mathbf{w}) - t_n\right)^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$

Show the closed form solution (vectorized) for the weights $\mathbf{w}$ that minimize $J(\mathbf{w})$.

(d) Train and evaluate the linear regression model in the following scenarios:

1. Without regularization: Use the training data to infer the parameters $\mathbf{w}$ for all values of $M \in [0, 9]$. For each order M, compute the RMSE separately for the training and test data, and plot all the values on the same graph, as shown in class.
2. With regularization: Fixing $M = 9$, use the training data to infer the parameters $\mathbf{w}$, one parameter vector for each value of $\ln \lambda \in [-50, 0]$ in steps of 5. For each parameter vector (lambda value), compute the RMSE separately for the training and validation data, and plot all the values on the same graph, as shown in class. Select the regularization parameter that leads to the parameter vector that obtains the lowest RMSE on the validation data, and use it to evaluate the model on the test data. Report and compare the test RMSE with the one obtained without regularization.

3 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw01.zip file that contains the hw01 folder in which your code is in the 3 required files.

On a Linux system, creating the archive can be done using the command:
`> zip -r hw01.zip hw01`.

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. On the theory assignment, **clear and complete explanations and proofs of your results are as important as getting the right answer**.
4. Make sure your code runs correctly when used in the directory structure shown above.

HW Assignment 2 (Due by 10:30am on Oct 5)

1 Implementation (90 points)

In this exercise, you are asked to implement de gradient descent algorithm and use it to train linear regression models for the Athens houses dataset and an artificial dataset with and without L2 regularization. The input data is available at `http://ace.cs.ohio.edu/~razvan/courses/ml4900/hw02.zip`. Make sure that you organize your code in folders as shown in the table below. Write code only in the Python files indicated in bold.

```
ml4900/
    hw02/
        code/
            univariate.py
            multivariate.py
            polyfit.py
            train_test_line.png
        data/
            univariate/
                train.txt, test.txt
            multivariate/
                train.txt, test.txt
            polyfit/
                train.txt, test.txt, devel.txt
```

1. [**Feature Scaling**, 10 points]
   To improve the convergence of gradient descent, it is important that the features are scaled so that they have similar ranges. Implement the standard scaling method using the skeleton code in `scaling.py`.

2. [**Univariate Regression**, 20 points]
   Train a univariate linear regression model to predict house prices as a function of their floor size, by running gradient descent for 200 epochs with a learning rate of 0.1. Use the dataset from the folder `hw02/data/univariate`. Plot $J(\mathbf{w})$ vs. the number of epochs in increments of 10 (i.e. after 0 epochs, 10 epochs, 20 epochs, ...). After training print the parameters and compare with solution from normal equations. Plot the training using the default blue circles and test examples using lime green triangles. On the same graph also plot the linear approximation.

3. [**Multivariate Regression**, 20 points]
   Train a multivariate linear regression model to predict house prices as a function of their floor size, number of bedrooms, and year. Run gradient descent for 500 epochs with a learning rate of 0.1. Use the the dataset from the folder `hw02/data/multivariate`. Plot $J(\mathbf{w})$ vs. the number of epochs in increments of 10 (i.e. after 0 epochs, 10 epochs, 20 epochs, ...). After training print the parameters and compare with solution from normal equations.

4. [**Polynomial Curve Fitting**, 40 points]
In this exercise, you are asked to use gradient descent to train a linear regression model, with and without regularization, on the artificial dataset from the folder `hw02/data/polyfit`.

   (a) Select 30 values for $x \in [0, 1]$ uniformly spaced, and generate corresponding t values according to $t(x) = sin(2\pi x) + x(x + 1)/4 + \epsilon$, where $\epsilon = N(0, 0.005)$ is a zero mean Gaussian with variance 0.005. Save and plot all the values. Done in `dataset.txt`.

   (b) Split the 30 samples (x_n, t_n) in three sets: 10 samples for training, 10 samples for validation, and 10 samples for testing. Save and plot the 3 datasets separately. Done in `train.txt, test.txt, devel.txt`.

   (c) Consider a linear regression model with polynomial basis functions, trained with the objective shown below:

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \left(h(x_n, \mathbf{w}) - t_n\right)^2 + \frac{\lambda}{2} ||\mathbf{w}||^2$$

   Show the gradient update used for minimizing $J(\mathbf{w})$.

   (d) Train and evaluate the linear regression model in the following scenarios:

      1. Without regularization: Using for M the value that obtained the lowest test RMSE in the first homework, run gradient descent with a learning rate that is tuned on the training data using the following consecutive powers of 10: {0.0001, 0.001, 0.01, 0.1, 1, 10}. Plot $J(\mathbf{w})$ vs. epochs and select the largest learning rate that leads to a smooth, decreasing behavior of $J(\mathbf{w})$. If convergence appears to be too slow, you may consider consecutive powers of 2 in the same range for the learning rate. Once the learning rate is selected, run gradient descent as long as $J(\mathbf{w})$ decreases by at least 1e-10 after each epoch. Report and compare the RMSE and the trained parameters with the solution from the normal equations for the same degree M.
      2. With regularization: Fixing $M = 9$ and λ to the value tuned in the first homework, repeat the experiments above, this time with regularization. Report and compare the RMSE and the trained parameters with the solution from the normal equations for the same degree M and λ.

5. [**Stochastic Gradient Descent (*)**, 20 points]
Implement SGD and run it for problems 2, 3, and 4 above, using the same hyperparameters (learning rate, number of epochs, M, and *lambda*). Compare the SGD solution to the batch GD solution. Does SGD need fewer or more epochs to arrive at a similar RMSE as batch GD? If fewer, how many epochs are sufficient?

2 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw02.zip file that contains the hw02 folder in which your code is in the 3 required files.

On a Linux system, creating the archive can be done using the command:
`> zip -r hw02.zip hw02`.

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. On the theory assignment, **clear and complete explanations and proofs of your results are as important as getting the right answer**.
4. Make sure your code runs correctly when used in the directory structure shown above.

HW Assignment 3 (Due by 10:30am on Oct 12)

1 Theory (100 points)

1. [**Maximum Likelihood**, 20 points]
The Poisson distribution specifies the probability of observing k events in an interval, as follows:

$$P(k \text{ events in interval}) = e^{-\lambda}\frac{\lambda^k}{k!} \tag{1}$$

For example, k can be the number of meteors greater than 1 meter diameter that strike Earth in a year, or the number of patients arriving in an emergency room between 10 and 11 pm[1].

Suppose we observe N samples k_1, k_2, ..., k_N from this distribution (i.e. numbers of meteors that strike Earth over a period of N years). Derive the maximum likelihood estimate of the event rate λ.

2. [**Logistic Regression**, 20 points]
Consider a dataset that contains the 4 examples below i.e., the truth table of the logical XOR function. Prove that no logistic regression model can perfectly classify this dataset. Do not forget the bias feature $x_0 = 1$.

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	0

Hint: Prove that there cannot be a vector of parameters $\mathbf{w}$ such that $P(t = 1|\mathbf{x}, \mathbf{w}) \geq 0.5$ for all examples $\mathbf{x}$ that are positive, and $P(t = 1|\mathbf{x}, \mathbf{w}) < 0.5$ for all examples $\mathbf{x}$ that are negative.

3. [**Logistic Regression**, 20 points]
Prove that the gradient (with respect to $\mathbf{w}$) of the negative log-likelihood error function for logistic regression corresponds to the formula shown in lecture 4:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \sum_{n=1}^{N}(h_n - t_n)\mathbf{x}_n \tag{2}$$

4. [**Logistic Regression**, 20 points]
In `scikit`, the objective function for logistic regression expresses the trade-off between training error and model complexity through a parameter C that is multiplied with the error term, as shown below. See the `scikit` documentation at `http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression`.

$$E(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C * \sum_{n=1}^{N}\ln(e^{-t_n(\mathbf{w}^T\mathbf{x}_n)} + 1) \tag{3}$$

[1] https://en.wikipedia.org/wiki/Poisson_distribution

- Show that the sum in the second term is equal with the negative log-likelihood.
- Compute the C parameter such that the objective is equivalent with the standard formulation shown on the slides in which the regularization parameter λ is multiplied with the L2 norm term.

5. [**Softmax Regression**, 20 points]
   Show that Logistic Regression is a special case of Softmax Regression. That is to say, if $\mathbf{w_1}$ and $\mathbf{w_2}$ are the parameter vectors of a Softmax Regression model for the case of two classes, then there exists a parameter vector $\mathbf{w}$ for Logistic Regression that results in the same classification as the Softmax Regression model.

6. [**Softmax Regression (*)**, 20 points]
   Prove that the gradient (with respect to $\mathbf{w}_k$) of the negative log-likelihood error function for regularized softmax regression corresponds to the formula shown in lecture 4, for any class $k \in [1..K]$:

$$\nabla_{\mathbf{w}_k} E(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} (\delta_k(t_n) - p(C_k|\mathbf{x}_n))\mathbf{x}_n + \alpha \mathbf{w}_k \tag{4}$$

2 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. On this theory assignment, **clear and complete explanations and proofs of your results are as important as getting the right answer**.

HW Assignment 4 (Due by 10:30am on Oct 19)

1 Implementation (150 points)

Implement two versions of the softmax regression model in Python, using (1) SciPy and (2) scikit-learn, and evaluate them on the MNIST digit recognition task. Starter code and the MNIST dataset are available at `http://ace.cs.ohio.edu/~razvan/courses/ml4900/hw04.zip`. Make sure that you organize your code in folders as shown in the table below. Write code only in the Python files indicated in bold.

```
ml4900/
    hw03/
        code/
            scipy/
                sofmax.py
                computeNumericalGradient.py
                output.txt
                softmaxExercise.py
                checkNumericalGradient.py
            scikit/
                sofmaxExercise.py
                output.txt
        data/
            mnist/
```

1.1 SciPy Implementation (100 points)

Coding effort: my implementation has 10 lines of code in softmax.py and 7 lines of code in computeNumericalGradient.py.

1. **Cost & Gradient:** You will need to write code for two functions in sofmax.py:

   (a) The `softmaxCost()` function, which computes the cost and the gradient.

   (b) The `softmaxPredict()` function, which computes the softmax predictions on the input data.

   The cost and gradient should be computed according to the formulas shown on the slides in Lecture 4.

2. **Vectorization:** It is important to vectorize your code so that it runs quickly.

3. **Ground truth:** The `groundTruth` is a matrix M such that M[c, n] = 1 if sample n has label c, and 0 otherwise. This can be done quickly, without a loop, using the SciPy function sparse.coo_matrix(). Specifically, `coo_matrix((data, (i, j)))` constructs a matrix A such that A[i[k], j[k]] = data[k], where the shape is inferred from the index arrays. The code for cumputing the ground truth matrix has been provided for you.

4. **Overflow:** Make sure that you prevent overflow when computing the softmax probabilities, as shown on the slides in Lecture 4.

5. **Numerical gradient:** Once you implemented the cost and the gradient in `softmaxCost`, implement code for computing the gradient numerically in computeNumericalGradient.py, as shown on the slides in Lecture 4. Code is provided in checkNumericalGradient.py for you to test your numerical gradient implementation.

6. **Gradient checking:** Use computeNumericalGradient.py to make sure that your softmaxCost.py is computing gradients correctly. This is done by running the main program in Debug mode, i.e. `python3 softmaxExercise.py --debug`. When debugging, you can speed up gradient checking by reducing the number of parameters your model uses. In this case, the code reduces the size of the input data, using the first 8 pixels of the images instead of the full 28x28 image.

   In general, whenever implementing a learning algorithm, you should always check your gradients numerically before proceeding to train the model. The norm of the difference between the numerical gradient and your analytical gradient should be small, on the order of 10^{-9}.

7. **Training:** Training your softmax regression is done using L-BFGS for 100 epochs, through the SciPy function scipy.optimize.fmin_l_bfgs_b(). Training the model on the entire MNIST training set of 60,000 28x28 images should be rather quick, and take less than 5 minutes for 100 iterations.

8. **Testing:** Now that you've trained your model, you will test it against the MNIST test set, comprising 10,000 28x28 images. However, to do so, you will first need to complete the function `softmaxPredict()` in softmax.py, a function which generates predictions for input data under a trained softmax model. Once that is done, you will be able to compute the accuracy of your model using the code provided. My implementation achieved an accuracy of 92.6%. If your model's accuracy is significantly less (less than 91%), check your code, ensure that you are using the trained weights, and that you are training your model on the full 60,000 training images.

1.2 Scikit Implementation (50 points)

Coding effort: my implementation has 4 lines of code in softmaxExercise.py.

You will need to write code for the following 3 functionalities:

1. **C parameter:** As explined in homework 3, `scikit`'s objective function expresses the trade-off between training error and model complexity through a parameter C that is multiplied with the erorr term. Compute the C parameter such that the objective is equivalent with the standard formulation used in `scipy` that multiplies the regularization parameter (called 'decay' in the code) with the L2 norm term.

2. **Softmax training:** Train a softmax regression model using the 'multinomial' option for multiclass classification, and the C parameter computed above. Specify training with the L-BFGS solver for 100 max iterations. For this, you will instantiate the class `linear_model.LinearRegression`.

3. **Softmax testing:** USe the trained softmax model to compute labels on hte test images.

The code also computes and prints the accuracy on the test images.

2 Bonus (25 points)

Create and evaluate a new version of the SciPy code that trains the softmax regression model using minibatch gradient descent for 2000 updates, where the size of a minibatch is 100.

3 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw04.zip file that contains the hw04 folder in which you write code **only in the 3 required files**. The screen output produced when running the softmaxExercise.py code should be redirected to (saved into) the **output.txt** files.

On a Linux system, creating the archive can be done using the command:
`> zip -r hw04.zip hw04`.

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. Make sure your code runs correctly when used in the directory structure shown above.

HW Assignment 5 (Due by 10:30am on Nov 2)

1 Theory (110 points)

1. [**Properties of Linear Discriminants**, 20 points]
We have proven in class that the distance between origin and the decision hyperplane $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$ is equal with $-w_0/\left\|\mathbf{w}\right\|$. Prove that the margin between a point $\mathbf{x}$ and the same decision hyperplane is equal with $h(\mathbf{x})/\left\|\mathbf{w}\right\|$.

2. [**Bonus**, 20 points]
Prove the two properties above for the general n-dimensional case.

3. [**Fisher Criterion and Least Squares**, 30 points]
Show that the Fisher criterion can be written in the vectorized form shown below:
$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}}$$

4. [**Fisher Criterion (*)**, 20 points]
Reference the PRML Chapter 4 material available on Blackboard under Content.
Using the definitions of the between-class and within-class covariance matrices given by (4.27) and (4.28), respectively, together with (4.34) and (4.36) and the choice of target values described in Section 4.1.5, show that the expression (4.33) that minimizes the sum-of-squares error function can be written in the form (4.37).

5. [**Perceptrons**, 40 points]
Consider a training set that contains the following 8 examples:

$\mathbf{x}$	x_1	x_2	x_3	$t(x)$
$\mathbf{x}^{(1)}$	0	0	0	+1
$\mathbf{x}^{(2)}$	0	1	0	+1
$\mathbf{x}^{(3)}$	1.5	0	-1.5	+1
$\mathbf{x}^{(4)}$	1.5	1	-1.5	+1
$\mathbf{x}^{(5)}$	1.5	0	0	–1
$\mathbf{x}^{(6)}$	1.5	1	0	–1
$\mathbf{x}^{(7)}$	0	0	-1.5	–1
$\mathbf{x}^{(8)}$	0	1	-1.5	–1

   (a) Prove that the perceptron algorithm does not converge on this dataset. Do not forget to include the bias.

   (b) Consider a kernel perceptron that uses a polynomial kernel $k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T\mathbf{y})^d$. What is the smallest degree d for which the kernel perceptron would converge on this dataset?

6. [**Perceptrons**, 10 points]
A kernel perceptron for binary classification is run for a number of epochs E on a training dataset containing N examples, resulting in the dual parameters α_1, α_2, ..., α_N. What is the total number of mistakes that are made during training?

7. [**Matrix Computations**, 10 points]
   Let $U \in R_{k \times m}$ and $X \in R_{n \times m}$. Let u_i and x_i be the i-th columns of U and X, respectively, for $1 \leq i \leq m$. Prove that $UX^T = \sum_{i=1}^{m} u_i x_i^T$.

2 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. On this theory assignment, **clear and complete explanations and proofs of your results are as important as getting the right answer**.

HW Assignment 6 (Due by 10:30am on Nov 16)

1 Implementation (150 points)

Implement the 3 versions of the perceptron algorithm discussed in class: the perceptron, the average perceptron, and the kernel perceptron. The algorithms should stop after achieving convergence, or after a predefined number of epochs T, whichever comes first. Make sure that you organize your code in folders as shown in the table below. Write code only in the Python files indicated in bold.

```
ml4900/
    hw06/
        code/
            perceptron.py
            exercise5.py
            spam_exercise.py
            newsgroups_exercise.py
            spam_model_p.txt
            spam_model_ap.txt
            newsgroups_model_p1.txt
            newsgroups_model_ap1.txt
            newsgroups_model_p2.txt
            newsgroups_model_ap2.txt
            output.txt
        data/
            spam/
                spam_train.txt
                spam_train_svm.txt
                spam_test.txt
                spam_test_svm.txt
                spam_vocab.txt
            newsgroups/
                stopwords.txt
                newsgroups_train1.txt
                newsgroups_test2.txt
                newsgroups_train2.txt
                newsgroups_test2.txt
                newsgroups_vocab.txt
```

1. **Perceptron Convergence:** Validate experimentally the conclusions you reached for Exercise 5 in the previous assignment. For 5(a), show that the perceptron algorithm hits the same weight vector at different epochs during training and thus it will run forever. For 5(b), show the number of epochs the kernel perceptron needed to converge and the values of the dual parameters at the end. Make sure you process the training examples in the order given in the assignment. In **perceptron.py** you should implement these five functions:

(a) `perceptron_train(data, labels, epochs)` trains the vanilla perceptron algorithm and returns the weight vector.

(b) `aperceptron_train(data, labels, epochs)` trains the average perceptron algorithm and returns the average weight vector.

(c) `perceptron_test(w, data)` test a perceptron with weights w and returns a vector with the labels on the test examples in data.

(d) `kperceptron_train(data, labels, epochs, kernel)` trains the kernel perceptron algorithm and returns the vector α of parameters.

(e) `kperceptron_test(alpha, tlabels, tdata, data, kernel)` tests the kernel perceptron algorithm with parameters alpha, support vectors tdata and their labels tlabels, and returns a vector with the labels on the test examples in data.

Write the code for validating Exercise 5 in the file **exercise5.py**.

2. **Spam vs. Non-Spam:** In this problem, you will train and evaluate spam classifiers using the perceptron and average perceptron algorithms. The dataset contains two files: `spam_train.txt` with 4,000 training examples and `spam_test.txt` with 1,000 test examples. The dataset is based on a subset of the SpamAssassin Public Corpus. Each line in the training and test files contains the pre-processed version of one email. The line starts with the label, followed by the email tokens separated by spaces.

   Figure 1 shows a sample source email, while Figure 2 shows its pre-processed version in which web addresses are replaced with the "httpaddr" token, numbers are replaced with a "number" token, dollar amounts are replaced with "dollarnumb", and emaill addresses are replaced with "emailaddr". Furthermore, all words are lower-cased, HTML tags are removed, and words are reduced to their stems i.e. "expecting", "expected", "expectation" are all replaced with "expect". Non-words and punctuation symbols are removed.

> Anyone knows how much it costs to host a web portal ?
> Well, it depends on how many visitors youre expecting. This can be anywhere from less than 10 bucks a month to a couple of $100. You should checkout http://www.rackspace.com/ or perhaps Amazon EC2 if youre running something big..
>
> To unsubscribe yourself from this mailing list, send an email to: groupname-unsubscribe@egroups.com

Figure 1: Sample email from the SpamAssassin corpus.

anyon know how much it cost to host a web portal well it depend on how mani visitor your expect thi can be anywher from less than number buck a month to a coupl of dollarnumb you should checkout httpaddr or perhap amazon ecnumb if your run someth big to unsubscrib yourself from thi mail list send an email to emailaddr

Figure 2: Pre-processed version of email from the SpamAssassin corpus.

(a) Create a vocabulary file `spam_vocab.txt` that contains a list of only the (preprocessed) tokens that appear at least 30 times in the training examples. The file should contain one token per line in the format `<id> <token>`, where each token is associated a unique integer identifier. The tokens should be listed in increasing order of their identifiers, starting from 1. See for example the vocabulary file `newsgroups_vocab.txt` that we generated for the newsgroup classification problem. Implement the function that creates the vocabulary in **spam_exercise.py**.

(b) For each training and test example, create a sparse feature vector representation wherein each example is represented as one line in the file using the format `<label> <id1>:<val1> <id2>:<val2> ...`, where the id's are listed in increasing order and correspond only to tokens that appear in that example (use 1 for all values, representing that fact that the corresponding token appeared in the example). An example of this sparse representation can be seen in the file `newsgroups_train1.txt` that we generated for the newsgroup classification problem. Save the new version of the dataset in the files `spam_train_svm.txt` and `spam_test_svm.txt`. Implement the function that creates the sparse feature vector representations in **spam_exercise.py**.

(c) Write an additional function `read_examples(file_name)` inside **perceptron.py** that reads all examples from a file with sparse feature vectors and returns a tuple `(data, labels)` where the `data` is a two dimensional array containing all feature vectors, one per row, in the same order as in the file, and the `labels` is a vector containing the corresponding labels.

(d) Train the perceptron algorithm until convergence, by reading the training examples from `spam_train_svm.txt` and by calling `perceptron_train(data, labels, epochs)` in the code you write inside **spam_exercise.py**. Process the examples in the order they are listed in the files. Report the number of epochs needed for convergence and the total number of mistakes made and save the returned parameter vector in `spam_model_p.txt`. Test the perceptron algorithm by reading the parameter vector from `spam_model_p.txt` and the test examples from `spam_test_svm.txt` and calling `perceptron_test(w, data)` in the code inside **spam_exercise.py**. Report the test accuracy.

(e) Train the average perceptron algorithm until the corresponding perceptron convergences, by reading the training examples from `spam_train_svm.txt` and by calling `aperceptron_train(data, labels, epochs)` in the code inside **spam_exercise.py**. Process the examples in the order they are listed in the files. Report the number of epochs needed for convergence and the total number of mistakes made and save the returned average parameter vector in `spam_model_ap.txt`. Test the average perceptron algorithm by reading the parameter vector from `spam_model_ap.txt` and the test examples from `spam_test_svm.txt` and calling `perceptron_test(w, data)` in the code inside **spam_exercise.py**. Report the test accuracy. Compare the test accuracy between the perceptron and the average perceptron.

3. **Atheism vs. Religion:** In this problem, you will train and evaluate the binary perceptron and average perceptron algorithms on a subset of the 20 newsgroups dataset. In this subset, there are 857 positive example and 570 test examples, on the topics of atheism and religion. Newsgroup postings on the topic of Atheism (`alt.atheism`) are given

label 1, whereas newsgroup posting on the topic of Religion (`talk.religion.misc`) are given label -1. Thus, the models will be trained to distinguish between postings on Atheism and postings on Religion.

The feature vectors have already been created for you and are stored in files using the sparse feature vector representation described above. To create these feature vectors, we stripped metadata, quotes, and headers from the documents. The words were stemmed and tokens that appeared less than 20 times in the training examples were filtered out. Common tokens from the `stopwords.txt` file were also removed. The remaining tokens are stored in the vocabulary file `newsgroups_vocab.txt` and are used to create two versions of the dataset:

- [**Version 1**] In this version, each token corresponds to a feature whose value for a particular document is computed using the standard *tf.idf* formula from Information Retrieval (think search engines). The term frequency *tf* refers to the number of times the token appears in the document, whereas the inverse document frequency *idf* refers to the inverse of the log of the total number of documents that contain the token. The *idf* numbers are computed using the entire 20 newsgroup dataset. The two quantities are multiplied into one *tf.idf* value and are meant to give more importance to words that are rare (i.e. large *idf*) and appear more frequently inside the corresponding document example (i.e. large *tf*). The training and test examples thus created are stored in `newsgroups_train1.txt` and `newsgroups_test1.txt` respectively.
- [**Version 2**] This is the same as version 1 above, except that the term frequencies are set to 1 for all tokens that appear in a document, i.e. the number of times a token appears in the document is irrelevant and the only thing that matters is whether the token appeared or not in the document, and also how rare it is (through the *idf* weight). The training and test examples for this version are stored in `newsgroups_train2.txt` and `newsgroups_test2.txt` respectively.

For more details on how the feature vectors were created, you can read the Scikit section at `http://scikit-learn.org/stable/datasets/twenty_newsgroups.html`.

For each version of the dataset, use the perceptron and average perceptron implementations that you wrote in **perceptron.py** and train the two algorithms for 10,000 epochs. Process the examples in the order they are listed in the files. Save the parameters in the corresponding `newsgroups_model_<x>.txt` files. Then evaluate the two perceptron algorithms on the corresponding test examples for each version. For each version, report and compare the accuracies of the two models.

4. **{Spam vs. Non-Spam} vs. {Atheism vs. Religion}:** Why is the accuracy on the second dataset (atheism vs. religion) much lower than the accuracy on the first dataset (spam vs. non-spam)? To answer this question accurately, you may want to download the original documents in the two datasets and look at some of them.

2 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw06.zip file that contains the hw06 folder in which you write code **only in the required files**. The screen output produced when running the code should be redirected to (saved into) the **output.txt** file.

On a Linux system, creating the archive can be done using the command:
`> zip -r hw06.zip hw06`.

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. Make sure your code runs correctly when used in the directory structure shown above.

HW Assignment 7 (Due by 10:30am on Nov 30)

1 Theory (140 points)

1. [**Normalized Kernels**, 50 + 25 points]
   Let $K : X \times X \to R$ be a kernel function defined over a sample space X.

   (a) Prove that the function below (a *normalized kernel*) is a valid kernel.

   $$\frac{K(x, y)}{\sqrt{K(x, x)K(y, y)}}$$

   (b) Why it would not make sense to normalize a Gaussian kernel?

   (c) [**Bonus**] Which types of input data would benefit from normalizing the kernel function? Explain why, and provide real world examples.

2. [**Support Vector Machines**, 50 points]
   Prove that the sum of slacks $\sum \xi_n$ from the objective function of the SVM formulation with soft margin is an upper bound on the number of misclassified training examples.

3. [**Max Margin Hyperplanes**, 20 points]
   Consider the constrained optimization SVM problem for the separable case shown on slide 12. Show that, if the 1 on the right-hand side of the inequality constraint is replaced by some arbitrary constant $\gamma > 0$, the resulting maximum margin hyperplane is unchanged.

4. [**Kernel Techniques**, 20 + 20 points]
   (a) Show that if $k_1(\mathbf{x}, \mathbf{y})$ and $k_2(\mathbf{x}, \mathbf{y})$ are valid kernel functions, then $k(\mathbf{x}, \mathbf{y}) = k_1(\mathbf{x}, \mathbf{y})k_2(\mathbf{x}, \mathbf{y})$ is also a valid kernel.
   (b) **(*)** Show that if A is a symmetric positive semidefinite matrix, then $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A\mathbf{y}$ is a valid kernel. *Hint: Inderjit Dhillon's Linear Algebra Background describes some useful properties of symmetric positive semidefinite matrices.*

5. [**Positive Definite Matrices (*)**, 20 points]
   Show that a diagonal matrix $\mathbf{W}$ whose elements satisfy $0 < W_{ii} < 1$ is positive definite. Show that the sum of two positive definite matrices is itself positive definite.

6. [**Large Margin Perceptron (*)**, 30 points]
   Let $\mathbf{u}$ be a current current vector of parameters and $\mathbf{x}$ and $\mathbf{y}$ two training examples such that $\mathbf{u}^T(\mathbf{x} - \mathbf{y}) < 1$. Use the technique of Lagrange multipliers to find a new vector of parameters $\mathbf{w}$ as the solution to the convex optimization problem below:

   minimize:
   $J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w} - \mathbf{u}\|^2$

   subject to:
   $\mathbf{w}^T(\mathbf{x} - \mathbf{y}) \geq 1$

2 Text Classification (100 points)

Train and test the SVM algorithm on the *Spam vs. Non-spam* and *Atheism vs. Religion* classification problems, using the datasets provided for the previous assignment. Use a linear kernel, with the cost parameter $C = 5$. Report and compare the accuracy of the trained SVM models with the perceptron and average perceptron accuracies from the previous assignment.

3 Digit Recognition (200 points)

In this exercise, you are asked to run an experimental evaluation of SVMs and the perceptron algorithm, with and without kernels, on the problem of classifying images representing digits.

1. The UCI Machine Learning Repository at *www.ics.uci.edu/~mlearn* maintains datasets for a wide variety of machine learning problems. For this assignment, you are supposed to work with the Optical Recognition of Handwritten Digits Data Set. The webpage for this dataset is at:
   http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits
   The actual dataset is located at:
   http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/

   Read the description of the dataset. Download the training set `optdigits.tra` and the test set `optdigits.tes`. Use the first 1000 examples in `optdigits.tra` for *development* and the rest of 2823 examples for *training*. Use all 1797 examples in `optdigits.tes` for *testing*. Scale all the features between [0, 1], as discussed in class, using the min and max computed over the training examples. Create training files for each of the 10 digits, setting the class to 1 for instances of that digit, and to -1 for instances of other digits, i.e. *one-vs-rest* scenario.

2. Train first the linear perceptron, with the number of epochs set to $T \in \{1, 2, ..., 20\}$. After training each linear perceptron, normalize the learned weight vector. Select for T the value that obtains the best overall accuracy on the development data, and use this value for the remaining perceptron experiments.

   Run experiments with the linear and kernel perceptron algorithms. For the kernel perceptron, experiment with polynomial kernels $k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T\mathbf{y})^d$ with degrees $d \in \{2, 3, 4, 5, 6\}$, and with Gaussian kernels $k(\mathbf{x}, \mathbf{y}) = exp(-||\mathbf{x} - \mathbf{y}||^2/2\sigma^2)$ with the width $\sigma \in \{0.1, 0.5, 2, 5, 10\}$. For each hyper-parameter value, you will have trained 10 models, one for each digit. In order to compute the label for a test or development example, you will run the 10 trained models and output the label that obtains the highest score. Compute the accuracy on the development data and identify the hyper-parameter value that obtains the best accuracy. Use the tuned hyper-parameter (d for poly-kernel, σ for Gaussian) to compute the overall performance on the test data.

   For each of the three perceptrons (linear, poly kernel and Gaussian kernel) report the total training time, the overall accuracy, and the number of support vectors. Show and compare the corresponding 4 confusion matrices. Which digit seems to be the hardest to classify? Which perceptron / kernel combination achieves the best performance? Which algorithms are slower at training time, and why?

3. Run the same experiments using SVMs instead of perceptrons, i.e. linear SVMs and SVMs with polynomial and Gaussian kernels. Use the same tuning scenarios for the hyper-parameters of the polynomial and Gaussian kernels. Use $C = 1$ in all SVM experiments. Report the same types of results and analysis as above, and compare with the perceptron results.

4 Tools

You are free to use MATLAB, R, or packages written in C++/Java/Python such as SVM-LIGHT (C), LIBSVM (C++, Java), or SCIKIT-LEARN (Python) to complete the implementation part of this assignment. Their web sites contain plenty of documentation on how to use them. If you use SCIKIT-LEARN, the following functionality from the `sklearn.svm` will be useful:

1. `SVC()`: This is the main class used for SVM classification models. Its implementation is based on LIBSVM. Make sure that you properly map the SVM hyper-parameters to the parameters in the constructor of this class. For example, the *gamma* parameter in the constructor corresponds to our $1/2\sigma^2$ coefficient in the Gaussian kernel. The formulas for the kernels implemented by SVC are described in this User Guide.

2. `decision_function(x)`: Once the classifier is trained, this will compute the distance between a sample `x` and the decision hyperplance. This is the quantity that you can use to determine the highest scoring class when training the 10 $one - vs - rest$ classifiers: once a classifier is trained for all 10 digits, given a sample `x` you compute this quantity for all 10 classifiers and select the class that corresponds to the classifier with largest decision function value.

3. `fit()`: This is the function used to train the classifier.

4. `predict(x)`: This is used to calculate the (binary) label for sample `x`.

LIBSVM, and therefore SCIKIT-LEARN too, already implement the *one-vs-rest* classification scheme. In this scheme, you can directly use the training dataset with the 10 original labels, and SCIKIT-LEARN will train the 10 binary classifiers for you. You can use this capability for this assignment, however bonus points will be given if you train the 10 binary classifiers directly, as described for the perceptron algorithm above, by creating a binary training dataset for each class.

5 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Blackboard a hw07.zip file that contains the hw07 folder in which you place the code and the datasets. Make sure you include a README.txt file explaining how the code is supposed to be used to replicate the results included in the report. The screen output produced when running the code should be redirected to (saved into) an **output.txt** file.

On a Linux system, creating the archive can be done using the command:

```
> zip -r hw07.zip hw07
```

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. **Do not submit third-party ML packages on Blackboard!** Just explain in the REAMDE file how you use external packages.
4. Make sure your code runs correctly when used in the directory structure shown above.
5. **Type and nicely format the project report**, including discussion points, tables, graphs etc. so that it is presentable and easy to read.
6. Working code and/or correct answers is only one part of the assignment. The project report, including discussion of the specific issues which the assignment asks about, is also a very important part of the assignment. Take the time and space to make an adequate and clear project report. On the non-programming learning-theory assignment, clear and complete explanations and proofs of your results are as important as getting the right answer.

HW Assignment 8 (Due by 10:30am on Dec 7)

1 Theory (150 points)

1. [**Kernel Nearest Neighbor**, 50 points]
   The nearest-neighbour classifier 1-NN assigns a new input vector $\mathbf{x}$ to the same class as that of the nearest input vector $\mathbf{x}_n$ from the training set, where in the simplest case, the distance is defined by the Euclidean metric $\|\mathbf{x} - \mathbf{x}_n\|^2$. By expressing this rule in terms of scalar products and then making use of kernel substitution, formulate the nearest-neighbour classifier for a general nonlinear kernel.

2. [**Distance-Weighted Nearest Neighbor**, 50 points]
   We have seen how to use kernels to formulate a distance-weighted nearest neighbor algorithm, when the labels are binary. Formulate a kernel-based, distance-weighted nearest neighbor that works for K classes, where $K \geq 2$.

3. [**Naive Bayes**, 50 points]
   The Naive Bayes algorithm for text categorization presented in class treats all sections of a document equally, ignoring the fact that words in the title are often more important than words in the text in determining the document category. Describe how you would modify the Naive Bayes algorithm for text categorization to reflect the constraint that words in the title are K times more important than the other words in the document for deciding the category, where K is an input parameter (include pseudocode).

4. [**Logistic Regression (*)**, 50 points]
   Assume that a binary feature x_i is equal to 1 for all training examples $\mathbf{x}$ belonging to a particular class C_k, and zero otherwise (i.e. x_i perfectly separates examples from class C_k from all other examples). Show that in this case the magnitude of the ML solution for $\mathbf{w}_k$ goes to infinity, thus motivating the use of a prior over the parameters (Hint: use the fact that the gradient on slide 24 must vanish at the solution).

2 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. On this theory assignment, **clear and complete explanations and proofs of your results are as important as getting the right answer**.