CS 4780/5780 Machine Learning Due November 1, 2011

Assignment 4: Kernels and Generative Models

Instructor: Thorsten Joachims *Total Points: 100*

Course Policy: *Assignments are due at the start of class on due-date in hard copy.*
Write your NetIDs on the first page of the submitted hard-copy.
Late assignments lose 5 points for each late day and can be submitted directly to the TAs.
Assignments submitted after solutions are made available will not be accepted.
Groups of two are strongly encouraged.
All sources of material must be cited.
Assignment solutions will be made available along with the graded homework solutions.
The University Academic Code of Conduct will be strictly enforced, including running cheating detection software.
Submitted code must follow a certain directory structure. Please read the relevant section carefully for instructions.

Problem 1: Kernels [35 points]

(a) [**5 points**] Consider the unbiased perceptron algorithm. Suppose we are given a kernel function $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. Modify the perceptron learning algorithm to use K. Provide both the updates during training and the decision rule during prediction. You **do not** need to implement the modified algorithm in code; just work out the algorithm on paper.

(b) [**10 points**] A TA claimed during office hours that kernels allow us to fit non-linear classifiers in the instance space; lets examine if there is any benefit to be achieved in practice. The task we shall attempt is automatic recognition of handwritten digits. The dataset available at UCI Machine Learning Repository has been preprocessed to adhere to SVM-light format and made available on CMS as *digits.zip*. The zipped folder contains 10 training sets and 10 validation sets (one pair for each digit) and one overall train, validation and test set. To start the experiment, we intend to learn 10 models (one for each digit) which predicts whether that digit was drawn or not. We will use the 10 pairs of training and validation sets and the overall test set for this experiment.

Use SVM^{light} to train 10 different biased classifiers with a linear kernel for each of the training sets. Use the validation set for each digit to arrive at good values for C (you may consider the range of $C = \{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$).

Note, the best value of C for a classifier for a digit can be different from the best C for another digit. Tabulate the accuracies you get on the validation set for each digit as you vary C. Keep aside the best performing model for each digit for the next step (provide a brief explanation for why you picked a model to be the best performing model).

For each of your models (0-9), run `svm_classify` on the overall test set. After you have run all the classifications, find the highest of the 10 predictions for each test instance, and assign it the corresponding class (if the highest prediction is a negative value, count that instance as an error). What is the accuracy on the overall test set?

Include representative commands (ie, the command-line arguments) you use for training for one of the digits.

(c) [**10 points**] We will now see the effect of non-linear kernels on performance. Train 10 different biased SVMs with polynomial kernel for each of the training sets, use the corresponding digit's validation set to tune the value of C and the degree of the polynomial, d (you may consider the range of $d = \{2, 3, 4, 5\}$). Note that, the d value will be same across the 10 classifiers (so, fix a value for d, tune C as you did in the first experiment; pick that value of d that produces least errors across all digits). Keep aside the best performing model for each digit for overall prediction. Run each model on the overall test set and use the highest of the 10 predictions to label each test instance. What is the accuracy on the overall test set now?

Include representative commands (ie, the command-line arguments) you use for training for one of the digits.

(d) [**10 points**] We will now try to formulate an SVM that will directly optimize for multiclass prediction. The corresponding quadratic program looks like

$$\begin{aligned} &\min_{\vec{w_1},\vec{w_2},\ldots,\vec{w_k},\xi} && \sum_{j=1}^{k} \frac{||\vec{w_j}||^2}{2} + C\sum_{i=1}^{n} \xi_i \\ &s.t. \forall i = 1, \ldots, n && \\ &\forall y = 1, \ldots, k && \vec{w_{y_i}} \cdot \vec{\phi}(\vec{x_i}) \geq \vec{w_y} \cdot \vec{\phi}(\vec{x_i}) + \mathbb{1}_{[y_i \neq y]} - \xi_i \end{aligned} \tag{1}$$

The program has a very similar form to the binary classification optimization problem, the major difference is there is a weight vector associated with each class.[1] Let each instance be drawn from $\vec{x} \in \mathbb{R}^N$ and let there be k classes for our multiclass prediction. To predict a class, we pick $h(\vec{x}) = \arg\max_{y \in \{1 \ldots k\}} \vec{w_y} \cdot \vec{\phi}(\vec{x})$.

Use $SVM^{multiclass}$ to train a *linear kernel* multiclass SVM on the overall training set. Use the overall validation set to tune C. (Note that C scales differently in multiclass

[1]This treatment is identical to the earlier formulation in the handout(if we concatenate the k weight vectors in this formulation into one long vector)

SVM's, so you may wish to consider

the range of $C = \{0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5\}$). Report the accuracy you get with this classifier on the overall test set.

Include representative commands (ie, the command-line arguments) you use for training and testing.

Problem 2: Generative Models [25 points]

(a) [**5 points**] Linear Discriminant Analysis is named suggestively: perhaps it can be rewritten as a linear rule. Remember, the decision function for LDA to classify an input vector $\vec{x} \in \mathbb{R}^N$ given class means $\vec{\mu_+}$ and $\vec{\mu_-}$ is

$$h(\vec{x}) = \arg\max_{y \in \{+1,-1\}} Pr(Y = y)e^{\frac{-1}{2}(\vec{x}-\mu_y)^2} \tag{2}$$

Assuming we have estimates for all the appropriate probabilities (none of them zero) and we have the class means, can we rewrite $h(\vec{x}) = sign(\vec{v} \cdot \vec{x} + b)$? If so, provide $\vec{v}$ and b.

(b) [**10 points**] We will attempt to write the *multinomial* Naïve Bayes Classifier Decision rule as a linear rule. Remember, a document is viewed as a sequence $d = (w_1, w_2, \ldots, w_l)$ of l words and is classified as

$$h(d) = \arg\max_{y \in \{+1,-1\}} Pr(Y = y) \prod_{i=1}^{l} Pr(W = w_i | Y = y) \tag{3}$$

Assuming we have estimates for all the appropriate probabilities and none of them are zero, can we rewrite $h(d) = sign(\vec{v} \cdot \vec{x} + b)$? If so, provide $\vec{v}, \vec{x}$ and b. *Hint for $\vec{x}$: You might find the construction of $\vec{x}$ such that each component corresponds to a word in the vocabulary particularly helpful.*

(c) [**10 points**] We learnt in class that the Naïve Bayes Classifier makes an independance assumption which, given the suggestive name, is probably very naïve. Consider the *multivariate* Naive Bayes Classifier for a binary classification problem, where the input features are all binary. Construct a training set (ie, provide the necessary probability estimates for the class priors and model for each class) and a test point $\vec{x}$ such that the naïve Bayes labelling and the Bayes-optimal labelling of $\vec{x}$ differ. Provide calculations to prove your construction answers the question. *Hint: You may wish to consider the case when two of the input attributes are dependant (say, equal).*

Problem 3: Naïve Bayes Classifier Implementation [40 points]

(a) [**10 points**] We will attempt automatic classification of documents as pertaining to *Astrophysics* or not; a subset of the arXiv corpus of scientific papers has been preprocessed to adhere to SVM-light format and made available on CMS as *arxiv.zip*.

The zipped folder contains four files: Each line in the *arxiv.train/test* file is one instance, the first field indicating if the instance was in the Astrophysics category (+1) or not (-1). The remaining fields in a line indicate

`<Index of word in dictionary>:<Number of occurrences of word in doc>`.

Every instance in the *arxiv.norm.train/test* has the class label and remaining fields indicate the *TFIDF* score of each word (Term Frequency $*$ Inverse Document Frequency; Number of occurrences of word in doc weighted by the logarithm of the inverse of that term's document frequency in the corpus) normalised to unit length.

You can use the maximum index you see in the training file as the size of the vocabulary. To start things off, use SVM^{light} to train on *arxiv.norm.train* and test on *arxiv.norm.test*. Report the accuracies obtained for the default value of C. Next, use 4 fold cross validation to arrive at a good value for C (you may consider

the range of $C = \{0.001, 0.01, 0.1, 1\}$).

Report the best C you get, and the accuracy on the test set for this C.

Also run the experiment using *arxiv.train* as training set (crossvalidating C as before) and test on *arxiv.test*. Does normalization of feature vector lengths seem to help? Explicitly list out the number of false positives and false negatives. (A false positive is when our algorithm predicts the class to be "True" but the actual label is "False", and a false negative is when we predict "False" when the actual value is "True").

(b) [**10 points**] Next, implement a Multinomial Naïve Bayes Classifier in your favorite programming language. Remember, the conditional probability of a word w being in a document d is estimated as

$$Pr(W = w|Y = y) = \frac{\text{Number of occurrences of w in documents in class y} + 1}{\text{Number of all words in documents in class y} + \text{Size of vocabulary}} \tag{4}$$

$$Pr(Y = y) = \frac{\text{Number of documents in class y}}{\text{Number of all documents}} \tag{5}$$

The decision rule is given in an earlier question. In case of ties, predict the class that occurs more often in the training set.

To avoid underflow, you should not compare products of probabilities directly but rather compare their logarithms instead (remember, the logarithm of a product is a sum of logarithms). What accuracy do you get on the test set? Explicitly list out the number of false positives and false negatives.

(c) **[10 points]** We will now examine cost sensitive classification; in several real-world classification tasks, the cost of a false positive is not equal to the cost of a false negative; let mislabelling a true instance as false have cost c_{10}, and mislabelling a false instance as true have cost c_{01}. There are also real-life costs for attaining positive examples (c_{11}) and negative examples (c_{00}). The setting discussed in class is the case when $c_{00} = c_{11} = 0; c_{01} = c_{10} = 1$. In our current setting, we note that a user looking for articles on astrophysics will tend to have non-symmetric costs for false positives and false negatives: for instance, missing one document talking about astrophysics could be just as bad as having to look at a bunch of documents about things other than astrophysics. So, rather than optimizing for accuracy on the test set, we must optimize for this user's cost. For this question, $c_{00} = c_{11} = 0; c_{01} = 1; c_{10} = 10$. The decision rule for the multinomial naïve bayes classifier now becomes,

$$\text{Label instance as } +1 \text{ if } (c_{10}-c_{11}) \cdot Pr(Y = +1|X = \vec{x}) \geq (c_{01}-c_{00}) \cdot Pr(Y = -1|X = \vec{x}) \tag{6}$$

Implement the cost sensitive classifier for the given c values and report your accuracy on the test set. Explicitly list out the number of false positives and false negatives. Can you provide an intuitive connect between the number of false positives, false negatives and c_{01}, c_{10}?

(d) **[5 points]** The SVM formulation we discussed in class can also handle unequal costs for false positives and false negatives (visit SVM^{light} and observe the `-j` option. Remember, there is a slack variable associated with each instance in the SVM optimization problem. With the `-j` option, the objective of the quadratic program is modified so that the coefficient of each slack variable corresponding to a positive instance in the training set is multiplied by the supplied value. The SVM program looks like,

$$\begin{aligned} \min_{\vec{w},\xi} \quad & ||\vec{w}|| + C \cdot \gamma \sum_{y_i=+1} \xi_i + C \sum_{y_i=-1} \xi_i \\ s.t. \forall i = 1, \ldots, n \quad & y_i \cdot (\vec{w} \cdot \vec{\phi}(\vec{x_i})) \geq 1 - \xi_i \end{aligned} \tag{7}$$

For our case, the cost quotient will be c_{10}/c_{01}. Use SVM^{light} to train on *arxiv.norm.train* and test on *arxiv.norm.test*. Use 4 fold cross validation to arrive at a good value for C (you may consider the range of C as in the first subquestion). *In your report, state what value you are optimizing for during cross-validation.* List the number of false positives and false negatives for each fold of the cross-validation you perform.

Report the best C you get, and the accuracy on the test set for this C. Use the predictions file generated to list out the number of false positives and false negatives on the test set.

(e) [**5 points**] Suggest two benefits and two disadvantages of Naïve Bayes Classifiers versus Support Vector Machines.

(f) **Code Organization:** Your code should be organized in the following directory structure. Upload a file called *code.zip* which contains four folders: *bin*, *doc*, *lib*, and *src*. If you used any third-party binaries, they go in the *bin* folder, the *README.txt* and any other documents go in the *doc* folder, any third party libraries used go to the *lib* folder. Organize all your source code (including Makefiles and other build scripts) in the *src* directory.