Machine Learning - Homework 3

Parita Pooj (psp2133)

October 12, 2016

Problem 1

(a) **Centering**: (No) The transformation does not affect the learning algorithm. Centering will basically make the mean zero. This can be shown as below:
Let $\hat{\boldsymbol{\mu}}'$ be the new mean parameter trained on the transformed data.
$\hat{\boldsymbol{\mu}}' = \frac{1}{n}\sum_{i=0}^{n}(\boldsymbol{x} - \hat{\boldsymbol{\mu}})$
$\hat{\boldsymbol{\mu}}' = \frac{1}{n}\sum_{i=0}^{n}\boldsymbol{x} - \dfrac{1}{n}\sum_{i=0}^{n}\hat{\boldsymbol{\mu}})$
$\hat{\boldsymbol{\mu}}' = \hat{\boldsymbol{\mu}} - \hat{\boldsymbol{\mu}}$
$\hat{\boldsymbol{\mu}}' = 0$
Thus, centering essentially transforms the mean to zero, but the distribution still remains the same and hence, the classification won't be affected.

Standardization: (No) Standardization does not affect the learning algorithm. Standardization makes the standard deviation 1 for each feature, thus not affecting the classification, same as above.

(b) **Centering**: (No) Centering preserves the order of the Euclidean distance between every pair of points. Hence, 1-NN classifier will not be affected. The preservation of the order of distances can be shown by considering three points $\boldsymbol{x_p}$, $\boldsymbol{x_q}$ and $\boldsymbol{x_r}$ such that:
$\sum_{i=1}^{n}(x_{p,i} - x_{q,i})^2 \leq \sum_{i=1}^{n}(x_{p,i} - x_{r,i})^2$
For transformed points, $\boldsymbol{x'_p}$, $\boldsymbol{x'_q}$ and $\boldsymbol{x'_r}$ we see that
$\sum_{i=1}^{n}(x'_{p,i} - x'_{q,i})^2 \leq \sum_{i=1}^{n}(x'_{p,i} - x'_{r,i})^2$
since,
$\sum_{i=1}^{n}(x_{p,i} - \hat{\mu}_i - x_{q,i} + \hat{\mu}_i)^2 \leq \sum_{i=1}^{n}(x_{p,i} - \hat{\mu}_i - x_{r,i} + \hat{\mu}_i)^2$

Standardization: (No) Standardization also doesn't affect the learning algorithm since it preserves the order. As above, for transformed points, $\boldsymbol{x'_p}$, $\boldsymbol{x'_q}$ and $\boldsymbol{x'_r}$ we see that

$$\sum_{i=1}^{n}(x'_{p,i} - x'_{q,i})^2 \leq \sum_{i=1}^{n}(x'_{p,i} - x'_{r,i})^2$$

since,

$$\sum_{i=1}^{n}(\frac{x_{p,i} - \hat{\mu}_i - x_{q,i} + \hat{\mu}_i}{\sigma_i})^2 \leq \sum_{i=1}^{n}(\frac{x_{p,i} - \hat{\mu}_i - x_{r,i} + \hat{\mu}_i}{\sigma_i})^2$$

(c) **Centering**: (No) The transformation does not affect the learning algorithm. The Gini Index uncertainty measure considers the probability distribution of the points with respect to the axis split.
Centering essentially doesn't affect the algorithm because the axis split undergoes an equivalent transformation, preserving the distribution with respect to the split. Due to this, the classification still remains the same.
This can be shown with a simple example for feature i. Let $x_i > c$ be one of the axis splits, which classifies some data points $\boldsymbol{x_p}$, $\boldsymbol{x_q}$, $\boldsymbol{x_r}$, $\boldsymbol{x_s}$ as below:
$x_{p,i} < x_{q,i} < c < x_{r,i} < x_{s,i}$ for feature i,
where the first two points are in the left tree and the last two in the right tree.
For the transformed data, $x_i - \mu_i > c$ or $x_i > c + \mu_i$ corresponds to the transformed axis which preserves the classification since it preserves the inequality.
Thus, the transformed axis split corresponds to the same distribution and hence, the same Gini Index uncertainty measure.

Standardization: (No) The transformation does not affect the learning algorithm. From above, we can see that dividing by σ_i preserves the inequality. Thus, the distribution for classification remains the same since the particular axis split will give the same Gini Index uncertainty measure which corresponds to the maximally reduced uncertainty.

(d) **Centering**: (Yes) Centering does affect the learning algorithm. For the intractable ERM linear classifier, we update the weights and threshold till we obtain a perfect linear classifier. For the transformed data, the final weight vector should be transformed by the same amount when the learning algorithm is run on the transformed data.
If the weight vector for $\hat{f_S} : \mathcal{X} \rightarrow \mathcal{Y}$ is $\hat{\boldsymbol{w}}$ and threshold is $\hat{t}$, then with $\phi : \mathcal{X} \rightarrow \mathcal{X}'$ as the transformation,
the weight vector for $\hat{f'_S} : \mathcal{X}' \rightarrow \mathcal{Y}$ must be $\hat{\boldsymbol{w}}' = \hat{\boldsymbol{w}} - \hat{\boldsymbol{\mu}}$ and threshold is $\hat{t}$
But, this is not guaranteed with the learning algorithm. We may get the

final weight vector $\boldsymbol{\hat{w}'} = \boldsymbol{\hat{w}} + \sum_{t=1}^{n} y_t \boldsymbol{\hat{\mu}} + \sum_{j=1}^{k} y_j \boldsymbol{x_j}$
where the last term is determined based on the sequence in which the unclassified points are considered and based on how many times they are considered.
Thus, if a linear classifier exists, even though the algorithm will find one that fits for the transformed data, it may not be the same as what we get when running the algorithm on the original data.
Thus, the classification may not be the same.

Standardization: (Yes) Standardization affects the learning algorithm. Similar to above, we can say that scaling the data further, i.e., standardization will not guarantee the same weight vector and threshold are obtained for the transformed as we got for the original data.
Thus, the classification won't be the same.

Problem 2

Note: Only the first 200,000 features have been used for this problem.
Fourth feature representation:
For this feature, I have ignored all the single letter words, and for the rest the feature is represented by the tf-idf as defined below:
$tf(w; d) \times (\ln(idf(w; D)) + 1)$
where, $tf(w; d)$ and $idf(w; D)$ are as defined in the homework assignment

Cross-validation error rates:
Averaged-Perceptron:
The cross-validation error rates for the various representations are as below:
1. Unigram Representation: Error Rate = 11.2845 %
2. Term frequency-inverse document frequency (tf-idf) weighting: Error Rate = 12.4535 %
3. Bigram Representation: Error Rate = 11.3489 %
4. Fourth feature Representation: Error Rate = 11.5385 %

Naive Bayes Classifier:
The cross-validation error rates for unigram representation is 18.1125 %

Chosen Procedure:
The **unigram representation with Averaged-Perceptron** was selected as the best classifier based on the above error rates.

Error Rates for the chosen classifier:
Training Error Rate: 10.3595 %

Test Error Rate: 11.1101392594 %

Source code and scripts:
1. *hw3_preprocess.py*: This code must be run first. It forms the datasets and saves them as .npy files to make the loading faster.
2. *hw3_avg_perceptron_1_unigram.py*: Main code for getting cross-validation error rate for Avergaged-Perceptron with Unigram representation.
3. *hw3_avg_perceptron_2_tfidf.py*: Main code for getting cross-validation error rate for Averaged-Perceptron with tf-idf representation.
4. *hw3_avg_perceptron_3_bigram.py*: Main code for getting cross-validation error rate for Averaged-Perceptron with bigram tf representation.
5. *hw3_avg_perceptron_4.py*: Main code for getting cross-validation error rate for the fourth representation.
6. *hw3_naive_bayes_unigram.py*: Main code for getting cross-validation errro rate for Bayes Classifier with unigram representation.
7. *hw3_final_classifier.py*: Final classifier hardcoded to Averaged Perceptron using unigram representation.
8. README: Problem 2 details are listed there as well just as above.

Packages used from sklearn:[3,4,5]
sklearn.feature_extraction.text.CountVectorizer
sklearn.feature_extraction.text.TfidfVectorizer
sklearn.feature_extraction.text.TfidfTransformer
sklearn.cross_validation.KFold
sklearn.naive_bayes.MultinomialNB

Additional packages:
scipy.sparse.csr_matrix[6]
random.shuffle

Problem 3

(a) The multivariate Gaussian distribution can be written as:

$$P_{\mu,\sigma^2} = \prod_{i=1}^{n} \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{\frac{-1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

where $\Sigma = \sigma^2 I$

$$\ln P_{\mu,\sigma^2} = \sum_{i=1}^{n} \frac{-1}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma|) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$$

$$\ln P_{\mu,\sigma^2} = -\frac{1}{2}\sum_{i=1}^{n} \ln(2\pi) + \ln(|\Sigma|) + (\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$$

By matrix derivation rules[1],

$$\frac{\partial \ln P_{\mu,\sigma^2}}{\partial \Sigma} = -\frac{1}{2}\sum_{i=1}^{n} 0 + |\Sigma|^{-T} + (-\Sigma^{-T}(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})\Sigma^{-T})$$

$$\frac{\partial \ln P_{\mu,\sigma^2}}{\partial \Sigma} = 0$$

$$-\frac{1}{2}\sum_{i=1}^{n}[|\Sigma|^{-T} - \Sigma^{-T}(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-T}] = 0$$

Since, Σ is a diagonal matrix: $\Sigma^{-T} = \Sigma^{-1}$

$$\sum_{i=1}^{n}[|\Sigma|^{-1} - \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}] = 0$$

$$\sum_{i=1}^{n}|\Sigma|^{-1} = \sum_{i=1}^{n}[\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}]$$

$$\sum_{i=1}^{n} I = \sum_{i=1}^{n}[(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T\Sigma^{-1}]$$

$$nI = \sum_{i=1}^{n}[(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T]\Sigma^{-1}$$

$$n\Sigma = \sum_{i=1}^{n}[(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T]$$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}[(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T]$$

$$\sigma^2 I = \frac{1}{n}\sum_{i=1}^{n}[(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^T]$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}[\sum_{j=1}^{d}(\boldsymbol{x_j}-\boldsymbol{\mu_j})^2]$$

(b)

References

[1] https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

[2] http://cs229.stanford.edu/section/gaussians.pdf

[3] http://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text

[4] http://scikit-learn.org/0.17/modules/generated/sklearn.cross_validation.KFold.html

[5] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[6] http://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html