INF554 - MACHINE LEARNING I
ÉCOLE POLYTECHNIQUE
Lab: Feature Selection and Dimensionality Reduction

Stratis Limnios, Jesse Read, Nikolaos Tziortziotis and Michalis Vazirgiannis

1 Evaluation of Feature Selection Methods

We will study the effects of *feature selection* techniques in learning tasks. More precisely, we will implement and apply the χ^2 and Information Gain measures to examine their performance in a classification task.

We have dataset $\mathcal{D}$, features $\{X_j\}_{j=1}^{D}$, where each X_j takes values in $\mathcal{X}_j = \{1, \ldots, V\}$, and label Y takes classes in $C = \{1, \ldots, K\}$. We want an importance score for each attribute X_j.

A very well known measure for feature selection is the χ^2 ***measure***:

$$\chi^2(X_j, Y) = \sum_{v \in \mathcal{X}_j} \sum_{c \in C} \frac{(O_{vc} - E_{vc})^2}{E_{vc}} \tag{1}$$

Recall that in the case of independence we would expect that $P(X, Y) = P(X)P(Y)$. Thus the Expected number of times that X_j takes value v *and* Y takes value c; and the Observed number of times of this combination is given as:

$$\begin{aligned}
O_{vc} &= N \cdot P(X_j = v, Y = c) \\
&= \sum_{i=1}^{N} \Big(\mathbb{I}(x_j^{(i)} = v) \cdot \mathbb{I}(y_i = c) \Big) \\
E_{vc} &= N \cdot P(X_j = v) \cdot P(Y = c) \\
&= N \cdot \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(x_j^{(i)} = v) \cdot \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i = c)
\end{aligned}$$

noting of course that P is the empirical probability. $\mathbb{I}(A)$ is an indicator function that returns 1 iff condition A holds.

In Information Theory, ***entropy*** is the average amount of information contained in each message received. In the case of feature selection, let Y be the class label. Then, the entropy of Y characterizes the purity of the class in a set S:

$$H(S) = -\sum_{c=1}^{C} \big(P(Y = c) \cdot \log_2 P(Y = c) \big),$$

For example, suppose that we have to deal with a dataset of 16 instances, and each of them belongs to one of the two possible classes: $\{Y, N\}$ (Yes, No). Let $S = \{Y, N, Y, N, Y, Y, Y, Y, N, N, Y, N, N, N, Y, Y\}$

be the set of the class labels. Then, the entropy will be equal to

$$H(S) = -\Big[\Big(\frac{9}{16}\Big)\log_2\Big(\frac{9}{16}\Big) + \Big(\frac{7}{16}\Big)\log_2\Big(\frac{7}{16}\Big)\Big] \approx 0.9836.$$

In this case, entropy is very close to one, which is expected since each class contains almost half part of the dataset.

The measure of ***information gain*** is used to compute the importance of a feature with respect to the class labels. More precisely, the information gain (IG) is defined as the expected reduction in entropy caused by partitioning the instances of the data according to a given feature. Let Y be the target variable and j-th feature be of interest. Then, $\text{IG}(S, j)$ is given by the following formula:

$$\text{IG}(S, j) = H(S) - \sum_{u \in \mathcal{X}_j} \frac{|S_u|}{|S|} H(S_u), \tag{2}$$

where $S_u, \forall u \in \mathcal{X}_j$ are the sets after the splitting of the dataset according to the possible values of attribute X_j.

1.1 The data

The dataset (`data.csv`) describes a set of 102 molecules of which 39 are judged by human experts to be musks (class 1) and the remaining 63 molecules are judged to be non-musks (class 2). The goal is to learn to predict whether new molecules will be musks or non-musks. However, the 166 features that describe these molecules depend upon the exact shape, or conformation, of the molecule. Because bonds can rotate, a single molecule can adopt many different shapes. To generate this data set, all the conformations of the molecules were generated to produce 6598 conformations. Then, a feature vector was extracted that describes each conformation. That way, the final dataset has 6598 instances and 166 features.

1.2 The pipeline

The file `main.py` loads the data into `X` and extracts the class labels into `Y`. We enable or disable feature selection setting `True` or `False` the `featureSelection` variable. The idea is to examine the performance of the classifier for both cases – with and without feature selection. For the first case, we also need to set the number of features that will be selected (`num_feat` variable). For feature selection, we implement and then apply the χ^2 and Information Gain measures, in the `chiSQ(X,Y)` and `infogain(X,Y)` functions; in files `chiSQ.py` and `infogain.py`. Then, the features are ranked based on their importance (as returned by the feature selection techniques).

The function `logisticRegression(X,Y)` trains the Logistic Regression classifier using the data that are stored on matrix X and the labels Y. The trained classifier can be used to predict the class of new data instances (i.e., molecules in our case). The data that will be used for prediction are stored in the `test.csv` file.

The last step involves the prediction of the class labels for the test data (note that, the prediction is based on the parameters `w` of the classifier). Finally, the `py` variable contains the predicted labels.

Note that, the prediction produced by the classifier is not binary (i.e., $0-1$ values), but in the range $[0, 1]$. We use 100 different threshold values (from 0 to 1 with step 0.01) to assign class labels (if the prediction is less that the threshold, then we assign the first label). Then, for each case we compute the precision and recall and we plot the corresponding curve (see next paragraph for more details on the evaluation).

How to evaluate the results? In order to evaluate the feature selection methods, we apply the *precision* and *recall* metrics. The precision and recall are given by the following formulas:

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \qquad \text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}.$$

Additionally, we compute the precision-recall curve. Note that, the higher the area under this curve, the better the model. In order to compute the area under curve, we apply the trapezoidal rule.

1.3 The tasks

After studying each part of the code and understand the basic functionality, we will examine the following points:

1. Run the classifier without feature selection. Observe the precision-recall curve (and the area under curve), as well as the running time.
2. Implement the χ^2 measure in `ChiSQ.py`.
3. Apply feature selection using the χ^2 measure. For different number of features (e.g., $20, 40, 60, 80, 100, 150$) examine the precision-recall curve (and the area under curve), as well as the running time. Does feature selection improve the accuracy? What about the execution time?
4. Implement the Information Gain measure in `infogain.py`.
5. Repeat step 3 using the Information Gain measure.
6. How correlated is the ranking of the features created by χ^2 and Information Gain? Hint: use the *Kendall tau*[1] rank correlation coefficient for determining the similarity of the orderings of the features, as produced by the different feature selection methods. You can use the function `scipy.stats.kendalltau()`.
7. Create a figure (plot) of the area under curve versus the number of selected features. Repeat the same for the running time. Observe the results.

2 Dimensionality Reduction with PCA

Principal components analysis (PCA) is a very popular technique for dimensionality reduction. Given a set of data on n dimensions, PCA aims to find a linear subspace of dimension d lower than n such that the data points lie mainly on this linear subspace. Such a reduced subspace attempts to maintain most of the variance of the data. The linear subspace can be specified by d orthogonal vectors that form a new coordinate system, called the *principal components*. The principal components are linear transformations of the original data points, so there can be no more than n of them. However, the hope is that only $d < n$ principal components are needed to approximate the space spanned by the n original axes. Thus, for a given set of data vectors $x_i, i \in 1 \dots t$, the d principal axes are those orthonormal axes onto which the variance retained under projection is maximal. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components.

[1]`http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient`

Recall that the variance of a random variable is given by the following formula: $V(X) = \sigma^2 = E[(X - \mu)^2]$. PCA can be computed using the eigenvalue decomposition of the covariance matrix as follows:

1. Suppose that the data is organized into an $m \times n$ matrix $\mathbf{A}$, initially subtract mean values from columns: $\mathbf{C} = \mathbf{A} - \mathbf{M}$
2. Calculate the covariance matrix $\mathbf{W} = \mathbf{C^T C}$
3. Find eigenvalues and eigenvectors of the covariance matrix $\mathbf{W}$
4. *Principal Components*: the k eigenvectors $\mathbf{U}_{[1\ldots k]}$ that correspond to the k largest eigenvalues
5. Project the data to the new space: $\mathbf{CU}_{[1\ldots k]}$

We compute the principal components using the SVD decomposition. Recall that the square root of the eigenvalues of the covariance matrix $\mathbf{C^T C}$ corresponds to the singular values of $\mathbf{C}$. That way, in step 2, we can compute the SVD decomposition of $\mathbf{C}$, and the principal components will correspond to the singular vectors that correspond to the k largest singular values.

As we have already discussed, PCA transforms the data into a lower dimensional space preserving the variance. The fraction of variance that is preserved in the transformed data can be captured by the following formula:

$$var = \frac{\sum_{i=0}^{k} \lambda_i}{\sum_{j=0}^{n} \lambda_j},$$

where n is the number of dimensions in the original dataset, k is the number of dimensions in the new representation space, and λ are the eigenvalues of the covariance matrix sorted in descending order.

2.1 The Data

Here, we will analyze a dataset about cities in the USA. This dataset is composed of 329 observations, describing the quality of life with the usage of 9 parameters (housing, climate, education, etc.). The number of dimensions of the dataset is 9.

2.2 The Pipeline

In `main_pca.py`, PCA will be applied to a dataset in order to project the data into a k-dimensional space defined by the first k principal components. It performs PCA for all possible top-k projections and visualizes the projection of the data to the first 2 components.

2.3 The Tasks

- Implement PCA in the file `pcaImp.py` under the directory `Code/Implementation/PCA/`
- Run `main_pca.py`. What do you observe? What is the city with the greatest "variation"? Which of the original features affects more the new feature space?

3 Dimensionality Reduction with Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a matrix decomposition method that leads to a low-dimensional representation of a high-dimensional matrix. Let $\mathbf{A}$ be a $m \times n$ matrix. Then the Singular Value Decomposition of matrix $\mathbf{A}$ is defined as

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$$

where

- $\mathbf{U}$: $m \times m$ matrix has as columns the eigenvectors of $\mathbf{A}\mathbf{A}^{\mathbf{T}}$
- Σ : $m \times n$ is a diagonal matrix with the singular values of $\mathbf{A}$ in the diagonal (= square roots of $\mathbf{A}\mathbf{A}^{\mathbf{T}}$ eigenvalues)
- $\mathbf{V}$: $n \times n$ matrix has as columns the eigenvectors of $\mathbf{A}^{\mathbf{T}}\mathbf{A}$.

SVD allows an exact representation of any matrix, and also makes it easy to eliminate the less important parts of that representation in order to produce an approximate representation with any desired number of dimensions – leading to the concept of *low rank approximation* of a matrix. Let $\mathbf{A}$ be a $m \times n$ matrix, where $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$. Then, a r rank approximation of $\mathbf{A}$ is given by

$$\mathbf{Y} = \mathbf{U}_{m\times r}\text{diag}(\sigma_1, \ldots, \sigma_r)\mathbf{V}^{\mathbf{T}}{}_{r\times n}.$$

How to perform dimensionality reduction with SVD?

The goal of dimensionality reduction is to find an approximation $\mathbf{Y}$ of $\mathbf{A}$ using r dimensions instead of the original n, $r < n$. This can be done using the properties of SVD: $\mathbf{Y}_{m\times n} = \mathbf{U}_{m\times r}\mathbf{\Sigma}_{r\times r}\mathbf{V}^{\mathbf{T}}{}_{r\times n}$. In practice, however, the purpose is not to actually reconstruct the original matrix, but to use the reduced dimensionality representation $\mathbf{U}_{m\times r}\mathbf{\Sigma}_{r\times r}\mathbf{V}^{\mathbf{T}}{}_{r\times n}$ in order to analyze the data.

The quality of low rank approximation $\mathbf{Y}$ of a matrix $\mathbf{A}$ can be evaluated using the *Frobenious norm* $\| \mathbf{A} - \mathbf{Y} \|_{\mathbf{F}}$, where

$$\| \mathbf{X} \|_{\mathbf{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |x_{ij}|^2} = \sqrt{\sum_{i=1}^{\min(m,n)} \sigma_i^2}.$$

The best low rank approximation is the one that minimizes the Frobenius norm $\| \mathbf{A} - \mathbf{Y} \|_{\mathbf{F}}$.

How many singular values should we retain?

A useful rule of thumb is to retain enough singular values to make up 90% of the energy in $\mathbf{\Sigma}$. That is, the sum of the squares of the retained singular values should be at least 90% of the sum of the squares of all singular values.

3.1 The Tasks

Fill in the Python script `my_svd.py` under the directory `Code/Bonus/SVD`, in order to implement and apply the SVD decomposition on a matrix $\mathbf{X}$ that corresponds to an image. Then, you will need to reconstruct the original matrix (image) $\mathbf{X}$ using the top (largest) $k = \{10, 20, 50, 100, 200\}$ singular values. Compare (visually) these approximations to the original image. What is the error of each approximation? What do you observe? Hint: examine the distribution of the singular values of the original matrix $\mathbf{X}$.

4 Non-negative Matrix Factorization (NMF)

Given a non-negative $n \times m$ matrix $\mathbf{V}$, find non-negative matrix factors $\mathbf{W}$ and $\mathbf{H}$ such that:

$$\mathbf{V} \approx \mathbf{WH}$$

with $\mathbf{W} \in \mathbb{R}^{n\times r}$ and $\mathbf{H} \in \mathbb{R}^{r\times m}$.
To find this approximation we must define first a cost function that quantifies the quality of the approximation; the reconstruction error:

$$\|\mathbf{V} - \mathbf{WH}\|^2 = \sum_{i,j}(\mathbf{V}_{i,j} - \mathbf{WH}_{i,j})^2$$

We define in this configuration the following problem: *Minimize* $\|\mathbf{V} - \mathbf{WH}\|^2$ with respect to $\mathbf{W}$ and $\mathbf{H}$, subject to the constraints $\mathbf{W}, \mathbf{H} \geq 0$. In order to approximate those solutions, we will implement the ”multiplicative update rules” which is a good compromise between speed and ease of implementation. Hence the Euclidian distance $\|\mathbf{V} - \mathbf{WH}\|$ is nonincreasing under the update rules

$$\mathbf{H}^{(k+1)}_{a\mu} \leftarrow \mathbf{H}^{(k)}_{a\mu} \frac{(\mathbf{W}^{(k)T}\mathbf{V})_{a\mu}}{(\mathbf{W}^{(k)T}\mathbf{W}^{(k)}\mathbf{H}^{(k)})_{a\mu}}$$

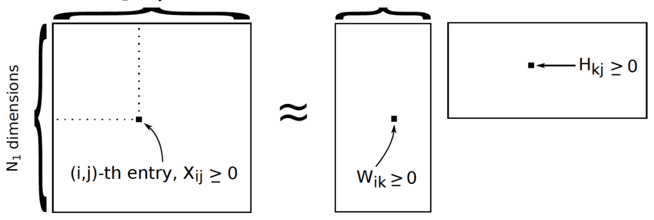
$$\mathbf{W}^{(k+1)}_{ia} \leftarrow \mathbf{W}^{(k)}_{ia} \frac{(\mathbf{V}\mathbf{H}^{(k+1)T})_{ia}}{(\mathbf{W}^{(k)}\mathbf{H}^{(k+1)}\mathbf{H}^{(k+1)T})_{ia}}$$

With $\forall\mu \in \{1,\dots,m\}$, $\forall\mu \in \{1,\dots,m\}$ and $\forall a \in \{1,\dots,r\}$. The idea being to update $\mathbf{H}$ first, row after row (a fixed). And then use the update to compute $\mathbf{W}$ column by column.

Regarding the initialization of $\mathbf{W}^{(0)}$ and $\mathbf{H}^{(0)}$ we need to ensure that those matrices are non negative. A first approach is to define random non-negative matrices, but a more efficient and sofisticated method is using the SVD decompostition of the matrice $\mathbf{V}_{n\times m} = \mathbf{W}_{n\times r}\mathbf{\Sigma}_{r\times r}\mathbf{H^T}_{r\times m}$. Setting $\mathbf{W}^{(0)} = |\mathbf{W}|$ and $\mathbf{H}^{(0)} = |\mathbf{H^T}|$. Moreover, similarly to the SVD as rule of thumb for the rank parameter is to retain enough singular values to make up 90% of the energy in $\mathbf{\Sigma}$.

4.1 The Task

We will do topic modeling among New York Times articles. In order to do so, our matrix $\mathbf{V}$ will consist of a term/document matrix. Each row corresponds to a term and each feature is its occurence in the given document.



Hence the $\mathbf{W}_{ij}$ value corresponds to the impact of the word i for the topic j. In order to model every topic, we will use the top t words that have the highest weight for every column of $\mathbf{W}$.

- First, implement the NMF of the term/document matrix inside `nmf.py` under the `Code/NMF` directory,
- Run `main.py` to model the key words for every topic computed among the New York Times articles.

5 Bonus

Apply the Principal components analysis (PCA), the SVD decomposition, and the Non-negative Matrix Factorization (NMF) on image `Code/Data/gatlin.csv` (the same image used in the SVD task). Then, reconstruct the original matrix (image) using the top $k = \{10, 20, 50, 100, 200\}$ singular values based on PCA, SVD and NMF. Finally, compare these three dimensionality reduction methodologies based on the next three error metrics: i) Frobenius norm, ii) Variance (useful note: usualy we compare the ratio of the sum of first k eigenvectors to all the eigenvectors but since this is going to be used for a comparison of non-pca appraches we are going to compare all the eigenvectors between the compressed and the original one), iii) Stress.

References

[1] Guyon, Isabelle, and André Elisseeff. ”An introduction to variable and feature selection”. The Journal of Machine Learning Research 3 (2003): 1157-1182.

[2] Dash, Manoranjan, and Huan Liu. ”Feature selection for classification”. Intelligent data analysis 1.3 (1997): 131-156.

[3] Lee, Daniel D and Seung, H Sebastian. ”Algorithms for non-negative matrix factorization”. Advances in neural information processing systems, pages 556-562,2001.

[4] Wikipedia’s article for feature selection (December 2014). `http://en.wikipedia.org/wiki/Feature_selection`.