Universidad Nacional de Rosario

Deep Learning

Trabajo Práctico I

Integrantes:
Crosetti Mariano
Tenaglia Felipe
Villagra Martín

Crosetti Mariano
Tenaglia Felipe
Villagra Martín

Introducción

En el presente trabajo se introducen los conceptos de pre-procesamiento de datos mediante métodos de normalización, necesarios para el posterior análisis. Particularmente tratamos los casos de procesamiento de imágenes y texto.

En síntesis en nuestros análisis cada entrada será un vector de $\mathbb{R}^d$. Los procesos de normalización se encargarán de que estos vectores posean características deseadas (como media cero y varianza unitaria) para los algoritmos de aprendizaje subsiguientes.

Ejercicio 1

Las funciones que leen y homogeneizan los datos se encuentran en el archivo `ejercicio1-2-4.py`. Aquí una breve explicación de las más relevantes:

- `get_images_all`: carga los archivos .jpg que se encuentran en la carpeta `./Caltech101/`
- `get_images_borges`: carga los archivos .png que se encuentran en la carpeta `./borges-words/`
- `to_float`: transforma las matrices de enteros que están en el rango $[0, 255]$ a matrices con números de punto flotante en el rango $[0, 1]$.
- `mean_by_channel(face)`: retorna le media de cada canal.
  Análogamente `var_by_channel(face)` con la varianza. Son utilizadas por las funciones `zero_mean(face)` y `one_variance(face)` para que, respectivamente, la media sea cero y la varianza unitaria, en cada canal.
- Un problema que debimos solucionar fue el hecho de que las imágenes, en su gran mayoría, son rectangulares y de diferentes tamaños. Para homogeneizarlas, decidimos redimensionarlas al menor cuadrado que la contenga. De esto se encarga `make_squared(face)`. El color elegido para rellenar los nuevos espacios fue el color medio (canal por canal) de toda la imagen. Luego `resize_image(face,tam)` la escala al tamaño especificado por la variable `final_tam`.
- El segundo problema a resolver lo encontramos al probar el programa con todo el dataset (durante el desarrollo utilizamos un subconjunto del él). Allí descubrimos que existían imágenes en escala de grises, y nuestras funciones traba-

jaban con imágenes RGB. Para solucionarlo, definimos la función `add_depth(face)`, que convierte la imagen a RGB, en caso de ser necesario.

La función `normalize(face)` es la encargada de realizar todo el proceso de normalización en la imagen dada como argumento.

Ejercicio 2

Para este ejercicio se ha reutilizado la función de normalización del ejercicio 1. Además se creó la función `get_slice` para extraer subsecciones de las imágenes. Para observar los efectos de la normalización, la hemos aplicado a un ejemplo y creamos un GIF que nos permitió observar la diferencia. Para mostrar las imágenes, la función `show` hace un corrimiento y escalamiento adecuado para asegurarse que las entradas de la matriz se encuentren en el intervalo $[0, 1]$.

Para observar las diferencias entre la imagen y luego de modificarla, decidimos utilizarla en su tamaño original y no escalarla ni recortarla.

Luego de mover la media a cero, podemos notar que la imagen se aclara (aumenta el brillo) y pierde su "color dominante", es decir, si todo tiende a una tonalidad azul, luego de este proceso se de-satura y deja de resaltar este color. Al forzar que la varianza sea uno, el cambio es más sutil, y vemos que aumenta un poco el contraste.

Estos procesos son importantes para tratar de limpiar las particularidades de la imagen y obtener una más uniforme, evitando así que los métodos de aprendizaje sobreajusten y "aprendan" a reconocer elementos que son específicos de la entrada y no características de la clase.

Adjunto a este PDF se puede ver la animación que muestra los cambios (Archivo `Image_ZeroMeanUnitVariance.gif`).

Ejercicio 3

Para gran mayoría de los algoritmos de procesamiento de datos vectoriales es fundamental la normalización de los mismos con procesos de Whitening. Nuestra información son *muestras* de vectores aleatorios d dimensionales (que suponemos repre-

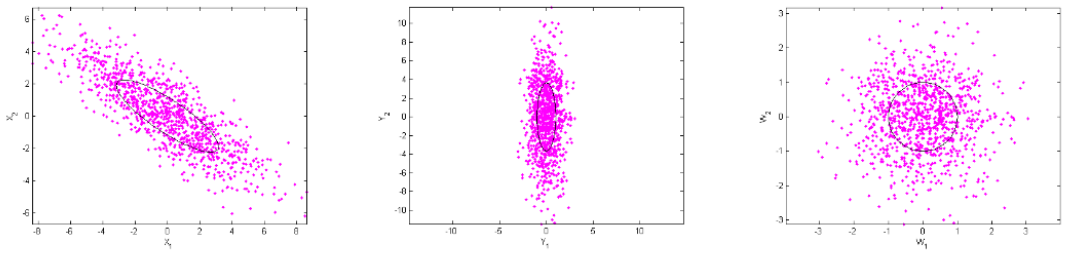
sentativa del espacio muestral). Para el posterior análisis necesitamos que el vector se encuentre *centrado*, o sea que la variable aleatoria posea media cero, podemos lograrlo substrayendo la media (generalmente una estimación de la misma con las muestras que disponemos).

Dichos datos generalmente poseen una dimensionalidad representativa alta, pero su dimensionalidad intrínseca puede ser significativamente menor. Esto es debido a que las componentes elegidas se encuentran, muy a menudo, correlacionadas. El objetivo es hacer que la matriz de covarianza sea diagonal. Esto se logra mediante un adecuado cambio de base, proyectando a los autovectores de la matriz de covarianza. Probablemente algunos autovectores corresponderán a autovalores relativamente pequeños, por lo que podemos descartar estas componentes si queremos reducir la dimensionalidad sin pérdida significativa de la información. Si queremos quedarnos con k componentes, se elegirán las que sean respectivas a autovalores más grandes (recordar que los autovalores son todos positivos, siendo la matriz de covarianza simétrica). La importancia de alguna de las componentes puede observarse de manera más visual en las primeras dos imágenes de la Figura 1. Esto es lo que se conoce cómo Principal Components Analysis (PCA). Lo que estamos haciendo es expresar los datos en términos de los patrones de correlación que existen entre ellos.

Además estos nuevos vectores serán escalados componente a componente dividiendo por los autovectores respectivos. Con lo cuál obtenemos una covarianza no sólo diagonal sino unitaria. Gráficamente esta transformación puede verse en la última imagen de la Figura 1. Este proceso entero es el que se denomina como Whitening. Debe su nombre a que los vectores que se obtienen son *white noise vectors* (vectores aleatorios dónde sus componentes no están correlacionadas y cada una posee varianza 1).

Esta transformación hace que ya no tenga sentido interpretar los datos del modo que veníamos haciéndolo (pierden su semántica original), pero la de-correlación de los mismos es fundamental para su análisis. Podemos volver a la representación anterior (si se han eliminado componentes, con pérdida de cierta información) con otras transformaciones inversas (ver colouring). A menudo las componentes descartadas son ruido e información intrascendente que existe en los datos.

En la Figura 2 podemos observar el resultado de la aplicación de ZCA Whitening en un conjunto de imágenes (y posterior reconstrucción). Intuitivamente podemos observar que sólo información relevante ha perdurado (lo cuál es positivo para el



(a) (b) (c)

Figura 1: (a) muestra la nube de datos original. (b) muestra luego de expresarlo en sus componentes principales, de-correlacionándolos. (c) luego de haber aplicado el escalamiento por componentes respectivo al whitening.

entrenamiento posterior).

Ejercicio 4

Para este ejercicio también se ha reutilizado la función de normalización descripta en el ejercicio 1. Primero creamos otra función que realiza la carga de imágenes, ya que la estructura de directorios difiere de la del otro ejercicio. Además se creó la función `drop_transparency(face)` para eliminar el canal alfa de los png (No tenía utilidad pues todas sus entradas eran 255).

Los resultados que podemos observar en este ejercicio son similares a los descriptos en el ejercicio 2. Podemos notar en las palabras como, al tener la media 0, se tornan más grises, lo que compensa los problemas de iluminación (vimos que, en algunos casos, las palabras estaban azuladas o amarillentas, resultado de la calibración de blanco del dispositivo que capturó las imágenes). Con varianza 1, el cambio es, nuevamente, más sutil, y realza levemente el color negro del trazo.

Adjunto a este PDF se puede ver la animación que muestra los cambios (Archivo `Aleph_ZeroMeanUnitVariance.gif`).

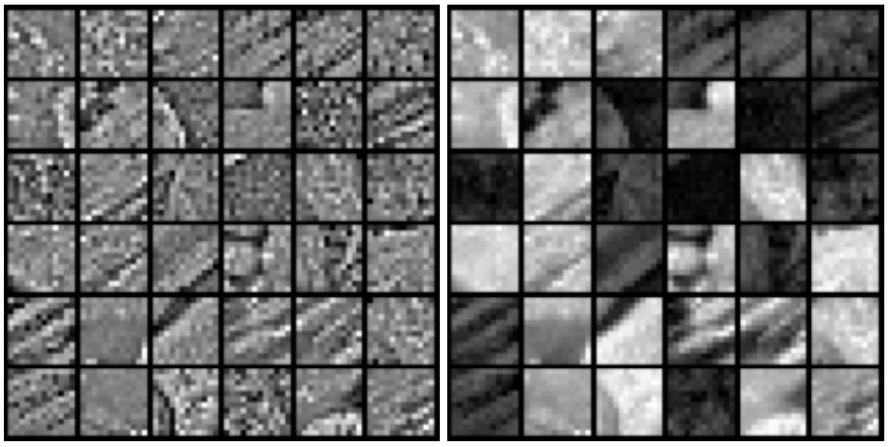


Figura 2: A la derecha las imágenes originales; a la izquierda luego de ZCA whitened.

Ejercicio 5

En primera instancia tenemos que procesar el texto, separando las oraciones y las palabras. Se buscó hacer código lo más simple posible, dejamos la elegancia para los sastres y zapateros. El resultado es una lista de oraciones, donde cada una de ellas es una lista de palabras (implementado en `utils.py:parse_text`).

Una vez cargados los datos, se calcula la frecuencia de cada palabra (implementado en `utils.py:calc_freq`). Esto es usado para encontrar las palabras más/menos frecuentes.

Ahora, dado un conjunto de n palabras interesantes (más frecuentes, menos frecuentes), el objetivo es crear para cada oración un vector de n números que contenga información para cada una de estas palabras. Si la palabra no está en la oración, ponemos un 0, en caso contrario es tentador colocar la frecuencia de esa palabra en la oración. Para realizar esto usamos la siguiente función auxiliar:

```
def make_vector(sentence, interesting_words, weightfunc):
    l = np.array([0]*len(interesting_words))
    for word, freq_in_sentence in calc_freq(sentence):
        if word in interesting_words:
            l[interesting_words[word]] +=
                weightfunc(word, freq_in_sentence)
    return l
```

Notar que en lugar de directamente colocar la frecuencia de la palabra, utilizamos una función para generalizar.

Teniendo estas funciones la construcción de los vectores se vuelve mucho más simple, la implementación se provee en los archivos `ejercicio5-6.a.py` y `ejercicio5-6.b.py`.

Ejercicio 6

Para este ejercicio nos valimos de la biblioteca `sklearn` para implementar `PCADimReduction(X,n)`, que toma una matriz con las muestras como filas y reduce la dimensionalidad a n dimensiones. También programamos `kNeighbours(v,X,k)` que encuentra los k vecinos más cercanos de v en el espacio de objetos X.
El código está distribuido en la última sección de código de los archivos `ejercicio5-6.a.py` y `ejercicio5-6.b.py`, y el archivo `ejercicio6.caperucita.py`.

PCA en 5.a

Si corremos el programa `ejercicio5-6.a.py` y ampliamos la Figura 3 se observan un ligero alineamiento de los puntos. No obstante el agrupamiento parece no muy significativo. Si usamos la función de k-vecinos más próximos hallamos oraciones largas y con una relación semántica dudosa. Al ver las palabras más frecuentes, podemos observar que el predominio de palabras intrascendentes (como artículos y preposiciones) repercuten negativamente, relacionando oraciones que semánticamente tratan de temas muy variados. Además dichas palabras aparecen frecuentemente en las oraciones, generando que las componentes respectivas sean grandes y cobrando importancia por sobre otras más significativas.

PCA en 5.b

Los resultados obtenidos del análisis de los vectores obtenidos en el ejercicio 5.b no son para nada útiles. Como podemos observar, las 1000 palabras menos frecuentes son usadas exactamente una vez en el texto. Es por ello que nuestros vectores tendrán a lo sumo una componente no nula. La matriz de covarianza será por ende, la identidad. Entonces la componente respectiva a cada autovector será igual de significativa (los autovalores serán todos 1). Es por ello que al proyectar sobre dos cualesquiera de ellos (lo que esencialmente hace PCA) y graficar el resultado vemos tres puntos: dos puntos sobre un sólo eje que contienen exactamente un elemento

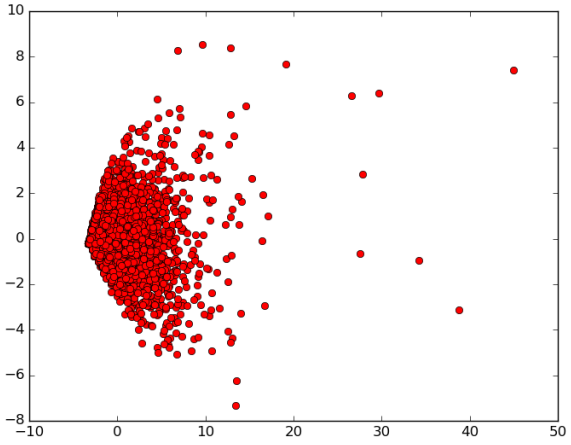


Figura 3: Si bien se observa un muy ligero alineamiento (más aún si se ampliase la imagen) las palabras intrascendentes genere una correlaciones indeseadas como se ha explicado

cada uno (la oración donde aparece la palabra respectiva) y un cúmulo de puntos en el (0,0), conteniendo el resto de las oraciones (Ver Figura 4).

Experimentos posteriores

Los resultados anteriores no fueron satisfactorios. En parte adjudicamos esto a la poca representatividad de la información escogida para el análisis que queríamos hacer. Nosotros esperaríamos que los textos de las temáticas que estén correlacionadas se encuentren agrupados y diferenciados del resto. Pero en el apartado 5.a consideramos palabras intrascendentes que no hacen a la semántica de las oraciones (como artículos y preposiciones). Y en el apartado 5.b palabras que aparecen en sólo una oración (por lo que la correlación entre oraciones es nula en este sentido).
La elección de los vectores que representan nuestra información es inadecuada, y partiendo de dicha base inadecuada no se puede llegar a resultados satisfactorios. Más aún en un texto complejo como es el compilado de Nieztche.

Como consideramos que el criterio con el cuál armamos nuestros vectores en el 5.a fue comparativamente mejor, usamos este mismo repitiendo el experimento con dos textos uno en español (*caperucita.txt*) y otro en idioma inglés (*machinelearning.ingles.txt*). Al tratarse de dos textos de idiomas (y además temáticas) dife-

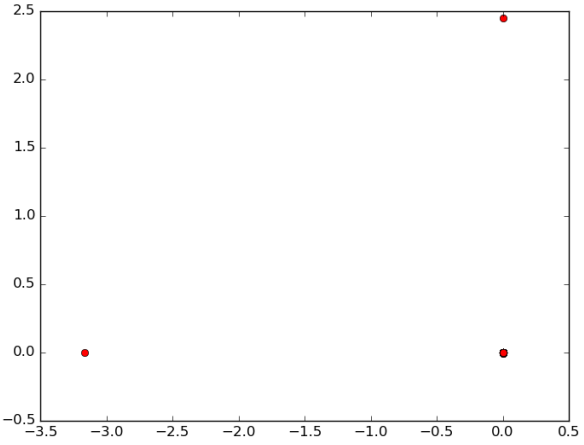


Figura 4: Sólo vemos tres puntos. La pérdida de información es muy significativa.

rentes, la agrupación de los vectores correspondientes a uno y otro es clara, como podemos ver en la Figura 5.
Hemos analizado también el caso de dos textos en español pero de temáticas diferentes: *caperucita.txt* (un cuento infantil) y *machinelearning.txt* (descripción de machine learning sacada de Wikipedia). Pese a la existencia de palabras intrascendentes que relacionan ambos textos (artículos, preposiciones) la separación de ambos grupos sigue siendo clara como podemos ver en la Figura 6.

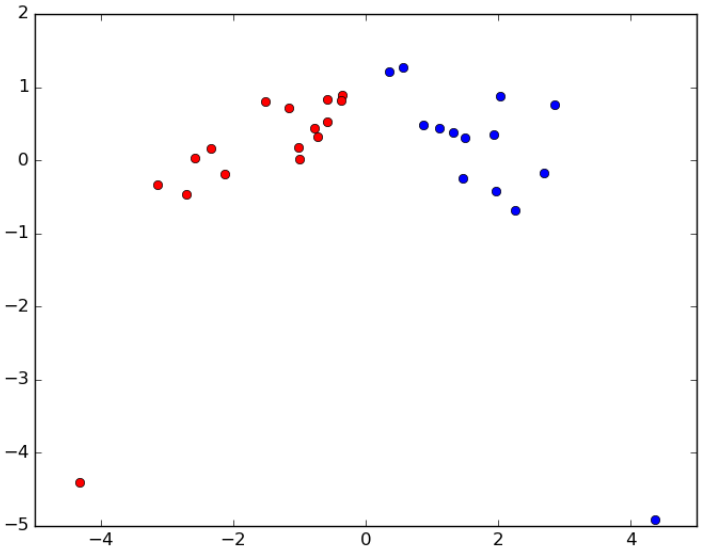


Figura 5: Los puntos rojos representan el texto de Caperucita Roja y los azules los de Machine Learning, en español e inglés respectivamente

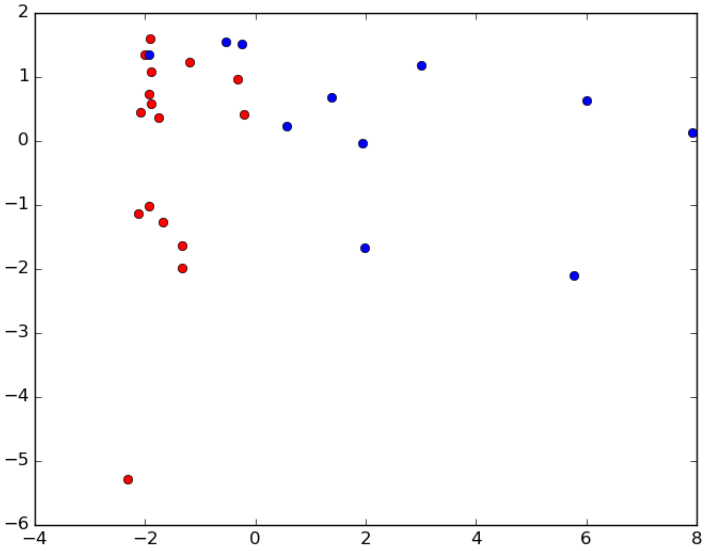


Figura 6: Los puntos rojos representan el texto de Caperucita Roja y los azules los de Machine Learning, ambos en español

Referencias

- *"A tutorial on Principal Components Analysis"* Lindsay I Smith, `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`
- *"Whitening and coloring transformations for multivariate gaussian data"* Maliha Hossain, `https://www.projectrhea.org/rhea/images/1/15/Slecture_ECE662_Whitening_and_Coloring_Transforms_S14_MH.pdf`