Faculté des Sciences et Technique de Mohammedia

Mémoire de projet cadre

La réalisation d'un classificateur NLP hybride entre la machine learning et le deep learning .

Author:
Mr ELKAISSI SOUHAIL

Supervisor:
Mme K.DOUZI

Ce Rapport correspond au stage de la deuxième année du cycle d'ingénieur Ingénierie Logicielle et Intégration des Systèmes Informatiques

in the

La Faculté des Sciences et Technique de Mohammedia

22 juin 2018

REMERCIEMENTS

Je tiens à remercier les personnes qui m'ont aidé lors de mon stage au sein de l'équipe du SANADTECH et notamment les trois personnes qui m'ont accompagné, structuré et aidé pendent ses six mois:

Mon encadrant en entreprise, M. ABDELHAKIM RHANIZAR, Fondateur et directeur technique de l'entreprise qui m'a accompagné tout au long de ce stage, en me faisant découvrir toutes les technologies que j'ai pu maitriser au cours des semaines .
Mon encadrant à la faculté des sciences et techniques de Mohammedia, Mme KHADIJA DOUZI, qui a travers ses nombreuses conseils, et grâce aussi à sa confiance j'ai pu m'accomplir totalement dans mes missions.
M. CHATRIA Zakaria, le second développeur avec lequel j'étais en contact direct, qui m'a aidé dans mes tâches et qui m'a fait sortir de nombreuses situations délicates en m'aidant à résoudre les problèmes rencontrés.

Et je tiens à laisser une pensée pour les autres membres de l'équipe que je n'ai pas eu l'occasion de côtoyer souvent, mais que j'ai trouvé sympathique.

Résumé

Dans le cadre de mon stage à la société SanadTech, j'ai eu la chance de participer au projet Nacharat. Nacharat, en fait c'est une application mobile qui permet à ses utilisateurs de consulter les nouveautés sous format vidéo.
Dans mon entreprise d'acceuil SanadTech, j'ai travaillé au sein d'une équipe dirigée par Mr RHANIZAR, le développeur front-end Mr CHATRIA, et moi-même comme Data Scientist.
La méthode de gestion de projets appliquée pour le processus est la méthode agile Scrum. Cette méthode privilégie les objetctifs à court terme et donne lieu à des interactions régulières avec les membres de l'équipe afin de discuter de l'avancement sur le projet.
Puisque l'application reçois les nouveautés depuis plusieurs plateformes, elle a besoin d'un classificateur intelligent qui va lui permettre de catégoriser son contenu d'une manière logique et tends vers la réalité, d'où le besoin d'utiliser le machine learning ou le deep learning.
Au cours de mon stage et après une formation aux différents algorithmes, et théorie dans la machine learning et deep learning, notamment les réseaux de neurones de convolution, et le support vecteur machine, ou encore le word to vector, j'ai en une première étape utilisé et rénové la solution primitive, qui était le CNN, un algorithme du type deep learning. Ensuite j'ai mené une étude sur d'autres méthodes et approches comme le SVM qui s'inclut dans la catégorie du machine learning, et enfin j'ai intégré le réseau neurenal word2vect dans le premier algorithme, afin d'augmenter la précision.

Mots-Clés : application mobile, machine learning, deep learning, SVM, CNN, WORD2VEC, Bayes, méthodologie Agile/Scrum.

Abstract

As part of my internship at SANADTECH, I had the chance to participate in the NACHARAT project. NACHARAT is a mobile application that allows these users to see what's new in video form easily. My host company is the company SANADTECH, led by Mr RHANIZAR. So I joined the development team composed of Mr CHATRIA and myself.
The project management method applied for the process is the agile method Scrum. This method favors short-term objectives and gives rise to interactions members of the team to discuss progress on the project.

After training in differents algorithms, and theory in machine learning and deep learning, I was able to participate active in the development of the platform.

Keywords: mobile application, machine learning, Agile , Scrum methodology.

Table des matières

Table des figures

Abbréviations

LSTM :	short term momory
cnn :	convolutional neural network
rnn :	recurrent neural network
svm :	Support vector machine
word2vect :	word to vector
nlp :	neuro-languistic processing

Introduction

Depuis le début de l'informatique, l'homme cherche à communiquer avec les machines. Si les nombreux langages de programmation permettent une forme d'échange entre l'homme et la machine, on aimerait que cette communication se fasse de façon plus naturelle. Pour que cela soit possible, il faut d'abord que la machine "comprenne" ce que l'utilisateur lui dit ensuite qu'elle soit capable de répondre d'une manière compréhensible par l'homme. La discipline derrière ce processus s'appelle le Natural Langage Processing (NLP) ou traitement Automatique du Langage Naturel (TALN) en français. Elle étudie la compréhension, la manipulation et la génération du langage naturel par les machines. Par langage naturel, on entend le langage utilisé par les humains dans leur communication de tous les jours par opposition aux langages artificiels comme les langages de programmation ou les notations mathématiques.
C'est dans ce contexte que s'inscrit le projet cadre en sein de l'entreprise SANADTECH, qui envisage d'établir un classificateur de texte à étiquettes multiples pour analyser les titres de son application mobile NACHARAT.
NACHARAT est une application mobile qui permet à ses utilisateurs d'être au courant des nouveautés en temps réel, à travers un ensemble de panoplies de news (hespress, hibapress, ChoufTV, ya bila...etc.).
Éventuellement, ce document décrit les étapes de la mise en œuvre de ce projet structuré en six chapitres principales, à savoir:

- La première Chapitre présente la partie management de projet en passant par une présentation rapide des acteurs du projet, méthode agile utilisée, et la planification du projet.
- Le deuxième chapitre contient la conception globale du projet, en la présente par un ensemble de diagrammes, et descriptions.
- Le troisième chapitre présente les données traitées, et la façon dont ils ont été manipulées.

– Le quatrième,cinquième et sixième chapitre exposent de façon détaillée les différentes étapes de la mise en place du classificateur, de l'analyse globale du projet, vers les différents algorithmes utilisés pour le classificateur.

Chapitre 1

Contexte générale du Projet

1.1 Présentation de l'organisme d'accueil

1.1.1 SanadTech

SanadTech, Société d'ingénierie et de conseil à taille humaine, spécialisée dans les solutions Cloud et mobiles. Elle est située à Hay Agdal, Rabat. dans des environnements très contraints (sécurité, performance, disponibilité, volumétrie).

SanadTech, Partenaire de Fujitsu RunMyProcess depuis 2013, qui à travers sa plate-forme RunMyProcess aide les organisations à transformer leur approche de travail, grâce à des expériences user interface de bout en bout, et à l'automatisation couvrant les systèmes cloud, sur sites web ou sur mobiles.

Le personnel de SanadTech forme une petite équipe de développeurs et d'architectes qualifiés. Ils croient que chaque membre de l'équipe est important, et a son mot à dire dans le développement de produits. Ils travaillent en étroite collaboration pour résoudre les problèmes difficiles.

Ils croient au partage des connaissances, et organisent une «pause-café» hebdomadaire pour partager de nouvelles pratiques exemplaires en matière de développement de logiciels. Ils assistent et parlent pendant les rencontres locales (groupes de développeurs Google, ...).

1.1.2 L'équipe

L'entreprise se compose du personnel suivant :

ABDELHAKIM RHANIZAR Fondateur , et Directeur technique .

MOHAMMED ERRAYSY Développeur front-end.

HAMZA HAJJI Développeur front-end.

1.2 Présentation du sujet

Il y a de plus en plus des demandes à avoir l'accès à des nouveautés en temps réel, ceci est dû à la transformation digitale de la presse, qui a commencé dans les années 1994. Au Maroc, la vague de la digitalisation de la presse n'a commencé qu'à vers les années 2005, et avec le temps, le nombre de presses a augmenté, et n'a pas cessé de progresser avec la demande, et les exigences du secteur.

Il est aujourd'hui difficile de trouver un site web d'informations qui ne permette pas aux internautes de visionner des contenus audiovisuels, sous la forme de vidéos, de sons, de diaporamas sonores, de « reportages web » ou de « récits multimédia » mêlant texte, audio et vidéo. l'importance est devenue éminent, par l'intégration et l'inclusion de la vidéo dans les grands réseaux sociaux, ce qui permet à la presse de partager son contenu, et d'avoir plus de portée .

La vidéo est un composant vif, elle permet de bien diminuer la distance entre l'internaute et le sujet, et elle le permet plus, si elle est accompagnée avec un titre significatif, qui pourra donner à l'internaute, un préjugement de ce que va contenir la vidéo, elle va l'aider à choisir s'il va voir la vidéo ou non. Mais parfois, l'internaute souhaite voir qu'une seule catégorie de vidéo de nouveautés, mais malheureusement les catégories n'accompagnent pas la vidéo tout le temps.

L'entreprise SANADTECH souhaite réaliser une application mobile, qui va permettre à ses utilisateurs d'être au courant des nouveautés en temps réel, à travers un ensemble de panoplies de news (hespress ,hibapress ,ChoufTV ,yabiladi , elbotola ,hibasport , alyaoum24 , medi1tv ...etc). L'application va offrir à ses utilisateurs la possibilité de visualiser les différents contenus, par catégories .

Afin de classifier les vidéos en des catégories, on va utiliser un module intelligent à l'aide des algorithmes de machine Learning.

L'application va contenir les caractéristiques suivantes :

- L'application va permettre de visualiser les dernières nouvelles sous format vidéo du Maroc provenant de dizaines de sources d'informations nationales et régionales diverses.
- L'application va permettre de sélectionner les sources d'actualités utilisées, pour que par la suite, l'utilisateur verra que les nouveautés de celui-ci dans la file d'actualité.
- L'application va permettre au client de faire un classement automatique des nouvelles par sujets, ou il peut faire un classement manuelle, selon ces besoins.
- L'utilisateur pourra recevoir des notifications lorsque des événements importants se produisent, dans son smartphone, pour l'inviter à les voir en exclusivité.
- L'utilisateur va avoir la possibilité d'ajuster les paramètres de notification et les thèmes à son goût.
- L'application aussi permettra à ces clients d'enregistrer les vidéos à regarder plus tard.

1.3 Technologies

La réalisation de ce projet est divisée en trois parties :

1. Front-end
2. Back-end
3. Développement du modèle

et dans chaque partie, on va utiliser un ensemble d'outils, qu' on va détailler par la suite.

1.3.1 la partie front-end

Le développement front-end ou web frontal correspond aux production HTML, CSS et JavaScript d'une page internet ou d'une application qu'un utilisateur peut voir et avec lesquelles il peut interagir directement.

Javascript : JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs avec l'utilisation (par exemple)

de Node.js.

ReactXP : ReactXP est une bibliothèque pour le développement d'applications multiplate-formes utilisant React et React Native.

Redux : Redux est un conteneur d'état prévisible pour les applications JavaScript.Il aide à écrire des applications qui se comportent de manière cohérente, s'exécutent dans des environnements différents (client, serveur et natif) et sont faciles à tester.

Progressive Web Apps PWA : est une application web qui consiste en des pages ou des sites web, et qui peuvent apparaître à l'utilisateur de la même manière que les applications natives ou les applications mobiles.

1.3.2 Pour la partie back-end

Le développeur back-end travaille sur les éléments invisibles aux utilisateurs depuis le navigateur, mais indispensables au bon fonctionnement du site/application web. C'est ici qu'intervient la notion de site dynamique, qui est en constante évolution et mis à jour en temps réel avec des informations.

JEE : Positionnement de Java EE vs Java SE.Java Platform, Enterprise Edition, Java EE ou Jakarta EE (anciennement Java 2 Platform, Enterprise Edition, ou J2EE), est une spécification pour la plate-forme Java d'Oracle, destinée aux applications d'entreprise.

1.3.3 Pour la partie Développement du modèle

Python : Python est un langage de programmation objet, multi-paradigme et multiplate-formes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

Tensorflow : TensorFlow est un outil open source d'apprentissage automatique développé par Google. Le code source a été ouvert le 9 novembre 2015 par Google et publié sous licence Apache.Il est basé sur l'infrastructure DistBelief, initié par Google en 2011, et est doté d'une interface Python.TensorFlow est l'un des outils les plus utilisés en IA dans le domaine de l'apprentissage machine.

Ml-Engine : Google Cloud Machine Learning est un service géré qui vous permet de créer facilement des modèles de machine learning adaptés à tous les types et volumes de données. Concever votre modèle avec le framework ultra-performant TensorFlow, qui est utilisé dans de nombreux produits Google comme Google Photos ou Google Cloud Speech.

1.4 Méthodologie Adoptée

Dans le cadre de ce stage, on a choisi comme méthodologie de pilotage pour notre projet la méthodologie SCRUM, car elle a plusieurs atouts, et on peut les résumés comme suit :

- Plus de souplesse et de réactivité
- La grande capacité d'adaptation au changement grâce à des itérations courtes
- La chose la plus importante, c'est que SCRUM rassemble les deux côtés théorique et pratique et se rapproche beaucoup de la réalité.

La méthodologie SCRUM est composée de quatre phases (on parle aussi de réunion):

- Planification du Sprint
- Revue de Sprint
- Rétrospective de Sprint
- Mêlée quotidienne

1.4.1 Planification SCRUM

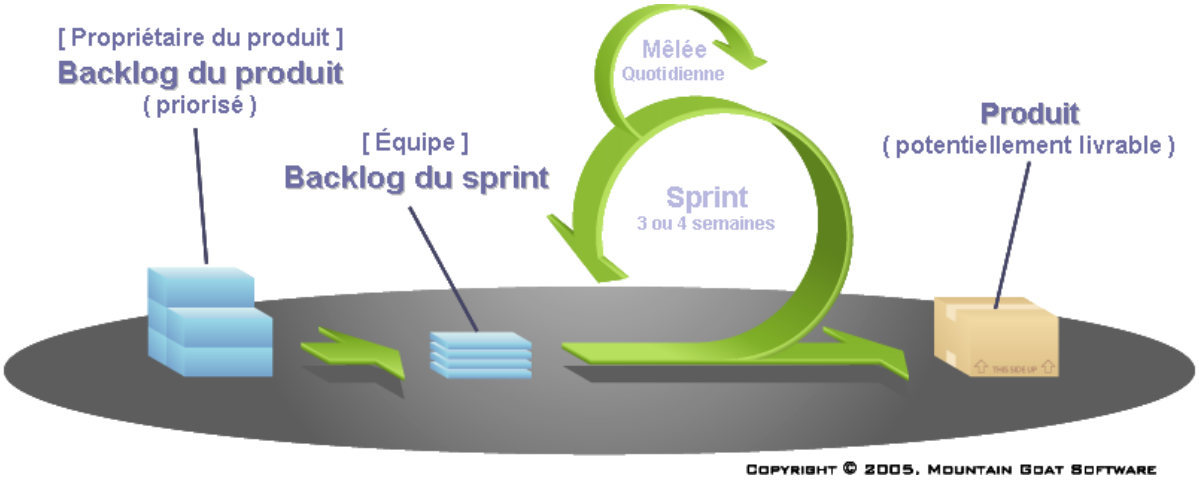


Fig. 1.1: Processus scrum

La planification du sprint correspond à lister les points prioritaires que l'équipe pense pouvoir réaliser au cours d'un sprint.

La revue du sprint a lieu en fin de sprint, l'équipe de développement présente les fonctionnalités terminées au cours du sprint et recueille les retours du représentant des utilisateurs finaux, c'est aussi à ce moment que la mise en place des prochains sprints peut être anticipée.

La Rétrospective de Sprint permet de faire un point sur le sprint en lui-même (productivité, efficacité, qualité...) afin de pouvoir s'améliorer pour les prochains sprints.

Enfin la mêlée quotidienne permet de faire un point sur les avancements de chacun, elle est courte et chacun réponds à trois questions principales : Qu'est-ce que j'ai terminé depuis la dernière mêlée ? Qu'est-ce que j'aurai terminé d'ici la prochaine mêlée ? Quels obstacles me retardent ?

1.4.2 L'équipe SCRUM

L'équipe a un rôle capital dans SCRUM : elle est constituée dans le but d'optimiser la flexibilité et la productivité; pour cela, elle s'organise elle-même et doit avoir toutes les compétences nécessaires au développement du produit. Elle est investie avec le pouvoir et l'autorité pour faire ce qu'elle a à faire.

Scrum definit les rôles suivants :

Le Product Owner (le propriétaire du produit) : c'est une personne qui porte la vision du produit à réaliser, généralement c'est un expert dans le domaine.

Le SCRUM Master (le directeur de produit) : c'est la personne qui doit assurer le bon déroulement des différents sprints du release, et qui doit impérativement maitriser SCRUM.

Le SCRUM Team (l'équipe de SCRUM) : constitué de personnes qui seront chargées d'implémenter les différents besoins du client. Bien évidemment, cette équipe sera constituée des développeurs, des testeurs, etc.

Dans le contexte de notre projet, Mr RHANIZAR est le propriétaire du produit, et le directeur du produit puisqu'il satisfait les différents prés-requis de ces rôles cités précédemment et nous formons moi ELKAISSI SOUHAIL et CHATRIA ZAKARIA, les deux membres de l'équipe SCRUM.

1.5 Langage de modélisation

Le langage de modélisation adopté durant ce stage sera le UML ,comme étant un outil comportant multiples points forts notamment sa standardisation ainsi que les divers diagrammes qu'il propose, et qui permettent la schématisation des systèmes complexes sous un format graphique et textuel simplifié et normalisé.
Généralement, le recours à la modélisation UML et plus particulièrement UML 2.0 procure de nombreux avantages qui agissent sur :

- La modularité.
- L'abstraction.
- La dissimulation.
- La structuration cohérente des fonctionnalités et des données.

1.6 TensorFlow : Librairie du machine learining et deep learning

TensorFlow est une bibliothèque de logiciels open source pour le calcul numérique utilisant des graphes de flux de données. Il a été initialement développé par l'équipe Google Brain au sein de l'organisation de recherche Machine Intelligence de Google pour l'apprentissage automatique et la recherche sur les réseaux neuronaux profonds, mais le système est assez général pour être applicable dans une grande variété d'autres domaines. Il a atteint la version 1.0 en février 2017 et a poursuivi son développement rapide, avec plus de 21 000 engagements à ce jour, dont un grand nombre de contributeurs externes.

TensorFlow est multi-plateforme. Il fonctionne sur presque tout : les processeurs graphiques et les processeurs normaux, y compris les plates-formes mobiles et embarquées, et même les unités de traitement tensoriel (TPU), qui sont des matériels spécialisés pour faire des calculs tensoriels. Ils ne sont pas encore largement disponibles, mais nous avons récemment lancé un programme alpha.

1.7 Planification prévisionnelle

Nom de la tâche	**Début de la tache**	**Date de la fin**	**Durée**
Début du stage cadre	23/04/2018	23/04/2018	1j
Phase de preparation	23/04/2018	01/05/2018	5j
Étude de la Solution actuelle	*22/04/2018*	*24/04/2018*	*2j*
Compréhension la totalité du projet	*25/04/2018*	*01/05/2018*	*3j*
Sprint 1: Réalisation d'un modèle de prédiction avec l'algorithme CNN	02/05/2018	23/05/2018	18j
Rédaction du Rapport partie 1	25/05/2018	25/05/2018	1j
Sprint 2: Réalisation d'un modèle de prédiction avec la méthode SVM	26/05/2018	04/06/2018	7j
Rédaction du Rapport partie 2	05/06/2018	06/06/2018	2j
Sprint 3: Amélioration du modèle CNN en ajoutant une couche de word2vec	07/06/2018	13/06/2018	8j
Rédaction du Rapport partie finale	14/06/2018	16/05/2018	3j
Préparation de la soutenance	17/06/2018	21/06/2018	5j
Soutenance du Projet Cadre	22/06/2018	22/06/2018	1j
Sprint 4: Réalisation d'un modèle hybride qui regroupe CNN avec word2vec et le SVM	23/06/2018	06/07/2018	17j
Sprint 5: Déploiement de la solution finale	09/07/2018	14/07/2018	5j

FIG. 1.2: planification prévisionnelle

Chapitre 2

Analyse et Conception

2.1 Le Backlog du Produit

Le backlog du produit est la liste des fonctionnalités attendues d'un produit. Plus exactement, au-delà de cet aspect fonctionnel, il contient tous les éléments qui vont nécessiter du travail pour l'équipe. Les éléments y sont classés par priorité ce qui permet de définir l'ordre de réalisation.

Comme il est mentionné sur la figure 2.1, le backlog est élaboré avant le lancement des sprints, dans la phase de préparation (ou sprint 0). Il est utilisé pour planifier la release, puis à chaque sprint, lors de la réunion de planification du sprint pour décider du sous-ensemble qui sera réalisé.

C'est donc un outil essentiel pour la planification. Mais il est aussi, par sa nature, un maillon de la gestion des exigences, puisqu'on y collecte ce que doit faire le produit.

A tout moment, le backlog est visible par tout le monde.

2.1.1 La méthode de priorisation

Pour prioriser le besoin, on a utilisé la méthode **MoSCow** ; c'est une technique visant à prioriser des besoins ou des exigences en matière d'assistance à maîtrise d'ouvrage et de développement logiciel. L'objectif est que le maître d'œuvre et le maître d'ouvrage s'accordent sur l'importance des tâches à réaliser par rapport aux délais prévus.

Les lettres majuscules de l'acronyme MoSCoW signifient (en anglais):

Story	**Priorité2**	**Effort ou chargen**
En tant qu'utilisateur, je souhaite pouvoir voir les news dans une fil d'actualité,et de sélectionner un contenu pour le visualiser.	M	22
En tant qu'utilisateur, je souhaite pouvoir ajouter un contenu au favoris pour que je puisse le visualiser plus tard.	C	11
En tant qu'utilisateur, je souhaite pouvoir visiter les différents contenus par catégories,et de les sélectionner selon mon besoin.	M	60

FIG. 2.1: Le Backlog du Produit

M must have this, c'est-à-dire 'doit être fait' (vital).

S should have this if at all possible, c'est-à-dire devrait être fait dans la mesure du possible (essentiel).

C could have this if it does not affect anything else, pourrait être fait dans la mesure où cela n'a pas d'impact sur les autres tâches (confort).

W won't have this time but would like in the future, ne sera pas fait cette fois mais sera fait plus tard' (luxe, c'est votre zone d'optimisation budgétaire).

Chapitre 3

Sprint 0 : Preparation et étude de projet

3.1 Le Backlog du Sprint

Sur la figure 3.1, nous présentons la backlog du sprint 0.

Story	**Priorité2**	**Effort ou chargen**
En tant qu'utilisateur, je souhaite pouvoir voir les news dans une fil d'actualité,et de sélectionner un contenu pour le visualiser.	M	22
En tant qu'utilisateur, je souhaite pouvoir ajouter un contenu au favoris pour que je puisse le visualiser plus tard.	C	11

FIG. 3.1: Le Backlog du Sprint 0

3.2 Conception

3.2.1 Diagramme de cas d'utilisation : services client

Comme on peut voir dans le digramme **FIG.3.2** , on a trois acteurs : le client, l'administrateur, et le modèle de machine learning de classification.

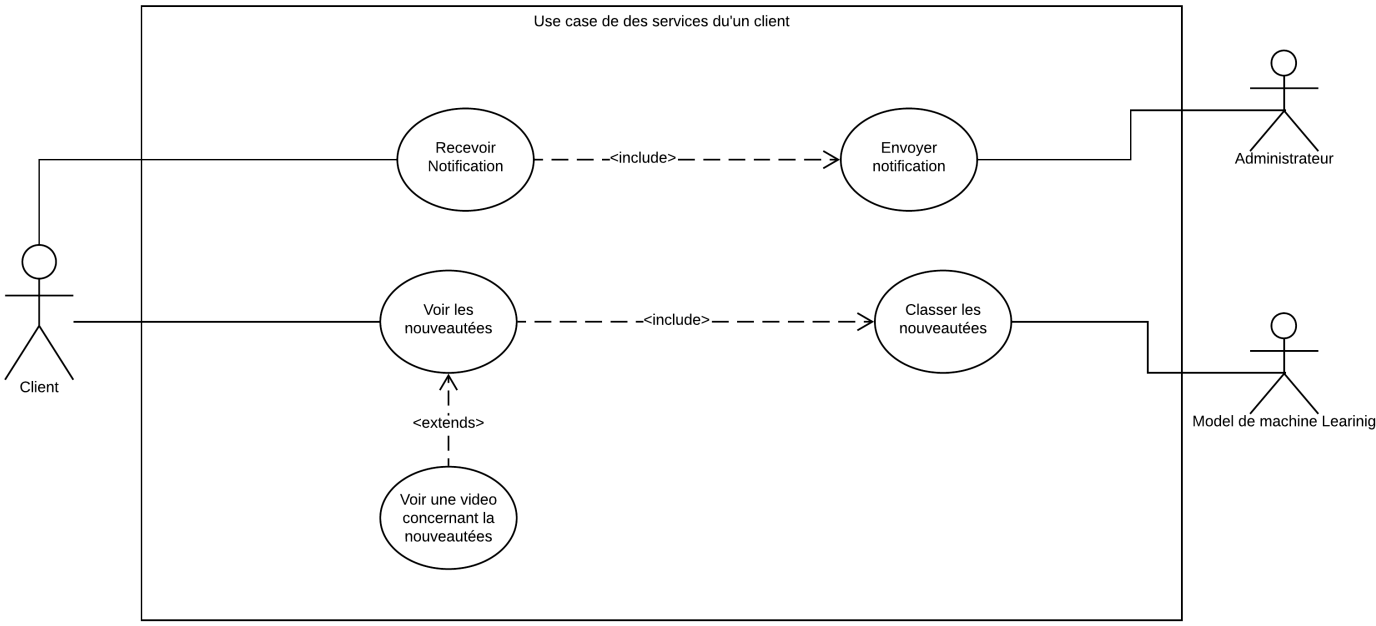


FIG. 3.2: Diagramme de cas d'utilisation

Pour que le client puisse visualiser les nouveautés , il faut que ces dernières soient classifier selon leurs catégories par le modèle, après cette manipulation, le client pourra naviguer dans la page d'accueil et voir les nouveautés .

Quand l'administrateur verra qu'une nouveauté est pertinente, il va lancer une notification, laquelle le client va recevoir, et pourra la visualiser en exclusivité.

3.2.2 Diagramme de classes

Comme on peut voir dans le digramme de la figure **FIG.3.3** , on a cinq classes : User, News, Catégorie; ApiAtacker et le modèle.

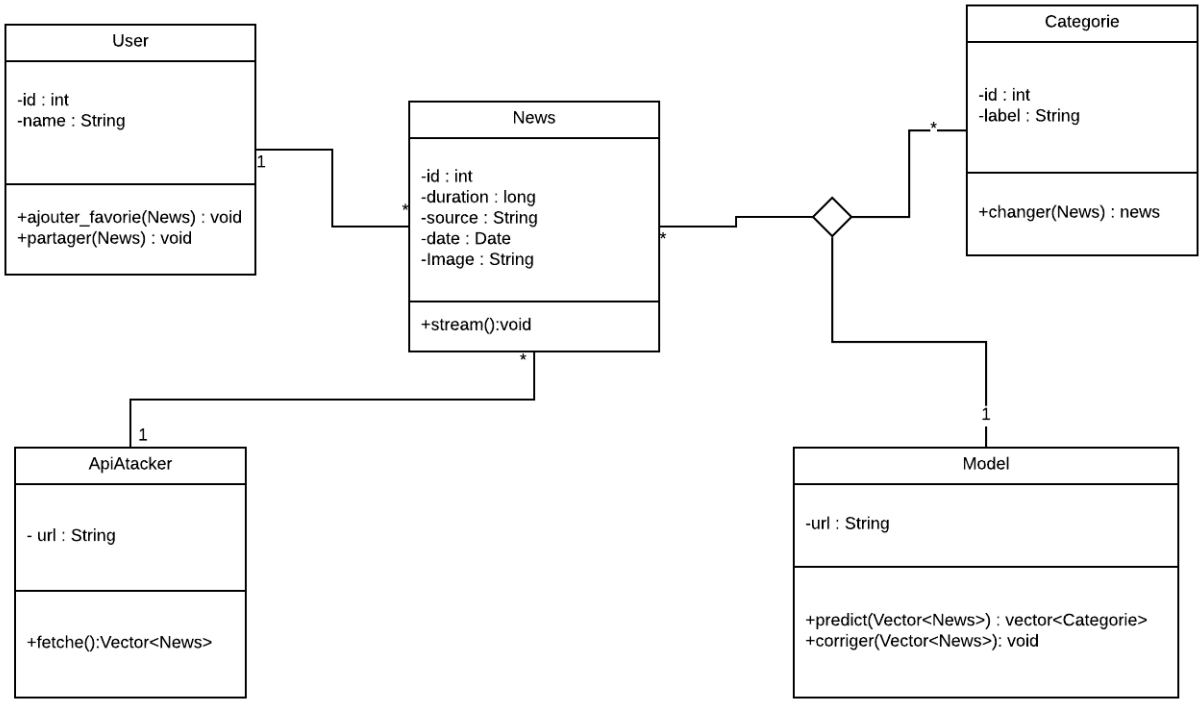


Fig. 3.3: Diagramme de classe

La classe **ApiAtacker**,c'est la classe responsable de l'acquisition des données depuis les différentes sources qu'on a, tels que : youtube, hesspress ...

La classe News représente une nouveauté dans notre système, et chaque nouveauté est liée par une et une seul catégorie.

Ici, la classe Model joue le rôle du classificateur, elle est celle qui transforme une nouveauté non étiquetée en une qui est étiquetée.

Par ces différentes méthodes, la classe User représente un client dans notre système, il décrit toutes ses actions possible.

3.2.3 Diagramme de séquence

Comme nous pouvant constater dans le diagramme de la figure **FIG.3.4**, nous avons quatre ligne de vie dont le Client, l'Application front-end, l'Application back-end,et le Model tensorFlow.

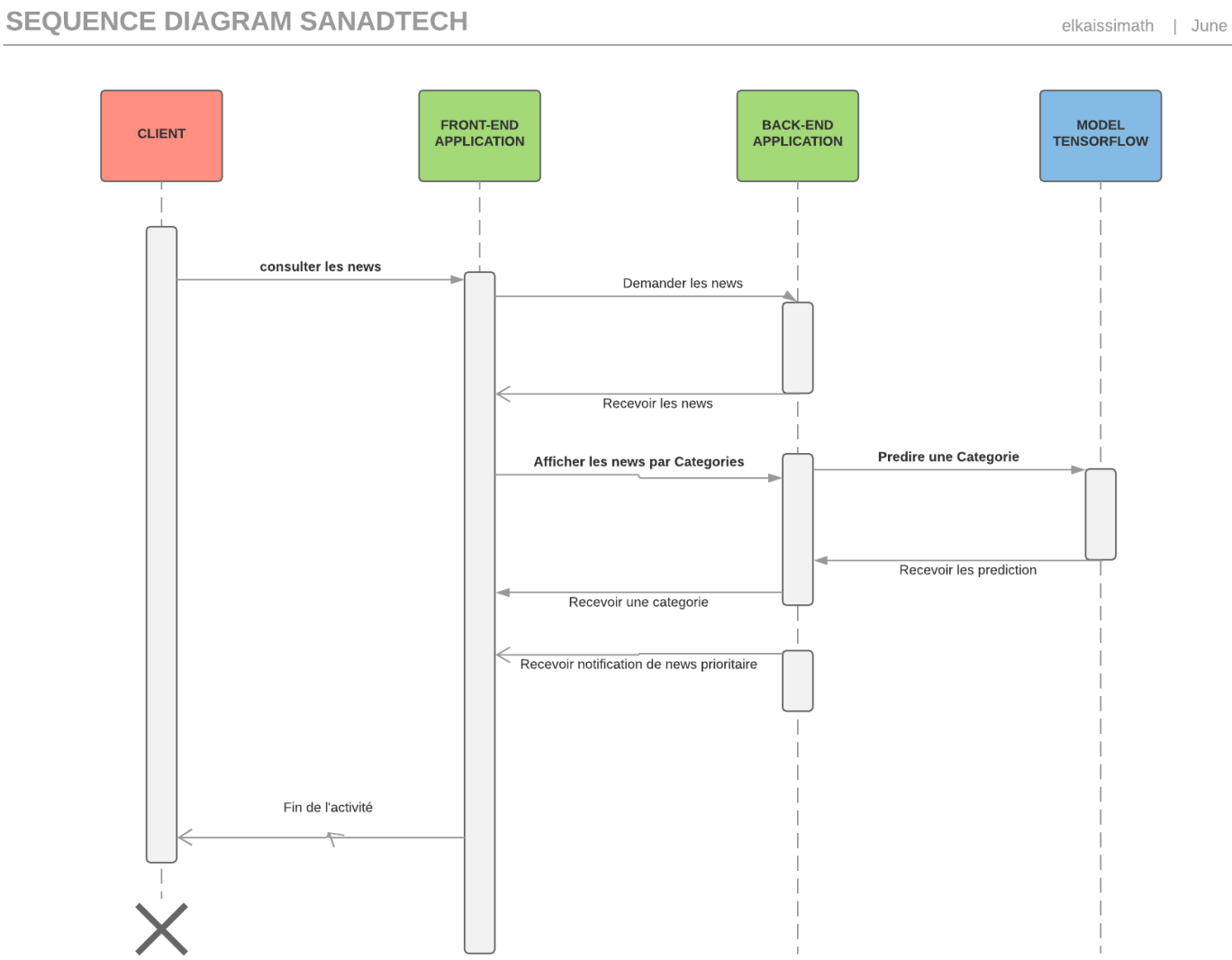


FIG. 3.4: Diagramme de séquence

L'activité de l'application au point de vue client commence, qu'on un utilisateur installe l'app dans son téléphone, il pourra ensuite recevoir les notifications des nouveautés les plus importantes.

Quand un utilisateur consulte les nouveautés , le front-end envoi une requête au back-end, et ce dernier lui envoi tout les news par ordre de date.

Si une news viens juste d'être intercepter par le back-end, elle envoie alors, une requête au modèle Tensorflow pour savoir à quelle catégorie cette news appartient. Et puis après, le back-end l'enregistre avec sa nouvelle classe ou catégorie.

Si une nouveauté est jugée comme importante, elle sera envoyée vers toutes les utilisateurs sous

la forme d'une notifications, depuis le back-end.

3.2.4 Diagramme de déploiement

Le diagramme de la figure **FIG.3.5** contient 4 nœuds principaux, on site : front-end server, back-end server, database server , Model tensorflow server.

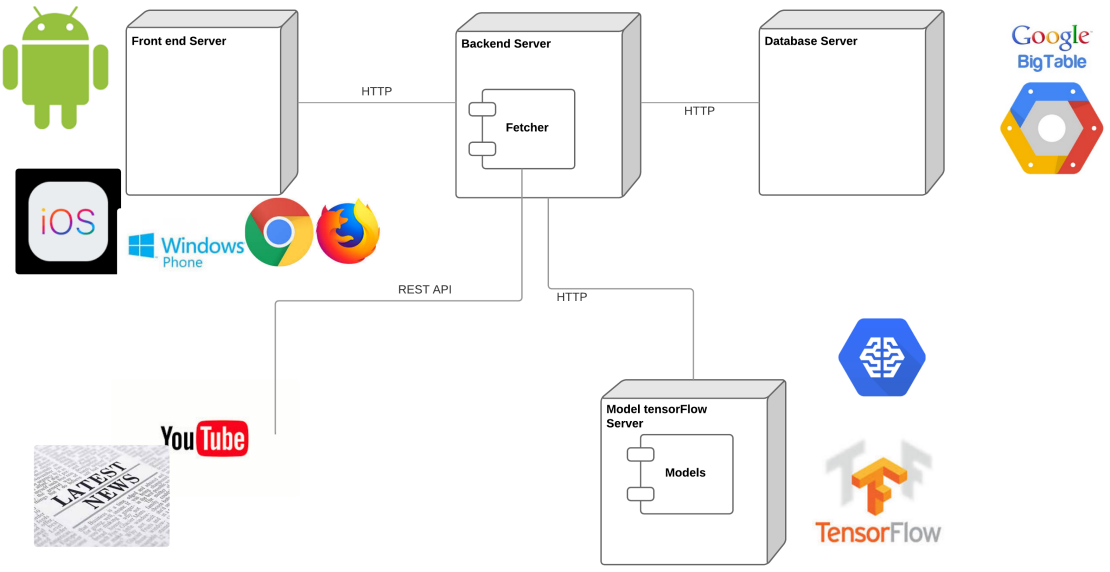


FIG. 3.5: Diagramme de déploiement

Front-end server , c'est où l'application finale est déployée, d'ici les utilisateurs peuvent la télécharger, depuis les différentes Stores, comme google play, apple store ...,ou la consulter dans le cas d'un navigateur.

Back-end server, c'est où le apiAtacker est situé , l'application envoie des requêtes vers ce nœud pour avoir les news.

Database server, ici on stocke nos données , on utilise la base de donnée de google Big Table.

Model Tensorflow Server, c'est où notre modèle de prédiction est stocké , le modèle est écrit par tensorflow et déployé dans le service google ML-engine.

3.3 NLP : Natural Language Processing

Le modèle de classification qu'on va réaliser entre dans la catégorie de la NLP, avant de le définir et l'expliquer, il faut d'abord rappeler qu'est ce que la machine learning et le deep learning.

3.3.1 Machine learning

L'apprentissage automatique (en anglais machine learning, littéralement « l'apprentissage machine ») ou apprentissage statistique, champ d'étude de l'intelligence artificielle, concerne la conception, l'analyse, le développement et l'implémentation de méthodes permettant à une machine (au sens large) d'évoluer par un processus systématique, et ainsi de remplir des tâches difficiles ou problématiques par des moyens algorithmiques plus classiques.

L'analyse peut concerner des graphes, arbres, ou courbes (par exemple, la courbe d'évolution temporelle d'une mesure ; on parle alors de données continues, par opposition aux données discrètes associées à des attributs-valeurs classiques) au même titre que de simples nombres. Voir l'article Inférence bayésienne.

Les algorithmes utilisés permettent, dans une certaine mesure, à un système piloté par ordinateur (un robot éventuellement), ou assisté par ordinateur, d'adapter ses analyses et ses comportements en réponse, en se fondant sur l'analyse de données empiriques provenant d'une base de données ou de capteurs.
La difficulté réside dans le fait que l'ensemble de tous les comportements possibles compte tenu de toutes les entrées possibles devient rapidement trop complexe à décrire (on parle d'explosion combinatoire). On confie donc à des programmes le soin d'ajuster un modèle pour simplifier cette complexité et de l'utiliser de manière opérationnelle. Idéalement, l'apprentissage visera à être non supervisé, c'est-à-dire que la nature des données d'entrainement n'est pas connue.

Ces programmes, selon leur degré de perfectionnement, intègrent éventuellement des capacités de traitement probabiliste des données, d'analyse de données issues de capteurs, de reconnaissance (reconnaissance vocale, reconnaissance de forme, d'écriture...), de data-mining, d'informatique théorique...

3.3.2 Deep Learning

L'apprentissage profond1 (en anglais deep learning, deep structured learning, hierarchical learning) est un ensemble de méthodes d'apprentissage automatique tentant à modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires[réf. souhaitée]. Ces techniques ont permis des progrès importants et rapides dans les domaines de l'analyse du signal sonore ou visuel et notamment de la reconnaissance faciale, de la reconnaissance vocale, de la vision par ordinateur, du traitement automatisé du langage. Dans les années 2000, ces progrès ont suscité des investissements privés, universitaires et publics importants, notamment de la part des GAFA (Google, Apple, Facebook, Amazon).

Les recherches dans ce domaine s'efforcent de construire de meilleures représentations du réel et de créer des modèles capables d'apprendre ces représentations à partir de données non labellisées à grande échelle. Certaines de ces représentations s'inspirent des dernières avancées en neuroscience. Il s'agit, grosso modo, d'interprétations du traitement de l'information et des modèles de communication du système nerveux, à l'image de la façon dont le système nerveux établit des connexions en fonction des messages reçus, de la réponse neuronale et du poids des connexions entre les neurones du cerveau.
Les différentes architectures de « deep learning » telles que les « deep neural networks », les « convolutional deep neural networks », et les « deep belief network » ont plusieurs champs d'application :

- la vision par ordinateur (reconnaissance de formes) ;
- la reconnaissance automatique de la parole ;
- le traitement automatique du langage naturel ;
- la reconnaissance audio et la bio-informatique.

Dans ces deux derniers domaines, notamment, elles ont obtenu des résultats très prometteurs.

3.3.3 NLP

Le traitement automatique du langage naturel (abr. TALN), ou traitement automatique de la langue naturelle1, est un domaine multidisciplinaire impliquant la linguistique, l'informatique et l'intelligence artificielle. Il vise à créer des outils de traitement de la langue naturelle pour

diverses applications. Il ne doit pas être confondu avec la linguistique informatique, qui vise à comprendre les langues au moyen d'outils informatiques.

Le TALN est sorti des laboratoires de recherche pour être progressivement mis en œuvre dans des applications informatiques nécessitant l'intégration du langage humain à la machine2. Aussi le TALN est-il parfois appelé ingénierie linguistique3. En France, le traitement automatique de la langue naturelle a sa revue, ATALA, publiée par l'Association pour le traitement automatique des langues.

3.4 Préparation des données

Dans un projet de machine learning, il est important de bien encadrer les données, parce que en sens large, il constitue l'enjeu majeur. Pour ce fait, on a suivi ces principaux étapes :

3.4.1 Identification des sources de données

Les données qu'on traite sont des données de type chaine de caractère qui comporte deux champs, le premier champs est celui de la catégorie ou la classe, et le deuxième champs est celui de la description ou le titre de la vidéo.

Category	Descript
sport	انتظارات من أسود الأطلس
politics	الجزائر.. وضعية بوتفليقة تطغى على افتتاح دورة البرلمان
society	روبورتاج.. أعباء الدخول المدرسي ثقل كاهل أولياء التلاميذ
society	توقعات المغاربة لمباراة الإياب بين المنخب المغربي والمنتخب المالي
society	أطباء عاطلون يحتجون بالرباط
sport	البطل محمد الصنهايي يتألق في الكيك بوكسينغ
society	خنيفرة - مشروع التكتل من اجل العدالة للنساء
society	الواجهة: العثور على جثة سبعيني ميت في بيته
society	جهة الدار البيضاء-سطات ومركز تحاقن الدم يدقان ناقوس الخطر: تبرعوا بالدم لإنقاذ إخوانكم
business	ملف.. الشراكة المغرب الإسبانية في المجال التجاري
various	سوق الصباط بالرباط
sport	أصغر بطلة فول كونتاكت
various	تافوغالت .. أريج الفجاج
culture	دردشة مع بدر سلطان
various	أبراج | أشنو قال زهرك اليوم : 03 يوليوز 2017 | شوف تيفي
politics	زوجة المهدوي تناقش حراك الريف
various	أحوال الطقس : 03 يوليوز 2017
politics	غضبة ملكية تطيح بمسؤولي الدرك بتطوان
politics	المغرب يوجه صفعة قوية للبوليساريو والجزائر بأثيوبيا

FIG. 3.6: Exemple de données

Le format des données actuellement, avec lequelle on va faire l'entrainement et les tests, sont en format **CSV**, l'avantage de ce format c'est qu'il est facilement manipulable par le langage de programmation python même si la taille de données peut être grande.

On récupère ces données là depuis plusieurs sources, et après on extrait les données jugées utiles pour les prédictions, dans ce cas, les descriptions des vidéo.

3.4.2 Définition du lieu d'entreposage

Les données transformées doivent être stockées, pour le fait de ne pas les prédire une deuxième fois, on utilise dans notre projet la base données google, Big Table, c'est un système de gestion de base de données compressées, haute performance, et propriétaire. C'est une base de données orientée colonnes, dont se sont inspirés plusieurs projets libres, comme HBase, Cassandra ou Hypertable.

FIG. 3.7: google cloud big table

Google ne distribue pas sa base de données mais propose une utilisation publique de BigTable via sa plateforme d'application Google App Engine.

3.4.3 Nettoyage des données

Les données utilisées proviennent de diverses sources et processus et peuvent contenir des irrégularités ou des données corrompues compromettant la qualité de l'ensemble de données.

Les problèmes typiques de qualité des données qui se posent sont les suivants:

Incomplet : Les données manquent d'attributs ou contiennent des valeurs manquantes.

Noisy : les données contiennent des enregistrements erronés ou des valeurs aberrantes.

Inconsistant : les données contiennent des enregistrements ou des divergences conflictuels.

Les données de qualité sont une condition préalable à la qualité des modèles prédictifs. Pour éviter les «déchets» et améliorer la qualité des données et donc la performance du modèle, nous avons trouvé qu'il est impératif de réaliser un écran d'intégrité des données afin de repérer rapidement les problèmes de données et de décider des étapes de traitement et de nettoyage correspondantes.

3.4.4 Features generation

C'est un processus consistant à prendre des données brutes non structurées et à définir des caractéristiques (c'est-à-dire des variables) pour une utilisation potentielle dans l'analyse statistique.
Par exemple, dans le cas de l'exploration de texte *(notre cas)*, on a commencé avec un journal brut de milliers de descriptions *(celles des vidéos)* et générer des fonctionnalités en supprimant des mots de faible valeur. des blocs de mots *(n-grammes)* ou en appliquant d'autres règles.

pour notre cas, on prend le corpus de données et on fait une relation symétrique entre les données de types chaine de caractère,et des entiers unique, qu'on le stocke par la suite dans un dictionnaire de données.

3.4.5 Feature extration

Une fois les fonctionnalités générées, il est souvent nécessaire de tester les transformations des entités d'origine et de sélectionner un sous-ensemble de données de caractéristiques potentielles originales et dérivées à utiliser dans notre modèle (extraction et sélection de caractéristiques).

Tester des valeurs dérivées est une étape courante car les données peuvent contenir des informations importantes qui ont un modèle ou une relation non linéaire avec notre résultat, ainsi l'importance de l'élément de données peut seulement apparaître dans son état transformé (par exemple des dérivés d'ordre supérieur).

L'utilisation d'un trop grand nombre de caractéristiques peut entraîner une colinéarité multiple ou confondre des modèles statistiques, alors qu'extraire le nombre minimum de caractéristiques correspondant au but de votre analyse suit le principe de la parcimonie.

On a divisé les données generées en deux parties, les données d'entrainement qu'on lui a attribué 60% des données du corpus, et les données de test qui formes un pourcentage de 40%.

Chapitre 4

Sprint 1 : Réalisation d'un modèle de prédiction avec l'algorithme CNN

4.1 Le Backlog du Sprint 1

La figure 4.1 décrit le baclog du sprint numéro 1.

Story	**Priorité2**	**Effort ou chargen**
En tant qu'utilisateur, je souhaite pouvoir visiter les différents contenus par catégories,et de les sélectionner selon mon besoin.	M	22

FIG. 4.1: Le Backlog du Sprint 1

4.2 Conception

4.2.1 Qu'est ce qu'un CNN/RNN ?

Réseau de neurones convolutifs

En apprentissage automatique, un réseau de neurones convolutifs ou réseau de neurones à convolution (en anglais CNN ou ConvNet pour Convolutional Neural Networks) est un type de réseau

de neurones artificiels acycliques (feed-forward), dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont arrangés de sorte qu'ils correspondent à des régions qui se chevauchent lors du pavage du champ visuelle.

Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilage multicouche de perceptrons, dont le but est de pré-traiter de petites quantités d'informations. Les réseaux neuronaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation et le traitement du langage naturel.

Réseau de neurones récurrents

Un réseau de neurones récurrents est un réseau de neurones artificiels présentant des connexions récurrentes. Un réseau de neurones récurrents est constitué d'unités (neurones) inter-connectés interagissant non-linéairement et pour lequel il existe au moins un cycle dans la structure. Les unités sont reliées par des arcs (synapses) qui possèdent un poids. La sortie d'un neurone est une combinaison non linéaire de ces entrées.

Les réseaux de neurones récurrents sont adaptés pour des données d'entrée de taille variable. Ils conviennent en particulier pour l'analyse de séries temporelles. Ils sont utilisés en reconnaissance automatique de la parole ou de l'écriture manuscrite - plus en général en reconnaissance de formes - ou encore en traduction automatique. « Dépliés », ils sont comparables à des réseaux de neurones classiques avec des contraintes d'égalité entre les poids du réseau (voir schéma ci-dessous).

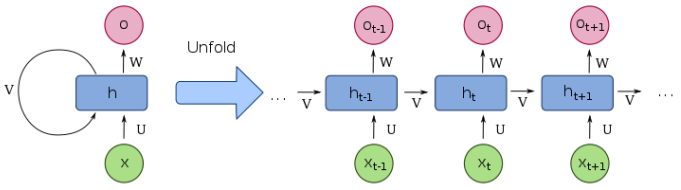


FIG. 4.2: Schéma d'un réseau de neurones récurrents à une unité reliant l'entrée et la sortie du réseau. A droite la version « dépliée » de la structure.

4.2.2 L'architecture de notre réseaux neuronal

Le réseaux neuronal qu'on a adopté est comme suite :

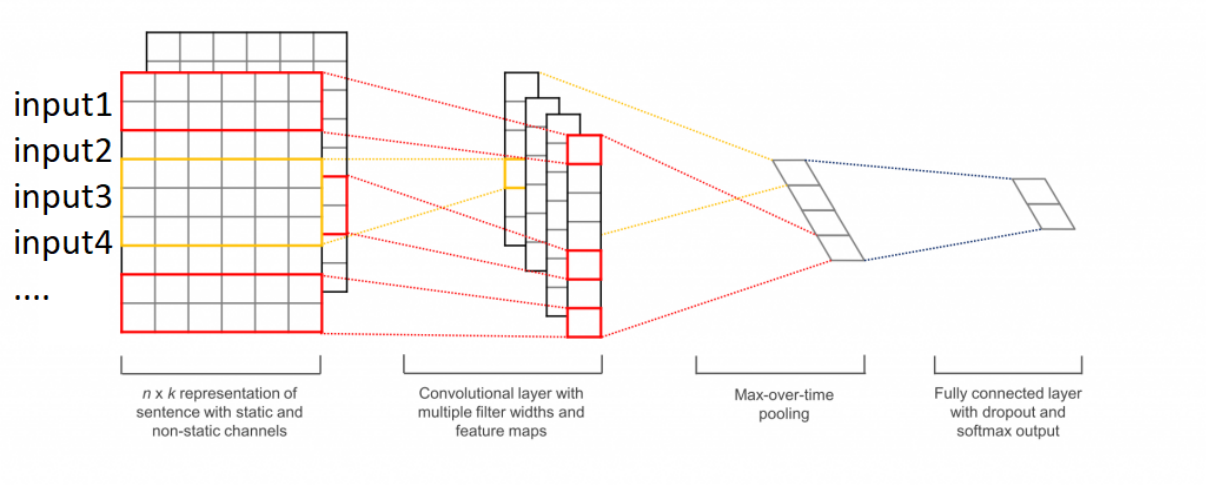


FIG. 4.3: La structure du réseaux neuronal ,réalisée par cnn.

Les premières couches incorporent des mots dans des vecteurs de basse dimension. La couche suivante effectue des convolutions sur les vecteurs de mots incorporés, en utilisant des tailles de filtres multiples.

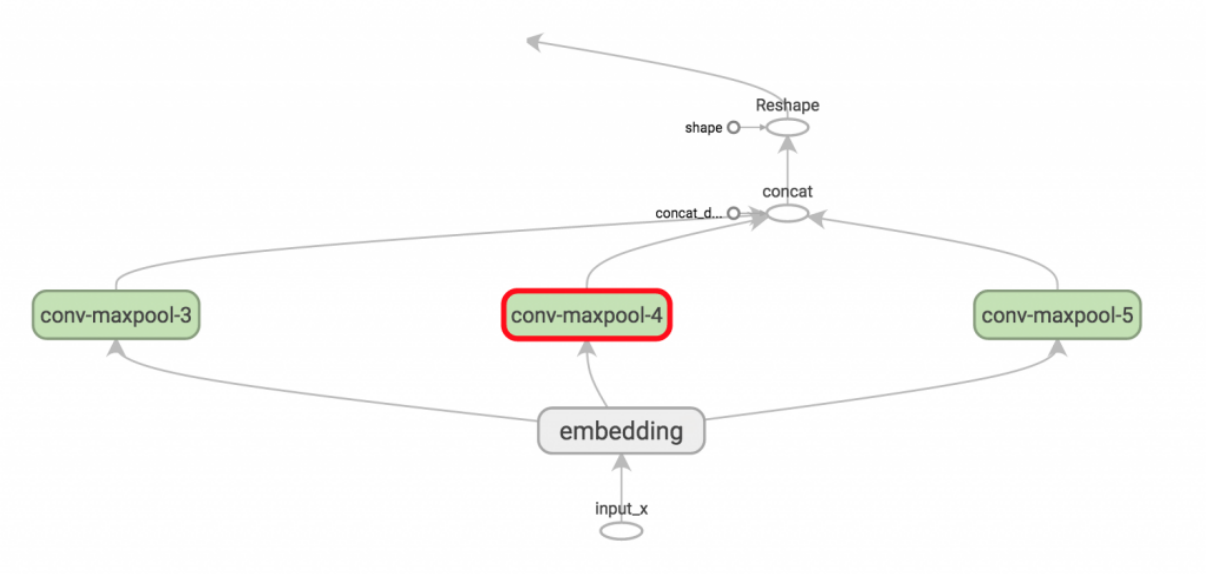


FIG. 4.4: La structure du réseaux neuronal, structure des couches de convolutions.

Par exemple, glisser sur 3, 4 ou 5 mots à la fois. Ensuite, nous (max-poolons) le résultat de la couche convolutionnelle dans un long vecteur de caractéristiques,et on ajoute une régularisation de décrochage, et enfin on classe le résultat en utilisant une couche softmax.

Puisque on ne peut pas traiter les chaînes de caractères dans un réseau neuronal, on doit transformer ces données là, en des données numériques, c'est pour cela, j'ai utilisé une transformation classique « Embeding» qui mappe chaque mot qui existe dans mon dictionnaire de mot ,en des numéros significatifs.

Pour bien comprendre le déroulement ,et le processus de fonctionnement de ce réseaux neuronal , voici une figure descriptive :

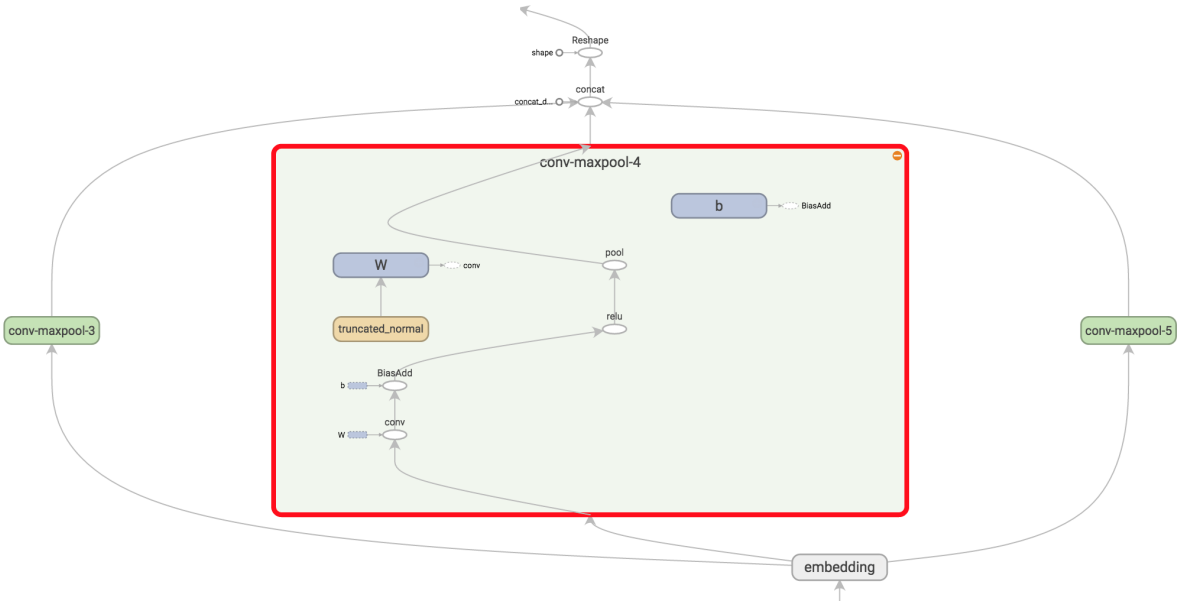


FIG. 4.5: Architecture des couches du réseaux neuronal.

Ici, **W** est notre matrice de filtre et h est le résultat de l'application de la non-linéarité à la sortie de la convolution. Chaque filtre glisse sur l'ensemble de l'incorporation, mais varie en nombre de mots. Le remplissage *VALIDE* signifie que nous glissons le filtre sur notre phrase sans capitonner les bords, en effectuant une convolution étroite qui nous donne une sortie de forme [1,$sequenceLengthFilterSize + 1$,1,1].

L'exécution de **max-pooling** sur la sortie d'une taille de filtre spécifique nous laisse avec un tenseur de forme [$batchSize$,1,1,$numFilters$]. C'est essentiellement un vecteur de caractéristiques, où la dernière dimension correspond à nos caractéristiques.

Une fois que nous avons tous les tenseurs de sortie regroupés de chaque taille de filtre, nous les combinons en un long vecteur de forme [$tailleBatch$,$numFiltersTotal$].

L'utilisation de -1 dans tf.reshape indique à TensorFlow d'aplatir la côte lorsque cela est possible.

Pour régulariser notre CNN, j'ai utilisé une couche **Dropout** ,l'idée derrière est simple. Une couche de décrochement stochastique **désactive** une fraction de ces neurones. Cela empêche les neurones de co-adapter, et les oblige à apprendre des fonctionnalités utiles individuellement.

La fraction de neurones que nous gardons activée est définie sur notre réseau. Nous avons réglé ceci à quelque chose comme *0.5* pendant l'entraînement, et à *1* (désactiver la couche) pendant l'évaluation.

En utilisant le vecteur de caractéristiques de **max-pooling** (avec **dropout** appliqué), nous pouvons générer des prédictions en faisant une multiplication matricielle et en choisissant la classe ayant le score le plus élevé. Nous pourrions également appliquer une fonction **softmax** pour convertir les scores bruts en probabilités normalisées, mais cela ne changerait pas nos prédictions finales.

En utilisant nos *scores*, nous pouvons définir la fonction de perte. La perte est une mesure de l'erreur que notre réseau fait, et notre objectif est de le minimiser. La fonction de perte standard pour la catégorisation pose des problèmes de perte d'*entropie* croisée.

Enfin le graphe final résultant ressemble à ça :

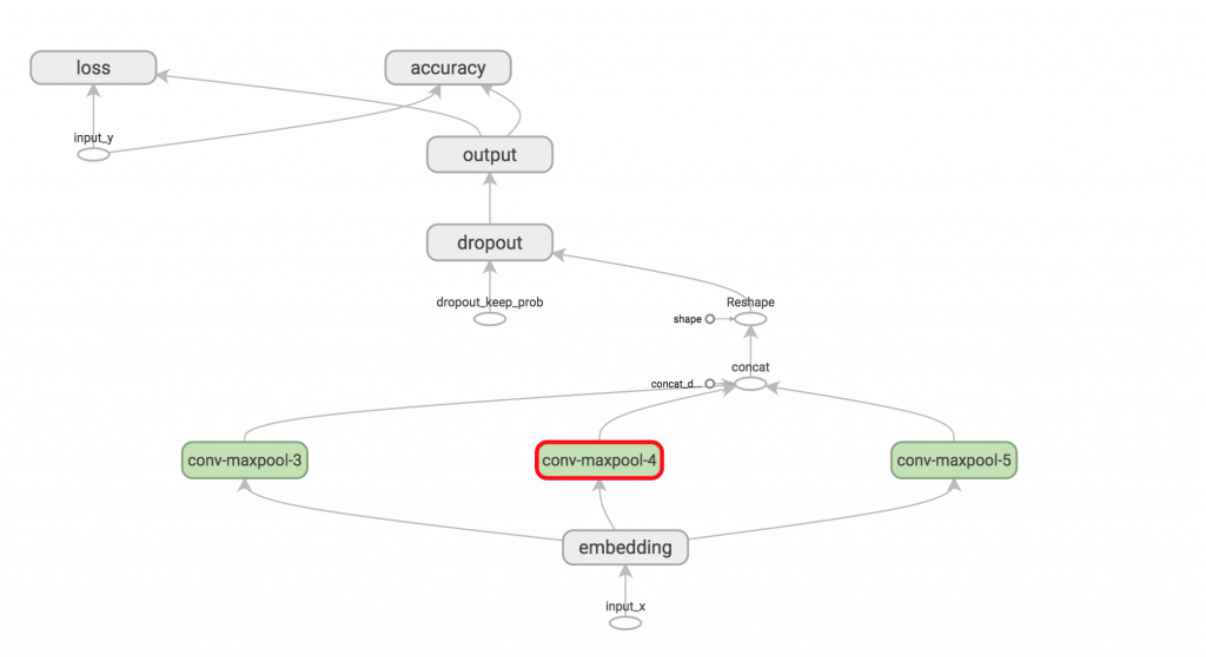


FIG. 4.6: Graphe qui représente les étapes du traitement du modèle.

L'état des pertes sur les variables de décision :

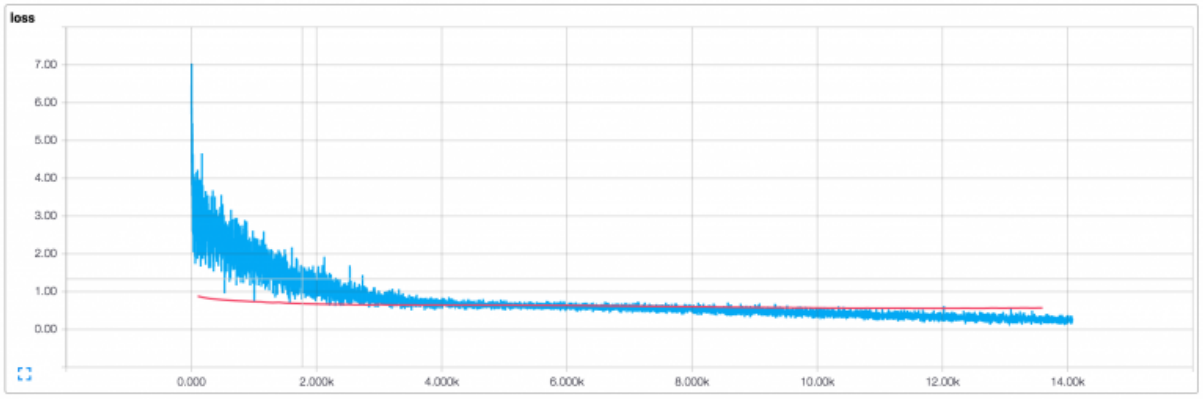


FIG. 4.7: Graphe qui représente les données perdues, du reseaux CNN/RNN.

l'état des précision qui est égale à 0,76:

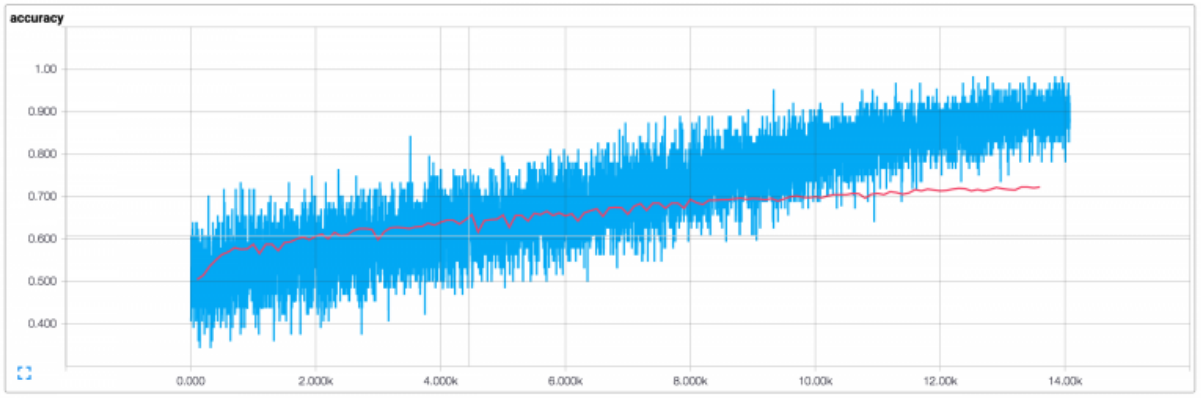


FIG. 4.8: Graphe qui représente la précision, du reseaux CNN/RNN

Chapitre 5

Sprint 2 : Réalisation d'un modèle de prédiction avec l'algorithme SVM

5.1 Le Backlog du Sprint 2

Dans la figure 5.1, on décrit le backlog du sprint numéro 2.

Story	**Priorité2**	**Effort ou chargen**
En tant qu'utilisateur, je souhaite pouvoir visiter les différents contenus par catégories,et de les sélectionner selon mon besoin.	M	22

FIG. 5.1: Le Backlog du Sprint 2

5.2 Conception

5.2.1 Qu'est ce que le SVM ?

Les machines à vecteurs de support ou séparateurs à vaste marge (en anglais support vector machine, SVM) sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination 1 et de régression. Les SVM sont une généralisation des classifieurs linéaires.

Les séparateurs à vaste marge ont été développés dans les années 1990 à partir des considérations théoriques de Vladimir Vapnik sur le développement d'une théorie statistique de l'apprentissage : la théorie de Vapnik-Chervonenkis. Ils ont rapidement été adoptés pour leur capacité à travailler avec des données de grandes dimensions, le faible nombre d'**hyperparamètres**, leurs garanties théoriques, et leurs bons résultats en pratique.

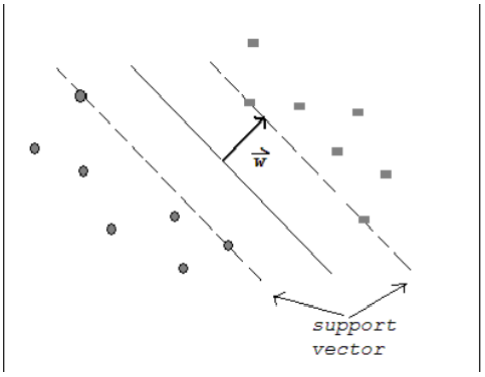


FIG. 5.2: Exemple d'un hyperplan pattern de SVM

Les SVM ont été appliqués à de très nombreux domaines (bio-informatique, recherche d'information, vision par ordinateur, finance...). Selon les données, la performance des machines à vecteurs de support est de même ordre, ou même supérieure, à celle d'un réseau de neurones ou d'un modèle de mélanges gaussiens.

Les machines à vecteurs de support sont basées sur le principe de la structure de minimisation de risque, dans la théorie de l'apprentissage aléatoire.L'idée de la structure de minimisation de risque est de trouver une hypothèse $\boldsymbol{h}$ avec laquelle on peut garantir, d'avoir le plus petit cout d'erreur.

La vrai erreur de $\boldsymbol{h}$, c'est la probabilité que $\boldsymbol{h}$ va faire une erreur en une donnée non traitée,et choisie aléatoirement. Un extremum haut, peut être utilisé, pour connecter, la vrai erreur de $\boldsymbol{h}$, avec l'erreur de $\boldsymbol{h}$ et avec les données de l'entrainement, est une complexité $\boldsymbol{H}$ (mesurer par VC-Dimension), l'hypothèse $\boldsymbol{h}$, laquelle approximativement, minimise l'extremum pour la vrai valeur de $\boldsymbol{h}$ en controllant the VC-Dimension de $\boldsymbol{H}$.

Les SVMs, sont des apprentis universels. Dans leur forme de base, ils apprennent par des fonctions seuil linéaires. Sauf que par une utilisation des fonction de kernel à la place des variables

unitaires, ils peuvent être utilisées pour un apprentissage polynomiale, et produire de classificateurs polynomiale.

Autre chose qui est magnifique chez le SVM,c'est que son pouvoir d'apprentissage est indépendante du dimension de l'espace du features, il mesure la complexité de l'hypothèse basé sur la marge qui sépare les données, non le nombre des features. Ceci signifie qu'on peut généraliser, si notre données est séparable avec une grande fonction de marge, même en présence de plusieurs features.

La marge suggère aussi d'avoir une heuristique pour sélectionner des valeurs exactes de traitement pour l'apprenti (comme le kernel avec le réseaux RBF). le meilleurs paramétrage, est celui qui produit l'hypothèse avec le plus petit VC-Dimension. Ceci permet d'automatiser les paramètres sans une perte de la cross-valisation.

5.2.2 implémentation et optimisation

On utilise le SVM dans les problèmes de classification binaires, or dans notre cas on a 7 catégories. Pour ce fait , il y a deux approches pour remédier à cette problématique :

Un contre tout le monde

Dans cette approche, on construit k classificateurs binaires séparés pour la classification k-class. Le classificateur binaire m-th est formé en utilisant les données de la m-ième classe comme exemples positifs et les classes k-1 restantes comme négatives exemples.

Pendant le test, l'étiquette de classe est déterminée par le classificateur binaire donne la valeur de sortie maximale. Un problème majeur de l'approche un versus repos est l'ensemble d'entraînement déséquilibré. Supposons que toutes les classes ont une taille égale d'entraînement exemples, le ratio d'exemples positifs à négatifs dans chaque classificateur est $1/(1-k)$. Dans ce cas, la symétrie du problème original est perdue.

Un contre Un

Une autre approche classique pour la classification multi-classe est le Un contre Un (1V1) ou

une décomposition par paires . Il évalue tous les classificateurs par paires possibles et ainsi induit $k * (k-1)/2$ classificateurs binaires individuels. Appliquer chaque classificateur à un test exemple donnerait un vote à la classe gagnante.

Un exemple de test est libellé pour la classe avec le plus de votes. La taille des classificateurs créés par le un contre un approche est beaucoup plus grande que celle de l'approche un versus repos.

Cependant, la taille de QP dans chaque classificateur est plus petit, ce qui permet de s'entraîner rapidement. De plus, par rapport à l'approche un contre le repos, la méthode un-contre-un est plus symétrique. Platt et al. a amélioré l'approche un-contre-un et proposé une méthode appelée SVM Directed Acyclic Graph SVM (DAGSVM) qui forme un arbre structure pour faciliter la phase de test. En conséquence, il ne faut que k - 1 individu évaluations pour décider de l'étiquette d'un exemple de test.

J'ai essayé les deux méthode, avec l'utilisation du kernel gaussiens pour les deux cas, et aussi en variant la valeur de la variables gamma, et j'ai conclu après les testes sur les deux modèles que le (Un contre Un) et mieux adapté pour notre cas.
Puisque qu'on a 7 catégories, le nombre de modèles utiliser est 21, chaque modèle sert à classifier la description entre deux classes, et c'est sûr qu'il va y avoir un résultat .

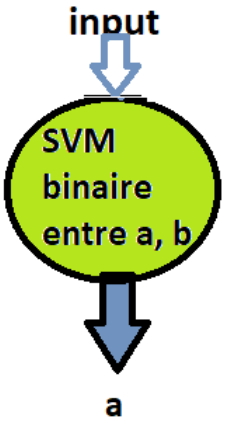


FIG. 5.3: Représentation d'un modèle binaire SVM

L'algorithme utilisé consiste à faire les prédictions dans chaque modèle qu'on a , et puis après en regroupe le résultat final dans un vecteur, qui contient lui aussi les valeurs d'apparition d'une

classe dans une prédiction, en le résultat final, est la classe qui est apparue le plus, dans le vecteur .

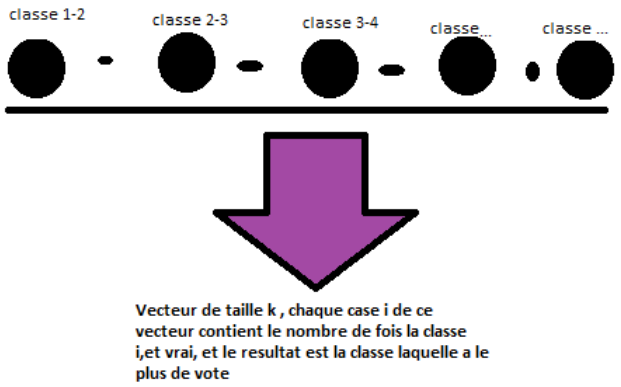


FIG. 5.4: Représentation globale du modèle

L'état des pertes sur les variables de décision :

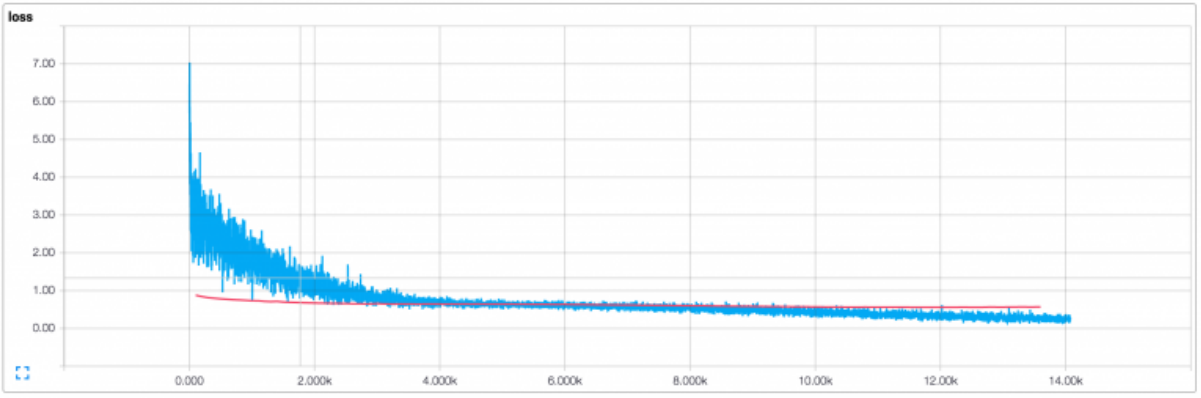


FIG. 5.5: Graphe qui représente les données perdues, du multiClass-SVM.

l'état des précision qui est égale à 0,65:

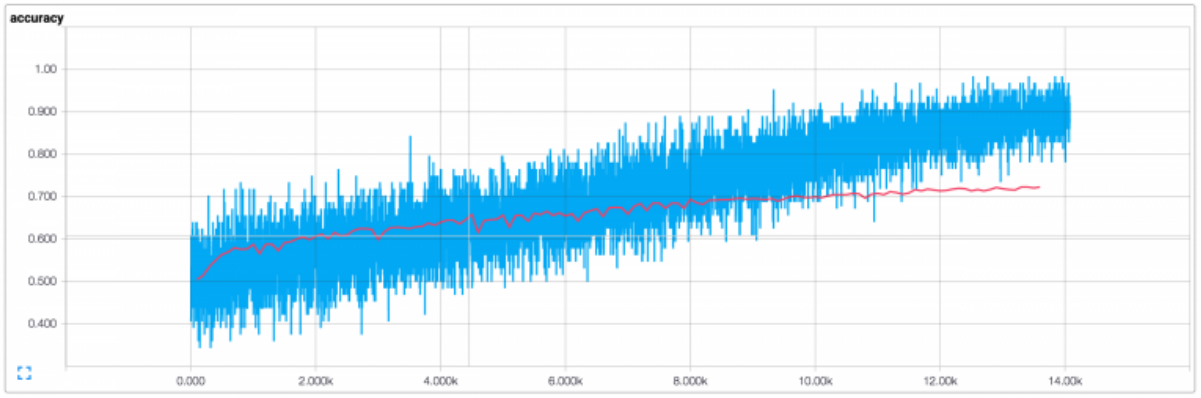


FIG. 5.6: Graphe qui représente la précision, du multiClass-SVM.

Chapitre 6

Sprint 3 : Amélioration du CNN avec word2vec

6.1 Le Backlog du Sprint 4

La figure 6.1, décrit le backlog du sprint numéro 3.

Story	**Priorité2**	**Effort ou chargen**
En tant qu'utilisateur, je souhaite pouvoir visiter les différents contenus par catégories,et de les sélectionner selon mon besoin.	M	22

FIG. 6.1: Le Backlog du Sprint 3

6.2 Conception

6.2.1 Qu'est ce qu'un word2vec ?

Les systèmes de traitement de l'image et du son fonctionnent avec des jeux de données riches et de grande dimension codés comme vecteurs des intensités de pixels brutes individuelles pour les données d'image, ou par exemple des coefficients de densité spectrale de puissance pour les données audio. Pour les tâches telles que la reconnaissance d'objet ou de reconnaissance vocale,

nous savons que toutes les informations nécessaires à l'exécution de la tâche sont codées dans les données (car les humains peuvent effectuer ces tâches à partir des données brutes).

Cependant, les systèmes de traitement du langage naturel traitent traditionnellement les mots comme des symboles atomiques discrets, et par conséquent, le terme 'chat' peut être représenté par Id537 'chien' Id143. Ces codages sont arbitraires et ne fournissent aucune information utile au système concernant les relations qui peuvent exister entre les symboles individuels. Cela signifie que le modèle peut tirer parti très peu de ce qu'il a appris sur les 'chats' lorsqu'il traite des données sur les 'chiens' (tels qu'ils sont à la fois des animaux, des quadrupèdes, des animaux domestiques, etc...). Le fait de représenter les mots comme des identificateurs uniques et discrets entraîne en outre un manque de données, et signifie généralement que nous aurons besoin de plus de données pour réussir la formation de modèles statistiques. L'utilisation de représentations vectorielles peut surmonter certains de ces obstacles.

Les modèles d'espace vectoriel (VSM) représentent (incorporent) des mots dans un espace vectoriel continu où des mots sémantiquement similaires sont mappés à des points voisins ('sont imbriqués les uns près des autres'). Les VSM ont une histoire longue et riche en PNL, mais toutes les méthodes dépendent d'une manière ou d'une autre de l' hypothèse distributionnelle , qui stipule que les mots qui apparaissent dans les mêmes contextes partagent une signification sémantique. Les différentes approches qui tirent parti de ce principe peuvent être divisées en deux catégories: les méthodes fondées sur le nombre (par exemple, l' analyse sémantique latente) et les méthodes prédictives (par exemple, les modèles de langage probabiliste neuronal).

Word2vec est un modèle prédictif particulièrement efficace sur le plan informatique pour l'apprentissage des plongées de mots à partir de texte brut. Il se présente sous deux formes: le modèle du sac continu de mots (CBOW) et le modèle de Skip-Gram (sections 3.1 et 3.2 de Mikolov et al.). Algorithmiquement, ces modèles sont similaires, sauf que CBOW prédit les mots cibles (par exemple 'mat') à partir des mots de contexte source ('le chat s'assoit sur le'), tandis que le skip-gram fait l'inverse et prédit les mots de contexte source de la cible mots. Cette inversion peut sembler un choix arbitraire, mais statistiquement elle a pour effet que le CBOW puise lire une grande partie de l'information distributive (en traitant un contexte entier comme une observation). Pour la plupart, cela s'avère être une chose utile pour les jeux de données plus petits. Cependant, skip-gram traite chaque paire contexte-cible comme une nouvelle observation, ce qui a tendance à être meilleur lorsque nous avons des ensembles de données plus volumineux.

6.2.2 Utilisation

Ce qui m'a motivé d'utiliser le word2vect est le problème de pertes de données au niveau de l'entrée du Convolutinal Neural Network du premier algorithme, et aussi le plus qui va ajouter l'approche sémantique à la precision de l'algorithme.
Avant l'algorithme CNN traité en input une taille de données qui est égale à 30, cette taille constante ne permettait pas de représenter la vrai valeur d'entrée, ce qui parfois déclenche une déviation sur la vrai identité de la description.

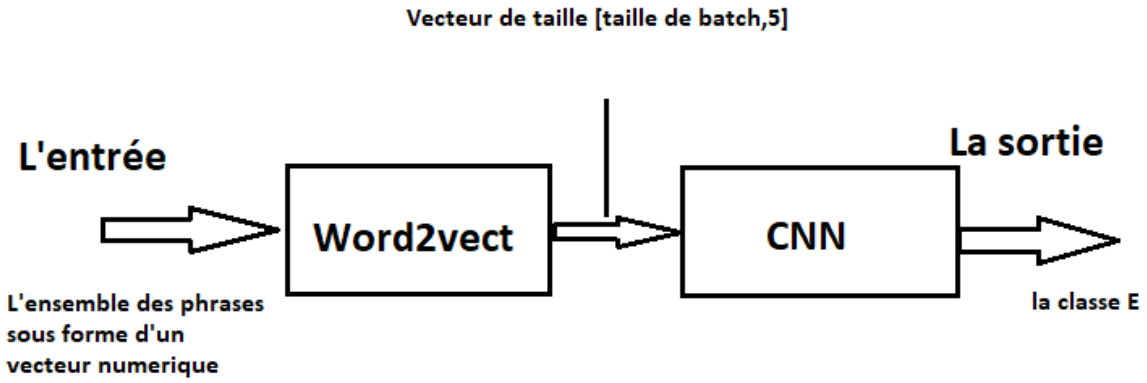


FIG. 6.2: Modélisation du modèle avec word2vect et cnn.

J'ai utilisé une taille de vecteur de word2vect égal à 5, parce que premièrement, mon corpus de données n'est pas assez grand, alors je vais économiser la mémoire, et deuxièmement pour minimiser la pertes de données au niveau de la fonction softmax.
La precision finale est presque égale à 80%, c'est bien meilleur que les deux approches discutées au paravent.

L'état des pertes sur les variables de décision :

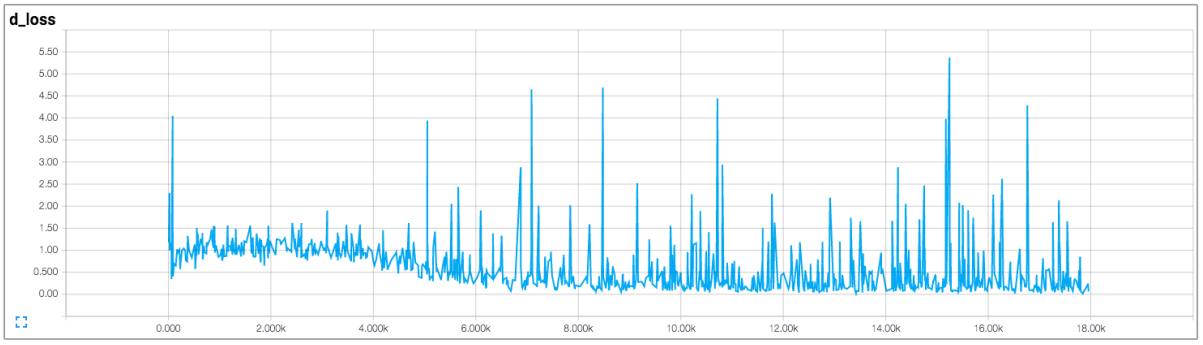


FIG. 6.3: Graphe qui représente les données perdues, du cnn avec le w2v.

l'état des précision qui est égale à 0,75:

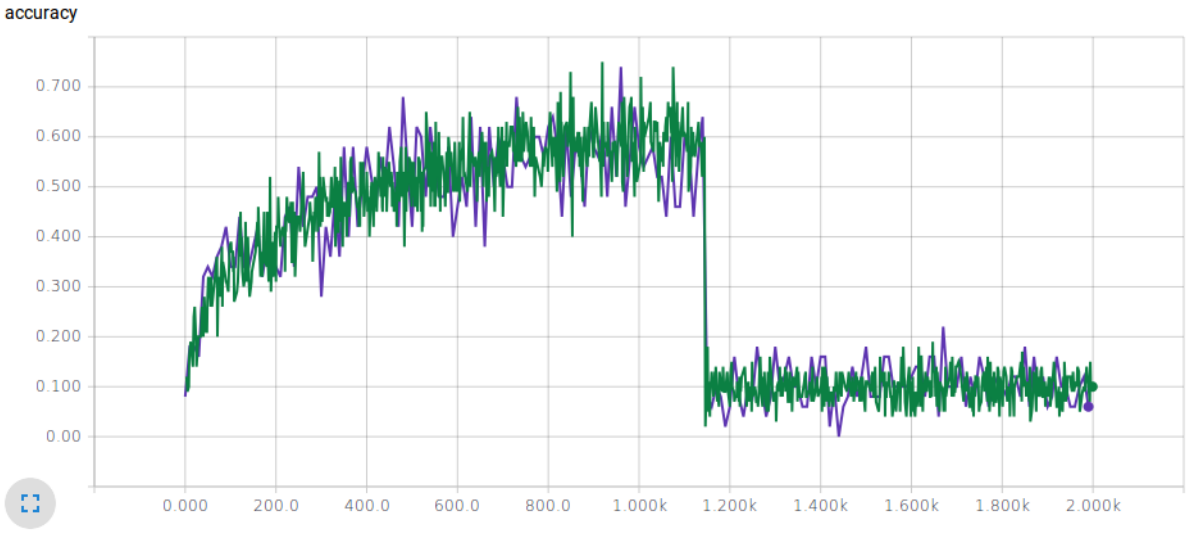


FIG. 6.4: Graphe qui représente la précision,du cnn avec le w2v.

Conclusion & Perspective

Pour conclure, j'ai effectué mon stage de deuxième année de cycle d'ingénierie logiciel et intégration des systèmes informatique en temps que Data Scientist pour l'application mobile Nacharat au sein de l'entreprise SanadTech. Lors de ce stage de 3 mois, j'ai pu mettre en pratique mes connaissances théoriques acquises durant ma formation et aussi ça m'a permis de m'introduire dans une nouvelle route, celle du big data et machine learning.

Pendant cette expérience, j'ai tous d'abord traité les données,en travaillant sur leur architecture, ce qui m'a permis de les transformer en des matrices numériques. Puis après j'ai commencé par aborder le problème de classification de type deep learning en utilisant le Réseau neuronal convolutif, ce qui m'a donné une precision en dessus de la moyenne, après j'ai changé l'approche du deep learning par celle du machine learning, en utilisant le Support vector machine avec un kernel gaussien. Il m'a donné un résultat satisfaisant mais pas assez convaincant, et à finalement, j'ai repris le premier algorithme développé, et j'ai travaillé à augmenter sa precision en utilisant le réseaux neuronal word2vec dans la phase d'entrée ce qui permet de minimiser la perte des données et aussi pour données à l'algorithme la capacité d'apprendre la sémantique des mots, afin d'avoir une precision bien meilleur.

Prochainement, en compte rassembler les deux approches en un seul algorithme de classification, ce dernier va contenir une seule entrée,alors que la première couche va contenir deux parties indépendantes le CNN plus le Word2vec et le SVM . La deuxième couche va ressembler les résultats obtenus par la première couche en un vecteur qu'on va l'analyser par la suite par la méthode de Bayes, pour enfin avoir un résultat avec un taux de precision très élevé.

webographie

The Elements of Statistical Learning - Stanford University

https://web.stanford.edu/ hastie/Papers/ESLII.pdf

Pattern Recognition and Machine Learning

http://users.isr.ist.utl.pt/ wurmd/ Livros/school/Bishop%20-%20Pattern%20Recognition %20And %20Machine%20Learning%20-%20Springer%20%202006.pdf

machine learning a probabilistic perspective

https://ai.google/research/pubs/pub38136

Machine Learning A Probabilistic Approach - Semantic Scholar

https://pdfs.semanticscholar.org/7bc7/54bc548f32b9ac53df67e3171e8e4df66d15.pdf

hands on machine learning with scikit learn and tensorflow concepts tools and techniques to build intelligent

http://shop.oreilly.com/product/0636920052289.do

Getting Started with TensorFlow

https://www.tensorflow.org/versions/r1.1/getstarted/getstarted

machine learning with tensorflow

https://www.amazon.com/Machine-Learning-TensorFlow-Nishant-Shukla/dp/1617293873

Convolutional Neural Networks for Sentence Classification

Yoon Kim
New York University
yhk255@nyu.edu

Annexe A

CNN

Abstract

We report on a series of experiments with convolutional neural networks (CNN) trained on top of pre-trained word vectors for sentence-level classification tasks. We show that a simple CNN with little hyperparameter tuning and static vectors achieves excellent results on multiple benchmarks. Learning task-specific vectors through fine-tuning offers further gains in performance. We additionally propose a simple modification to the architecture to allow for the use of both task-specific and static vectors. The CNN models discussed herein improve upon the state of the art on 4 out of 7 tasks, which include sentiment analysis and question classification.

1 Introduction

Deep learning models have achieved remarkable results in computer vision (Krizhevsky et al., 2012) and speech recognition (Graves et al., 2013) in recent years. Within natural language processing, much of the work with deep learning methods has involved learning word vector representations through neural language models (Bengio et al., 2003; Yih et al., 2011; Mikolov et al., 2013) and performing composition over the learned word vectors for classification (Collobert et al., 2011). Word vectors, wherein words are projected from a sparse, 1-of-V encoding (here V is the vocabulary size) onto a lower dimensional vector space via a hidden layer, are essentially feature extractors that encode semantic features of words in their dimensions. In such dense representations, semantically close words are likewise close—in euclidean or cosine distance—in the lower dimensional vector space.

Convolutional neural networks (CNN) utilize layers with convolving filters that are applied to local features (LeCun et al., 1998). Originally invented for computer vision, CNN models have subsequently been shown to be effective for NLP and have achieved excellent results in semantic parsing (Yih et al., 2014), search query retrieval (Shen et al., 2014), sentence modeling (Kalchbrenner et al., 2014), and other traditional NLP tasks (Collobert et al., 2011).

In the present work, we train a simple CNN with one layer of convolution on top of word vectors obtained from an unsupervised neural language model. These vectors were trained by Mikolov et al. (2013) on 100 billion words of Google News, and are publicly available.[1] We initially keep the word vectors static and learn only the other parameters of the model. Despite little tuning of hyperparameters, this simple model achieves excellent results on multiple benchmarks, suggesting that the pre-trained vectors are 'universal' feature extractors that can be utilized for various classification tasks. Learning task-specific vectors through fine-tuning results in further improvements. We finally describe a simple modification to the architecture to allow for the use of both pre-trained and task-specific vectors by having multiple channels.

Our work is philosophically similar to Razavian et al. (2014) which showed that for image classification, feature extractors obtained from a pre-trained deep learning model perform well on a variety of tasks—including tasks that are very different from the original task for which the feature extractors were trained.

2 Model

The model architecture, shown in figure 1, is a slight variant of the CNN architecture of Collobert et al. (2011). Let $\mathbf{x}_i \in \mathbb{R}^k$ be the k-dimensional word vector corresponding to the i-th word in the sentence. A sentence of length n (padded where

[1] https://code.google.com/p/word2vec/

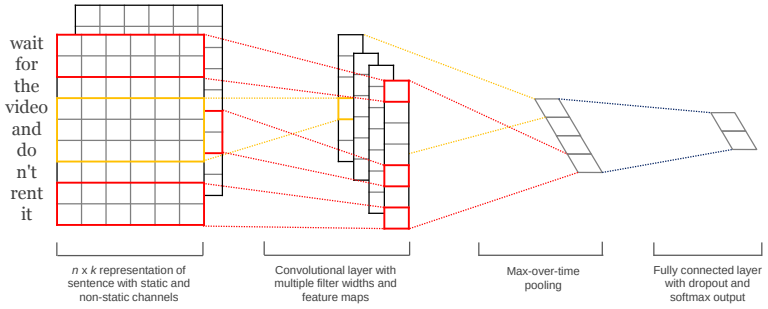


Figure 1: Model architecture with two channels for an example sentence.

necessary) is represented as

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n, \quad (1)$$

where $\oplus$ is the concatenation operator. In general, let $\mathbf{x}_{i:i+j}$ refer to the concatenation of words $\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+j}$. A convolution operation involves a *filter* $\mathbf{w} \in \mathbb{R}^{hk}$, which is applied to a window of h words to produce a new feature. For example, a feature c_i is generated from a window of words $\mathbf{x}_{i:i+h-1}$ by

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b). \quad (2)$$

Here $b \in \mathbb{R}$ is a bias term and f is a non-linear function such as the hyperbolic tangent. This filter is applied to each possible window of words in the sentence $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$ to produce a *feature map*

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}], \quad (3)$$

with $\mathbf{c} \in \mathbb{R}^{n-h+1}$. We then apply a max-over-time pooling operation (Collobert et al., 2011) over the feature map and take the maximum value $\hat{c} = \max\{\mathbf{c}\}$ as the feature corresponding to this particular filter. The idea is to capture the most important feature—one with the highest value—for each feature map. This pooling scheme naturally deals with variable sentence lengths.

We have described the process by which *one* feature is extracted from *one* filter. The model uses multiple filters (with varying window sizes) to obtain multiple features. These features form the penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over labels.

In one of the model variants, we experiment with having two 'channels' of word vectors—one that is kept static throughout training and one that is fine-tuned via backpropagation (section 3.2).[2] In the multichannel architecture, illustrated in figure 1, each filter is applied to both channels and the results are added to calculate c_i in equation (2). The model is otherwise equivalent to the single channel architecture.

2.1 Regularization

For regularization we employ dropout on the penultimate layer with a constraint on l_2-norms of the weight vectors (Hinton et al., 2012). Dropout prevents co-adaptation of hidden units by randomly dropping out—i.e., setting to zero—a proportion p of the hidden units during foward-backpropagation. That is, given the penultimate layer $\mathbf{z} = [\hat{c}_1, \ldots, \hat{c}_m]$ (note that here we have m filters), instead of using

$$y = \mathbf{w} \cdot \mathbf{z} + b \quad (4)$$

for output unit y in forward propagation, dropout uses

$$y = \mathbf{w} \cdot (\mathbf{z} \circ \mathbf{r}) + b, \quad (5)$$

where $\circ$ is the element-wise multiplication operator and $\mathbf{r} \in \mathbb{R}^m$ is a 'masking' vector of Bernoulli random variables with probability p of being 1. Gradients are backpropagated only through the unmasked units. At test time, the learned weight vectors are scaled by p such that $\hat{\mathbf{w}} = p\mathbf{w}$, and $\hat{\mathbf{w}}$ is used (without dropout) to score unseen sentences. We additionally constrain l_2-norms of the weight vectors by rescaling $\mathbf{w}$ to have $||\mathbf{w}||_2 = s$ whenever $||\mathbf{w}||_2 > s$ after a gradient descent step.

[2] We employ language from computer vision where a color image has red, green, and blue channels.

Data	c	l	N	$\|V\|$	$\|V_{pre}\|$	*Test*
MR	2	20	10662	18765	16448	CV
SST-1	5	18	11855	17836	16262	2210
SST-2	2	19	9613	16185	14838	1821
Subj	2	23	10000	21323	17913	CV
TREC	6	10	5952	9592	9125	500
CR	2	19	3775	5340	5046	CV
MPQA	2	3	10606	6246	6083	CV

Table 1: Summary statistics for the datasets after tokenization. c: Number of target classes. l: Average sentence length. N: Dataset size. $|V|$: Vocabulary size. $|V_{pre}|$: Number of words present in the set of pre-trained word vectors. *Test*: Test set size (CV means there was no standard train/test split and thus 10-fold CV was used).

3 Datasets and Experimental Setup

We test our model on various benchmarks. Summary statistics of the datasets are in table 1.

- **MR**: Movie reviews with one sentence per review. Classification involves detecting positive/negative reviews (Pang and Lee, 2005).[3]
- **SST-1**: Stanford Sentiment Treebank—an extension of MR but with train/dev/test splits provided and fine-grained labels (very positive, positive, neutral, negative, very negative), re-labeled by Socher et al. (2013).[4]
- **SST-2**: Same as SST-1 but with neutral reviews removed and binary labels.
- **Subj**: Subjectivity dataset where the task is to classify a sentence as being subjective or objective (Pang and Lee, 2004).
- **TREC**: TREC question dataset—task involves classifying a question into 6 question types (whether the question is about person, location, numeric information, etc.) (Li and Roth, 2002).[5]
- **CR**: Customer reviews of various products (cameras, MP3s etc.). Task is to predict positive/negative reviews (Hu and Liu, 2004).[6]
- **MPQA**: Opinion polarity detection subtask of the MPQA dataset (Wiebe et al., 2005).[7]

[3] https://www.cs.cornell.edu/people/pabo/movie-review-data/

[4] http://nlp.stanford.edu/sentiment/ Data is actually provided at the phrase-level and hence we train the model on both phrases and sentences but only score on sentences at test time, as in Socher et al. (2013), Kalchbrenner et al. (2014), and Le and Mikolov (2014). Thus the training set is an order of magnitude larger than listed in table 1.

[5] http://cogcomp.cs.illinois.edu/Data/QA/QC/

[6] http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

[7] http://www.cs.pitt.edu/mpqa/

3.1 Hyperparameters and Training

For all datasets we use: rectified linear units, filter windows (h) of 3, 4, 5 with 100 feature maps each, dropout rate (p) of 0.5, l_2 constraint (s) of 3, and mini-batch size of 50. These values were chosen via a grid search on the SST-2 dev set.

We do not otherwise perform any dataset-specific tuning other than early stopping on dev sets. For datasets without a standard dev set we randomly select 10% of the training data as the dev set. Training is done through stochastic gradient descent over shuffled mini-batches with the Adadelta update rule (Zeiler, 2012).

3.2 Pre-trained Word Vectors

Initializing word vectors with those obtained from an unsupervised neural language model is a popular method to improve performance in the absence of a large supervised training set (Collobert et al., 2011; Socher et al., 2011; Iyyer et al., 2014). We use the publicly available `word2vec` vectors that were trained on 100 billion words from Google News. The vectors have dimensionality of 300 and were trained using the continuous bag-of-words architecture (Mikolov et al., 2013). Words not present in the set of pre-trained words are initialized randomly.

3.3 Model Variations

We experiment with several variants of the model.

- **CNN-rand**: Our baseline model where all words are randomly initialized and then modified during training.
- **CNN-static**: A model with pre-trained vectors from `word2vec`. All words—including the unknown ones that are randomly initialized—are kept static and only the other parameters of the model are learned.
- **CNN-non-static**: Same as above but the pre-trained vectors are fine-tuned for each task.
- **CNN-multichannel**: A model with two sets of word vectors. Each set of vectors is treated as a 'channel' and each filter is applied

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Table 2: Results of our CNN models against other methods. **RAE**: Recursive Autoencoders with pre-trained word vectors from Wikipedia (Socher et al., 2011). **MV-RNN**: Matrix-Vector Recursive Neural Network with parse trees (Socher et al., 2012). **RNTN**: Recursive Neural Tensor Network with tensor-based feature function and parse trees (Socher et al., 2013). **DCNN**: Dynamic Convolutional Neural Network with k-max pooling (Kalchbrenner et al., 2014). **Paragraph-Vec**: Logistic regression on top of paragraph vectors (Le and Mikolov, 2014). **CCAE**: Combinatorial Category Autoencoders with combinatorial category grammar operators (Hermann and Blunsom, 2013). **Sent-Parser**: Sentiment analysis-specific parser (Dong et al., 2014). **NBSVM, MNB**: Naive Bayes SVM and Multinomial Naive Bayes with uni-bigrams from Wang and Manning (2012). **G-Dropout, F-Dropout**: Gaussian Dropout and Fast Dropout from Wang and Manning (2013). **Tree-CRF**: Dependency tree with Conditional Random Fields (Nakagawa et al., 2010). **CRF-PR**: Conditional Random Fields with Posterior Regularization (Yang and Cardie, 2014). **SVM_S**: SVM with uni-bi-trigrams, wh word, head word, POS, parser, hypernyms, and 60 hand-coded rules as features from Silva et al. (2011).

to both channels, but gradients are back-propagated only through one of the channels. Hence the model is able to fine-tune one set of vectors while keeping the other static. Both channels are initialized with `word2vec`.

In order to disentangle the effect of the above variations versus other random factors, we eliminate other sources of randomness—CV-fold assignment, initialization of unknown word vectors, initialization of CNN parameters—by keeping them uniform within each dataset.

4 Results and Discussion

Results of our models against other methods are listed in table 2. Our baseline model with all randomly initialized words (CNN-rand) does not perform well on its own. While we had expected performance gains through the use of pre-trained vectors, we were surprised at the magnitude of the gains. Even a simple model with static vectors (CNN-static) performs remarkably well, giving competitive results against the more sophisticated deep learning models that utilize complex pooling schemes (Kalchbrenner et al., 2014) or require parse trees to be computed beforehand (Socher et al., 2013). These results suggest that the pre-trained vectors are good, 'universal' feature extractors and can be utilized across datasets. Fine-tuning the pre-trained vectors for each task gives still further improvements (CNN-non-static).

4.1 Multichannel vs. Single Channel Models

We had initially hoped that the multichannel architecture would prevent overfitting (by ensuring that the learned vectors do not deviate too far from the original values) and thus work better than the single channel model, especially on smaller datasets. The results, however, are mixed, and further work on regularizing the fine-tuning process is warranted. For instance, instead of using an additional channel for the non-static portion, one could maintain a single channel but employ extra dimensions that are allowed to be modified during training.

	Most Similar Words for	
	Static Channel	Non-static Channel
bad	*good*	*terrible*
	terrible	*horrible*
	horrible	*lousy*
	lousy	*stupid*
good	*great*	*nice*
	bad	*decent*
	terrific	*solid*
	decent	*terrific*
n't	*os*	*not*
	ca	*never*
	ireland	*nothing*
	wo	*neither*
!	*2,500*	*2,500*
	entire	*lush*
	jez	*beautiful*
	changer	*terrific*
,	*decasia*	*but*
	abysmally	*dragon*
	demise	*a*
	valiant	*and*

Table 3: Top 4 neighboring words—based on cosine similarity—for vectors in the static channel (left) and fine-tuned vectors in the non-static channel (right) from the multichannel model on the SST-2 dataset after training.

4.2 Static vs. Non-static Representations

As is the case with the single channel non-static model, the multichannel model is able to fine-tune the non-static channel to make it more specific to the task-at-hand. For example, *good* is most similar to *bad* in `word2vec`, presumably because they are (almost) syntactically equivalent. But for vectors in the non-static channel that were fine-tuned on the SST-2 dataset, this is no longer the case (table 3). Similarly, *good* is arguably closer to *nice* than it is to *great* for expressing sentiment, and this is indeed reflected in the learned vectors.

For (randomly initialized) tokens not in the set of pre-trained vectors, fine-tuning allows them to learn more meaningful representations: the network learns that exclamation marks are associated with effusive expressions and that commas are conjunctive (table 3).

4.3 Further Observations

We report on some further experiments and observations:

- Kalchbrenner et al. (2014) report much worse results with a CNN that has essentially the same architecture as our single channel model. For example, their Max-TDNN (Time Delay Neural Network) with randomly initialized words obtains 37.4% on the SST-1 dataset, compared to 45.0% for our model. We attribute such discrepancy to our CNN having much more capacity (multiple filter widths and feature maps).
- Dropout proved to be such a good regularizer that it was fine to use a larger than necessary network and simply let dropout regularize it. Dropout consistently added 2%–4% relative performance.
- When randomly initializing words not in `word2vec`, we obtained slight improvements by sampling each dimension from $U[-a, a]$ where a was chosen such that the randomly initialized vectors have the same variance as the pre-trained ones. It would be interesting to see if employing more sophisticated methods to mirror the distribution of pre-trained vectors in the initialization process gives further improvements.
- We briefly experimented with another set of publicly available word vectors trained by Collobert et al. (2011) on Wikipedia,[8] and found that `word2vec` gave far superior performance. It is not clear whether this is due to Mikolov et al. (2013)'s architecture or the 100 billion word Google News dataset.
- Adadelta (Zeiler, 2012) gave similar results to Adagrad (Duchi et al., 2011) but required fewer epochs.

5 Conclusion

In the present work we have described a series of experiments with convolutional neural networks built on top of `word2vec`. Despite little tuning of hyperparameters, a simple CNN with one layer of convolution performs remarkably well. Our results add to the well-established evidence that unsupervised pre-training of word vectors is an important ingredient in deep learning for NLP.

Acknowledgments

We would like to thank Yann LeCun and the anonymous reviewers for their helpful feedback and suggestions.

[8] http://ronan.collobert.com/senna/

References

Y. Bengio, R. Ducharme, P. Vincent. 2003. Neural Probabilitistic Language Model. *Journal of Machine Learning Research* 3:1137–1155.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuglu, P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research 12:2493–2537*.

J. Duchi, E. Hazan, Y. Singer. 2011 Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research,* 12:2121–2159.

L. Dong, F. Wei, S. Liu, M. Zhou, K. Xu. 2014. A Statistical Parsing Framework for Sentiment Classification. *CoRR, abs/1401.6330.*

A. Graves, A. Mohamed, G. Hinton. 2013. Speech recognition with deep recurrent neural networks. *In Proceedings of ICASSP 2013.*

G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR, abs/1207.0580.*

K. Hermann, P. Blunsom. 2013. The Role of Syntax in Vector Space Models of Compositional Semantics. *In Proceedings of ACL 2013.*

M. Hu, B. Liu. 2004. Mining and Summarizing Customer Reviews. *In Proceedings of ACM SIGKDD 2004.*

M. Iyyer, P. Enns, J. Boyd-Graber, P. Resnik 2014. Political Ideology Detection Using Recursive Neural Networks. *In Proceedings of ACL 2014.*

N. Kalchbrenner, E. Grefenstette, P. Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. *In Proceedings of ACL 2014.*

A. Krizhevsky, I. Sutskever, G. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *In Proceedings of NIPS 2012.*

Q. Le, T. Mikolov. 2014. Distributed Represenations of Sentences and Documents. *In Proceedings of ICML 2014.*

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. 1998. Gradient-based learning applied to document recognition. *In Proceedings of the IEEE*, 86(11):2278–2324, November.

X. Li, D. Roth. 2002. Learning Question Classifiers. *In Proceedings of ACL 2002.*

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *In Proceedings of NIPS 2013.*

T. Nakagawa, K. Inui, S. Kurohashi. 2010. Dependency tree-based sentiment classification using CRFs with hidden variables. *In Proceedings of ACL 2010.*

B. Pang, L. Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. *In Proceedings of ACL 2004.*

B. Pang, L. Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *In Proceedings of ACL 2005.*

A.S. Razavian, H. Azizpour, J. Sullivan, S. Carlsson 2014. CNN Features off-the-shelf: an Astounding Baseline. *CoRR, abs/1403.6382.*

Y. Shen, X. He, J. Gao, L. Deng, G. Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. *In Proceedings of WWW 2014.*

J. Silva, L. Coheur, A. Mendes, A. Wichert. 2011. From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review*, 35(2):137–154.

R. Socher, J. Pennington, E. Huang, A. Ng, C. Manning. 2011. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. *In Proceedings of EMNLP 2011.*

R. Socher, B. Huval, C. Manning, A. Ng. 2012. Semantic Compositionality through Recursive Matrix-Vector Spaces. *In Proceedings of EMNLP 2012.*

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, C. Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. *In Proceedings of EMNLP 2013.*

J. Wiebe, T. Wilson, C. Cardie. 2005. Annotating Expressions of Opinions and Emotions in Language. *Language Resources and Evaluation*, 39(2-3): 165–210.

S. Wang, C. Manning. 2012. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. *In Proceedings of ACL 2012.*

S. Wang, C. Manning. 2013. Fast Dropout Training. *In Proceedings of ICML 2013.*

B. Yang, C. Cardie. 2014. Context-aware Learning for Sentence-level Sentiment Analysis with Posterior Regularization. *In Proceedings of ACL 2014.*

W. Yih, K. Toutanova, J. Platt, C. Meek. 2011. Learning Discriminative Projections for Text Similarity Measures. *Proceedings of the Fifteenth Conference on Computational Natural Language Learning, 247–256.*

W. Yih, X. He, C. Meek. 2014. Semantic Parsing for Single-Relation Question Answering. *In Proceedings of ACL 2014.*

M. Zeiler. 2012. Adadelta: An adaptive learning rate method. *CoRR, abs/1212.5701.*

Annexe B

CS229 Lecture notes

Andrew Ng

SVM

Part V
Support Vector Machines

This set of notes presents the Support Vector Machine (SVM) learning algorithm. SVMs are among the best (and many believe are indeed the best) "off-the-shelf" supervised learning algorithm. To tell the SVM story, we'll need to first talk about margins and the idea of separating data with a large "gap." Next, we'll talk about the optimal margin classifier, which will lead us into a digression on Lagrange duality. We'll also see kernels, which give a way to apply SVMs efficiently in very high dimensional (such as infinite-dimensional) feature spaces, and finally, we'll close off the story with the SMO algorithm, which gives an efficient implementation of SVMs.

1 Margins: Intuition

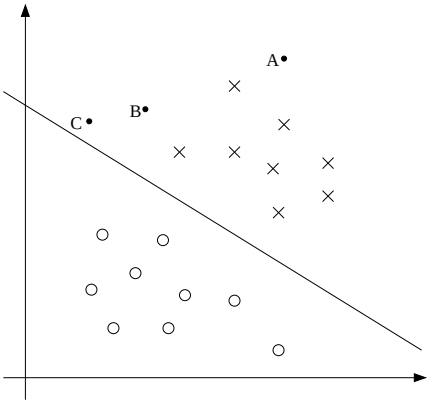
We'll start our story on SVMs by talking about margins. This section will give the intuitions about margins and about the "confidence" of our predictions; these ideas will be made formal in Section 3.

Consider logistic regression, where the probability $p(y = 1|x; \theta)$ is modeled by $h_\theta(x) = g(\theta^T x)$. We would then predict "1" on an input x if and only if $h_\theta(x) \geq 0.5$, or equivalently, if and only if $\theta^T x \geq 0$. Consider a positive training example ($y = 1$). The larger $\theta^T x$ is, the larger also is $h_\theta(x) = p(y = 1|x; w, b)$, and thus also the higher our degree of "confidence" that the label is 1. Thus, informally we can think of our prediction as being a very confident one that $y = 1$ if $\theta^T x \gg 0$. Similarly, we think of logistic regression as making a very confident prediction of $y = 0$, if $\theta^T x \ll 0$. Given a training set, again informally it seems that we'd have found a good fit to the training data if we can find θ so that $\theta^T x^{(i)} \gg 0$ whenever $y^{(i)} = 1$, and

2

$\theta^T x^{(i)} \ll 0$ whenever $y^{(i)} = 0$, since this would reflect a very confident (and correct) set of classifications for all the training examples. This seems to be a nice goal to aim for, and we'll soon formalize this idea using the notion of functional margins.

For a different type of intuition, consider the following figure, in which x's represent positive training examples, o's denote negative training examples, a decision boundary (this is the line given by the equation $\theta^T x = 0$, and is also called the **separating hyperplane**) is also shown, and three points have also been labeled A, B and C.



Notice that the point A is very far from the decision boundary. If we are asked to make a prediction for the value of y at A, it seems we should be quite confident that $y = 1$ there. Conversely, the point C is very close to the decision boundary, and while it's on the side of the decision boundary on which we would predict $y = 1$, it seems likely that just a small change to the decision boundary could easily have caused our prediction to be $y = 0$. Hence, we're much more confident about our prediction at A than at C. The point B lies in-between these two cases, and more broadly, we see that if a point is far from the separating hyperplane, then we may be significantly more confident in our predictions. Again, informally we think it'd be nice if, given a training set, we manage to find a decision boundary that allows us to make all correct and confident (meaning far from the decision boundary) predictions on the training examples. We'll formalize this later using the notion of geometric margins.

3

2 Notation

To make our discussion of SVMs easier, we'll first need to introduce a new notation for talking about classification. We will be considering a linear classifier for a binary classification problem with labels y and features x. From now, we'll use $y \in \{-1, 1\}$ (instead of $\{0, 1\}$) to denote the class labels. Also, rather than parameterizing our linear classifier with the vector θ, we will use parameters w, b, and write our classifier as

$$h_{w,b}(x) = g(w^T x + b).$$

Here, $g(z) = 1$ if $z \geq 0$, and $g(z) = -1$ otherwise. This "w, b" notation allows us to explicitly treat the intercept term b separately from the other parameters. (We also drop the convention we had previously of letting $x_0 = 1$ be an extra coordinate in the input feature vector.) Thus, b takes the role of what was previously θ_0, and w takes the role of $[\theta_1 \dots \theta_n]^T$.

Note also that, from our definition of g above, our classifier will directly predict either 1 or -1 (cf. the perceptron algorithm), without first going through the intermediate step of estimating the probability of y being 1 (which was what logistic regression did).

3 Functional and geometric margins

Let's formalize the notions of the functional and geometric margins. Given a training example $(x^{(i)}, y^{(i)})$, we define the **functional margin** of (w, b) with respect to the training example

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b).$$

Note that if $y^{(i)} = 1$, then for the functional margin to be large (i.e., for our prediction to be confident and correct), we need $w^T x + b$ to be a large positive number. Conversely, if $y^{(i)} = -1$, then for the functional margin to be large, we need $w^T x + b$ to be a large negative number. Moreover, if $y^{(i)}(w^T x + b) > 0$, then our prediction on this example is correct. (Check this yourself.) Hence, a large functional margin represents a confident and a correct prediction.

For a linear classifier with the choice of g given above (taking values in $\{-1, 1\}$), there's one property of the functional margin that makes it not a very good measure of confidence, however. Given our choice of g, we note that if we replace w with $2w$ and b with $2b$, then since $g(w^T x + b) = g(2w^T x + 2b)$,

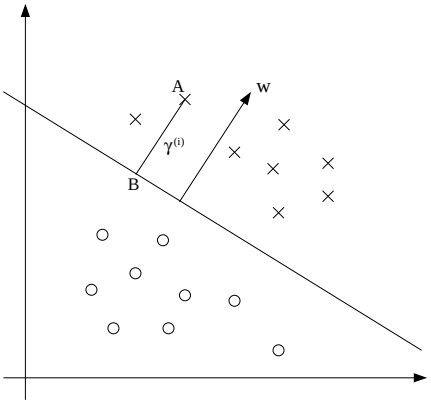
4

this would not change $h_{w,b}(x)$ at all. I.e., g, and hence also $h_{w,b}(x)$, depends only on the sign, but not on the magnitude, of $w^T x + b$. However, replacing (w, b) with $(2w, 2b)$ also results in multiplying our functional margin by a factor of 2. Thus, it seems that by exploiting our freedom to scale w and b, we can make the functional margin arbitrarily large without really changing anything meaningful. Intuitively, it might therefore make sense to impose some sort of normalization condition such as that $||w||_2 = 1$; i.e., we might replace (w, b) with $(w/||w||_2, b/||w||_2)$, and instead consider the functional margin of $(w/||w||_2, b/||w||_2)$. We'll come back to this later.

Given a training set $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$, we also define the function margin of (w, b) with respect to S to be the smallest of the functional margins of the individual training examples. Denoted by $\hat{\gamma}$, this can therefore be written:

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)}.$$

Next, let's talk about **geometric margins**. Consider the picture below:



The decision boundary corresponding to (w, b) is shown, along with the vector w. Note that w is orthogonal (at 90°) to the separating hyperplane. (You should convince yourself that this must be the case.) Consider the point at A, which represents the input $x^{(i)}$ of some training example with label $y^{(i)} = 1$. Its distance to the decision boundary, $\gamma^{(i)}$, is given by the line segment AB.

How can we find the value of $\gamma^{(i)}$? Well, $w/||w||$ is a unit-length vector pointing in the same direction as w. Since A represents $x^{(i)}$, we therefore

find that the point B is given by $x^{(i)} - \gamma^{(i)} \cdot w/||w||$. But this point lies on the decision boundary, and all points x on the decision boundary satisfy the equation $w^T x + b = 0$. Hence,

$$w^T \left(x^{(i)} - \gamma^{(i)} \frac{w}{||w||} \right) + b = 0.$$

Solving for $\gamma^{(i)}$ yields

$$\gamma^{(i)} = \frac{w^T x^{(i)} + b}{||w||} = \left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||}.$$

This was worked out for the case of a positive training example at A in the figure, where being on the "positive" side of the decision boundary is good. More generally, we define the geometric margin of (w, b) with respect to a training example $(x^{(i)}, y^{(i)})$ to be

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||} \right).$$

Note that if $||w|| = 1$, then the functional margin equals the geometric margin—this thus gives us a way of relating these two different notions of margin. Also, the geometric margin is invariant to rescaling of the parameters; i.e., if we replace w with $2w$ and b with $2b$, then the geometric margin does not change. This will in fact come in handy later. Specifically, because of this invariance to the scaling of the parameters, when trying to fit w and b to training data, we can impose an arbitrary scaling constraint on w without changing anything important; for instance, we can demand that $||w|| = 1$, or $|w_1| = 5$, or $|w_1 + b| + |w_2| = 2$, and any of these can be satisfied simply by rescaling w and b.

Finally, given a training set $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$, we also define the geometric margin of (w, b) with respect to S to be the smallest of the geometric margins on the individual training examples:

$$\gamma = \min_{i=1,\ldots,m} \gamma^{(i)}.$$

4 The optimal margin classifier

Given a training set, it seems from our previous discussion that a natural desideratum is to try to find a decision boundary that maximizes the (geometric) margin, since this would reflect a very confident set of predictions

6

on the training set and a good "fit" to the training data. Specifically, this will result in a classifier that separates the positive and the negative training examples with a "gap" (geometric margin).

For now, we will assume that we are given a training set that is linearly separable; i.e., that it is possible to separate the positive and negative examples using some separating hyperplane. How we we find the one that achieves the maximum geometric margin? We can pose the following optimization problem:

$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m \\ & ||w|| = 1. \end{aligned}$$

I.e., we want to maximize γ, subject to each training example having functional margin at least γ. The $||w|| = 1$ constraint moreover ensures that the functional margin equals to the geometric margin, so we are also guaranteed that all the geometric margins are at least γ. Thus, solving this problem will result in (w, b) with the largest possible geometric margin with respect to the training set.

If we could solve the optimization problem above, we'd be done. But the "$||w|| = 1$" constraint is a nasty (non-convex) one, and this problem certainly isn't in any format that we can plug into standard optimization software to solve. So, let's try transforming the problem into a nicer one. Consider:

$$\begin{aligned} \max_{\hat{\gamma},w,b} \quad & \frac{\hat{\gamma}}{||w||} \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \ldots, m \end{aligned}$$

Here, we're going to maximize $\hat{\gamma}/||w||$, subject to the functional margins all being at least $\hat{\gamma}$. Since the geometric and functional margins are related by $\gamma = \hat{\gamma}/||w|$, this will give us the answer we want. Moreover, we've gotten rid of the constraint $||w|| = 1$ that we didn't like. The downside is that we now have a nasty (again, non-convex) objective $\frac{\hat{\gamma}}{||w||}$ function; and, we still don't have any off-the-shelf software that can solve this form of an optimization problem.

Let's keep going. Recall our earlier discussion that we can add an arbitrary scaling constraint on w and b without changing anything. This is the key idea we'll use now. We will introduce the scaling constraint that the functional margin of w, b with respect to the training set must be 1:

$$\hat{\gamma} = 1.$$

Since multiplying w and b by some constant results in the functional margin being multiplied by that same constant, this is indeed a scaling constraint, and can be satisfied by rescaling w, b. Plugging this into our problem above, and noting that maximizing $\hat{\gamma}/||w|| = 1/||w||$ is the same thing as minimizing $||w||^2$, we now have the following optimization problem:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m \end{aligned}$$

We've now transformed the problem into a form that can be efficiently solved. The above is an optimization problem with a convex quadratic objective and only linear constraints. Its solution gives us the **optimal margin classifier**. This optimization problem can be solved using commercial quadratic programming (QP) code.[1]

While we could call the problem solved here, what we will instead do is make a digression to talk about Lagrange duality. This will lead us to our optimization problem's dual form, which will play a key role in allowing us to use kernels to get optimal margin classifiers to work efficiently in very high dimensional spaces. The dual form will also allow us to derive an efficient algorithm for solving the above optimization problem that will typically do much better than generic QP software.

5 Lagrange duality

Let's temporarily put aside SVMs and maximum margin classifiers, and talk about solving constrained optimization problems.

Consider a problem of the following form:

$$\begin{aligned} \min_{w} \quad & f(w) \\ \text{s.t.} \quad & h_i(w) = 0, \quad i = 1, \ldots, l. \end{aligned}$$

Some of you may recall how the method of Lagrange multipliers can be used to solve it. (Don't worry if you haven't seen it before.) In this method, we define the **Lagrangian** to be

$$\mathcal{L}(w, \beta) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

[1] You may be familiar with linear programming, which solves optimization problems that have linear objectives and linear constraints. QP software is also widely available, which allows convex quadratic objectives and linear constraints.

8

Here, the β_i's are called the **Lagrange multipliers**. We would then find and set $\mathcal{L}$'s partial derivatives to zero:

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0; \quad \frac{\partial \mathcal{L}}{\partial \beta_i} = 0,$$

and solve for w and β.

In this section, we will generalize this to constrained optimization problems in which we may have inequality as well as equality constraints. Due to time constraints, we won't really be able to do the theory of Lagrange duality justice in this class,[2] but we will give the main ideas and results, which we will then apply to our optimal margin classifier's optimization problem.

Consider the following, which we'll call the **primal** optimization problem:

$$\begin{aligned} \min_w \quad & f(w) \\ \text{s.t.} \quad & g_i(w) \leq 0, \quad i = 1, \ldots, k \\ & h_i(w) = 0, \quad i = 1, \ldots, l. \end{aligned}$$

To solve it, we start by defining the **generalized Lagrangian**

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w).$$

Here, the α_i's and β_i's are the Lagrange multipliers. Consider the quantity

$$\theta_{\mathcal{P}}(w) = \max_{\alpha, \beta \,:\, \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta).$$

Here, the "$\mathcal{P}$" subscript stands for "primal." Let some w be given. If w violates any of the primal constraints (i.e., if either $g_i(w) > 0$ or $h_i(w) \neq 0$ for some i), then you should be able to verify that

$$\begin{aligned} \theta_{\mathcal{P}}(w) &= \max_{\alpha, \beta \,:\, \alpha_i \geq 0} f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w) \qquad (1) \\ &= \infty. \qquad (2) \end{aligned}$$

Conversely, if the constraints are indeed satisfied for a particular value of w, then $\theta_{\mathcal{P}}(w) = f(w)$. Hence,

$$\theta_{\mathcal{P}}(w) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{otherwise.} \end{cases}$$

[2] Readers interested in learning more about this topic are encouraged to read, e.g., R. T. Rockarfeller (1970), Convex Analysis, Princeton University Press.

9

Thus, $\theta_{\mathcal{P}}$ takes the same value as the objective in our problem for all values of w that satisfies the primal constraints, and is positive infinity if the constraints are violated. Hence, if we consider the minimization problem

$$\min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta),$$

we see that it is the same problem (i.e., and has the same solutions as) our original, primal problem. For later use, we also define the optimal value of the objective to be $p^* = \min_w \theta_{\mathcal{P}}(w)$; we call this the **value** of the primal problem.

Now, let's look at a slightly different problem. We define

$$\theta_{\mathcal{D}}(\alpha, \beta) = \min_w \mathcal{L}(w, \alpha, \beta).$$

Here, the "$\mathcal{D}$" subscript stands for "dual." Note also that whereas in the definition of $\theta_{\mathcal{P}}$ we were optimizing (maximizing) with respect to α, β, here are are minimizing with respect to w.

We can now pose the **dual** optimization problem:

$$\max_{\alpha,\beta\,:\,\alpha_i \geq 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta).$$

This is exactly the same as our primal problem shown above, except that the order of the "max" and the "min" are now exchanged. We also define the optimal value of the dual problem's objective to be $d^* = \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \theta_{\mathcal{D}}(w)$.

How are the primal and the dual problems related? It can easily be shown that

$$d^* = \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta) \leq \min_w \max_{\alpha,\beta\,:\,\alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = p^*.$$

(You should convince yourself of this; this follows from the "max min" of a function always being less than or equal to the "min max.") However, under certain conditions, we will have

$$d^* = p^*,$$

so that we can solve the dual problem in lieu of the primal problem. Let's see what these conditions are.

Suppose f and the g_i's are convex,[3] and the h_i's are affine.[4] Suppose further that the constraints g_i are (strictly) feasible; this means that there exists some w so that $g_i(w) < 0$ for all i.

[3] When f has a Hessian, then it is convex if and only if the Hessian is positive semi-definite. For instance, $f(w) = w^T w$ is convex; similarly, all linear (and affine) functions are also convex. (A function f can also be convex without being differentiable, but we won't need those more general definitions of convexity here.)

[4] I.e., there exists a_i, b_i, so that $h_i(w) = a_i^T w + b_i$. "Affine" means the same thing as linear, except that we also allow the extra intercept term b_i.

Under our above assumptions, there must exist w^*, α^*, β^* so that w^* is the solution to the primal problem, α^*, β^* are the solution to the dual problem, and moreover $p^* = d^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$. Moreover, w^*, α^* and β^* satisfy the **Karush-Kuhn-Tucker (KKT) conditions**, which are as follows:

$$\begin{aligned}
\frac{\partial}{\partial w_i}\mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, n && (3)\\
\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*, \alpha^*, \beta^*) &= 0, \quad i = 1, \ldots, l && (4)\\
\alpha_i^* g_i(w^*) &= 0, \quad i = 1, \ldots, k && (5)\\
g_i(w^*) &\leq 0, \quad i = 1, \ldots, k && (6)\\
\alpha^* &\geq 0, \quad i = 1, \ldots, k && (7)
\end{aligned}$$

Moreover, if some w^*, α^*, β^* satisfy the KKT conditions, then it is also a solution to the primal and dual problems.

We draw attention to Equation (5), which is called the KKT **dual complementarity** condition. Specifically, it implies that if $\alpha_i^* > 0$, then $g_i(w^*) = 0$. (I.e., the "$g_i(w) \leq 0$" constraint is **active**, meaning it holds with equality rather than with inequality.) Later on, this will be key for showing that the SVM has only a small number of "support vectors"; the KKT dual complementarity condition will also give us our convergence test when we talk about the SMO algorithm.

6 Optimal margin classifiers

Previously, we posed the following (primal) optimization problem for finding the optimal margin classifier:

$$\begin{aligned}
\min_{\gamma, w, b} \quad & \frac{1}{2}||w||^2 \\
\text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m
\end{aligned}$$

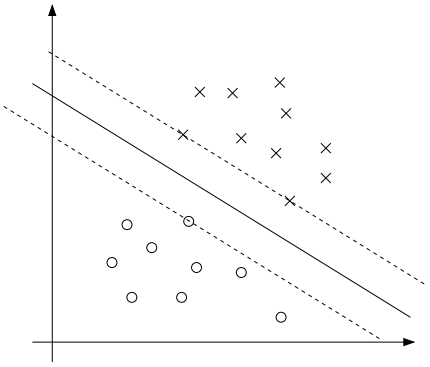
We can write the constraints as

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0.$$

We have one such constraint for each training example. Note that from the KKT dual complementarity condition, we will have $\alpha_i > 0$ only for the training examples that have functional margin exactly equal to one (i.e., the ones

11

corresponding to constraints that hold with equality, $g_i(w) = 0$). Consider the figure below, in which a maximum margin separating hyperplane is shown by the solid line.



The points with the smallest margins are exactly the ones closest to the decision boundary; here, these are the three points (one negative and two positive examples) that lie on the dashed lines parallel to the decision boundary. Thus, only three of the α_i's—namely, the ones corresponding to these three training examples—will be non-zero at the optimal solution to our optimization problem. These three points are called the **support vectors** in this problem. The fact that the number of support vectors can be much smaller than the size the training set will be useful later.

Let's move on. Looking ahead, as we develop the dual form of the problem, one key idea to watch out for is that we'll try to write our algorithm in terms of only the inner product $\langle x^{(i)}, x^{(j)} \rangle$ (think of this as $(x^{(i)})^T x^{(j)}$) between points in the input feature space. The fact that we can express our algorithm in terms of these inner products will be key when we apply the kernel trick.

When we construct the Lagrangian for our optimization problem we have:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(w^T x^{(i)} + b) - 1 \right]. \tag{8}$$

Note that there're only "α_i" but no "β_i" Lagrange multipliers, since the problem has only inequality constraints.

Let's find the dual form of the problem. To do so, we need to first minimize $\mathcal{L}(w, b, \alpha)$ with respect to w and b (for fixed α), to get $\theta_{\mathcal{D}}$, which

we'll do by setting the derivatives of $\mathcal{L}$ with respect to w and b to zero. We have:

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0$$

This implies that

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}. \tag{9}$$

As for the derivative with respect to b, we obtain

$$\frac{\partial}{\partial b} \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0. \tag{10}$$

If we take the definition of w in Equation (9) and plug that back into the Lagrangian (Equation 8), and simplify, we get

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - b \sum_{i=1}^{m} \alpha_i y^{(i)}.$$

But from Equation (10), the last term must be zero, so we obtain

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}.$$

Recall that we got to the equation above by minimizing $\mathcal{L}$ with respect to w and b. Putting this together with the constraints $\alpha_i \geq 0$ (that we always had) and the constraint (10), we obtain the following dual optimization problem:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

You should also be able to verify that the conditions required for $p^* = d^*$ and the KKT conditions (Equations 3–7) to hold are indeed satisfied in our optimization problem. Hence, we can solve the dual in lieu of solving the primal problem. Specifically, in the dual problem above, we have a maximization problem in which the parameters are the α_i's. We'll talk later

about the specific algorithm that we're going to use to solve the dual problem, but if we are indeed able to solve it (i.e., find the α's that maximize $W(\alpha)$ subject to the constraints), then we can use Equation (9) to go back and find the optimal w's as a function of the α's. Having found w^*, by considering the primal problem, it is also straightforward to find the optimal value for the intercept term b as

$$b^* = -\frac{\max_{i:y^{(i)}=-1} w^{*T} x^{(i)} + \min_{i:y^{(i)}=1} w^{*T} x^{(i)}}{2}. \tag{11}$$

(Check for yourself that this is correct.)

Before moving on, let's also take a more careful look at Equation (9), which gives the optimal value of w in terms of (the optimal value of) α. Suppose we've fit our model's parameters to a training set, and now wish to make a prediction at a new point input x. We would then calculate $w^T x + b$, and predict $y = 1$ if and only if this quantity is bigger than zero. But using (9), this quantity can also be written:

$$w^T x + b = \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b \tag{12}$$

$$= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. \tag{13}$$

Hence, if we've found the α_i's, in order to make a prediction, we have to calculate a quantity that depends only on the inner product between x and the points in the training set. Moreover, we saw earlier that the α_i's will all be zero except for the support vectors. Thus, many of the terms in the sum above will be zero, and we really need to find only the inner products between x and the support vectors (of which there is often only a small number) in order calculate (13) and make our prediction.

By examining the dual form of the optimization problem, we gained significant insight into the structure of the problem, and were also able to write the entire algorithm in terms of only inner products between input feature vectors. In the next section, we will exploit this property to apply the kernels to our classification problem. The resulting algorithm, **support vector machines**, will be able to efficiently learn in very high dimensional spaces.

7 Kernels

Back in our discussion of linear regression, we had a problem in which the input x was the living area of a house, and we considered performing regres-

sion using the features x, x^2 and x^3 (say) to obtain a cubic function. To distinguish between these two sets of variables, we'll call the "original" input value the input **attributes** of a problem (in this case, x, the living area). When that is mapped to some new set of quantities that are then passed to the learning algorithm, we'll call those new quantities the input **features**. (Unfortunately, different authors use different terms to describe these two things, but we'll try to use this terminology consistently in these notes.) We will also let ϕ denote the **feature mapping**, which maps from the attributes to the features. For instance, in our example, we had

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}.$$

Rather than applying SVMs using the original input attributes x, we may instead want to learn using some features $\phi(x)$. To do so, we simply need to go over our previous algorithm, and replace x everywhere in it with $\phi(x)$.

Since the algorithm can be written entirely in terms of the inner products $\langle x, z\rangle$, this means that we would replace all those inner products with $\langle \phi(x), \phi(z)\rangle$. Specificically, given a feature mapping ϕ, we define the corresponding **Kernel** to be

$$K(x, z) = \phi(x)^T \phi(z).$$

Then, everywhere we previously had $\langle x, z\rangle$ in our algorithm, we could simply replace it with $K(x, z)$, and our algorithm would now be learning using the features ϕ.

Now, given ϕ, we could easily compute $K(x, z)$ by finding $\phi(x)$ and $\phi(z)$ and taking their inner product. But what's more interesting is that often, $K(x, z)$ may be very inexpensive to calculate, even though $\phi(x)$ itself may be very expensive to calculate (perhaps because it is an extremely high dimensional vector). In such settings, by using in our algorithm an efficient way to calculate $K(x, z)$, we can get SVMs to learn in the high dimensional feature space space given by ϕ, but without ever having to explicitly find or represent vectors $\phi(x)$.

Let's see an example. Suppose $x, z \in \mathbb{R}^n$, and consider

$$K(x, z) = (x^T z)^2.$$

15

We can also write this as

$$\begin{aligned} K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

Thus, we see that $K(x,z) = \phi(x)^T\phi(z)$, where the feature mapping ϕ is given (shown here for the case of $n = 3$) by

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}.$$

Note that whereas calculating the high-dimensional $\phi(x)$ requires $O(n^2)$ time, finding $K(x,z)$ takes only $O(n)$ time—linear in the dimension of the input attributes.

For a related kernel, also consider

$$\begin{aligned} K(x,z) &= (x^T z + c)^2 \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n} (\sqrt{2c}x_i)(\sqrt{2c}z_i) + c^2. \end{aligned}$$

(Check this yourself.) This corresponds to the feature mapping (again shown

16

for $n = 3$)

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ \sqrt{2c}x_3 \\ c \end{bmatrix},$$

and the parameter c controls the relative weighting between the x_i (first order) and the x_ix_j (second order) terms.

More broadly, the kernel $K(x, z) = (x^T z + c)^d$ corresponds to a feature mapping to an $\binom{n+d}{d}$ feature space, corresponding of all monomials of the form $x_{i_1}x_{i_2}\ldots x_{i_k}$ that are up to order d. However, despite working in this $O(n^d)$-dimensional space, computing $K(x, z)$ still takes only $O(n)$ time, and hence we never need to explicitly represent feature vectors in this very high dimensional feature space.

Now, let's talk about a slightly different view of kernels. Intuitively, (and there are things wrong with this intuition, but nevermind), if $\phi(x)$ and $\phi(z)$ are close together, then we might expect $K(x, z) = \phi(x)^T\phi(z)$ to be large. Conversely, if $\phi(x)$ and $\phi(z)$ are far apart—say nearly orthogonal to each other—then $K(x, z) = \phi(x)^T\phi(z)$ will be small. So, we can think of $K(x, z)$ as some measurement of how similar are $\phi(x)$ and $\phi(z)$, or of how similar are x and z.

Given this intuition, suppose that for some learning problem that you're working on, you've come up with some function $K(x, z)$ that you think might be a reasonable measure of how similar x and z are. For instance, perhaps you chose

$$K(x, z) = \exp\left(-\frac{||x - z||^2}{2\sigma^2}\right).$$

This is a resonable measure of x and z's similarity, and is close to 1 when x and z are close, and near 0 when x and z are far apart. Can we use this definition of K as the kernel in an SVM? In this particular example, the answer is yes. (This kernel is called the **Gaussian kernel**, and corresponds

17

to an infinite dimensional feature mapping ϕ.) But more broadly, given some function K, how can we tell if it's a valid kernel; i.e., can we tell if there is some feature mapping ϕ so that $K(x, z) = \phi(x)^T\phi(z)$ for all x, z?

Suppose for now that K is indeed a valid kernel corresponding to some feature mapping ϕ. Now, consider some finite set of m points (not necessarily the training set) $\{x^{(1)}, \ldots, x^{(m)}\}$, and let a square, m-by-m matrix K be defined so that its (i, j)-entry is given by $K_{ij} = K(x^{(i)}, x^{(j)})$. This matrix is called the **Kernel matrix**. Note that we've overloaded the notation and used K to denote both the kernel function $K(x, z)$ and the kernel matrix K, due to their obvious close relationship.

Now, if K is a valid Kernel, then $K_{ij} = K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T\phi(x^{(j)}) = \phi(x^{(j)})^T\phi(x^{(i)}) = K(x^{(j)}, x^{(i)}) = K_{ji}$, and hence K must be symmetric. Moreover, letting $\phi_k(x)$ denote the k-th coordinate of the vector $\phi(x)$, we find that for any vector z, we have

$$\begin{aligned} z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\ &= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\ &= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\ &= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\ &= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\ &\geq 0. \end{aligned}$$

The second-to-last step above used the same trick as you saw in Problem set 1 Q1. Since z was arbitrary, this shows that K is positive semi-definite ($K \geq 0$).

Hence, we've shown that if K is a valid kernel (i.e., if it corresponds to some feature mapping ϕ), then the corresponding Kernel matrix $K \in \mathbb{R}^{m \times m}$ is symmetric positive semidefinite. More generally, this turns out to be not only a necessary, but also a sufficient, condition for K to be a valid kernel (also called a Mercer kernel). The following result is due to Mercer.[5]

[5] Many texts present Mercer's theorem in a slightly more complicated form involving L^2 functions, but when the input attributes take values in $\mathbb{R}^n$, the version given here is equivalent.

18

Theorem (Mercer). Let $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x^{(1)}, \ldots, x^{(m)}\}$, $(m < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite.

Given a function K, apart from trying to find a feature mapping ϕ that corresponds to it, this theorem therefore gives another way of testing if it is a valid kernel. You'll also have a chance to play with these ideas more in problem set 2.

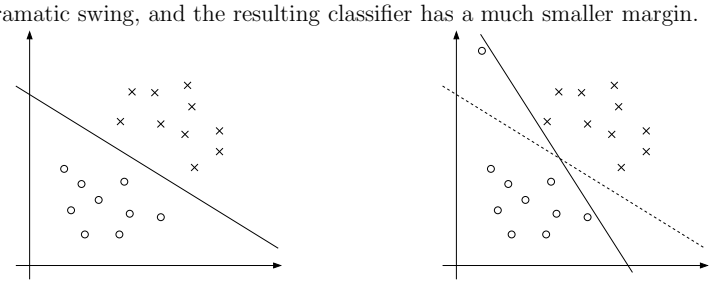
In class, we also briefly talked about a couple of other examples of kernels. For instance, consider the digit recognition problem, in which given an image (16x16 pixels) of a handwritten digit (0-9), we have to figure out which digit it was. Using either a simple polynomial kernel $K(x, z) = (x^T z)^d$ or the Gaussian kernel, SVMs were able to obtain extremely good performance on this problem. This was particularly surprising since the input attributes x were just a 256-dimensional vector of the image pixel intensity values, and the system had no prior knowledge about vision, or even about which pixels are adjacent to which other ones. Another example that we briefly talked about in lecture was that if the objects x that we are trying to classify are strings (say, x is a list of amino acids, which strung together form a protein), then it seems hard to construct a reasonable, "small" set of features for most learning algorithms, especially if different strings have different lengths. However, consider letting $\phi(x)$ be a feature vector that counts the number of occurrences of each length-k substring in x. If we're considering strings of english letters, then there are 26^k such strings. Hence, $\phi(x)$ is a 26^k dimensional vector; even for moderate values of k, this is probably too big for us to efficiently work with. (e.g., $26^4 \approx 460000$.) However, using (dynamic programming-ish) string matching algorithms, it is possible to efficiently compute $K(x, z) = \phi(x)^T \phi(z)$, so that we can now implicitly work in this 26^k-dimensional feature space, but without ever explicitly computing feature vectors in this space.

The application of kernels to support vector machines should already be clear and so we won't dwell too much longer on it here. Keep in mind however that the idea of kernels has significantly broader applicability than SVMs. Specifically, if you have any learning algorithm that you can write in terms of only inner products $\langle x, z \rangle$ between input attribute vectors, then by replacing this with $K(x, z)$ where K is a kernel, you can "magically" allow your algorithm to work efficiently in the high dimensional feature space corresponding to K. For instance, this kernel trick can be applied with the perceptron to to derive a kernel perceptron algorithm. Many of the

algorithms that we'll see later in this class will also be amenable to this method, which has come to be known as the "kernel trick."

8 Regularization and the non-separable case

The derivation of the SVM as presented so far assumed that the data is linearly separable. While mapping data to a high dimensional feature space via ϕ does generally increase the likelihood that the data is separable, we can't guarantee that it always will be so. Also, in some cases it is not clear that finding a separating hyperplane is exactly what we'd want to do, since that might be susceptible to outliers. For instance, the left figure below shows an optimal margin classifier, and when a single outlier is added in the upper-left region (right figure), it causes the decision boundary to make a dramatic swing, and the resulting classifier has a much smaller margin.



To make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers, we reformulate our optimization (using ℓ_1 **regularization**) as follows:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

Thus, examples are now permitted to have (functional) margin less than 1, and if an example has functional margin $1 - \xi_i$ (with $\xi > 0$), we would pay a cost of the objective function being increased by $C\xi_i$. The parameter C controls the relative weighting between the twin goals of making the $||w||^2$ small (which we saw earlier makes the margin large) and of ensuring that most examples have functional margin at least 1.

As before, we can form the Lagrangian:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(x^T w + b) - 1 + \xi_i \right] - \sum_{i=1}^{m} r_i \xi_i.$$

Here, the α_i's and r_i's are our Lagrange multipliers (constrained to be ≥ 0). We won't go through the derivation of the dual again in detail, but after setting the derivatives with respect to w and b to zero as before, substituting them back in, and simplifying, we obtain the following dual form of the problem:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

As before, we also have that w can be expressed in terms of the α_i's as given in Equation (9), so that after solving the dual problem, we can continue to use Equation (13) to make our predictions. Note that, somewhat surprisingly, in adding ℓ_1 regularization, the only change to the dual problem is that what was originally a constraint that $0 \leq \alpha_i$ has now become $0 \leq \alpha_i \leq C$. The calculation for b^* also has to be modified (Equation 11 is no longer valid); see the comments in the next section/Platt's paper.

Also, the KKT dual-complementarity conditions (which in the next section will be useful for testing for the convergence of the SMO algorithm) are:

$$\alpha_i = 0 \quad \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 \tag{14}$$

$$\alpha_i = C \quad \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \leq 1 \tag{15}$$

$$0 < \alpha_i < C \quad \Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1. \tag{16}$$

Now, all that remains is to give an algorithm for actually solving the dual problem, which we will do in the next section.

9 The SMO algorithm

The SMO (sequential minimal optimization) algorithm, due to John Platt, gives an efficient way of solving the dual problem arising from the derivation

of the SVM. Partly to motivate the SMO algorithm, and partly because it's interesting in its own right, let's first take another digression to talk about the coordinate ascent algorithm.

9.1 Coordinate ascent

Consider trying to solve the unconstrained optimization problem

$$\max_{\alpha} W(\alpha_1, \alpha_2, \ldots, \alpha_m).$$

Here, we think of W as just some function of the parameters α_i's, and for now ignore any relationship between this problem and SVMs. We've already seen two optimization algorithms, gradient ascent and Newton's method. The new algorithm we're going to consider here is called **coordinate ascent**:

Loop until convergence: {

For $i = 1, \ldots, m$, {

$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$.

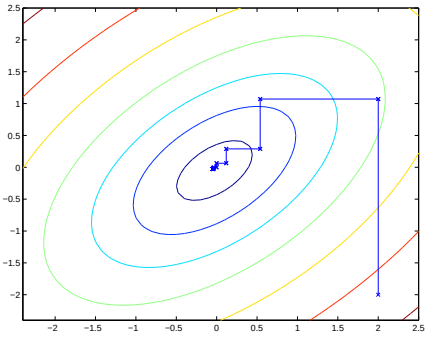
}

}

Thus, in the innermost loop of this algorithm, we will hold all the variables except for some α_i fixed, and reoptimize W with respect to just the parameter α_i. In the version of this method presented here, the inner-loop reoptimizes the variables in order $\alpha_1, \alpha_2, \ldots, \alpha_m, \alpha_1, \alpha_2, \ldots$. (A more sophisticated version might choose other orderings; for instance, we may choose the next variable to update according to which one we expect to allow us to make the largest increase in $W(\alpha)$.)

When the function W happens to be of such a form that the "arg max" in the inner loop can be performed efficiently, then coordinate ascent can be a fairly efficient algorithm. Here's a picture of coordinate ascent in action:

22



The ellipses in the figure are the contours of a quadratic function that we want to optimize. Coordinate ascent was initialized at $(2, -2)$, and also plotted in the figure is the path that it took on its way to the global maximum. Notice that on each step, coordinate ascent takes a step that's parallel to one of the axes, since only one variable is being optimized at a time.

9.2 SMO

We close off the discussion of SVMs by sketching the derivation of the SMO algorithm. Some details will be left to the homework, and for others you may refer to the paper excerpt handed out in class.

Here's the (dual) optimization problem that we want to solve:

$$\max_\alpha \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \tag{17}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \tag{18}$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0. \tag{19}$$

Let's say we have set of α_i's that satisfy the constraints (18-19). Now, suppose we want to hold $\alpha_2, \ldots, \alpha_m$ fixed, and take a coordinate ascent step and reoptimize the objective with respect to α_1. Can we make any progress? The answer is no, because the constraint (19) ensures that

$$\alpha_1 y^{(1)} = -\sum_{i=2}^{m} \alpha_i y^{(i)}.$$

23

Or, by multiplying both sides by $y^{(1)}$, we equivalently have

$$\alpha_1 = -y^{(1)} \sum_{i=2}^{m} \alpha_i y^{(i)}.$$

(This step used the fact that $y^{(1)} \in \{-1, 1\}$, and hence $(y^{(1)})^2 = 1$.) Hence, α_1 is exactly determined by the other α_i's, and if we were to hold $\alpha_2, \ldots, \alpha_m$ fixed, then we can't make any change to α_1 without violating the constraint (19) in the optimization problem.

Thus, if we want to update some subject of the α_i's, we must update at least two of them simultaneously in order to keep satisfying the constraints. This motivates the SMO algorithm, which simply does the following:

Repeat till convergence {

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Reoptimize $W(\alpha)$ with respect to α_i and α_j, while holding all the other α_k's $(k \neq i, j)$ fixed.

}

To test for convergence of this algorithm, we can check whether the KKT conditions (Equations 14-16) are satisfied to within some *tol*. Here, *tol* is the convergence tolerance parameter, and is typically set to around 0.01 to 0.001. (See the paper and pseudocode for details.)

The key reason that SMO is an efficient algorithm is that the update to α_i, α_j can be computed very efficiently. Let's now briefly sketch the main ideas for deriving the efficient update.

Let's say we currently have some setting of the α_i's that satisfy the constraints (18-19), and suppose we've decided to hold $\alpha_3, \ldots, \alpha_m$ fixed, and want to reoptimize $W(\alpha_1, \alpha_2, \ldots, \alpha_m)$ with respect to α_1 and α_2 (subject to the constraints). From (19), we require that

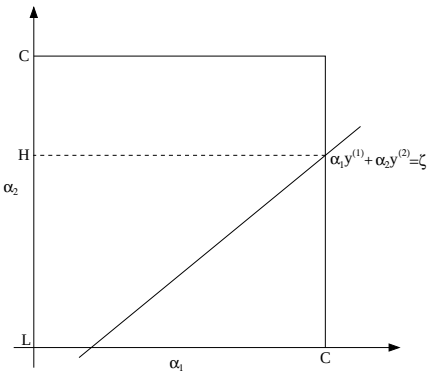
$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{m} \alpha_i y^{(i)}.$$

Since the right hand side is fixed (as we've fixed $\alpha_3, \ldots \alpha_m$), we can just let it be denoted by some constant ζ:

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta. \tag{20}$$

We can thus picture the constraints on α_1 and α_2 as follows:

24



From the constraints (18), we know that α_1 and α_2 must lie within the box $[0, C] \times [0, C]$ shown. Also plotted is the line $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$, on which we know α_1 and α_2 must lie. Note also that, from these constraints, we know $L \leq \alpha_2 \leq H$; otherwise, (α_1, α_2) can't simultaneously satisfy both the box and the straight line constraint. In this example, $L = 0$. But depending on what the line $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$ looks like, this won't always necessarily be the case; but more generally, there will be some lower-bound L and some upper-bound H on the permissable values for α_2 that will ensure that α_1, α_2 lie within the box $[0, C] \times [0, C]$.

Using Equation (20), we can also write α_1 as a function of α_2:

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}.$$

(Check this derivation yourself; we again used the fact that $y^{(1)} \in \{-1, 1\}$ so that $(y^{(1)})^2 = 1$.) Hence, the objective $W(\alpha)$ can be written

$$W(\alpha_1, \alpha_2, \ldots, \alpha_m) = W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \ldots, \alpha_m).$$

Treating $\alpha_3, \ldots, \alpha_m$ as constants, you should be able to verify that this is just some quadratic function in α_2. I.e., this can also be expressed in the form $a\alpha_2^2 + b\alpha_2 + c$ for some appropriate a, b, and c. If we ignore the "box" constraints (18) (or, equivalently, that $L \leq \alpha_2 \leq H$), then we can easily maximize this quadratic function by setting its derivative to zero and solving. We'll let $\alpha_2^{new,unclipped}$ denote the resulting value of α_2. You should also be able to convince yourself that if we had instead wanted to maximize W with respect to α_2 but subject to the box constraint, then we can find the resulting value optimal simply by taking $\alpha_2^{new,unclipped}$ and "clipping" it to lie in the

25

$[L, H]$ interval, to get

$$\alpha_2^{new} = \begin{cases} H & \text{if } \alpha_2^{new,unclipped} > H \\ \alpha_2^{new,unclipped} & \text{if } L \leq \alpha_2^{new,unclipped} \leq H \\ L & \text{if } \alpha_2^{new,unclipped} < L \end{cases}$$

Finally, having found the α_2^{new}, we can use Equation (20) to go back and find the optimal value of α_1^{new}.

There're a couple more details that are quite easy but that we'll leave you to read about yourself in Platt's paper: One is the choice of the heuristics used to select the next α_i, α_j to update; the other is how to update b as the SMO algorithm is run.

Annexe C

Word2vect

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling of the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling.

An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the meanings of "Canada" and "Air" cannot be easily combined to obtain "Air Canada". Motivated by this example, we present a simple method for finding phrases in text, and show that learning good vector representations for millions of phrases is possible.

1 Introduction

Distributed representations of words in a vector space help learning algorithms to achieve better performance in natural language processing tasks by grouping similar words. One of the earliest use of word representations dates back to 1986 due to Rumelhart, Hinton, and Williams [13]. This idea has since been applied to statistical language modeling with considerable success [1]. The follow up work includes applications to automatic speech recognition and machine translation [14, 7], and a wide range of NLP tasks [2, 20, 15, 3, 18, 19, 9].

Recently, Mikolov et al. [8] introduced the Skip-gram model, an efficient method for learning high-quality vector representations of words from large amounts of unstructured text data. Unlike most of the previously used neural network architectures for learning word vectors, training of the Skip-gram model (see Figure 1) does not involve dense matrix multiplications. This makes the training extremely efficient: an optimized single-machine implementation can train on more than 100 billion words in one day.

The word representations computed using neural networks are very interesting because the learned vectors explicitly encode many linguistic regularities and patterns. Somewhat surprisingly, many of these patterns can be represented as linear translations. For example, the result of a vector calculation vec("Madrid") - vec("Spain") + vec("France") is closer to vec("Paris") than to any other word vector [9, 8].

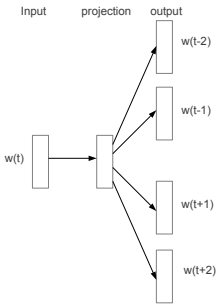


Figure 1: The Skip-gram model architecture. The training objective is to learn word vector representations that are good at predicting the nearby words.

In this paper we present several extensions of the original Skip-gram model. We show that subsampling of frequent words during training results in a significant speedup (around 2x - 10x), and improves accuracy of the representations of less frequent words. In addition, we present a simplified variant of Noise Contrastive Estimation (NCE) [4] for training the Skip-gram model that results in faster training and better vector representations for frequent words, compared to more complex hierarchical softmax that was used in the prior work [8].

Word representations are limited by their inability to represent idiomatic phrases that are not compositions of the individual words. For example, “Boston Globe” is a newspaper, and so it is not a natural combination of the meanings of “Boston” and “Globe”. Therefore, using vectors to represent the whole phrases makes the Skip-gram model considerably more expressive. Other techniques that aim to represent meaning of sentences by composing the word vectors, such as the recursive autoencoders [15], would also benefit from using phrase vectors instead of the word vectors.

The extension from word based to phrase based models is relatively simple. First we identify a large number of phrases using a data-driven approach, and then we treat the phrases as individual tokens during the training. To evaluate the quality of the phrase vectors, we developed a test set of analogical reasoning tasks that contains both words and phrases. A typical analogy pair from our test set is “Montreal”:“Montreal Canadiens”::“Toronto”:“Toronto Maple Leafs”. It is considered to have been answered correctly if the nearest representation to vec(“Montreal Canadiens”) - vec(“Montreal”) + vec(“Toronto”) is vec(“Toronto Maple Leafs”).

Finally, we describe another interesting property of the Skip-gram model. We found that simple vector addition can often produce meaningful results. For example, vec(“Russia”) + vec(“river”) is close to vec(“Volga River”), and vec(“Germany”) + vec(“capital”) is close to vec(“Berlin”). This compositionality suggests that a non-obvious degree of language understanding can be obtained by using basic mathematical operations on the word vector representations.

2 The Skip-gram Model

The training objective of the Skip-gram model is to find word representations that are useful for predicting the surrounding words in a sentence or a document. More formally, given a sequence of training words $w_1, w_2, w_3, \ldots, w_T$, the objective of the Skip-gram model is to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{1}$$

where c is the size of the training context (which can be a function of the center word w_t). Larger c results in more training examples and thus can lead to a higher accuracy, at the expense of the

training time. The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^\top v_{w_I}\right)} \tag{2}$$

where v_w and v'_w are the "input" and "output" vector representations of w, and W is the number of words in the vocabulary. This formulation is impractical because the cost of computing $\nabla \log p(w_O|w_I)$ is proportional to W, which is often large (10^5–10^7 terms).

2.1 Hierarchical Softmax

A computationally efficient approximation of the full softmax is the hierarchical softmax. In the context of neural network language models, it was first introduced by Morin and Bengio [12]. The main advantage is that instead of evaluating W output nodes in the neural network to obtain the probability distribution, it is needed to evaluate only about $\log_2(W)$ nodes.

The hierarchical softmax uses a binary tree representation of the output layer with the W words as its leaves and, for each node, explicitly represents the relative probabilities of its child nodes. These define a random walk that assigns probabilities to words.

More precisely, each word w can be reached by an appropriate path from the root of the tree. Let $n(w, j)$ be the j-th node on the path from the root to w, and let $L(w)$ be the length of this path, so $n(w, 1) = \text{root}$ and $n(w, L(w)) = w$. In addition, for any inner node n, let $\text{ch}(n)$ be an arbitrary fixed child of n and let $[\![x]\!]$ be 1 if x is true and -1 otherwise. Then the hierarchical softmax defines $p(w_O|w_I)$ as follows:

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\left([\![n(w, j+1) = \text{ch}(n(w, j))]\!] \cdot {v'_{n(w,j)}}^\top v_{w_I}\right) \tag{3}$$

where $\sigma(x) = 1/(1 + \exp(-x))$. It can be verified that $\sum_{w=1}^{W} p(w|w_I) = 1$. This implies that the cost of computing $\log p(w_O|w_I)$ and $\nabla \log p(w_O|w_I)$ is proportional to $L(w_O)$, which on average is no greater than $\log W$. Also, unlike the standard softmax formulation of the Skip-gram which assigns two representations v_w and v'_w to each word w, the hierarchical softmax formulation has one representation v_w for each word w and one representation v'_n for every inner node n of the binary tree.

The structure of the tree used by the hierarchical softmax has a considerable effect on the performance. Mnih and Hinton explored a number of methods for constructing the tree structure and the effect on both the training time and the resulting model accuracy [10]. In our work we use a binary Huffman tree, as it assigns short codes to the frequent words which results in fast training. It has been observed before that grouping words together by their frequency works well as a very simple speedup technique for the neural network based language models [5, 8].

2.2 Negative Sampling

An alternative to the hierarchical softmax is Noise Contrastive Estimation (NCE), which was introduced by Gutmann and Hyvarinen [4] and applied to language modeling by Mnih and Teh [11]. NCE posits that a good model should be able to differentiate data from noise by means of logistic regression. This is similar to hinge loss used by Collobert and Weston [2] who trained the models by ranking the data above noise.

While NCE can be shown to approximately maximize the log probability of the softmax, the Skip-gram model is only concerned with learning high-quality vector representations, so we are free to simplify NCE as long as the vector representations retain their quality. We define Negative sampling (NEG) by the objective

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I})\right] \tag{4}$$

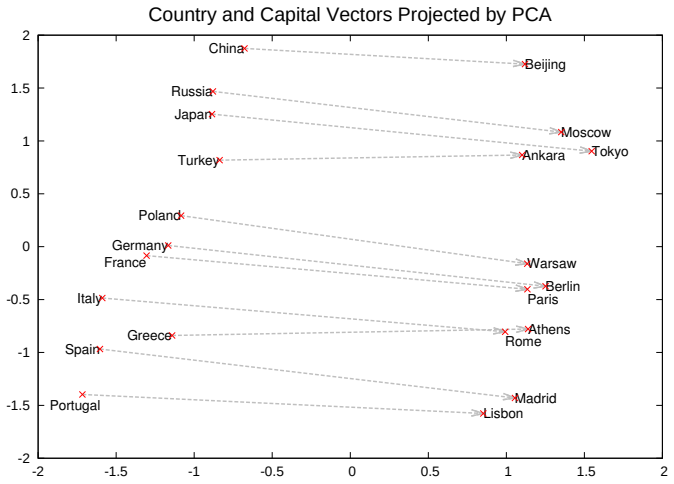


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

which is used to replace every $\log P(w_O|w_I)$ term in the Skip-gram objective. Thus the task is to distinguish the target word w_O from draws from the noise distribution $P_n(w)$ using logistic regression, where there are k negative samples for each data sample. Our experiments indicate that values of k in the range 5–20 are useful for small training datasets, while for large datasets the k can be as small as 2–5. The main difference between the Negative sampling and NCE is that NCE needs both samples and the numerical probabilities of the noise distribution, while Negative sampling uses only samples. And while NCE approximately maximizes the log probability of the softmax, this property is not important for our application.

Both NCE and NEG have the noise distribution $P_n(w)$ as a free parameter. We investigated a number of choices for $P_n(w)$ and found that the unigram distribution $U(w)$ raised to the $3/4$rd power (i.e., $U(w)^{3/4}/Z$) outperformed significantly the unigram and the uniform distributions, for both NCE and NEG on every task we tried including language modeling (not reported here).

2.3 Subsampling of Frequent Words

In very large corpora, the most frequent words can easily occur hundreds of millions of times (e.g., “in”, “the”, and “a”). Such words usually provide less information value than the rare words. For example, while the Skip-gram model benefits from observing the co-occurrences of “France” and “Paris”, it benefits much less from observing the frequent co-occurrences of “France” and “the”, as nearly every word co-occurs frequently within a sentence with “the”. This idea can also be applied in the opposite direction; the vector representations of frequent words do not change significantly after training on several million examples.

To counter the imbalance between the rare and frequent words, we used a simple subsampling approach: each word w_i in the training set is discarded with probability computed by the formula

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{5}$$

Method	Time [min]	Syntactic [%]	Semantic [%]	Total accuracy [%]
NEG-5	38	63	54	59
NEG-15	97	63	58	**61**
HS-Huffman	41	53	40	47
NCE-5	38	60	45	53
The following results use 10^{-5} subsampling				
NEG-5	14	61	58	60
NEG-15	36	61	61	**61**
HS-Huffman	21	52	59	55

Table 1: Accuracy of various Skip-gram 300-dimensional models on the analogical reasoning task as defined in [8]. NEG-k stands for Negative Sampling with k negative samples for each positive sample; NCE stands for Noise Contrastive Estimation and HS-Huffman stands for the Hierarchical Softmax with the frequency-based Huffman codes.

where $f(w_i)$ is the frequency of word w_i and t is a chosen threshold, typically around 10^{-5}. We chose this subsampling formula because it aggressively subsamples words whose frequency is greater than t while preserving the ranking of the frequencies. Although this subsampling formula was chosen heuristically, we found it to work well in practice. It accelerates learning and even significantly improves the accuracy of the learned vectors of the rare words, as will be shown in the following sections.

3 Empirical Results

In this section we evaluate the Hierarchical Softmax (HS), Noise Contrastive Estimation, Negative Sampling, and subsampling of the training words. We used the analogical reasoning task[1] introduced by Mikolov et al. [8]. The task consists of analogies such as "Germany" : "Berlin" :: "France" : ?, which are solved by finding a vector $\mathbf{x}$ such that vec($\mathbf{x}$) is closest to vec("Berlin") - vec("Germany") + vec("France") according to the cosine distance (we discard the input words from the search). This specific example is considered to have been answered correctly if $\mathbf{x}$ is "Paris". The task has two broad categories: the syntactic analogies (such as "quick" : "quickly" :: "slow" : "slowly") and the semantic analogies, such as the country to capital city relationship.

For training the Skip-gram models, we have used a large dataset consisting of various news articles (an internal Google dataset with one billion words). We discarded from the vocabulary all words that occurred less than 5 times in the training data, which resulted in a vocabulary of size 692K. The performance of various Skip-gram models on the word analogy test set is reported in Table 1. The table shows that Negative Sampling outperforms the Hierarchical Softmax on the analogical reasoning task, and has even slightly better performance than the Noise Contrastive Estimation. The subsampling of the frequent words improves the training speed several times and makes the word representations significantly more accurate.

It can be argued that the linearity of the skip-gram model makes its vectors more suitable for such linear analogical reasoning, but the results of Mikolov et al. [8] also show that the vectors learned by the standard sigmoidal recurrent neural networks (which are highly non-linear) improve on this task significantly as the amount of the training data increases, suggesting that non-linear models also have a preference for a linear structure of the word representations.

4 Learning Phrases

As discussed earlier, many phrases have a meaning that is not a simple composition of the meanings of its individual words. To learn vector representation for phrases, we first find words that appear frequently together, and infrequently in other contexts. For example, "New York Times" and "Toronto Maple Leafs" are replaced by unique tokens in the training data, while a bigram "this is" will remain unchanged.

[1] `code.google.com/p/word2vec/source/browse/trunk/questions-words.txt`

Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Canadiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon

Table 2: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples). The goal is to compute the fourth phrase using the first three. Our best model achieved an accuracy of 72% on this dataset.

This way, we can form many reasonable phrases without greatly increasing the size of the vocabulary; in theory, we can train the Skip-gram model using all n-grams, but that would be too memory intensive. Many techniques have been previously developed to identify phrases in the text; however, it is out of scope of our work to compare them. We decided to use a simple data-driven approach, where phrases are formed based on the unigram and bigram counts, using

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}. \tag{6}$$

The δ is used as a discounting coefficient and prevents too many phrases consisting of very infrequent words to be formed. The bigrams with score above the chosen threshold are then used as phrases. Typically, we run 2-4 passes over the training data with decreasing threshold value, allowing longer phrases that consists of several words to be formed. We evaluate the quality of the phrase representations using a new analogical reasoning task that involves phrases. Table 2 shows examples of the five categories of analogies used in this task. This dataset is publicly available on the web[2].

4.1 Phrase Skip-Gram Results

Starting with the same news data as in the previous experiments, we first constructed the phrase based training corpus and then we trained several Skip-gram models using different hyperparameters. As before, we used vector dimensionality 300 and context size 5. This setting already achieves good performance on the phrase dataset, and allowed us to quickly compare the Negative Sampling and the Hierarchical Softmax, both with and without subsampling of the frequent tokens. The results are summarized in Table 3.

The results show that while Negative Sampling achieves a respectable accuracy even with $k = 5$, using $k = 15$ achieves considerably better performance. Surprisingly, while we found the Hierarchical Softmax to achieve lower performance when trained without subsampling, it became the best performing method when we downsampled the frequent words. This shows that the subsampling can result in faster training and can also improve accuracy, at least in some cases.

[2] `code.google.com/p/word2vec/source/browse/trunk/questions-phrases.txt`

Method	Dimensionality	No subsampling [%]	10^{-5} subsampling [%]
NEG-5	300	24	27
NEG-15	300	27	42
HS-Huffman	300	19	**47**

Table 3: Accuracies of the Skip-gram models on the phrase analogy dataset. The models were trained on approximately one billion words from the news dataset.

	NEG-15 with 10^{-5} subsampling	HS with 10^{-5} subsampling
Vasco de Gama	Lingsugur	Italian explorer
Lake Baikal	Great Rift Valley	Aral Sea
Alan Bean	Rebbeca Naomi	moonwalker
Ionian Sea	Ruegen	Ionian Islands
chess master	chess grandmaster	Garry Kasparov

Table 4: Examples of the closest entities to the given short phrases, using two different models.

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

To maximize the accuracy on the phrase analogy task, we increased the amount of the training data by using a dataset with about 33 billion words. We used the hierarchical softmax, dimensionality of 1000, and the entire sentence for the context. This resulted in a model that reached an accuracy of **72%**. We achieved lower accuracy 66% when we reduced the size of the training dataset to 6B words, which suggests that the large amount of the training data is crucial.

To gain further insight into how different the representations learned by different models are, we did inspect manually the nearest neighbours of infrequent phrases using various models. In Table 4, we show a sample of such comparison. Consistently with the previous results, it seems that the best representations of phrases are learned by a model with the hierarchical softmax and subsampling.

5 Additive Compositionality

We demonstrated that the word and phrase representations learned by the Skip-gram model exhibit a linear structure that makes it possible to perform precise analogical reasoning using simple vector arithmetics. Interestingly, we found that the Skip-gram representations exhibit another kind of linear structure that makes it possible to meaningfully combine words by an element-wise addition of their vector representations. This phenomenon is illustrated in Table 5.

The additive property of the vectors can be explained by inspecting the training objective. The word vectors are in a linear relationship with the inputs to the softmax nonlinearity. As the word vectors are trained to predict the surrounding words in the sentence, the vectors can be seen as representing the distribution of the context in which a word appears. These values are related logarithmically to the probabilities computed by the output layer, so the sum of two word vectors is related to the product of the two context distributions. The product works here as the AND function: words that are assigned high probabilities by both word vectors will have high probability, and the other words will have low probability. Thus, if "Volga River" appears frequently in the same sentence together with the words "Russian" and "river", the sum of these two word vectors will result in such a feature vector that is close to the vector of "Volga River".

6 Comparison to Published Word Representations

Many authors who previously worked on the neural network based representations of words have published their resulting models for further use and comparison: amongst the most well known authors are Collobert and Weston [2], Turian et al. [17], and Mnih and Hinton [10]. We downloaded their word vectors from the web[3]. Mikolov et al. [8] have already evaluated these word representations on the word analogy task, where the Skip-gram models achieved the best performance with a huge margin.

[3] `http://metaoptimize.com/projects/wordreprs/`

Model (training time)	Redmond	Havel	ninjutsu	graffiti	capitulate
Collobert (50d) (2 months)	conyers lubbock keene	plauen dzerzhinsky osterreich	reiki kohona karate	cheesecake gossip dioramas	abdicate accede rearm
Turian (200d) (few weeks)	McCarthy Alston Cousins	Jewell Arzu Ovitz	- - -	gunfire emotion impunity	- - -
Mnih (100d) (7 days)	Podhurst Harlang Agarwal	Pontiff Pinochet Rodionov	- - -	anaesthetics monkeys Jews	Mavericks planning hesitated
Skip-Phrase (1000d, 1 day)	Redmond Wash. Redmond Washington Microsoft	Vaclav Havel president Vaclav Havel Velvet Revolution	ninja martial arts swordsmanship	spray paint grafitti taggers	capitulation capitulated capitulating

Table 6: Examples of the closest tokens given various well known models and the Skip-gram model trained on phrases using over 30 billion training words. An empty cell means that the word was not in the vocabulary.

To give more insight into the difference of the quality of the learned vectors, we provide empirical comparison by showing the nearest neighbours of infrequent words in Table 6. These examples show that the big Skip-gram model trained on a large corpus visibly outperforms all the other models in the quality of the learned representations. This can be attributed in part to the fact that this model has been trained on about 30 billion words, which is about two to three orders of magnitude more data than the typical size used in the prior work. Interestingly, although the training set is much larger, the training time of the Skip-gram model is just a fraction of the time complexity required by the previous model architectures.

7 Conclusion

This work has several key contributions. We show how to train distributed representations of words and phrases with the Skip-gram model and demonstrate that these representations exhibit linear structure that makes precise analogical reasoning possible. The techniques introduced in this paper can be used also for training the continuous bag-of-words model introduced in [8].

We successfully trained models on several orders of magnitude more data than the previously published models, thanks to the computationally efficient model architecture. This results in a great improvement in the quality of the learned word and phrase representations, especially for the rare entities. We also found that the subsampling of the frequent words results in both faster training and significantly better representations of uncommon words. Another contribution of our paper is the Negative sampling algorithm, which is an extremely simple training method that learns accurate representations especially for frequent words.

The choice of the training algorithm and the hyper-parameter selection is a task specific decision, as we found that different problems have different optimal hyperparameter configurations. In our experiments, the most crucial decisions that affect the performance are the choice of the model architecture, the size of the vectors, the subsampling rate, and the size of the training window.

A very interesting result of this work is that the word vectors can be somewhat meaningfully combined using just simple vector addition. Another approach for learning representations of phrases presented in this paper is to simply represent the phrases with a single token. Combination of these two approaches gives a powerful yet simple way how to represent longer pieces of text, while having minimal computational complexity. Our work can thus be seen as complementary to the existing approach that attempts to represent phrases using recursive matrix-vector operations [16].

We made the code for training the word and phrase vectors based on the techniques described in this paper available as an open-source project[4].

[4] `code.google.com/p/word2vec`

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[2] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[3] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*, 513–520, 2011.

[4] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13:307–361, 2012.

[5] Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[6] Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukas Burget and Jan Cernocky. Strategies for Training Large Scale Neural Network Language Models. In Proc. *Automatic Speech Recognition and Understanding*, 2011.

[7] Tomas Mikolov. Statistical Language Models Based on Neural Networks. *PhD thesis, PhD Thesis, Brno University of Technology*, 2012.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *ICLR Workshop*, 2013.

[9] Tomas Mikolov, Wen-tau Yih and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of NAACL HLT*, 2013.

[10] Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. *Advances in neural information processing systems*, 21:1081–1088, 2009.

[11] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

[12] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the international workshop on artificial intelligence and statistics*, pages 246–252, 2005.

[13] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[14] Holger Schwenk. Continuous space language models. *Computer Speech and Language*, vol. 21, 2007.

[15] Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, volume 2, 2011.

[16] Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2012.

[17] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[18] Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. In *Journal of Artificial Intelligence Research*, 37:141-188, 2010.

[19] Peter D. Turney. Distributional semantics beyond words: Supervised learning of analogy and paraphrase. In *Transactions of the Association for Computational Linguistics (TACL)*, 353–366, 2013.

[20] Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2764–2770. AAAI Press, 2011.